

# Loop freedom of the (untimed) AODV routing protocol

Timothy Bourke<sup>1</sup>

Peter Höfner<sup>2</sup>

December 17, 2016

<sup>1</sup>Inria, École normale supérieure, and NICTA

<sup>2</sup>NICTA and Computer Science and Engineering, UNSW

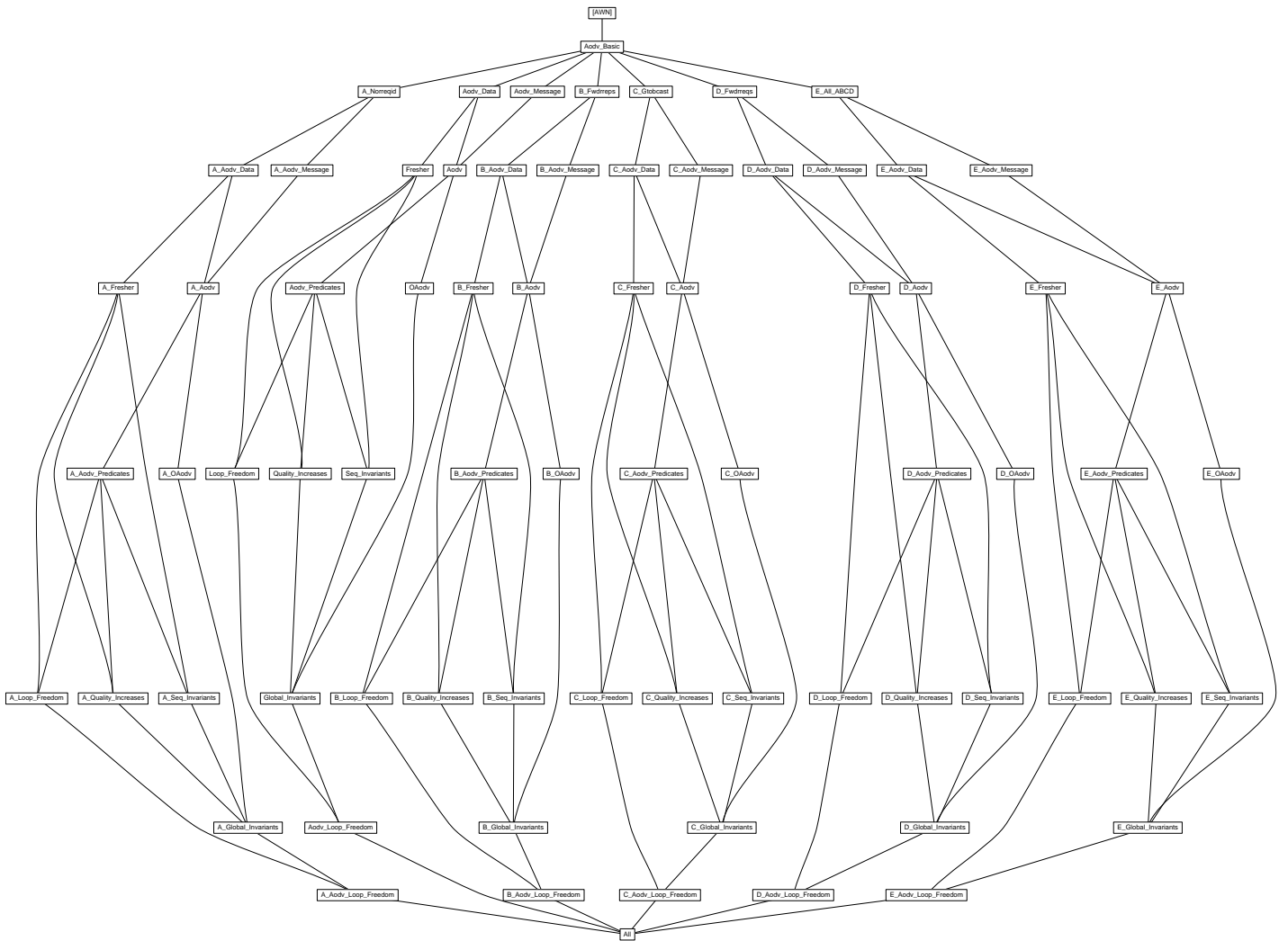
## Abstract

The Ad hoc On-demand Distance Vector (AODV) routing protocol [6] allows the nodes in a Mobile Ad hoc Network (MANET) or a Wireless Mesh Network (WMN) to know where to forward data packets. Such a protocol is ‘loop free’ if it never leads to routing decisions that forward packets in circles.

This development mechanises an existing pen-and-paper proof of loop freedom of AODV [4]. The protocol is modelled in the Algebra of Wireless Networks (AWN), which is the subject of an earlier paper [3] and mechanization [1]. The proof relies on a novel compositional approach for lifting invariants to networks of nodes.

We exploit the mechanization to analyse several variants of AODV and show that Isabelle/HOL can re-establish most proof obligations automatically and identify exactly the steps that are no longer valid. Each of the variants is essentially a modified copy of the main development.

Further documentation is available in [2].



# Contents

0.1	Basic data types and constants	5
0.2	Predicates and functions used in the AODV model	5
0.2.1	Sequence Numbers	5
0.2.2	Modelling Routes	6
0.2.3	Routing Tables	6
0.2.4	Updating Routing Tables	9
0.2.5	Route Requests	16
0.2.6	Queued Packets	16
0.2.7	Comparison with the original technical report	17
0.3	AODV protocol messages	17
0.4	The AODV protocol	18
0.4.1	Data state	18
0.4.2	Auxilliary message handling definitions	20
0.4.3	The protocol process	21
0.5	Invariant assumptions and properties	26
0.6	Quality relations between routes	28
0.6.1	Net sequence numbers	28
0.6.2	Comparing routes	30
0.6.3	Comparing routing tables	31
0.6.4	Strictly comparing routing tables	34
0.7	Invariant proofs on individual processes	36
0.8	The quality increases predicate	40
0.9	The ‘open’ AODV model	43
0.10	Global invariant proofs over sequential processes	44
0.11	Routing graphs and loop freedom	47
0.12	Lift and transfer invariants to show loop freedom	48
0.12.1	Lift to parallel processes with queues	48
0.12.2	Lift to nodes	49
0.12.3	Lift to partial networks	49
0.12.4	Lift to closed networks	50
0.12.5	Transfer into the standard model	50
0.12.6	Loop freedom of AODV	50
<b>1</b>	<b>Variant A: Skipping the RREQ ID</b>	<b>51</b>
1.1	Predicates and functions used in the AODV model	51
1.1.1	Sequence Numbers	51
1.1.2	Modelling Routes	51
1.1.3	Routing Tables	52
1.1.4	Updating Routing Tables	54
1.1.5	Route Requests	61
1.1.6	Queued Packets	62
1.1.7	Comparison with the original technical report	63
1.2	AODV protocol messages	63
1.3	The AODV protocol	64
1.3.1	Data state	64
1.3.2	Auxilliary message handling definitions	65
1.3.3	The protocol process	67

1.4	Invariant assumptions and properties . . . . .	72
1.5	Quality relations between routes . . . . .	74
1.5.1	Net sequence numbers . . . . .	74
1.5.2	Comparing routes . . . . .	76
1.5.3	Comparing routing tables . . . . .	76
1.5.4	Strictly comparing routing tables . . . . .	80
1.6	Invariant proofs on individual processes . . . . .	82
1.7	The quality increases predicate . . . . .	86
1.8	The ‘open’ AODV model . . . . .	89
1.9	Global invariant proofs over sequential processes . . . . .	90
1.10	Routing graphs and loop freedom . . . . .	93
1.11	Lift and transfer invariants to show loop freedom . . . . .	94
1.11.1	Lift to nodes . . . . .	94
1.11.2	Lift to partial networks . . . . .	95
1.11.3	Lift to closed networks . . . . .	95
1.11.4	Transfer into the standard model . . . . .	95
1.11.5	Loop freedom of AODV . . . . .	96
<b>2</b>	<b>Variant B: Forwarding the Route Reply</b> . . . . .	<b>97</b>
2.1	Predicates and functions used in the AODV model . . . . .	97
2.1.1	Sequence Numbers . . . . .	97
2.1.2	Modelling Routes . . . . .	97
2.1.3	Routing Tables . . . . .	98
2.1.4	Updating Routing Tables . . . . .	100
2.1.5	Route Requests . . . . .	108
2.1.6	Queued Packets . . . . .	108
2.1.7	Comparison with the original technical report . . . . .	109
2.2	AODV protocol messages . . . . .	109
2.3	The AODV protocol . . . . .	110
2.3.1	Data state . . . . .	110
2.3.2	Auxilliary message handling definitions . . . . .	111
2.3.3	The protocol process . . . . .	113
2.4	Invariant assumptions and properties . . . . .	118
2.5	Quality relations between routes . . . . .	120
2.5.1	Net sequence numbers . . . . .	120
2.5.2	Comparing routes . . . . .	122
2.5.3	Comparing routing tables . . . . .	123
2.5.4	Strictly comparing routing tables . . . . .	126
2.6	Invariant proofs on individual processes . . . . .	128
2.7	The quality increases predicate . . . . .	132
2.8	The ‘open’ AODV model . . . . .	135
2.9	Global invariant proofs over sequential processes . . . . .	136
2.10	Routing graphs and loop freedom . . . . .	139
2.11	Lift and transfer invariants to show loop freedom . . . . .	140
2.11.1	Lift to parallel processes with queues . . . . .	140
2.11.2	Lift to nodes . . . . .	141
2.11.3	Lift to partial networks . . . . .	141
2.11.4	Lift to closed networks . . . . .	141
2.11.5	Transfer into the standard model . . . . .	142
2.11.6	Loop freedom of AODV . . . . .	142
<b>3</b>	<b>Variant C: From Groupcast to Broadcast</b> . . . . .	<b>143</b>
3.1	Predicates and functions used in the AODV model . . . . .	143
3.1.1	Sequence Numbers . . . . .	143
3.1.2	Modelling Routes . . . . .	143
3.1.3	Routing Tables . . . . .	144
3.1.4	Updating Routing Tables . . . . .	146

3.1.5	Route Requests . . . . .	152
3.1.6	Queued Packets . . . . .	152
3.1.7	Comparison with the original technical report . . . . .	153
3.2	AODV protocol messages . . . . .	153
3.3	The AODV protocol . . . . .	154
3.3.1	Data state . . . . .	154
3.3.2	Auxilliary message handling definitions . . . . .	155
3.3.3	The protocol process . . . . .	157
3.4	Invariant assumptions and properties . . . . .	162
3.5	Quality relations between routes . . . . .	164
3.5.1	Net sequence numbers . . . . .	164
3.5.2	Comparing routes . . . . .	165
3.5.3	Comparing routing tables . . . . .	166
3.5.4	Strictly comparing routing tables . . . . .	169
3.6	Invariant proofs on individual processes . . . . .	172
3.7	The quality increases predicate . . . . .	175
3.8	The ‘open’ AODV model . . . . .	178
3.9	Global invariant proofs over sequential processes . . . . .	179
3.10	Routing graphs and loop freedom . . . . .	182
3.11	Lift and transfer invariants to show loop freedom . . . . .	183
3.11.1	Lift to parallel processes with queues . . . . .	183
3.11.2	Lift to nodes . . . . .	183
3.11.3	Lift to partial networks . . . . .	184
3.11.4	Lift to closed networks . . . . .	184
3.11.5	Transfer into the standard model . . . . .	185
3.11.6	Loop freedom of AODV . . . . .	185
<b>4</b>	<b>Variant D: Forwarding the Route Request</b> . . . . .	<b>186</b>
4.1	Predicates and functions used in the AODV model . . . . .	186
4.1.1	Sequence Numbers . . . . .	186
4.1.2	Modelling Routes . . . . .	186
4.1.3	Routing Tables . . . . .	187
4.1.4	Updating Routing Tables . . . . .	189
4.1.5	Route Requests . . . . .	197
4.1.6	Queued Packets . . . . .	197
4.1.7	Comparison with the original technical report . . . . .	198
4.2	AODV protocol messages . . . . .	198
4.3	The AODV protocol . . . . .	199
4.3.1	Data state . . . . .	199
4.3.2	Auxilliary message handling definitions . . . . .	200
4.3.3	The protocol process . . . . .	202
4.4	Invariant assumptions and properties . . . . .	207
4.5	Quality relations between routes . . . . .	209
4.5.1	Net sequence numbers . . . . .	209
4.5.2	Comparing routes . . . . .	211
4.5.3	Comparing routing tables . . . . .	212
4.5.4	Strictly comparing routing tables . . . . .	215
4.6	Invariant proofs on individual processes . . . . .	218
4.7	The quality increases predicate . . . . .	221
4.8	The ‘open’ AODV model . . . . .	224
4.9	Global invariant proofs over sequential processes . . . . .	225
4.10	Routing graphs and loop freedom . . . . .	228
4.11	Lift and transfer invariants to show loop freedom . . . . .	229
4.11.1	Lift to parallel processes with queues . . . . .	229
4.11.2	Lift to nodes . . . . .	230
4.11.3	Lift to partial networks . . . . .	231
4.11.4	Lift to closed networks . . . . .	231

4.11.5	Transfer into the standard model . . . . .	231
4.11.6	Loop freedom of AODV . . . . .	231
<b>5</b>	<b>Variants A–D: All proposed modifications</b>	<b>232</b>
5.1	Predicates and functions used in the AODV model . . . . .	232
5.1.1	Sequence Numbers . . . . .	232
5.1.2	Modelling Routes . . . . .	232
5.1.3	Routing Tables . . . . .	233
5.1.4	Updating Routing Tables . . . . .	235
5.1.5	Route Requests . . . . .	240
5.1.6	Queued Packets . . . . .	241
5.1.7	Comparison with the original technical report . . . . .	241
5.2	AODV protocol messages . . . . .	242
5.3	The AODV protocol . . . . .	243
5.3.1	Data state . . . . .	243
5.3.2	Auxilliary message handling definitions . . . . .	244
5.3.3	The protocol process . . . . .	246
5.4	Invariant assumptions and properties . . . . .	250
5.5	Quality relations between routes . . . . .	252
5.5.1	Net sequence numbers . . . . .	253
5.5.2	Comparing routes . . . . .	254
5.5.3	Comparing routing tables . . . . .	255
5.5.4	Strictly comparing routing tables . . . . .	258
5.6	Invariant proofs on individual processes . . . . .	260
5.7	The quality increases predicate . . . . .	264
5.8	The ‘open’ AODV model . . . . .	267
5.9	Global invariant proofs over sequential processes . . . . .	268
5.10	Routing graphs and loop freedom . . . . .	271
5.11	Lift and transfer invariants to show loop freedom . . . . .	272
5.11.1	Lift to parallel processes with queues . . . . .	272
5.11.2	Lift to nodes . . . . .	272
5.11.3	Lift to partial networks . . . . .	273
5.11.4	Lift to closed networks . . . . .	273
5.11.5	Transfer into the standard model . . . . .	273
5.11.6	Loop freedom of AODV . . . . .	274

## 0.1 Basic data types and constants

```
theory Aodv_Basic
imports Main "../AWN/AWN_SOS"
begin
```

These definitions are shared with all variants.

```
type_synonym rreqid = nat
type_synonym sqn = nat
```

```
datatype k = Known | Unknown
abbreviation kno where "kno  $\equiv$  Known"
abbreviation unk where "unk  $\equiv$  Unknown"
```

```
datatype p = NoRequestRequired | RequestRequired
abbreviation noreq where "noreq  $\equiv$  NoRequestRequired"
abbreviation req where "req  $\equiv$  RequestRequired"
```

```
datatype f = Valid | Invalid
abbreviation val where "val  $\equiv$  Valid"
abbreviation inv where "inv  $\equiv$  Invalid"
```

```
lemma not_ks [simp]:
  "(x  $\neq$  kno) = (x = unk)"
  "(x  $\neq$  unk) = (x = kno)"
  <proof>
```

```
lemma not_ps [simp]:
  "(x  $\neq$  noreq) = (x = req)"
  "(x  $\neq$  req) = (x = noreq)"
  <proof>
```

```
lemma not_ffs [simp]:
  "(x  $\neq$  val) = (x = inv)"
  "(x  $\neq$  inv) = (x = val)"
  <proof>
```

end

## 0.2 Predicates and functions used in the AODV model

```
theory Aodv_Data
imports Aodv_Basic
begin
```

### 0.2.1 Sequence Numbers

Sequence numbers approximate the relative freshness of routing information.

```
definition inc :: "sqn  $\Rightarrow$  sqn"
  where "inc sn  $\equiv$  if sn = 0 then sn else sn + 1"
```

```
lemma less_than_inc [simp]: "x  $\leq$  inc x"
  <proof>
```

```
lemma inc_minus_suc_0 [simp]:
  "inc x - Suc 0 = x"
  <proof>
```

```
lemma inc_never_one' [simp, intro]: "inc x  $\neq$  Suc 0"
  <proof>
```

```
lemma inc_never_one [simp, intro]: "inc x  $\neq$  1"
  <proof>
```

## 0.2.2 Modelling Routes

A route is a 6-tuple,  $(dsn, dsk, flag, hops, nhip, pre)$  where  $dsn$  is the ‘destination sequence number’,  $dsk$  is the ‘destination-sequence-number status’,  $flag$  is the route status,  $hops$  is the number of hops to the destination,  $nhip$  is the next hop toward the destination, and  $pre$  is the set of ‘precursor nodes’—those interested in hearing about changes to the route.

```
type_synonym r = "sqn × k × f × nat × ip × ip set"
```

```
definition proj2 :: "r ⇒ sqn" ("π2")
  where "π2 ≡ λ(dsn, _, _, _, _). dsn"
```

```
definition proj3 :: "r ⇒ k" ("π3")
  where "π3 ≡ λ(_, dsk, _, _, _). dsk"
```

```
definition proj4 :: "r ⇒ f" ("π4")
  where "π4 ≡ λ(_, _, flag, _, _). flag"
```

```
definition proj5 :: "r ⇒ nat" ("π5")
  where "π5 ≡ λ(_, _, _, hops, _, _). hops"
```

```
definition proj6 :: "r ⇒ ip" ("π6")
  where "π6 ≡ λ(_, _, _, _, nhip, _). nhip"
```

```
definition proj7 :: "r ⇒ ip set" ("π7")
  where "π7 ≡ λ(_, _, _, _, _, pre). pre"
```

```
lemma projs [simp]:
  "π2(dsn, dsk, flag, hops, nhip, pre) = dsn"
  "π3(dsn, dsk, flag, hops, nhip, pre) = dsk"
  "π4(dsn, dsk, flag, hops, nhip, pre) = flag"
  "π5(dsn, dsk, flag, hops, nhip, pre) = hops"
  "π6(dsn, dsk, flag, hops, nhip, pre) = nhip"
  "π7(dsn, dsk, flag, hops, nhip, pre) = pre"
  <proof>
```

```
lemma proj3_pred [intro]: "[ P kno; P unk ] ⇒ P (π3 x)"
  <proof>
```

```
lemma proj4_pred [intro]: "[ P val; P inv ] ⇒ P (π4 x)"
  <proof>
```

```
lemma proj6_pair_snd [simp]:
  fixes dsn' r
  shows "π6(dsn', snd (r)) = π6(r)"
  <proof>
```

## 0.2.3 Routing Tables

Routing tables map ip addresses to route entries.

```
type_synonym rt = "ip → r"
```

```
syntax
  "_Sigma_route" :: "rt ⇒ ip → r" ("σroute'(_, _)")
```

```
translations
  "σroute(rt, dip)" => "rt dip"
```

```
definition sqn :: "rt ⇒ ip ⇒ sqn"
  where "sqn rt dip ≡ case σroute(rt, dip) of Some r ⇒ π2(r) | None ⇒ 0"
```

```
definition sqnf :: "rt ⇒ ip ⇒ k"
  where "sqnf rt dip ≡ case σroute(rt, dip) of Some r ⇒ π3(r) | None ⇒ unk"
```



```

abbreviation flag :: "rt ⇒ ip → f"
  where "flag rt dip ≡ map_option π4 (σroute(rt, dip))"

abbreviation dhops :: "rt ⇒ ip → nat"
  where "dhops rt dip ≡ map_option π5 (σroute(rt, dip))"

abbreviation nhop :: "rt ⇒ ip → ip"
  where "nhop rt dip ≡ map_option π6 (σroute(rt, dip))"

abbreviation precs :: "rt ⇒ ip → ip set"
  where "precs rt dip ≡ map_option π7 (σroute(rt, dip))"

definition vD :: "rt ⇒ ip set"
  where "vD rt ≡ {dip. flag rt dip = Some val}"

definition iD :: "rt ⇒ ip set"
  where "iD rt ≡ {dip. flag rt dip = Some inv}"

definition kD :: "rt ⇒ ip set"
  where "kD rt ≡ {dip. rt dip ≠ None}"

lemma kD_is_vD_and_iD: "kD rt = vD rt ∪ iD rt"
  ⟨proof⟩

lemma vD_iD_gives_kD [simp]:
  "∧ip rt. ip ∈ vD rt ⇒ ip ∈ kD rt"
  "∧ip rt. ip ∈ iD rt ⇒ ip ∈ kD rt"
  ⟨proof⟩

lemma kD_Some [dest]:
  fixes dip rt
  assumes "dip ∈ kD rt"
  shows "∃ dsn dsk flag hops nhip pre.
    σroute(rt, dip) = Some (dsn, dsk, flag, hops, nhip, pre)"
  ⟨proof⟩

lemma kD_None [dest]:
  fixes dip rt
  assumes "dip ∉ kD rt"
  shows "σroute(rt, dip) = None"
  ⟨proof⟩

lemma vD_Some [dest]:
  fixes dip rt
  assumes "dip ∈ vD rt"
  shows "∃ dsn dsk hops nhip pre.
    σroute(rt, dip) = Some (dsn, dsk, val, hops, nhip, pre)"
  ⟨proof⟩

lemma vD_empty [simp]: "vD Map.empty = {}"
  ⟨proof⟩

lemma iD_Some [dest]:
  fixes dip rt
  assumes "dip ∈ iD rt"
  shows "∃ dsn dsk hops nhip pre.
    σroute(rt, dip) = Some (dsn, dsk, inv, hops, nhip, pre)"
  ⟨proof⟩

lemma val_is_vD [elim]:
  fixes ip rt
  assumes "ip ∈ kD(rt)"
  and "the (flag rt ip) = val"
  shows "ip ∈ vD(rt)"

```

```

⟨proof⟩

lemma inv_is_iD [elim]:
  fixes ip rt
  assumes "ip∈kD(rt)"
    and "the (flag rt ip) = inv"
  shows "ip∈iD(rt)"
⟨proof⟩

lemma iD_flag_is_inv [elim, simp]:
  fixes ip rt
  assumes "ip∈iD(rt)"
  shows "the (flag rt ip) = inv"
⟨proof⟩

lemma kD_but_not_vD_is_iD [elim]:
  fixes ip rt
  assumes "ip∈kD(rt)"
    and "ip∉vD(rt)"
  shows "ip∈iD(rt)"
⟨proof⟩

lemma vD_or_iD [elim]:
  fixes ip rt
  assumes "ip∈kD(rt)"
    and "ip∈vD(rt) ⇒ P rt ip"
    and "ip∈iD(rt) ⇒ P rt ip"
  shows "P rt ip"
⟨proof⟩

lemma proj5_eq_dhops: "∧dip rt. dip∈kD(rt) ⇒ π5(the (rt dip)) = the (dhops rt dip)"
⟨proof⟩

lemma proj4_eq_flag: "∧dip rt. dip∈kD(rt) ⇒ π4(the (rt dip)) = the (flag rt dip)"
⟨proof⟩

lemma proj2_eq_sqn: "∧dip rt. dip∈kD(rt) ⇒ π2(the (rt dip)) = sqn rt dip"
⟨proof⟩

lemma kD_sqnf_is_proj3 [simp]:
  "∧ip rt. ip∈kD(rt) ⇒ sqnf rt ip = π3(the (rt ip))"
⟨proof⟩

lemma vD_flag_val [simp]:
  "∧dip rt. dip ∈ vD (rt) ⇒ the (flag rt dip) = val"
⟨proof⟩

lemma kD_update [simp]:
  "∧rt nip v. kD (rt(nip ↦ v)) = insert nip (kD rt)"
⟨proof⟩

lemma kD_empty [simp]: "kD Map.empty = {}"
⟨proof⟩

lemma ip_equal_or_known [elim]:
  fixes rt ip ip'
  assumes "ip = ip' ∨ ip∈kD(rt)"
    and "ip = ip' ⇒ P rt ip ip'"
    and "[ ip ≠ ip'; ip∈kD(rt) ] ⇒ P rt ip ip'"
  shows "P rt ip ip'"
⟨proof⟩

```

## 0.2.4 Updating Routing Tables

Routing table entries are modified through explicit functions. The properties of these functions are important in invariant proofs.

### Updating Precursor Lists

```
definition addpre :: "r  $\Rightarrow$  ip set  $\Rightarrow$  r"  
  where "addpre r npre  $\equiv$  let (dsn, dsk, flag, hops, nhip, pre) = r in  
        (dsn, dsk, flag, hops, nhip, pre  $\cup$  npre)"
```

```
lemma proj2_addpre:  
  fixes v pre  
  shows " $\pi_2$ (addpre v pre) =  $\pi_2$ (v)"  
  <proof>
```

```
lemma proj3_addpre:  
  fixes v pre  
  shows " $\pi_3$ (addpre v pre) =  $\pi_3$ (v)"  
  <proof>
```

```
lemma proj4_addpre:  
  fixes v pre  
  shows " $\pi_4$ (addpre v pre) =  $\pi_4$ (v)"  
  <proof>
```

```
lemma proj5_addpre:  
  fixes v pre  
  shows " $\pi_5$ (addpre v pre) =  $\pi_5$ (v)"  
  <proof>
```

```
lemma proj6_addpre:  
  fixes dsn dsk flag hops nhip pre npre  
  shows " $\pi_6$ (addpre v npre) =  $\pi_6$ (v)"  
  <proof>
```

```
lemma proj7_addpre:  
  fixes dsn dsk flag hops nhip pre npre  
  shows " $\pi_7$ (addpre v npre) =  $\pi_7$ (v)  $\cup$  npre"  
  <proof>
```

```
lemma addpre_empty: "addpre r {} = r"  
  <proof>
```

```
lemma addpre_r:  
  "addpre (dsn, dsk, fl, hops, nhip, pre) npre = (dsn, dsk, fl, hops, nhip, pre  $\cup$  npre)"  
  <proof>
```

```
lemmas addpre_simps [simp] = proj2_addpre proj3_addpre proj4_addpre proj5_addpre  
  proj6_addpre proj7_addpre addpre_empty addpre_r
```

```
definition addpreRT :: "rt  $\Rightarrow$  ip  $\Rightarrow$  ip set  $\rightarrow$  rt"  
  where "addpreRT rt dip npre  $\equiv$   
        map_option ( $\lambda$ s. rt (dip  $\mapsto$  addpre s npre)) ( $\sigma_{route}$ (rt, dip))"
```

```
lemma snd_addpre [simp]:  
  " $\bigwedge$ dsn dsn' v pre. (dsn, snd(addpre (dsn', v) pre)) = addpre (dsn, v) pre"  
  <proof>
```

```
lemma proj2_addpreRT [simp]:  
  fixes ip rt ip' npre  
  assumes "ip  $\in$  kD rt"  
  and "ip'  $\in$  kD rt"  
  shows " $\pi_2$ (the (the (addpreRT rt ip' npre) ip)) =  $\pi_2$ (the (rt ip))"
```

*<proof>*

**lemma** *proj3\_addpreRT [simp]:*

fixes *ip rt ip' npre*

assumes "*ip ∈ kD rt*"

and "*ip' ∈ kD rt*"

shows " $\pi_3(\text{the } (\text{the } (\text{addpreRT } rt \ ip' \ npre) \ ip)) = \pi_3(\text{the } (rt \ ip))$ "

*<proof>*

**lemma** *proj5\_addpreRT [simp]:*

" $\bigwedge rt \ dip \ ip \ npre. \ dip \in kD(rt) \implies \pi_5(\text{the } (\text{the } (\text{addpreRT } rt \ dip \ npre) \ ip)) = \pi_5(\text{the } (rt \ ip))$ "

*<proof>*

**lemma** *flag\_addpreRT [simp]:*

fixes *rt pre ip dip*

assumes "*dip ∈ kD rt*"

shows "*flag (the (addpreRT rt dip pre)) ip = flag rt ip*"

*<proof>*

**lemma** *kD\_addpreRT [simp]:*

fixes *rt dip npre*

assumes "*dip ∈ kD rt*"

shows "*kD (the (addpreRT rt dip npre)) = kD rt*"

*<proof>*

**lemma** *vD\_addpreRT [simp]:*

fixes *rt dip npre*

assumes "*dip ∈ kD rt*"

shows "*vD (the (addpreRT rt dip npre)) = vD rt*"

*<proof>*

**lemma** *iD\_addpreRT [simp]:*

fixes *rt dip npre*

assumes "*dip ∈ kD rt*"

shows "*iD (the (addpreRT rt dip npre)) = iD rt*"

*<proof>*

**lemma** *nhop\_addpreRT [simp]:*

fixes *rt pre ip dip*

assumes "*dip ∈ kD rt*"

shows "*nhop (the (addpreRT rt dip pre)) ip = nhop rt ip*"

*<proof>*

**lemma** *sqn\_addpreRT [simp]:*

fixes *rt pre ip dip*

assumes "*dip ∈ kD rt*"

shows "*sqn (the (addpreRT rt dip pre)) ip = sqn rt ip*"

*<proof>*

**lemma** *dhops\_addpreRT [simp]:*

fixes *rt pre ip dip*

assumes "*dip ∈ kD rt*"

shows "*dhops (the (addpreRT rt dip pre)) ip = dhops rt ip*"

*<proof>*

**lemma** *sqnf\_addpreRT [simp]:*

" $\bigwedge ip \ dip. \ ip \in kD(rt \ \xi) \implies sqnf(\text{the } (\text{addpreRT } (rt \ \xi) \ ip \ npre)) \ dip = sqnf(rt \ \xi) \ dip$ "

*<proof>*

## Updating route entries

**lemma** *in\_kD\_case [simp]:*

fixes *dip rt*

assumes "*dip ∈ kD(rt)*"

```

  shows "(case rt dip of None  $\Rightarrow$  en | Some r  $\Rightarrow$  es r) = es (the (rt dip))"
  <proof>

lemma not_in_kD_case [simp]:
  fixes dip rt
  assumes "dip  $\notin$  kD(rt)"
  shows "(case rt dip of None  $\Rightarrow$  en | Some r  $\Rightarrow$  es r) = en"
  <proof>

lemma rt_Some_sqn [dest]:
  fixes rt and ip dsn dsk flag hops nhip pre
  assumes "rt ip = Some (dsn, dsk, flag, hops, nhip, pre)"
  shows "sqn rt ip = dsn"
  <proof>

lemma not_kD_sqn [simp]:
  fixes dip rt
  assumes "dip  $\notin$  kD(rt)"
  shows "sqn rt dip = 0"
  <proof>

definition update_arg_wf :: "r  $\Rightarrow$  bool"
where "update_arg_wf r  $\equiv$   $\pi_4(r) = \text{val} \wedge$ 
      ( $\pi_2(r) = 0$ ) = ( $\pi_3(r) = \text{unk}$ )  $\wedge$ 
      ( $\pi_3(r) = \text{unk} \longrightarrow \pi_5(r) = 1$ )"

lemma update_arg_wf_gives_cases:
  " $\bigwedge r$ . update_arg_wf r  $\implies$  ( $\pi_2(r) = 0$ ) = ( $\pi_3(r) = \text{unk}$ )"
  <proof>

lemma update_arg_wf_tuples [simp]:
  " $\bigwedge$ nhip pre. update_arg_wf (0, unk, val, Suc 0, nhip, pre)"
  " $\bigwedge$ n hops nhip pre. update_arg_wf (Suc n, kno, val, hops, nhip, pre)"
  <proof>

lemma update_arg_wf_tuples' [elim]:
  " $\bigwedge$ n hops nhip pre. Suc 0  $\leq$  n  $\implies$  update_arg_wf (n, kno, val, hops, nhip, pre)"
  <proof>

lemma wf_r_cases [intro]:
  fixes P r
  assumes "update_arg_wf r"
  and c1: " $\bigwedge$ nhip pre. P (0, unk, val, Suc 0, nhip, pre)"
  and c2: " $\bigwedge$ dsn hops nhip pre. dsn > 0  $\implies$  P (dsn, kno, val, hops, nhip, pre)"
  shows "P r"
  <proof>

definition update :: "rt  $\Rightarrow$  ip  $\Rightarrow$  r  $\Rightarrow$  rt"
where
  "update rt ip r  $\equiv$ 
  case  $\sigma_{\text{route}}(\text{rt}, \text{ip})$  of
    None  $\Rightarrow$  rt (ip  $\mapsto$  r)
  | Some s  $\Rightarrow$ 
    if  $\pi_2(s) < \pi_2(r)$  then rt (ip  $\mapsto$  addpre r ( $\pi_7(s)$ ))
    else if  $\pi_2(s) = \pi_2(r) \wedge (\pi_5(s) > \pi_5(r) \vee \pi_4(s) = \text{inv})$ 
    then rt (ip  $\mapsto$  addpre r ( $\pi_7(s)$ ))
    else if  $\pi_3(r) = \text{unk}$ 
    then rt (ip  $\mapsto$  ( $\pi_2(s)$ , snd (addpre r ( $\pi_7(s)$ ))))
    else rt (ip  $\mapsto$  addpre s ( $\pi_7(r)$ ))"

lemma update_simps [simp]:
  fixes r s nrt nr nr' ns rt ip
  defines "s  $\equiv$  the  $\sigma_{\text{route}}(\text{rt}, \text{ip})$ "
  and "nr  $\equiv$  addpre r ( $\pi_7(s)$ )"

```

```

and "nr'  $\equiv$  ( $\pi_2(s)$ ,  $\pi_3(nr)$ ,  $\pi_4(nr)$ ,  $\pi_5(nr)$ ,  $\pi_6(nr)$ ,  $\pi_7(nr)$ )"
and "ns  $\equiv$  addpre s ( $\pi_7(r)$ )"
shows
"[[ip  $\notin$  kD(rt)]]  $\implies$  update rt ip r = rt (ip  $\mapsto$  r)"
"[[ip  $\in$  kD(rt); sqn rt ip <  $\pi_2(r)$ ]]  $\implies$  update rt ip r = rt (ip  $\mapsto$  nr)"
"[[ip  $\in$  kD(rt); sqn rt ip =  $\pi_2(r)$ ;
  the (dhops rt ip) >  $\pi_5(r)$ ]]  $\implies$  update rt ip r = rt (ip  $\mapsto$  nr)"
"[[ip  $\in$  kD(rt); sqn rt ip =  $\pi_2(r)$ ;
  flag rt ip = Some inv]]  $\implies$  update rt ip r = rt (ip  $\mapsto$  nr)"
"[[ip  $\in$  kD(rt);  $\pi_3(r)$  = unk; ( $\pi_2(r)$  = 0) = ( $\pi_3(r)$  = unk)]]  $\implies$  update rt ip r = rt (ip  $\mapsto$  nr)"
"[[ip  $\in$  kD(rt); sqn rt ip  $\geq$   $\pi_2(r)$ ;  $\pi_3(r)$  = kno;
  sqn rt ip =  $\pi_2(r)$ ]]  $\implies$  the (dhops rt ip)  $\leq$   $\pi_5(r)$   $\wedge$  the (flag rt ip) = val]]
 $\implies$  update rt ip r = rt (ip  $\mapsto$  ns)"
<proof>

```

lemma update\_cases [elim]:

```

assumes "( $\pi_2(r)$  = 0) = ( $\pi_3(r)$  = unk)"
and c1: "[[ip  $\notin$  kD(rt)]]  $\implies$  P (rt (ip  $\mapsto$  r))"

and c2: "[[ip  $\in$  kD(rt); sqn rt ip <  $\pi_2(r)$ ]]
 $\implies$  P (rt (ip  $\mapsto$  addpre r ( $\pi_7$ (the  $\sigma_{route}$ (rt, ip)))))"
and c3: "[[ip  $\in$  kD(rt); sqn rt ip =  $\pi_2(r)$ ; the (dhops rt ip) >  $\pi_5(r)$ ]]
 $\implies$  P (rt (ip  $\mapsto$  addpre r ( $\pi_7$ (the  $\sigma_{route}$ (rt, ip)))))"
and c4: "[[ip  $\in$  kD(rt); sqn rt ip =  $\pi_2(r)$ ; the (flag rt ip) = inv]]
 $\implies$  P (rt (ip  $\mapsto$  addpre r ( $\pi_7$ (the  $\sigma_{route}$ (rt, ip)))))"
and c5: "[[ip  $\in$  kD(rt);  $\pi_3(r)$  = unk]]
 $\implies$  P (rt (ip  $\mapsto$  ( $\pi_2$ (the  $\sigma_{route}$ (rt, ip)),  $\pi_3(r)$ ,
 $\pi_4(r)$ ,  $\pi_5(r)$ ,  $\pi_6(r)$ ,  $\pi_7$ (addpre r ( $\pi_7$ (the  $\sigma_{route}$ (rt, ip)))))))"
and c6: "[[ip  $\in$  kD(rt); sqn rt ip  $\geq$   $\pi_2(r)$ ;  $\pi_3(r)$  = kno;
  sqn rt ip =  $\pi_2(r)$ ]]  $\implies$  the (dhops rt ip)  $\leq$   $\pi_5(r)$   $\wedge$  the (flag rt ip) = val]]
 $\implies$  P (rt (ip  $\mapsto$  addpre (the  $\sigma_{route}$ (rt, ip)) ( $\pi_7(r)$ )))"
shows "(P (update rt ip r))"
<proof>

```

lemma update\_cases\_kD:

```

assumes "( $\pi_2(r)$  = 0) = ( $\pi_3(r)$  = unk)"
and "ip  $\in$  kD(rt)"
and c2: "[[sqn rt ip <  $\pi_2(r)$ ]]  $\implies$  P (rt (ip  $\mapsto$  addpre r ( $\pi_7$ (the  $\sigma_{route}$ (rt, ip)))))"
and c3: "[[sqn rt ip =  $\pi_2(r)$ ; the (dhops rt ip) >  $\pi_5(r)$ ]]
 $\implies$  P (rt (ip  $\mapsto$  addpre r ( $\pi_7$ (the  $\sigma_{route}$ (rt, ip)))))"
and c4: "[[sqn rt ip =  $\pi_2(r)$ ; the (flag rt ip) = inv]]
 $\implies$  P (rt (ip  $\mapsto$  addpre r ( $\pi_7$ (the  $\sigma_{route}$ (rt, ip)))))"
and c5: "[ $\pi_3(r)$  = unk]  $\implies$  P (rt (ip  $\mapsto$  ( $\pi_2$ (the  $\sigma_{route}$ (rt, ip)),  $\pi_3(r)$ ,
 $\pi_4(r)$ ,  $\pi_5(r)$ ,  $\pi_6(r)$ ,
 $\pi_7$ (addpre r ( $\pi_7$ (the  $\sigma_{route}$ (rt, ip)))))))"
and c6: "[[sqn rt ip  $\geq$   $\pi_2(r)$ ;  $\pi_3(r)$  = kno;
  sqn rt ip =  $\pi_2(r)$ ]]  $\implies$  the (dhops rt ip)  $\leq$   $\pi_5(r)$   $\wedge$  the (flag rt ip) = val]]
 $\implies$  P (rt (ip  $\mapsto$  addpre (the  $\sigma_{route}$ (rt, ip)) ( $\pi_7(r)$ )))"
shows "(P (update rt ip r))"
<proof>

```

lemma in\_kD\_after\_update [simp]:

```

fixes rt nip dsn dsk flag hops nhip pre
shows "kD (update rt nip (dsn, dsk, flag, hops, nhip, pre)) = insert nip (kD rt)"
<proof>

```

lemma nhop\_of\_update [simp]:

```

fixes rt dip dsn dsk flag hops nhip
assumes "rt  $\neq$  update rt dip (dsn, dsk, flag, hops, nhip, {})"
shows "the (nhop (update rt dip (dsn, dsk, flag, hops, nhip, {})) dip) = nhip"
<proof>

```

lemma sqn\_if\_updated:

```

fixes rip v rt ip

```

```

shows "sqn ( $\lambda x. \text{if } x = \text{rip} \text{ then Some } v \text{ else rt } x$ ) ip
      = (if ip = rip then  $\pi_2(v)$  else sqn rt ip)"
⟨proof⟩

lemma update_sqn [simp]:
  fixes rt dip rip dsn dsk hops nhip pre
  assumes "(dsn = 0) = (dsk = unk)"
  shows "sqn rt dip  $\leq$  sqn (update rt rip (dsn, dsk, val, hops, nhip, pre)) dip"
⟨proof⟩

lemma sqn_update_bigger [simp]:
  fixes rt ip ip' dsn dsk flag hops nhip pre
  assumes "1  $\leq$  hops"
  shows "sqn rt ip  $\leq$  sqn (update rt ip' (dsn, dsk, flag, hops, nhip, pre)) ip"
⟨proof⟩

lemma dhops_update [intro]:
  fixes rt dsn dsk flag hops ip rip nhip pre
  assumes ex: " $\forall ip \in kD \text{ rt. the (dhops rt ip)} \geq 1$ "
  and ip: "(ip = rip  $\wedge$  Suc 0  $\leq$  hops)  $\vee$  (ip  $\neq$  rip  $\wedge$  ip  $\in kD \text{ rt})$ "
  shows "Suc 0  $\leq$  the (dhops (update rt rip (dsn, dsk, flag, hops, nhip, pre)) ip)"
⟨proof⟩

lemma update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip  $\neq$  dip"
  shows "(update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = rt ip"
⟨proof⟩

lemma nhop_update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip  $\neq$  dip"
  shows "nhop (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = nhop rt ip"
⟨proof⟩

lemma dhops_update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip  $\neq$  dip"
  shows "dhops (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = dhops rt ip"
⟨proof⟩

lemma sqn_update_same [simp]:
  " $\wedge$ rt dsn dsk flag hops nhip pre. sqn (rt(ip  $\mapsto$  v)) ip =  $\pi_2(v)$ "
⟨proof⟩

lemma dhops_update_changed [simp]:
  fixes rt dip osn hops nhip
  assumes "rt  $\neq$  update rt dip (osn, kno, val, hops, nhip, {})"
  shows "the (dhops (update rt dip (osn, kno, val, hops, nhip, {})) dip) = hops"
⟨proof⟩

lemma nhop_update_unk_val [simp]:
  " $\wedge$ rt dip ip dsn hops npre.
  the (nhop (update rt dip (dsn, unk, val, hops, ip, npre)) dip) = ip"
⟨proof⟩

lemma nhop_update_changed [simp]:
  fixes rt dip dsn dsk flg hops sip
  assumes "update rt dip (dsn, dsk, flg, hops, sip, {})  $\neq$  rt"
  shows "the (nhop (update rt dip (dsn, dsk, flg, hops, sip, {})) dip) = sip"
⟨proof⟩

lemma update_rt_split_asm:
  " $\wedge$ rt ip dsn dsk flag hops sip.

```

```

P (update rt ip (dsn, dsk, flag, hops, sip, {}))
=
(¬(rt = update rt ip (dsn, dsk, flag, hops, sip, {}) ∧ ¬P rt
  ∨ rt ≠ update rt ip (dsn, dsk, flag, hops, sip, {}))
  ∧ ¬P (update rt ip (dsn, dsk, flag, hops, sip, {})))
⟨proof⟩

```

```

lemma sqn_update [simp]: "∧rt dip dsn flg hops sip.
rt ≠ update rt dip (dsn, kno, flg, hops, sip, {})
⇒ sqn (update rt dip (dsn, kno, flg, hops, sip, {})) dip = dsn"
⟨proof⟩

```

```

lemma sqnf_update [simp]: "∧rt dip dsn dsk flg hops sip.
rt ≠ update rt dip (dsn, dsk, flg, hops, sip, {})
⇒ sqnf (update rt dip (dsn, dsk, flg, hops, sip, {})) dip = dsk"
⟨proof⟩

```

```

lemma update_kno_dsn_greater_zero:
"∧rt dip ip dsn hops npre. 1 ≤ dsn ⇒ 1 ≤ (sqn (update rt dip (dsn, kno, val, hops, ip, npre)) dip)"
⟨proof⟩

```

```

lemma proj3_update [simp]: "∧rt dip dsn dsk flg hops sip.
rt ≠ update rt dip (dsn, dsk, flg, hops, sip, {})
⇒ π3(the (update rt dip (dsn, dsk, flg, hops, sip, {})) dip) = dsk"
⟨proof⟩

```

```

lemma nhop_update_changed_kno_val [simp]: "∧rt ip dsn dsk hops nhip.
rt ≠ update rt ip (dsn, kno, val, hops, nhip, {})
⇒ the (nhop (update rt ip (dsn, kno, val, hops, nhip, {})) ip) = nhip"
⟨proof⟩

```

```

lemma flag_update [simp]: "∧rt dip dsn flg hops sip.
rt ≠ update rt dip (dsn, kno, flg, hops, sip, {})
⇒ the (flag (update rt dip (dsn, kno, flg, hops, sip, {})) dip) = flg"
⟨proof⟩

```

```

lemma the_flag_Some [dest!]:
  fixes ip rt
  assumes "the (flag rt ip) = x"
  and "ip ∈ kD rt"
  shows "flag rt ip = Some x"
⟨proof⟩

```

```

lemma kD_update_unchanged [dest]:
  fixes rt dip dsn dsk flag hops nhip pre
  assumes "rt = update rt dip (dsn, dsk, flag, hops, nhip, pre)"
  shows "dip ∈ kD(rt)"
⟨proof⟩

```

```

lemma nhop_update [simp]: "∧rt dip dsn dsk flg hops sip.
rt ≠ update rt dip (dsn, dsk, flg, hops, sip, {})
⇒ the (nhop (update rt dip (dsn, dsk, flg, hops, sip, {})) dip) = sip"
⟨proof⟩

```

```

lemma sqn_update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
  shows "sqn (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = sqn rt ip"
⟨proof⟩

```

```

lemma sqnf_update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
  shows "sqnf (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = sqnf rt ip"

```



*<proof>*

**lemma** *vD\_update\_val* [*dest*]:

" $\bigwedge \text{dip } \text{rt } \text{dip}' \text{ dsn } \text{dsk } \text{hops } \text{nhip } \text{pre}.$

$\text{dip} \in \text{vD}(\text{update } \text{rt } \text{dip}' (\text{dsn}, \text{dsk}, \text{val}, \text{hops}, \text{nhip}, \text{pre})) \implies (\text{dip} \in \text{vD}(\text{rt}) \vee \text{dip} = \text{dip}')$ "

*<proof>*

## Invalidating route entries

**definition** *invalidate* :: "*rt*  $\Rightarrow$  (*ip*  $\rightarrow$  *sqn*)  $\Rightarrow$  *rt*"

where "*invalidate* *rt* *dests*  $\equiv$

$\lambda \text{ip}. \text{case } (\text{rt } \text{ip}, \text{dests } \text{ip}) \text{ of}$

$(\text{None}, \_) \Rightarrow \text{None}$

$| (\text{Some } s, \text{None}) \Rightarrow \text{Some } s$

$| (\text{Some } (\_, \text{dsk}, \_, \text{hops}, \text{nhip}, \text{pre}), \text{Some } \text{rsn}) \Rightarrow$   
 $\text{Some } (\text{rsn}, \text{dsk}, \text{inv}, \text{hops}, \text{nhip}, \text{pre})"$

**lemma** *proj3\_invalidate* [*simp*]:

" $\bigwedge \text{dip}. \pi_3(\text{the } ((\text{invalidate } \text{rt } \text{dests}) \text{ dip})) = \pi_3(\text{the } (\text{rt } \text{dip}))"$

*<proof>*

**lemma** *proj5\_invalidate* [*simp*]:

" $\bigwedge \text{dip}. \pi_5(\text{the } ((\text{invalidate } \text{rt } \text{dests}) \text{ dip})) = \pi_5(\text{the } (\text{rt } \text{dip}))"$

*<proof>*

**lemma** *proj6\_invalidate* [*simp*]:

" $\bigwedge \text{dip}. \pi_6(\text{the } ((\text{invalidate } \text{rt } \text{dests}) \text{ dip})) = \pi_6(\text{the } (\text{rt } \text{dip}))"$

*<proof>*

**lemma** *proj7\_invalidate* [*simp*]:

" $\bigwedge \text{dip}. \pi_7(\text{the } ((\text{invalidate } \text{rt } \text{dests}) \text{ dip})) = \pi_7(\text{the } (\text{rt } \text{dip}))"$

*<proof>*

**lemma** *invalidate\_kD\_inv* [*simp*]:

" $\bigwedge \text{rt } \text{dests}. \text{kD } (\text{invalidate } \text{rt } \text{dests}) = \text{kD } \text{rt}"$

*<proof>*

**lemma** *invalidate\_sqn*:

fixes *rt* *dip* *dests*

assumes " $\forall \text{rsn}. \text{dests } \text{dip} = \text{Some } \text{rsn} \longrightarrow \text{sqn } \text{rt } \text{dip} \leq \text{rsn}"$

shows " $\text{sqn } \text{rt } \text{dip} \leq \text{sqn } (\text{invalidate } \text{rt } \text{dests}) \text{ dip}"$

*<proof>*

**lemma** *sqn\_invalidate\_in\_dests* [*simp*]:

fixes *dests* *ipa* *rsn* *rt*

assumes "*dests* *ipa* = *Some* *rsn*"

and "*ipa*  $\in$  *kD*(*rt*)"

shows " $\text{sqn } (\text{invalidate } \text{rt } \text{dests}) \text{ ipa} = \text{rsn}"$

*<proof>*

**lemma** *dhops\_invalidate* [*simp*]:

" $\bigwedge \text{dip}. \text{the } (\text{dhops } (\text{invalidate } \text{rt } \text{dests}) \text{ dip}) = \text{the } (\text{dhops } \text{rt } \text{dip})"$

*<proof>*

**lemma** *sqnf\_invalidate* [*simp*]:

" $\bigwedge \text{dip}. \text{sqnf } (\text{invalidate } (\text{rt } \xi) (\text{dests } \xi)) \text{ dip} = \text{sqnf } (\text{rt } \xi) \text{ dip}"$

*<proof>*

**lemma** *nhop\_invalidate* [*simp*]:

" $\bigwedge \text{dip}. \text{the } (\text{nhop } (\text{invalidate } (\text{rt } \xi) (\text{dests } \xi)) \text{ dip}) = \text{the } (\text{nhop } (\text{rt } \xi) \text{ dip})"$

*<proof>*

**lemma** *invalidate\_other* [*simp*]:

fixes *rt* *dests* *dip*

```

assumes "dip $\notin$ dom(dests)"
shows "invalidate rt dests dip = rt dip"
<proof>

```

```

lemma invalidate_none [simp]:
  fixes rt dests dip
  assumes "dip $\notin$ kD(rt)"
  shows "invalidate rt dests dip = None"
<proof>

```

```

lemma vD_invalidate_vD_not_dests:
  " $\bigwedge$ dip rt dests. dip $\in$ vD(invalidate rt dests)  $\implies$  dip $\in$ vD(rt)  $\wedge$  dests dip = None"
<proof>

```

```

lemma sqn_invalidate_not_in_dests [simp]:
  fixes dests dip rt
  assumes "dip $\notin$ dom(dests)"
  shows "sqn (invalidate rt dests) dip = sqn rt dip"
<proof>

```

```

lemma invalidate_changes:
  fixes rt dests dip dsn dsk flag hops nhip pre
  assumes "invalidate rt dests dip = Some (dsn, dsk, flag, hops, nhip, pre)"
  shows " dsn = (case dests dip of None  $\Rightarrow$   $\pi_2$ (the (rt dip)) | Some rsn  $\Rightarrow$  rsn)
     $\wedge$  dsk =  $\pi_3$ (the (rt dip))
     $\wedge$  flag = (if dests dip = None then  $\pi_4$ (the (rt dip)) else inv)
     $\wedge$  hops =  $\pi_5$ (the (rt dip))
     $\wedge$  nhip =  $\pi_6$ (the (rt dip))
     $\wedge$  pre =  $\pi_7$ (the (rt dip))"
<proof>

```

```

lemma proj3_inv: " $\bigwedge$ dip rt dests. dip $\in$ kD (rt)
 $\implies$   $\pi_3$ (the (invalidate rt dests dip)) =  $\pi_3$ (the (rt dip))"
<proof>

```

```

lemma dests_iD_invalidate [simp]:
  assumes "dests ip = Some rsn"
  and "ip $\in$ kD(rt)"
  shows "ip $\in$ iD(invalidate rt dests)"
<proof>

```

## 0.2.5 Route Requests

Generate a fresh route request identifier.

```

definition nrreqid :: "(ip  $\times$  rreqid) set  $\Rightarrow$  ip  $\Rightarrow$  rreqid"
  where "nrreqid rreqs ip  $\equiv$  Max ({n. (ip, n)  $\in$  rreqs}  $\cup$  {0}) + 1"

```

## 0.2.6 Queued Packets

Functions for sending data packets.

```

type_synonym store = "ip  $\rightarrow$  (p  $\times$  data list)"

```

```

definition sigma_queue :: "store  $\Rightarrow$  ip  $\Rightarrow$  data list" (" $\sigma_{\text{queue}}$ '(_, _)'")
  where " $\sigma_{\text{queue}}$ (store, dip)  $\equiv$  case store dip of None  $\Rightarrow$  [] | Some (p, q)  $\Rightarrow$  q"

```

```

definition qD :: "store  $\Rightarrow$  ip set"
  where "qD  $\equiv$  dom"

```

```

definition add :: "data  $\Rightarrow$  ip  $\Rightarrow$  store  $\Rightarrow$  store"
  where "add d dip store  $\equiv$  case store dip of
    None  $\Rightarrow$  store (dip  $\mapsto$  (req, [d]))
  | Some (p, q)  $\Rightarrow$  store (dip  $\mapsto$  (p, q @ [d]))"

```

```

lemma qD_add [simp]:
  fixes d dip store
  shows "qD(add d dip store) = insert dip (qD store)"
  <proof>

definition drop :: "ip  $\Rightarrow$  store  $\rightarrow$  store"
  where "drop dip store  $\equiv$ 
    map_option ( $\lambda(p, q). \text{if } tl\ q = [] \text{ then store (dip := None)
      else store (dip } \mapsto (p, tl\ q))$ ) (store dip)"

definition sigma_p_flag :: "store  $\Rightarrow$  ip  $\rightarrow$  p" (" $\sigma_{p\text{-flag}}'(\_, \_)$ ")
  where " $\sigma_{p\text{-flag}}(\text{store}, \text{dip}) \equiv \text{map\_option fst (store dip)}$ "

definition unsetRRF :: "store  $\Rightarrow$  ip  $\Rightarrow$  store"
  where "unsetRRF store dip  $\equiv$  case store dip of
    None  $\Rightarrow$  store
    | Some (p, q)  $\Rightarrow$  store (dip  $\mapsto$  (noreq, q))"

definition setRRF :: "store  $\Rightarrow$  (ip  $\rightarrow$  sqn)  $\Rightarrow$  store"
  where "setRRF store dests  $\equiv$   $\lambda$ dip. if dests dip = None then store dip
    else map_option ( $\lambda(\_, q). (\text{req}, q)$ ) (store dip)"

```

## 0.2.7 Comparison with the original technical report

The major differences with the AODV technical report of Fehnker et al are:

1. *nhop* is partial, thus a ‘*the*’ is needed, similarly for *dhops* and *addpreRT*.
2. *precs* is partial.
3.  $\sigma_{p\text{-flag}}(\text{store}, \text{dip})$  is partial.
4. The routing table (*rt*) is modelled as a map ( $\text{ip} \Rightarrow r\ \text{option}$ ) rather than a set of 7-tuples, likewise, the *r* is a 6-tuple rather than a 7-tuple, i.e., the destination ip-address (*dip*) is taken from the argument to the function, rather than a part of the result. Well-definedness then follows from the structure of the type and more related facts are available automatically, rather than having to be acquired through tedious proofs.
5. Similar remarks hold for the *dests* mapping passed to *invalidate*, and *store*.

end

## 0.3 AODV protocol messages

```

theory Aodv_Message
imports Aodv_Basic
begin

datatype msg =
  Rreq nat rreqid ip sqn k ip sqn ip
  | Rrep nat ip sqn ip ip
  | Rerr "ip  $\rightarrow$  sqn" ip
  | Newpkt data ip
  | Pkt data ip ip

instantiation msg :: msg
begin
  definition newpkt_def [simp]: "newpkt  $\equiv$   $\lambda(d, \text{dip}). \text{Newpkt } d\ \text{dip}"
  definition eq_newpkt_def: "eq_newpkt m  $\equiv$  case m of Newpkt d dip  $\Rightarrow$  True | _  $\Rightarrow$  False"

  instance <proof>
end$ 
```

The *msg* type models the different messages used within AODV. The instantiation as a *msg* is a technicality due to the special treatment of *newpkt* messages in the AWN SOS rules. This use of classes allows a clean separation of the AWN-specific definitions and these AODV-specific definitions.

```
definition rreq :: "nat × rreqid × ip × sqn × k × ip × sqn × ip ⇒ msg"
  where "rreq ≡ λ(hops, rreqid, dip, dsn, dsk, oip, osn, sip).
        Rreq hops rreqid dip dsn dsk oip osn sip"
```

```
lemma rreq_simp [simp]:
  "rreq(hops, rreqid, dip, dsn, dsk, oip, osn, sip) = Rreq hops rreqid dip dsn dsk oip osn sip"
  <proof>
```

```
definition rrep :: "nat × ip × sqn × ip × ip ⇒ msg"
  where "rrep ≡ λ(hops, dip, dsn, oip, sip). Rrep hops dip dsn oip sip"
```

```
lemma rrep_simp [simp]:
  "rrep(hops, dip, dsn, oip, sip) = Rrep hops dip dsn oip sip"
  <proof>
```

```
definition rerr :: "(ip → sqn) × ip ⇒ msg"
  where "rerr ≡ λ(dests, sip). Rerr dests sip"
```

```
lemma rerr_simp [simp]:
  "rerr(dests, sip) = Rerr dests sip"
  <proof>
```

```
lemma not_eq_newpkt_rreq [simp]: "¬eq_newpkt (Rreq hops rreqid dip dsn dsk oip osn sip)"
  <proof>
```

```
lemma not_eq_newpkt_rrep [simp]: "¬eq_newpkt (Rrep hops dip dsn oip sip)"
  <proof>
```

```
lemma not_eq_newpkt_rerr [simp]: "¬eq_newpkt (Rerr dests sip)"
  <proof>
```

```
lemma not_eq_newpkt_pkt [simp]: "¬eq_newpkt (Pkt d dip sip)"
  <proof>
```

```
definition pkt :: "data × ip × ip ⇒ msg"
  where "pkt ≡ λ(d, dip, sip). Pkt d dip sip"
```

```
lemma pkt_simp [simp]:
  "pkt(d, dip, sip) = Pkt d dip sip"
  <proof>
```

end

## 0.4 The AODV protocol

```
theory Aodv
imports Aodv_Data Aodv_Message
  "../AWN/AWN_SOS_Labels" "../AWN/AWN_Invariants"
begin
```

### 0.4.1 Data state

```
record state =
  ip      :: "ip"
  sn      :: "sqn"
  rt      :: "rt"
  rreqs   :: "(ip × rreqid) set"
  store   :: "store"

  msg     :: "msg"
```

```

data    :: "data"
destds  :: "ip  $\rightarrow$  sqn"
pre     :: "ip set"
rreqid  :: "rreqid"
dip     :: "ip"
oip     :: "ip"
hops    :: "nat"
dsn     :: "sqn"
dsk     :: "k"
osn     :: "sqn"
sip     :: "ip"

```

**abbreviation** aadv\_init :: "ip  $\Rightarrow$  state"

```

where "aadv_init i  $\equiv$  ( $\{$ 
  ip = i,
  sn = 1,
  rt = empty,
  rreqs = {},
  store = empty,

  msg    = (SOME x. True),
  data   = (SOME x. True),
  destds = (SOME x. True),
  pre    = (SOME x. True),
  rreqid = (SOME x. True),
  dip    = (SOME x. True),
  oip    = (SOME x. True),
  hops   = (SOME x. True),
  dsn    = (SOME x. True),
  dsk    = (SOME x. True),
  osn    = (SOME x. True),
  sip    = (SOME x. x  $\neq$  i)
 $\}$ "

```

**lemma** some\_neq\_not\_eq [simp]: " $\neg((\text{SOME } x :: \text{nat. } x \neq i) = i)$ "  
 <proof>

**definition** clear\_locals :: "state  $\Rightarrow$  state"

```

where "clear_locals  $\xi$  =  $\xi$  ( $\{$ 
  msg    := (SOME x. True),
  data   := (SOME x. True),
  destds := (SOME x. True),
  pre    := (SOME x. True),
  rreqid := (SOME x. True),
  dip    := (SOME x. True),
  oip    := (SOME x. True),
  hops   := (SOME x. True),
  dsn    := (SOME x. True),
  dsk    := (SOME x. True),
  osn    := (SOME x. True),
  sip    := (SOME x. x  $\neq$  ip  $\xi$ )
 $\}$ "

```

**lemma** clear\_locals\_sip\_not\_ip [simp]: " $\neg(\text{sip } (\text{clear\_locals } \xi) = \text{ip } \xi)$ "  
 <proof>

**lemma** clear\_locals\_but\_not\_globals [simp]:

```

"ip (clear_locals  $\xi$ ) = ip  $\xi$ "
"sn (clear_locals  $\xi$ ) = sn  $\xi$ "
"rt (clear_locals  $\xi$ ) = rt  $\xi$ "
"rreqs (clear_locals  $\xi$ ) = rreqs  $\xi$ "
"store (clear_locals  $\xi$ ) = store  $\xi$ "
<proof>

```

## 0.4.2 Auxilliary message handling definitions

**definition** *is\_newpkt*

where "*is\_newpkt*  $\xi \equiv \text{case msg } \xi \text{ of}$   
     *Newpkt* *data' dip'  $\Rightarrow \{ \xi(\text{data} := \text{data}', \text{dip} := \text{dip}') \}$   
     |  $\_ \Rightarrow \{\}$ "*

**definition** *is\_pkt*

where "*is\_pkt*  $\xi \equiv \text{case msg } \xi \text{ of}$   
     *Pkt* *data' dip' oip'  $\Rightarrow \{ \xi(\text{data} := \text{data}', \text{dip} := \text{dip}', \text{oip} := \text{oip}') \}$   
     |  $\_ \Rightarrow \{\}$ "*

**definition** *is\_rreq*

where "*is\_rreq*  $\xi \equiv \text{case msg } \xi \text{ of}$   
     *Rreq* *hops' rreqid' dip' dsn' dsk' oip' osn' sip'  $\Rightarrow$*   
     {  $\xi(\text{hops} := \text{hops}', \text{rreqid} := \text{rreqid}', \text{dip} := \text{dip}', \text{dsn} := \text{dsn}',$   
       *dsk* := *dsk'*, *oip* := *oip'*, *osn* := *osn'*, *sip* := *sip'* ) }  
     |  $\_ \Rightarrow \{\}$ "

**lemma** *is\_rreq\_asm [dest!]*:

assumes " $\xi' \in \text{is\_rreq } \xi$ "

shows " $(\exists \text{hops}' \text{rreqid}' \text{dip}' \text{dsn}' \text{dsk}' \text{oip}' \text{osn}' \text{sip}'.$   
     *msg*  $\xi = \text{Rreq } \text{hops}' \text{rreqid}' \text{dip}' \text{dsn}' \text{dsk}' \text{oip}' \text{osn}' \text{sip}' \wedge$   
      $\xi' = \xi(\text{hops} := \text{hops}', \text{rreqid} := \text{rreqid}', \text{dip} := \text{dip}', \text{dsn} := \text{dsn}',$   
       *dsk* := *dsk'*, *oip* := *oip'*, *osn* := *osn'*, *sip* := *sip'* )"

*<proof>*

**definition** *is\_rrep*

where "*is\_rrep*  $\xi \equiv \text{case msg } \xi \text{ of}$   
     *Rrep* *hops' dip' dsn' oip' sip'  $\Rightarrow$*   
     {  $\xi(\text{hops} := \text{hops}', \text{dip} := \text{dip}', \text{dsn} := \text{dsn}', \text{oip} := \text{oip}', \text{sip} := \text{sip}') \}$   
     |  $\_ \Rightarrow \{\}$ "

**lemma** *is\_rrep\_asm [dest!]*:

assumes " $\xi' \in \text{is\_rrep } \xi$ "

shows " $(\exists \text{hops}' \text{dip}' \text{dsn}' \text{oip}' \text{sip}'.$   
     *msg*  $\xi = \text{Rrep } \text{hops}' \text{dip}' \text{dsn}' \text{oip}' \text{sip}' \wedge$   
      $\xi' = \xi(\text{hops} := \text{hops}', \text{dip} := \text{dip}', \text{dsn} := \text{dsn}', \text{oip} := \text{oip}', \text{sip} := \text{sip}')"$

*<proof>*

**definition** *is\_rerr*

where "*is\_rerr*  $\xi \equiv \text{case msg } \xi \text{ of}$   
     *Rerr* *dests' sip'  $\Rightarrow \{ \xi(\text{dests} := \text{dests}', \text{sip} := \text{sip}') \}$*   
     |  $\_ \Rightarrow \{\}$ "

**lemma** *is\_rerr\_asm [dest!]*:

assumes " $\xi' \in \text{is\_rerr } \xi$ "

shows " $(\exists \text{dests}' \text{sip}'.$   
     *msg*  $\xi = \text{Rerr } \text{dests}' \text{sip}' \wedge$   
      $\xi' = \xi(\text{dests} := \text{dests}', \text{sip} := \text{sip}')"$

*<proof>*

**lemmas** *is\_msg\_defs =*

*is\_rerr\_def is\_rrep\_def is\_rreq\_def is\_pkt\_def is\_newpkt\_def*

**lemma** *is\_msg\_inv\_ip [simp]*:

$\xi' \in \text{is\_rerr } \xi \implies \text{ip } \xi' = \text{ip } \xi$   
 $\xi' \in \text{is\_rrep } \xi \implies \text{ip } \xi' = \text{ip } \xi$   
 $\xi' \in \text{is\_rreq } \xi \implies \text{ip } \xi' = \text{ip } \xi$   
 $\xi' \in \text{is\_pkt } \xi \implies \text{ip } \xi' = \text{ip } \xi$   
 $\xi' \in \text{is\_newpkt } \xi \implies \text{ip } \xi' = \text{ip } \xi$

*<proof>*

**lemma** *is\_msg\_inv\_sn [simp]*:

$\xi' \in \text{is\_rerr } \xi \implies \text{sn } \xi' = \text{sn } \xi$

```

" $\xi'$   $\in$  is_rrep  $\xi$   $\implies$  sn  $\xi'$  = sn  $\xi$ "
" $\xi'$   $\in$  is_rreq  $\xi$   $\implies$  sn  $\xi'$  = sn  $\xi$ "
" $\xi'$   $\in$  is_pkt  $\xi$   $\implies$  sn  $\xi'$  = sn  $\xi$ "
" $\xi'$   $\in$  is_newpkt  $\xi$   $\implies$  sn  $\xi'$  = sn  $\xi$ "
<proof>

```

lemma is\_msg\_inv\_rt [simp]:

```

" $\xi'$   $\in$  is_rerr  $\xi$   $\implies$  rt  $\xi'$  = rt  $\xi$ "
" $\xi'$   $\in$  is_rrep  $\xi$   $\implies$  rt  $\xi'$  = rt  $\xi$ "
" $\xi'$   $\in$  is_rreq  $\xi$   $\implies$  rt  $\xi'$  = rt  $\xi$ "
" $\xi'$   $\in$  is_pkt  $\xi$   $\implies$  rt  $\xi'$  = rt  $\xi$ "
" $\xi'$   $\in$  is_newpkt  $\xi$   $\implies$  rt  $\xi'$  = rt  $\xi$ "
<proof>

```

lemma is\_msg\_inv\_rreqs [simp]:

```

" $\xi'$   $\in$  is_rerr  $\xi$   $\implies$  rreqs  $\xi'$  = rreqs  $\xi$ "
" $\xi'$   $\in$  is_rrep  $\xi$   $\implies$  rreqs  $\xi'$  = rreqs  $\xi$ "
" $\xi'$   $\in$  is_rreq  $\xi$   $\implies$  rreqs  $\xi'$  = rreqs  $\xi$ "
" $\xi'$   $\in$  is_pkt  $\xi$   $\implies$  rreqs  $\xi'$  = rreqs  $\xi$ "
" $\xi'$   $\in$  is_newpkt  $\xi$   $\implies$  rreqs  $\xi'$  = rreqs  $\xi$ "
<proof>

```

lemma is\_msg\_inv\_store [simp]:

```

" $\xi'$   $\in$  is_rerr  $\xi$   $\implies$  store  $\xi'$  = store  $\xi$ "
" $\xi'$   $\in$  is_rrep  $\xi$   $\implies$  store  $\xi'$  = store  $\xi$ "
" $\xi'$   $\in$  is_rreq  $\xi$   $\implies$  store  $\xi'$  = store  $\xi$ "
" $\xi'$   $\in$  is_pkt  $\xi$   $\implies$  store  $\xi'$  = store  $\xi$ "
" $\xi'$   $\in$  is_newpkt  $\xi$   $\implies$  store  $\xi'$  = store  $\xi$ "
<proof>

```

lemma is\_msg\_inv\_sip [simp]:

```

" $\xi'$   $\in$  is_pkt  $\xi$   $\implies$  sip  $\xi'$  = sip  $\xi$ "
" $\xi'$   $\in$  is_newpkt  $\xi$   $\implies$  sip  $\xi'$  = sip  $\xi$ "
<proof>

```

### 0.4.3 The protocol process

datatype pseqp =

```

  PAadv
| PNewPkt
| PPkt
| PRreq
| PRrep
| PRerr

```

fun nat\_of\_seqp :: "pseqp  $\Rightarrow$  nat"

where

```

  "nat_of_seqp PAadv = 1"
| "nat_of_seqp PPkt = 2"
| "nat_of_seqp PNewPkt = 3"
| "nat_of_seqp PRreq = 4"
| "nat_of_seqp PRrep = 5"
| "nat_of_seqp PRerr = 6"

```

instantiation "pseqp" :: ord

begin

definition less\_eq\_seqp [iff]: " $l1 \leq l2 = (\text{nat\_of\_seqp } l1 \leq \text{nat\_of\_seqp } l2)$ "

definition less\_seqp [iff]: " $l1 < l2 = (\text{nat\_of\_seqp } l1 < \text{nat\_of\_seqp } l2)$ "

instance <proof>

end

abbreviation AODV

where

```

  "AODV  $\equiv$   $\lambda$ _. [[clear_locals]] call(PAadv)"

```

abbreviation PKT

where

"PKT args ≡

```
[[ξ. let (data, dip, oip) = args ξ in
  (clear_locals ξ) (| data := data, dip := dip, oip := oip |)]
call(PPkt)"
```

abbreviation NEWPKT

where

"NEWPKT args ≡

```
[[ξ. let (data, dip) = args ξ in
  (clear_locals ξ) (| data := data, dip := dip |)]
call(PNewPkt)"
```

abbreviation RREQ

where

"RREQ args ≡

```
[[ξ. let (hops, rreqid, dip, dsn, dsk, oip, osn, sip) = args ξ in
  (clear_locals ξ) (| hops := hops, rreqid := rreqid, dip := dip,
    dsn := dsn, dsk := dsk, oip := oip,
    osn := osn, sip := sip |)]
call(PRreq)"
```

abbreviation RREP

where

"RREP args ≡

```
[[ξ. let (hops, dip, dsn, oip, sip) = args ξ in
  (clear_locals ξ) (| hops := hops, dip := dip, dsn := dsn,
    oip := oip, sip := sip |)]
call(PRrep)"
```

abbreviation RERR

where

"RERR args ≡

```
[[ξ. let (dests, sip) = args ξ in
  (clear_locals ξ) (| dests := dests, sip := sip |)]
call(PRerr)"
```

fun  $\Gamma_{AODV} :: "(state, msg, pseqp, pseqp label) seqp\_env"$

where

" $\Gamma_{AODV}$  PAodv = labelled PAodv (

receive( $\lambda$ msg' ξ. ξ (| msg := msg' |)).

(  $\langle$ is\_newpkt $\rangle$  NEWPKT( $\lambda$ ξ. (data ξ, ip ξ))

⊕  $\langle$ is\_pkt $\rangle$  PKT( $\lambda$ ξ. (data ξ, dip ξ, oip ξ))

⊕  $\langle$ is\_rreq $\rangle$

[[ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {f}) |)]

RREQ( $\lambda$ ξ. (hops ξ, rreqid ξ, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, sip ξ))

⊕  $\langle$ is\_rrep $\rangle$

[[ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {f}) |)]

RREP( $\lambda$ ξ. (hops ξ, dip ξ, dsn ξ, oip ξ, sip ξ))

⊕  $\langle$ is\_rerr $\rangle$

[[ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {f}) |)]

RERR( $\lambda$ ξ. (dests ξ, sip ξ))

)

⊕  $\langle$  $\lambda$ ξ. { ξ(| dip := dip |) | dip. dip ∈ qD(store ξ) ∩ vD(rt ξ) } $\rangle$

[[ξ. ξ (| data := hd( $\sigma_{queue}$ (store ξ, dip ξ)) |)]

unicast( $\lambda$ ξ. the (nhop (rt ξ) (dip ξ)),  $\lambda$ ξ. pkt(data ξ, dip ξ, ip ξ)).

[[ξ. ξ (| store := the (drop (dip ξ) (store ξ)) |)]

AODV()

▷ [[ξ. ξ (| dests := ( $\lambda$ rip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
 then Some (inc (sqn (rt ξ) rip)) else None) |)]

[[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)]

[[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |)]



```

[[ξ. ξ (| pre := ∪{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } )]]
[[ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
then (dests ξ) rip else None) )]]
groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)). AODV()
⊕ ⟨λξ. { ξ(| dip := dip )
| dip. dip ∈ qD(store ξ) - vD(rt ξ) ∧ the (σp-flag(store ξ, dip)) = req }⟩
[[ξ. ξ (| store := unsetRRF (store ξ) (dip ξ) )]]
[[ξ. ξ (| sn := inc (sn ξ) )]]
[[ξ. ξ (| rreqid := nrreqid (rreqs ξ) (ip ξ) )]]
[[ξ. ξ (| rreqs := rreqs ξ ∪ {(ip ξ, rreqid ξ)} )]]
broadcast(λξ. rreq(0, rreqid ξ, dip ξ, sqn (rt ξ) (dip ξ), sqnf (rt ξ) (dip ξ),
ip ξ, sn ξ, ip ξ)). AODV()"

| "ΓAODV PNewPkt = labelled PNewPkt (
⟨ξ. dip ξ = ip ξ⟩
deliver(λξ. data ξ).AODV()
⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
[[ξ. ξ (| store := add (data ξ) (dip ξ) (store ξ) )]]
AODV()")

| "ΓAODV PPkt = labelled PPkt (
⟨ξ. dip ξ = ip ξ⟩
deliver(λξ. data ξ).AODV()
⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
(
⟨ξ. dip ξ ∈ vD (rt ξ)⟩
unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, oip ξ)).AODV()
▷
[[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
then Some (inc (sqn (rt ξ) rip)) else None) )]]
[[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) )]]
[[ξ. ξ (| store := setRRF (store ξ) (dests ξ) )]]
[[ξ. ξ (| pre := ∪{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } )]]
[[ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
then (dests ξ) rip else None) )]]
groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
⊕ ⟨ξ. dip ξ ∉ vD (rt ξ)⟩
(
⟨ξ. dip ξ ∈ iD (rt ξ)⟩
groupcast(λξ. the (precs (rt ξ) (dip ξ)),
λξ. rerr([dip ξ ↦ sqn (rt ξ) (dip ξ)], ip ξ)). AODV()
⊕ ⟨ξ. dip ξ ∉ iD (rt ξ)⟩
AODV()
)
)
))"

| "ΓAODV PRreq = labelled PRreq (
⟨ξ. (oip ξ, rreqid ξ) ∈ rreqs ξ⟩
AODV()
⊕ ⟨ξ. (oip ξ, rreqid ξ) ∉ rreqs ξ⟩
[[ξ. ξ (| rt := update (rt ξ) (oip ξ) (osn ξ, kno, val, hops ξ + 1, sip ξ, { }) )]]
[[ξ. ξ (| rreqs := rreqs ξ ∪ {(oip ξ, rreqid ξ)} )]]
(
⟨ξ. dip ξ = ip ξ⟩
[[ξ. ξ (| sn := max (sn ξ) (dsn ξ) )]]
unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(0, dip ξ, sn ξ, oip ξ, ip ξ)).AODV()
▷
[[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
then Some (inc (sqn (rt ξ) rip)) else None) )]]
[[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) )]]
[[ξ. ξ (| store := setRRF (store ξ) (dests ξ) )]]
[[ξ. ξ (| pre := ∪{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } )]]
[[ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
then (dests ξ) rip else None) )]]

```

```

    groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
(
  ⟨ξ. dip ξ ∈ vD (rt ξ) ∧ dsn ξ ≤ sqn (rt ξ) (dip ξ) ∧ sqnf (rt ξ) (dip ξ) = kno⟩
  [[ξ. ξ (| rt := the (addpreRT (rt ξ) (dip ξ) {sip ξ}) )]]
  [[ξ. ξ (| rt := the (addpreRT (rt ξ) (oip ξ) {the (nhop (rt ξ) (dip ξ))}) )]]
  unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(the (dhops (rt ξ) (dip ξ)), dip ξ,
    sqn (rt ξ) (dip ξ), oip ξ, ip ξ)).
  AODV()
▷
  [[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
    then Some (inc (sqn (rt ξ) rip)) else None) )]]
  [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) )]]
  [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) )]]
  [[ξ. ξ (| pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } )]]
  [[ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ { })
    then (dests ξ) rip else None) )]]
  groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
⊕ ⟨ξ. dip ξ ∉ vD (rt ξ) ∨ sqn (rt ξ) (dip ξ) < dsn ξ ∨ sqnf (rt ξ) (dip ξ) = unk⟩
  broadcast(λξ. rreq(hops ξ + 1, rreqid ξ, dip ξ, max (sqn (rt ξ) (dip ξ)) (dsn ξ),
    dsk ξ, oip ξ, osn ξ, ip ξ)).
  AODV()
)
)"

```

```

| "ΓAODV PRrep = labelled PRrep (
  ⟨ξ. rt ξ ≠ update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, { })⟩
  (
    [[ξ. ξ (| rt := update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, { }) )]]
    (
      ⟨ξ. oip ξ = ip ξ⟩
      AODV()
    ⊕ ⟨ξ. oip ξ ≠ ip ξ⟩
    (
      ⟨ξ. oip ξ ∈ vD (rt ξ)⟩
      [[ξ. ξ (| rt := the (addpreRT (rt ξ) (dip ξ) {the (nhop (rt ξ) (oip ξ))}) )]]
      [[ξ. ξ (| rt := the (addpreRT (rt ξ) (the (nhop (rt ξ) (dip ξ)))
        {the (nhop (rt ξ) (oip ξ))}) )]]
      unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(hops ξ + 1, dip ξ, dsn ξ, oip ξ, ip ξ)).
      AODV()
    ▷
      [[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
        then Some (inc (sqn (rt ξ) rip)) else None) )]]
      [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) )]]
      [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) )]]
      [[ξ. ξ (| pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } )]]
      [[ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ { })
        then (dests ξ) rip else None) )]]
      groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
    ⊕ ⟨ξ. oip ξ ∉ vD (rt ξ)⟩
      AODV()
    )
  )
)
⊕ ⟨ξ. rt ξ = update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, { })⟩
  AODV()
)"

```

```

| "ΓAODV PRerr = labelled PRerr (
  [[ξ. ξ (| dests := (λrip. case (dests ξ) rip of None ⇒ None
    | Some rsn ⇒ if rip ∈ vD (rt ξ) ∧ the (nhop (rt ξ) rip) = sip ξ
      ∧ sqn (rt ξ) rip < rsn then Some rsn else None) )]]
  [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) )]]
  [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) )]]

```

```

[[ξ. ξ (| pre := ∪{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } )]]
[[ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
    then (dests ξ) rip else None) )]]
groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)). AODV()"

declare ΓAODV.simps [simp del, code del]
lemmas ΓAODV.simps [simp, code] = ΓAODV.simps [simplified]

fun ΓAODV_skeleton
where
  "ΓAODV_skeleton PAodv = seqp_skeleton (ΓAODV PAodv)"
  | "ΓAODV_skeleton PNewPkt = seqp_skeleton (ΓAODV PNewPkt)"
  | "ΓAODV_skeleton PPkt = seqp_skeleton (ΓAODV PPkt)"
  | "ΓAODV_skeleton PRreq = seqp_skeleton (ΓAODV PRreq)"
  | "ΓAODV_skeleton PRrep = seqp_skeleton (ΓAODV PRrep)"
  | "ΓAODV_skeleton PRerr = seqp_skeleton (ΓAODV PRerr)"

lemma ΓAODV_skeleton_wf [simp]:
  "wellformed ΓAODV_skeleton"
  ⟨proof⟩

declare ΓAODV_skeleton.simps [simp del, code del]
lemmas ΓAODV_skeleton.simps [simp, code]
  = ΓAODV_skeleton.simps [simplified ΓAODV.simps seqp_skeleton.simps]

lemma aodv_proc_cases [dest]:
  fixes p pn
  shows "p ∈ ctermsl (ΓAODV pn) ⇒
    (p ∈ ctermsl (ΓAODV PAodv) ∨
     p ∈ ctermsl (ΓAODV PNewPkt) ∨
     p ∈ ctermsl (ΓAODV PPkt) ∨
     p ∈ ctermsl (ΓAODV PRreq) ∨
     p ∈ ctermsl (ΓAODV PRrep) ∨
     p ∈ ctermsl (ΓAODV PRerr))"
  ⟨proof⟩

definition σAODV :: "ip ⇒ (state × (state, msg, pseq, pseq label) seq) set"
where "σAODV i ≡ {(aodv_init i, ΓAODV PAodv)}"

abbreviation paodv
  :: "ip ⇒ (state × (state, msg, pseq, pseq label) seq, msg seq_action) automaton"
where
  "paodv i ≡ (| init = σAODV i, trans = seqp_sos ΓAODV |)"

lemma aodv_trans: "trans (paodv i) = seqp_sos ΓAODV"
  ⟨proof⟩

lemma aodv_control_within [simp]: "control_within ΓAODV (init (paodv i))"
  ⟨proof⟩

lemma aodv_wf [simp]:
  "wellformed ΓAODV"
  ⟨proof⟩

lemmas aodv_labels_not_empty [simp] = labels_not_empty [OF aodv_wf]

lemma aodv_ex_label [intro]: "∃l. l ∈ labels ΓAODV p"
  ⟨proof⟩

lemma aodv_ex_labelE [elim]:
  assumes "∀l ∈ labels ΓAODV p. P l p"
  and "∃p l. P l p ⇒ Q"
  shows "Q"
  ⟨proof⟩

```

```
lemma aadv_simple_labels [simp]: "simple_labels  $\Gamma_{AODV}$ "
  <proof>
```

```
lemma  $\sigma_{AODV}$ _labels [simp]: " $(\xi, p) \in \sigma_{AODV} i \implies \text{labels } \Gamma_{AODV} p = \{PAadv-:0\}$ "
  <proof>
```

```
lemma aadv_init_kD_empty [simp]:
  " $(\xi, p) \in \sigma_{AODV} i \implies kD (rt \xi) = \{\}$ "
  <proof>
```

```
lemma aadv_init_sip_not_ip [simp]: " $\neg(\text{sip } (aadv\_init\ i) = i)$ " <proof>
```

```
lemma aadv_init_sip_not_ip' [simp]:
  assumes " $(\xi, p) \in \sigma_{AODV} i$ "
  shows " $\text{sip } \xi \neq ip\ \xi$ "
  <proof>
```

```
lemma aadv_init_sip_not_i [simp]:
  assumes " $(\xi, p) \in \sigma_{AODV} i$ "
  shows " $\text{sip } \xi \neq i$ "
  <proof>
```

```
lemma clear_locals_sip_not_ip':
  assumes " $ip\ \xi = i$ "
  shows " $\neg(\text{sip } (clear\_locals\ \xi) = i)$ "
  <proof>
```

Stop the simplifier from descending into process terms.

```
declare seqp_congs [cong]
```

Configure the main invariant tactic for AODV.

```
declare
   $\Gamma_{AODV\_simps}$  [cterms_env]
  aadv_proc_cases [cterms1_cases]
  seq_invariant_ctermsI [OF aadv_wf aadv_control_within aadv_simple_labels aadv_trans,
    cterms_intros]
  seq_step_invariant_ctermsI [OF aadv_wf aadv_control_within aadv_simple_labels aadv_trans,
    cterms_intros]
```

```
end
```

## 0.5 Invariant assumptions and properties

```
theory Aadv_Predicates
  imports Aadv
  begin
```

Definitions for expression assumptions on incoming messages and properties of outgoing messages.

```
abbreviation not_Pkt :: "msg  $\Rightarrow$  bool"
where "not_Pkt m  $\equiv$  case m of Pkt _ _ _  $\Rightarrow$  False | _  $\Rightarrow$  True"
```

```
definition msg_sender :: "msg  $\Rightarrow$  ip"
where "msg_sender m  $\equiv$  case m of Rreq _ _ _ _ _ ipc  $\Rightarrow$  ipc
  | Rrep _ _ _ _ ipc  $\Rightarrow$  ipc
  | Rerr _ ipc  $\Rightarrow$  ipc
  | Pkt _ _ ipc  $\Rightarrow$  ipc"
```

```
lemma msg_sender_simps [simp]:
  " $\wedge$ hops rreqid dip dsn dsk oip osn sip.
    msg_sender (Rreq hops rreqid dip dsn dsk oip osn sip) = sip"
  " $\wedge$ hops dip dsn oip sip. msg_sender (Rrep hops dip dsn oip sip) = sip"
  " $\wedge$ dests sip. msg_sender (Rerr dests sip) = sip"
```

```
" $\wedge$ d dip sip.          msg_sender (Pkt d dip sip) = sip"
<proof>
```

```
definition msg_zhops :: "msg  $\Rightarrow$  bool"
where "msg_zhops m  $\equiv$  case m of
```

```
  Rreq hopsc _ dipc _ _ oipc _ sipc  $\Rightarrow$  hopsc = 0  $\longrightarrow$  oipc = sipc
  | Rrep hopsc dipc _ _ sipc  $\Rightarrow$  hopsc = 0  $\longrightarrow$  dipc = sipc
  | _  $\Rightarrow$  True"
```

```
lemma msg_zhops_simps [simp]:
```

```
" $\wedge$ hops rreqid dip dsn dsk oip osn sip.
  msg_zhops (Rreq hops rreqid dip dsn dsk oip osn sip) = (hops = 0  $\longrightarrow$  oip = sip)"
" $\wedge$ hops dip dsn oip sip. msg_zhops (Rrep hops dip dsn oip sip) = (hops = 0  $\longrightarrow$  dip = sip)"
" $\wedge$ dests sip.          msg_zhops (Rerr dests sip)          = True"
" $\wedge$ dip.               msg_zhops (Newpkt d dip)           = True"
" $\wedge$ dip sip.          msg_zhops (Pkt d dip sip)           = True"
<proof>
```

```
definition rreq_rrep_sn :: "msg  $\Rightarrow$  bool"
```

```
where "rreq_rrep_sn m  $\equiv$  case m of Rreq _ _ _ _ _ osnc _  $\Rightarrow$  osnc  $\geq$  1
  | Rrep _ _ dsnc _ _  $\Rightarrow$  dsnc  $\geq$  1
  | _  $\Rightarrow$  True"
```

```
lemma rreq_rrep_sn_simps [simp]:
```

```
" $\wedge$ hops rreqid dip dsn dsk oip osn sip.
  rreq_rrep_sn (Rreq hops rreqid dip dsn dsk oip osn sip) = (osn  $\geq$  1)"
" $\wedge$ hops dip dsn oip sip. rreq_rrep_sn (Rrep hops dip dsn oip sip) = (dsn  $\geq$  1)"
" $\wedge$ dests sip.          rreq_rrep_sn (Rerr dests sip) = True"
" $\wedge$ dip.               rreq_rrep_sn (Newpkt d dip) = True"
" $\wedge$ dip sip.          rreq_rrep_sn (Pkt d dip sip) = True"
<proof>
```

```
definition rreq_rrep_fresh :: "rt  $\Rightarrow$  msg  $\Rightarrow$  bool"
```

```
where "rreq_rrep_fresh crt m  $\equiv$  case m of Rreq hopsc _ _ _ oipc osnc ipcc  $\Rightarrow$  (ipcc  $\neq$  oipc  $\longrightarrow$ 
  oipc  $\in$  kD(crt)  $\wedge$  (sqn crt oipc > osnc
     $\vee$  (sqn crt oipc = osnc
       $\wedge$  the (dhops crt oipc)  $\leq$  hopsc
       $\wedge$  the (flag crt oipc) = val)))
  | Rrep hopsc dipc dsnc _ ipcc  $\Rightarrow$  (ipcc  $\neq$  dipc  $\longrightarrow$ 
  dipc  $\in$  kD(crt)
     $\wedge$  sqn crt dipc = dsnc
     $\wedge$  the (dhops crt dipc) = hopsc
     $\wedge$  the (flag crt dipc) = val)
  | _  $\Rightarrow$  True"
```

```
lemma rreq_rrep_fresh_simps [simp]:
```

```
" $\wedge$ hops rreqid dip dsn dsk oip osn sip.
  rreq_rrep_fresh crt (Rreq hops rreqid dip dsn dsk oip osn sip) =
  (sip  $\neq$  oip  $\longrightarrow$  oip  $\in$  kD(crt)
     $\wedge$  (sqn crt oip > osn
       $\vee$  (sqn crt oip = osn
         $\wedge$  the (dhops crt oip)  $\leq$  hops
         $\wedge$  the (flag crt oip) = val))))"
" $\wedge$ hops dip dsn oip sip. rreq_rrep_fresh crt (Rrep hops dip dsn oip sip) =
  (sip  $\neq$  dip  $\longrightarrow$  dip  $\in$  kD(crt)
     $\wedge$  sqn crt dip = dsn
     $\wedge$  the (dhops crt dip) = hops
     $\wedge$  the (flag crt dip) = val)"
" $\wedge$ dests sip.          rreq_rrep_fresh crt (Rerr dests sip) = True"
" $\wedge$ dip.               rreq_rrep_fresh crt (Newpkt d dip) = True"
" $\wedge$ dip sip.          rreq_rrep_fresh crt (Pkt d dip sip) = True"
<proof>
```

```
definition rerr_invalid :: "rt  $\Rightarrow$  msg  $\Rightarrow$  bool"
```



```

lemma nsqn_r_val [simp]:
  " $\bigwedge$  dsn dsk hops nhip pre. nsqn_r (dsn, dsk, val, hops, nhip, pre) = dsn"
  <proof>

lemma nsqn_r_inv [simp]:
  " $\bigwedge$  dsn dsk hops nhip pre. nsqn_r (dsn, dsk, inv, hops, nhip, pre) = dsn - 1"
  <proof>

lemma nsqn_r_lte_dsn [simp]:
  " $\bigwedge$  dsn dsk flag hops nhip pre. nsqn_r (dsn, dsk, flag, hops, nhip, pre)  $\leq$  dsn"
  <proof>

```

## On routes in routing tables

### definition

```
nsqn :: "rt  $\Rightarrow$  ip  $\Rightarrow$  sqn"
```

### where

```
"nsqn  $\equiv$   $\lambda$ rt dip. case  $\sigma_{route}(rt, dip)$  of None  $\Rightarrow$  0 | Some r  $\Rightarrow$  nsqn_r(r)"
```

### lemma nsqn\_sqn\_def:

```
" $\bigwedge$ rt dip. nsqn rt dip = (if flag rt dip = Some val  $\vee$  sqn rt dip = 0
  then sqn rt dip else sqn rt dip - 1)"
```

```
<proof>
```

### lemma not\_in\_kD\_nsqn [simp]:

```
assumes "dip  $\notin$  kD(rt)"
```

```
shows "nsqn rt dip = 0"
```

```
<proof>
```

### lemma kD\_nsqn:

```
assumes "dip  $\in$  kD(rt)"
```

```
shows "nsqn rt dip = nsqn_r(the ( $\sigma_{route}(rt, dip)$ ))"
```

```
<proof>
```

### lemma nsqnr\_r\_flag\_pred [simp, intro]:

```
fixes dsn dsk flag hops nhip pre
```

```
assumes "P (nsqn_r (dsn, dsk, val, hops, nhip, pre))"
```

```
and "P (nsqn_r (dsn, dsk, inv, hops, nhip, pre))"
```

```
shows "P (nsqn_r (dsn, dsk, flag, hops, nhip, pre))"
```

```
<proof>
```

### lemma nsqn\_r\_addpreRT\_inv [simp]:

```
" $\bigwedge$ rt dip npre dip'. dip  $\in$  kD(rt)  $\implies$ 
```

```
nsqn_r (the (the (addpreRT rt dip npre) dip')) = nsqn_r (the (rt dip'))"
```

```
<proof>
```

### lemma sqn\_nsqn:

```
" $\bigwedge$ rt dip. sqn rt dip - 1  $\leq$  nsqn rt dip"
```

```
<proof>
```

### lemma nsqn\_sqn: "nsqn rt dip $\leq$ sqn rt dip"

```
<proof>
```

### lemma val\_nsqn\_sqn [elim, simp]:

```
assumes "ip  $\in$  kD(rt)"
```

```
and "the (flag rt ip) = val"
```

```
shows "nsqn rt ip = sqn rt ip"
```

```
<proof>
```

### lemma vD\_nsqn\_sqn [elim, simp]:

```
assumes "ip  $\in$  vD(rt)"
```

```
shows "nsqn rt ip = sqn rt ip"
```

```
<proof>
```

```

lemma inv_nsqn_sqn [elim, simp]:
  assumes "ip ∈ kD(rt)"
    and "the (flag rt ip) = inv"
  shows "nsqn rt ip = sqn rt ip - 1"
  ⟨proof⟩

```

```

lemma iD_nsqn_sqn [elim, simp]:
  assumes "ip ∈ iD(rt)"
  shows "nsqn rt ip = sqn rt ip - 1"
  ⟨proof⟩

```

```

lemma nsqn_update_changed_kno_val [simp]: "∧rt ip dsn dsk hops nhip.
  rt ≠ update rt ip (dsn, kno, val, hops, nhip, { })
  ⇒ nsqn (update rt ip (dsn, kno, val, hops, nhip, { })) ip = dsn"
  ⟨proof⟩

```

```

lemma nsqn_addpreRT_inv [simp]:
  "∧rt dip npre dip'. dip ∈ kD(rt) ⇒
  nsqn (the (addpreRT rt dip npre)) dip' = nsqn rt dip'"
  ⟨proof⟩

```

```

lemma nsqn_update_other [simp]:
  fixes dsn dsk flag hops dip nhip pre rt ip
  assumes "dip ≠ ip"
  shows "nsqn (update rt ip (dsn, dsk, flag, hops, nhip, pre)) dip = nsqn rt dip"
  ⟨proof⟩

```

```

lemma nsqn_invalidate_eq:
  assumes "dip ∈ kD(rt)"
    and "dests dip = Some rsn"
  shows "nsqn (invalidate rt dests) dip = rsn - 1"
  ⟨proof⟩

```

```

lemma nsqn_invalidate_other [simp]:
  assumes "dip ∈ kD(rt)"
    and "dip ∉ dom dests"
  shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
  ⟨proof⟩

```

## 0.6.2 Comparing routes

**definition**

```

fresher :: "r ⇒ r ⇒ bool" ("(_/ ⊆ _)" [51, 51] 50)

```

where

```

"fresher r r' ≡ ((nsqnr r < nsqnr r') ∨ (nsqnr r = nsqnr r' ∧ π5(r) ≥ π5(r')))"

```

```

lemma fresherI1 [intro]:
  assumes "nsqnr r < nsqnr r'"
  shows "r ⊆ r'"
  ⟨proof⟩

```

```

lemma fresherI2 [intro]:
  assumes "nsqnr r = nsqnr r'"
    and "π5(r) ≥ π5(r')"
  shows "r ⊆ r'"
  ⟨proof⟩

```

```

lemma fresherI [intro]:
  assumes "(nsqnr r < nsqnr r') ∨ (nsqnr r = nsqnr r' ∧ π5(r) ≥ π5(r'))"
  shows "r ⊆ r'"
  ⟨proof⟩

```

```

lemma fresherE [elim]:

```



```

assumes "r ⊆ r'"
  and "nsqnr r < nsqnr r' ⇒ P r r'"
  and "nsqnr r = nsqnr r' ∧ π5(r) ≥ π5(r') ⇒ P r r'"
shows "P r r'"
⟨proof⟩

```

```

lemma fresher_refl [simp]: "r ⊆ r"
⟨proof⟩

```

```

lemma fresher_trans [elim, trans]:
"[[ x ⊆ y; y ⊆ z ]] ⇒ x ⊆ z"
⟨proof⟩

```

```

lemma not_fresher_trans [elim, trans]:
"[[ ¬(x ⊆ y); ¬(z ⊆ x) ]] ⇒ ¬(z ⊆ y)"
⟨proof⟩

```

```

lemma fresher_dsn_flag_hops_const [simp]:
fixes dsn dsk dsk' flag hops nhip nhip' pre pre'
shows "(dsn, dsk, flag, hops, nhip, pre) ⊆ (dsn, dsk', flag, hops, nhip', pre)"
⟨proof⟩

```

```

lemma addpre_fresher [simp]: "∧r npre. r ⊆ (addpre r npre)"
⟨proof⟩

```

### 0.6.3 Comparing routing tables

definition

```
rt_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"
```

where

```
"rt_fresher ≡ λdip rt rt'. (the (σroute(rt, dip))) ⊆ (the (σroute(rt', dip)))"
```

abbreviation

```
rt_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" ("(_/ ⊆_ _)" [51, 999, 51] 50)
```

where

```
"rt1 ⊆i rt2 ≡ rt_fresher i rt1 rt2"
```

lemma rt\_fresher\_def':

```
"(rt1 ⊆i rt2) = (nsqnr (the (rt1 i)) < nsqnr (the (rt2 i)) ∨
nsqnr (the (rt1 i)) = nsqnr (the (rt2 i)) ∧ π5 (the (rt2 i)) ≤ π5 (the (rt1 i)))"
```

⟨proof⟩

lemma single\_rt\_fresher [intro]:

```
assumes "the (rt1 ip) ⊆ the (rt2 ip)"
shows "rt1 ⊆ip rt2"
```

⟨proof⟩

lemma rt\_fresher\_single [intro]:

```
assumes "rt1 ⊆ip rt2"
shows "the (rt1 ip) ⊆ the (rt2 ip)"
```

⟨proof⟩

lemma rt\_fresher\_def2:

```
assumes "dip ∈ kD(rt1)"
  and "dip ∈ kD(rt2)"
shows "(rt1 ⊆dip rt2) = (nsqn rt1 dip < nsqn rt2 dip
∨ (nsqn rt1 dip = nsqn rt2 dip
∧ the (dhops rt1 dip) ≥ the (dhops rt2 dip)))"
```

⟨proof⟩

lemma rt\_fresherI1 [intro]:

```
assumes "dip ∈ kD(rt1)"
  and "dip ∈ kD(rt2)"
  and "nsqn rt1 dip < nsqn rt2 dip"
```

```

  shows "rt1  $\sqsubseteq_{dip}$  rt2"
  <proof>

lemma rt_fresherI2 [intro]:
  assumes "dip  $\in$  kD(rt1)"
    and "dip  $\in$  kD(rt2)"
    and "nsqn rt1 dip = nsqn rt2 dip"
    and "the (dhops rt1 dip)  $\geq$  the (dhops rt2 dip)"
  shows "rt1  $\sqsubseteq_{dip}$  rt2"
  <proof>

lemma rt_fresherE [elim]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
    and "dip  $\in$  kD(rt1)"
    and "dip  $\in$  kD(rt2)"
    and "[[ nsqn rt1 dip < nsqn rt2 dip ]  $\implies$  P rt1 rt2 dip]"
    and "[[ nsqn rt1 dip = nsqn rt2 dip;
      the (dhops rt1 dip)  $\geq$  the (dhops rt2 dip) ]  $\implies$  P rt1 rt2 dip]"
  shows "P rt1 rt2 dip"
  <proof>

lemma rt_fresher_refl [simp]: "rt  $\sqsubseteq_{dip}$  rt"
  <proof>

lemma rt_fresher_trans [elim, trans]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
    and "rt2  $\sqsubseteq_{dip}$  rt3"
  shows "rt1  $\sqsubseteq_{dip}$  rt3"
  <proof>

lemma rt_fresher_if_Some [intro!]:
  assumes "the (rt dip)  $\sqsubseteq$  r"
  shows "rt  $\sqsubseteq_{dip}$  ( $\lambda$ ip. if ip = dip then Some r else rt ip)"
  <proof>

definition rt_fresh_as :: "ip  $\Rightarrow$  rt  $\Rightarrow$  rt  $\Rightarrow$  bool"
where
  "rt_fresh_as  $\equiv$   $\lambda$ dip rt1 rt2. (rt1  $\sqsubseteq_{dip}$  rt2)  $\wedge$  (rt2  $\sqsubseteq_{dip}$  rt1)"

abbreviation
  rt_fresh_as_syn :: "rt  $\Rightarrow$  ip  $\Rightarrow$  rt  $\Rightarrow$  bool" ("(_/  $\approx$  _)" [51, 999, 51] 50)
where
  "rt1  $\approx_i$  rt2  $\equiv$  rt_fresh_as i rt1 rt2"

lemma rt_fresh_as_refl [simp]: " $\wedge$ rt dip. rt  $\approx_{dip}$  rt"
  <proof>

lemma rt_fresh_as_trans [simp, intro, trans]:
  " $\wedge$ rt1 rt2 rt3 dip. [[ rt1  $\approx_{dip}$  rt2; rt2  $\approx_{dip}$  rt3 ]  $\implies$  rt1  $\approx_{dip}$  rt3"
  <proof>

lemma rt_fresh_asI [intro!]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
    and "rt2  $\sqsubseteq_{dip}$  rt1"
  shows "rt1  $\approx_{dip}$  rt2"
  <proof>

lemma rt_fresh_as_fresherI [intro]:
  assumes "dip  $\in$  kD(rt1)"
    and "dip  $\in$  kD(rt2)"
    and "the (rt1 dip)  $\sqsubseteq$  the (rt2 dip)"
    and "the (rt2 dip)  $\sqsubseteq$  the (rt1 dip)"
  shows "rt1  $\approx_{dip}$  rt2"
  <proof>

```

```

lemma nsqn_rt_fresh_asI:
  assumes "dip ∈ kD(rt)"
    and "dip ∈ kD(rt'"
    and "nsqn rt dip = nsqn rt' dip"
    and "π5(the (rt dip)) = π5(the (rt' dip))"
  shows "rt ≈dip rt'"
  ⟨proof⟩

lemma rt_fresh_asE [elim]:
  assumes "rt1 ≈dip rt2"
    and "[[ rt1 ⊆dip rt2; rt2 ⊆dip rt1 ] ] ⇒ P rt1 rt2 dip"
  shows "P rt1 rt2 dip"
  ⟨proof⟩

lemma rt_fresh_asD1 [dest]:
  assumes "rt1 ≈dip rt2"
  shows "rt1 ⊆dip rt2"
  ⟨proof⟩

lemma rt_fresh_asD2 [dest]:
  assumes "rt1 ≈dip rt2"
  shows "rt2 ⊆dip rt1"
  ⟨proof⟩

lemma rt_fresh_as_sym:
  assumes "rt1 ≈dip rt2"
  shows "rt2 ≈dip rt1"
  ⟨proof⟩

lemma not_rt_fresh_asI1 [intro]:
  assumes "¬ (rt1 ⊆dip rt2)"
  shows "¬ (rt1 ≈dip rt2)"
  ⟨proof⟩

lemma not_rt_fresh_asI2 [intro]:
  assumes "¬ (rt2 ⊆dip rt1)"
  shows "¬ (rt1 ≈dip rt2)"
  ⟨proof⟩

lemma not_single_rt_fresher [elim]:
  assumes "¬(the (rt1 ip) ⊆ the (rt2 ip))"
  shows "¬(rt1 ⊆ip rt2)"
  ⟨proof⟩

lemmas not_single_rt_fresh_asI1 [intro] = not_rt_fresh_asI1 [OF not_single_rt_fresher]
lemmas not_single_rt_fresh_asI2 [intro] = not_rt_fresh_asI2 [OF not_single_rt_fresher]

lemma not_rt_fresher_single [elim]:
  assumes "¬(rt1 ⊆ip rt2)"
  shows "¬(the (rt1 ip) ⊆ the (rt2 ip))"
  ⟨proof⟩

lemma rt_fresh_as_nsqnr:
  assumes "dip ∈ kD(rt1)"
    and "dip ∈ kD(rt2)"
    and "rt1 ≈dip rt2"
  shows "nsqnr (the (rt2 dip)) = nsqnr (the (rt1 dip))"
  ⟨proof⟩

lemma rt_fresher_mapupd [intro!]:
  assumes "dip ∈ kD(rt)"
    and "the (rt dip) ⊆ r"
  shows "rt ⊆dip rt(dip ↦ r)"

```

*<proof>*

**lemma** *rt\_fresher\_map\_update\_other* [intro!]:  
 assumes "dip ∈ kD(rt)"  
 and "dip ≠ ip"  
 shows "rt  $\sqsubseteq_{\text{dip}}$  rt(ip ↦ r)"  
*<proof>*

**lemma** *rt\_fresher\_update\_other* [simp]:  
 assumes *inkD*: "dip ∈ kD(rt)"  
 and "dip ≠ ip"  
 shows "rt  $\sqsubseteq_{\text{dip}}$  update rt ip r"  
*<proof>*

**theorem** *rt\_fresher\_update* [simp]:  
 assumes "dip ∈ kD(rt)"  
 and "the (dhops rt dip) ≥ 1"  
 and "update\_arg\_wf r"  
 shows "rt  $\sqsubseteq_{\text{dip}}$  update rt ip r"  
*<proof>*

**theorem** *rt\_fresher\_invalidate* [simp]:  
 assumes "dip ∈ kD(rt)"  
 and *indests*: " $\forall \text{rip} \in \text{dom}(\text{dests}). \text{rip} \in \text{vD}(\text{rt}) \wedge \text{sqn rt rip} < \text{the}(\text{dests rip})$ "  
 shows "rt  $\sqsubseteq_{\text{dip}}$  invalidate rt dests"  
*<proof>*

**lemma** *nsqn<sub>r</sub>\_invalidate* [simp]:  
 assumes "dip ∈ kD(rt)"  
 and "dip ∈ dom(dests)"  
 shows "nsqn<sub>r</sub> (the (invalidate rt dests dip)) = the (dests dip) - 1"  
*<proof>*

**lemma** *rt\_fresh\_as\_inc\_invalidate* [simp]:  
 assumes "dip ∈ kD(rt)"  
 and " $\forall \text{rip} \in \text{dom}(\text{dests}). \text{rip} \in \text{vD}(\text{rt}) \wedge \text{the}(\text{dests rip}) = \text{inc}(\text{sqn rt rip})$ "  
 shows "rt  $\approx_{\text{dip}}$  invalidate rt dests"  
*<proof>*

**lemmas** *rt\_fresher\_inc\_invalidate* [simp] = *rt\_fresh\_as\_inc\_invalidate* [THEN *rt\_fresh\_asD1*]

**lemma** *rt\_fresh\_as\_addpreRT* [simp]:  
 assumes "ip ∈ kD(rt)"  
 shows "rt  $\approx_{\text{dip}}$  the (addpreRT rt ip npre)"  
*<proof>*

**lemmas** *rt\_fresher\_addpreRT* [simp] = *rt\_fresh\_as\_addpreRT* [THEN *rt\_fresh\_asD1*]

## 0.6.4 Strictly comparing routing tables

**definition** *rt\_strictly\_fresher* :: "ip ⇒ rt ⇒ rt ⇒ bool"

where

"rt\_strictly\_fresher  $\equiv \lambda \text{dip rt1 rt2}. (\text{rt1} \sqsubseteq_{\text{dip}} \text{rt2}) \wedge \neg(\text{rt1} \approx_{\text{dip}} \text{rt2})$ "

**abbreviation**

*rt\_strictly\_fresher\_syn* :: "rt ⇒ ip ⇒ rt ⇒ bool" (" $\_ / \sqsubseteq \_$ " [51, 999, 51] 50)

where

"rt1  $\sqsubseteq_i$  rt2  $\equiv \text{rt_strictly_fresher } i \text{ rt1 rt2}$ "

**lemma** *rt\_strictly\_fresher\_def''*:  
 "rt1  $\sqsubseteq_i$  rt2 = ((rt1  $\sqsubseteq_i$  rt2) ∧ ¬(rt2  $\sqsubseteq_i$  rt1))"  
*<proof>*

**lemma** *rt\_strictly\_fresherI'* [intro]:

```

assumes "rt1  $\sqsubseteq_i$  rt2"
  and " $\neg$ (rt2  $\sqsubseteq_i$  rt1)"
  shows "rt1  $\sqsubseteq_i$  rt2"
<proof>

lemma rt_strictly_fresherE' [elim]:
  assumes "rt1  $\sqsubseteq_i$  rt2"
  and "[[ rt1  $\sqsubseteq_i$  rt2;  $\neg$ (rt2  $\sqsubseteq_i$  rt1) ] ]  $\implies$  P rt1 rt2 i"
  shows "P rt1 rt2 i"
<proof>

lemma rt_strictly_fresherI [intro]:
  assumes "rt1  $\sqsubseteq_i$  rt2"
  and " $\neg$ (rt1  $\approx_i$  rt2)"
  shows "rt1  $\sqsubseteq_i$  rt2"
<proof>

lemmas rt_strictly_fresher_singleI [elim] = rt_strictly_fresherI [OF single_rt_fresher]

lemma rt_strictly_fresherE [elim]:
  assumes "rt1  $\sqsubseteq_i$  rt2"
  and "[[ rt1  $\sqsubseteq_i$  rt2;  $\neg$ (rt1  $\approx_i$  rt2) ] ]  $\implies$  P rt1 rt2 i"
  shows "P rt1 rt2 i"
<proof>

lemma rt_strictly_fresher_def':
  "rt1  $\sqsubseteq_i$  rt2 =
  (nsqnr (the (rt1 i)) < nsqnr (the (rt2 i))
   $\vee$  (nsqnr (the (rt1 i)) = nsqnr (the (rt2 i))  $\wedge$   $\pi_5$ (the (rt1 i)) >  $\pi_5$ (the (rt2 i))))"
<proof>

lemma rt_strictly_fresher_fresherD [dest]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
  shows "the (rt1 dip)  $\sqsubseteq$  the (rt2 dip)"
<proof>

lemma rt_strictly_fresher_not_fresh_asD [dest]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
  shows " $\neg$  rt1  $\approx_{dip}$  rt2"
<proof>

lemma rt_strictly_fresher_trans [elim, trans]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
  and "rt2  $\sqsubseteq_{dip}$  rt3"
  shows "rt1  $\sqsubseteq_{dip}$  rt3"
<proof>

lemma rt_strictly_fresher_irefl [simp]: " $\neg$  (rt  $\sqsubseteq_{dip}$  rt)"
<proof>

lemma rt_fresher_trans_rt_strictly_fresher [elim, trans]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
  and "rt2  $\sqsubseteq_{dip}$  rt3"
  shows "rt1  $\sqsubseteq_{dip}$  rt3"
<proof>

lemma rt_fresher_trans_rt_strictly_fresher' [elim, trans]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
  and "rt2  $\sqsubseteq_{dip}$  rt3"
  shows "rt1  $\sqsubseteq_{dip}$  rt3"
<proof>

lemma rt_fresher_imp_nsqn_le:
  assumes "rt1  $\sqsubseteq_{ip}$  rt2"

```

```

    and "ip ∈ kD rt1"
    and "ip ∈ kD rt2"
    shows "nsqn rt1 ip ≤ nsqn rt2 ip"
  ⟨proof⟩

lemma rt_strictly_fresher_ltI [intro]:
  assumes "dip ∈ kD(rt1)"
    and "dip ∈ kD(rt2)"
    and "nsqn rt1 dip < nsqn rt2 dip"
  shows "rt1 ⊑dip rt2"
  ⟨proof⟩

lemma rt_strictly_fresher_eqI [intro]:
  assumes "i ∈ kD(rt1)"
    and "i ∈ kD(rt2)"
    and "nsqn rt1 i = nsqn rt2 i"
    and "π5(the (rt2 i)) < π5(the (rt1 i))"
  shows "rt1 ⊑i rt2"
  ⟨proof⟩

lemma invalidate_rtsf_left [simp]:
  "∧ dests dip rt rt'. dests dip = None ⇒ (invalidate rt dests ⊑dip rt') = (rt ⊑dip rt')"
  ⟨proof⟩

lemma vD_invalidate_rt_strictly_fresher [simp]:
  assumes "dip ∈ vD(invalidate rt1 dests)"
  shows "(invalidate rt1 dests ⊑dip rt2) = (rt1 ⊑dip rt2)"
  ⟨proof⟩

lemma rt_strictly_fresher_update_other [elim!]:
  "∧ dip ip rt r rt'. [ dip ≠ ip; rt ⊑dip rt' ] ⇒ update rt ip r ⊑dip rt'"
  ⟨proof⟩

lemma addpreRT_strictly_fresher [simp]:
  assumes "dip ∈ kD(rt)"
  shows "(the (addpreRT rt dip npre) ⊑ip rt2) = (rt ⊑ip rt2)"
  ⟨proof⟩

lemma lt_sqn_imp_update_strictly_fresher:
  assumes "dip ∈ vD (rt2 nhip)"
    and *: "osn < sqn (rt2 nhip) dip"
    and **: "rt ≠ update rt dip (osn, kno, val, hops, nhip, {})"
  shows "update rt dip (osn, kno, val, hops, nhip, {}) ⊑dip rt2 nhip"
  ⟨proof⟩

lemma dhops_le_hops_imp_update_strictly_fresher:
  assumes "dip ∈ vD(rt2 nhip)"
    and sqn: "sqn (rt2 nhip) dip = osn"
    and hop: "the (dhops (rt2 nhip) dip) ≤ hops"
    and **: "rt ≠ update rt dip (osn, kno, val, Suc hops, nhip, {})"
  shows "update rt dip (osn, kno, val, Suc hops, nhip, {}) ⊑dip rt2 nhip"
  ⟨proof⟩

lemma nsqn_invalidate:
  assumes "dip ∈ kD(rt)"
    and "∀ ip ∈ dom(dests). ip ∈ vD(rt) ∧ the (dests ip) = inc (sqn rt ip)"
  shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
  ⟨proof⟩

end

```

## 0.7 Invariant proofs on individual processes

theory Seq\_Invariants

**imports** "../AWN/Invariants" Aodv Aodv\_Data Aodv\_Predicates Fresher

**begin**

The proposition numbers are taken from the December 2013 version of the Fehnker et al technical report.

Proposition 7.2

**lemma** *sequence\_number\_increases*:

"paodv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), \_, (\xi', \_)). sn \xi \leq sn \xi'$ )"  
 <proof>

**lemma** *sequence\_number\_one\_or\_bigger*:

"paodv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_). 1 \leq sn \xi$ )"  
 <proof>

We can get rid of the onl/onll if desired...

**lemma** *sequence\_number\_increases'*:

"paodv i  $\models_A$  ( $\lambda((\xi, \_), \_, (\xi', \_)). sn \xi \leq sn \xi'$ )"  
 <proof>

**lemma** *sequence\_number\_one\_or\_bigger'*:

"paodv i  $\models$  ( $\lambda(\xi, \_). 1 \leq sn \xi$ )"  
 <proof>

**lemma** *sip\_in\_kD*:

"paodv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). l \in (\{PAodv-:7\} \cup \{PAodv-:5\} \cup \{PRrep-:0..PRrep-:1\}$   
 $\cup \{PRreq-:0..PRreq-:3\}) \longrightarrow sip \xi \in kD (rt \xi)$ )"  
 <proof>

**lemma** *rrep\_1\_update\_changes*:

"paodv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). (l = PRrep-:1 \longrightarrow$   
 $rt \xi \neq update (rt \xi) (dip \xi) (dsn \xi, kno, val, hops \xi + 1, sip \xi, \{\}))$ )"  
 <proof>

**lemma** *addpreRT\_partly\_welldefined*:

"paodv i  $\models$   
 onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). (l \in \{PRreq-:16..PRreq-:18\} \cup \{PRrep-:2..PRrep-:6\} \longrightarrow dip \xi \in kD (rt \xi)$ )  
 $\wedge (l \in \{PRreq-:3..PRreq-:17\} \longrightarrow oip \xi \in kD (rt \xi))$ )"  
 <proof>

Proposition 7.38

**lemma** *includes\_nhip*:

"paodv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). \forall dip \in kD (rt \xi). the (nhop (rt \xi) dip) \in kD (rt \xi)$ )"  
 <proof>

Proposition 7.22: needed in Proposition 7.4

**lemma** *addpreRT\_welldefined*:

"paodv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). (l \in \{PRreq-:16..PRreq-:18\} \longrightarrow dip \xi \in kD (rt \xi)) \wedge$   
 $(l = PRreq-:17 \longrightarrow oip \xi \in kD (rt \xi)) \wedge$   
 $(l = PRrep-:5 \longrightarrow dip \xi \in kD (rt \xi)) \wedge$   
 $(l = PRrep-:6 \longrightarrow (the (nhop (rt \xi) (dip \xi))) \in kD (rt \xi))$ )"  
 (is " $\_ \models$  onl  $\Gamma_{AODV} ?P$ ")  
 <proof>

Proposition 7.4

**lemma** *known\_destinations\_increase*:

"paodv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), \_, (\xi', \_)). kD (rt \xi) \subseteq kD (rt \xi')$ )"  
 <proof>

Proposition 7.5

**lemma** *rreqs\_increase*:

"paodv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), \_, (\xi', \_)). rreqs \xi \subseteq rreqs \xi'$ )"  
 <proof>

**lemma** *dests\_bigger\_than\_sqn*:

"paodv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). l \in \{PAodv-:15..PAodv-:19\}$   
 $\cup \{PPkt-:7..PPkt-:11\}$   
 $\cup \{PRreq-:9..PRreq-:13\}$   
 $\cup \{PRreq-:21..PRreq-:25\}$   
 $\cup \{PRrep-:10..PRrep-:14\}$   
 $\cup \{PRerr-:1..PRerr-:5\}$   
 $\rightarrow (\forall ip \in \text{dom}(\text{dests } \xi). ip \in kD(\text{rt } \xi) \wedge \text{sqn}(\text{rt } \xi) ip \leq \text{the}(\text{dests } \xi ip))$ )"

*<proof>*

Proposition 7.6

**lemma** *sqns\_increase*:

"paodv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), \_, (\xi', \_)). \forall ip. \text{sqn}(\text{rt } \xi) ip \leq \text{sqn}(\text{rt } \xi') ip$ )"

*<proof>*

Proposition 7.7

**lemma** *ip\_constant*:

"paodv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_). ip \xi = i$ )"

*<proof>*

Proposition 7.8

**lemma** *sender\_ip\_valid'*:

"paodv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), a, \_). \text{anycast}(\lambda m. \text{not\_Pkt } m \rightarrow \text{msg\_sender } m = ip \xi) a$ )"

*<proof>*

**lemma** *sender\_ip\_valid*:

"paodv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), a, \_). \text{anycast}(\lambda m. \text{not\_Pkt } m \rightarrow \text{msg\_sender } m = i) a$ )"

*<proof>*

**lemma** *received\_msg\_inv*:

"paodv i  $\models$  ( $\text{recvm}sg P \rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). l \in \{PAodv-:1\} \rightarrow P(\text{msg } \xi)$ )"

*<proof>*

Proposition 7.9

**lemma** *sip\_not\_ip'*:

"paodv i  $\models$  ( $\text{recvm}sg(\lambda m. \text{not\_Pkt } m \rightarrow \text{msg\_sender } m \neq i) \rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_). \text{sip } \xi \neq ip \xi$ )"

*<proof>*

**lemma** *sip\_not\_ip*:

"paodv i  $\models$  ( $\text{recvm}sg(\lambda m. \text{not\_Pkt } m \rightarrow \text{msg\_sender } m \neq i) \rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_). \text{sip } \xi \neq i$ )"

*<proof>*

Neither *sip\_not\_ip'* nor *sip\_not\_ip* is needed to show loop freedom.

Proposition 7.10

**lemma** *hop\_count\_positive*:

"paodv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_). \forall ip \in kD(\text{rt } \xi). \text{the}(\text{dhops}(\text{rt } \xi) ip) \geq 1$ )"

*<proof>*

**lemma** *rreq\_dip\_in\_vD\_dip\_eq\_ip*:

"paodv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). (l \in \{PRreq-:16..PRreq-:18\} \rightarrow \text{dip } \xi \in vD(\text{rt } \xi))$   
 $\wedge (l \in \{PRreq-:5, PRreq-:6\} \rightarrow \text{dip } \xi = ip \xi)$   
 $\wedge (l \in \{PRreq-:15..PRreq-:18\} \rightarrow \text{dip } \xi \neq ip \xi)$ )"

*<proof>*

Proposition 7.11

**lemma** *anycast\_msg\_zhops*:

" $\bigwedge rreqid \text{ dip dsn dsk oip osn sip.}$   
paodv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda(\_, a, \_). \text{anycast } \text{msg\_zhops } a$ )"

*<proof>*

**lemma** *hop\_count\_zero\_oip\_dip\_sip*:



"paadv i  $\models$  (recvmsg msg\_zhops  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, 1)$ .  
 $(1 \in \{PAadv-:4..PAadv-:5\} \cup \{PRreq-:n/n. True\} \rightarrow$   
 $(hops \xi = 0 \rightarrow oip \xi = sip \xi))$   
 $\wedge$   
 $((1 \in \{PAadv-:6..PAadv-:7\} \cup \{PRrep-:n/n. True\} \rightarrow$   
 $(hops \xi = 0 \rightarrow dip \xi = sip \xi))))$ "

$\langle proof \rangle$

**lemma osn\_rreq:**

"paadv i  $\models$  (recvmsg rreq\_rrep\_sn  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, 1)$ .  
 $1 \in \{PAadv-:4, PAadv-:5\} \cup \{PRreq-:n/n. True\} \rightarrow 1 \leq osn \xi$ )"

$\langle proof \rangle$

**lemma osn\_rreq':**

"paadv i  $\models$  (recvmsg ( $\lambda m. rreq_rrep_sn m \wedge msg_zhops m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, 1)$ .  
 $1 \in \{PAadv-:4, PAadv-:5\} \cup \{PRreq-:n/n. True\} \rightarrow 1 \leq osn \xi$ )"

$\langle proof \rangle$

**lemma dsn\_rrep:**

"paadv i  $\models$  (recvmsg rreq\_rrep\_sn  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, 1)$ .  
 $1 \in \{PAadv-:6, PAadv-:7\} \cup \{PRrep-:n/n. True\} \rightarrow 1 \leq dsn \xi$ )"

$\langle proof \rangle$

**lemma dsn\_rrep':**

"paadv i  $\models$  (recvmsg ( $\lambda m. rreq_rrep_sn m \wedge msg_zhops m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, 1)$ .  
 $1 \in \{PAadv-:6, PAadv-:7\} \cup \{PRrep-:n/n. True\} \rightarrow 1 \leq dsn \xi$ )"

$\langle proof \rangle$

**lemma hop\_count\_zero\_oip\_dip\_sip':**

"paadv i  $\models$  (recvmsg ( $\lambda m. rreq_rrep_sn m \wedge msg_zhops m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, 1)$ .  
 $(1 \in \{PAadv-:4..PAadv-:5\} \cup \{PRreq-:n/n. True\} \rightarrow$   
 $(hops \xi = 0 \rightarrow oip \xi = sip \xi))$   
 $\wedge$   
 $((1 \in \{PAadv-:6..PAadv-:7\} \cup \{PRrep-:n/n. True\} \rightarrow$   
 $(hops \xi = 0 \rightarrow dip \xi = sip \xi))))$ "

$\langle proof \rangle$

Proposition 7.12

**lemma zero\_seq\_unk\_hops\_one':**

"paadv i  $\models$  (recvmsg ( $\lambda m. rreq_rrep_sn m \wedge msg_zhops m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_)$ .  
 $\forall dip \in kD(rt \xi). (sqn (rt \xi) dip = 0 \rightarrow sqnf (rt \xi) dip = unk)$   
 $\wedge (sqnf (rt \xi) dip = unk \rightarrow the (dhops (rt \xi) dip) = 1)$   
 $\wedge (the (dhops (rt \xi) dip) = 1 \rightarrow the (nhop (rt \xi) dip) = dip))$ "

$\langle proof \rangle$

**lemma zero\_seq\_unk\_hops\_one:**

"paadv i  $\models$  (recvmsg ( $\lambda m. rreq_rrep_sn m \wedge msg_zhops m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_)$ .  
 $\forall dip \in kD(rt \xi). (sqn (rt \xi) dip = 0 \rightarrow (sqnf (rt \xi) dip = unk$   
 $\wedge the (dhops (rt \xi) dip) = 1$   
 $\wedge the (nhop (rt \xi) dip) = dip))$ "

$\langle proof \rangle$

**lemma kD\_unk\_or\_atleast\_one:**

"paadv i  $\models$  (recvmsg rreq\_rrep\_sn  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, 1)$ .  
 $\forall dip \in kD(rt \xi). \pi_3(the (rt \xi dip)) = unk \vee 1 \leq \pi_2(the (rt \xi dip))$ )"

$\langle proof \rangle$

Proposition 7.13

**lemma rreq\_rrep\_sn\_any\_step\_invariant:**

"paadv i  $\models_A$  (recvmsg rreq\_rrep\_sn  $\rightarrow$ ) onll  $\Gamma_{AODV}$  ( $\lambda(\_, a, \_). anycast rreq_rrep_sn a$ )"

$\langle proof \rangle$

Proposition 7.14

**lemma rreq\_rrep\_fresh\_any\_step\_invariant:**

```
"paadv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), a, \_)$ . anycast (rreq_rrep_fresh (rt  $\xi$ )) a)"
<proof>
```

Proposition 7.15

**lemma rerr\_invalid\_any\_step\_invariant:**

```
"paadv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), a, \_)$ . anycast (rerr_invalid (rt  $\xi$ )) a)"
<proof>
```

Proposition 7.16

Some well-definedness obligations are irrelevant for the Isabelle development:

1. In each routing table there is at most one entry for each destination: guaranteed by type.
2. In each store of queued data packets there is at most one data queue for each destination: guaranteed by structure.
3. Whenever a set of pairs  $(rip, rsn)$  is assigned to the variable  $dests$  of type  $ip \rightarrow sqn$ , or to the first argument of the function  $rerr$ , this set is a partial function, i.e., there is at most one entry  $(rip, rsn)$  for each destination  $rip$ : guaranteed by type.

**lemma dests\_vD\_inc\_sqn:**

```
"paadv i  $\models$ 
  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l)$ . ( $l \in \{PAadv-:15, PPkt-:7, PRreq-:9, PRreq-:21, PRrep-:10\}$ 
     $\rightarrow (\forall ip \in \text{dom}(dests \ \xi)$ .  $ip \in vD(\text{rt } \xi) \wedge \text{the } (dests \ \xi \ ip) = \text{inc } (sqn \ (\text{rt } \xi) \ ip))$ )
     $\wedge (l = PRerr-:1$ 
     $\rightarrow (\forall ip \in \text{dom}(dests \ \xi)$ .  $ip \in vD(\text{rt } \xi) \wedge \text{the } (dests \ \xi \ ip) > sqn \ (\text{rt } \xi) \ ip))$ )"
<proof>
```

Proposition 7.27

**lemma route\_tables\_fresher:**

```
"paadv i  $\models_A$  (recvmmsg rreq_rrep_sn  $\rightarrow$ ) onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), \_, (\xi', \_))$ .
   $\forall dip \in kD(\text{rt } \xi)$ .  $\text{rt } \xi \sqsubseteq_{dip} \text{rt } \xi'$ )"
<proof>
```

end

## 0.8 The quality increases predicate

**theory Quality\_Increases**

**imports Aadv\_Predicates Fresher**

**begin**

**definition quality\_increases** :: "state  $\Rightarrow$  state  $\Rightarrow$  bool"

```
where "quality_increases  $\xi \ \xi' \equiv (\forall dip \in kD(\text{rt } \xi)$ .  $dip \in kD(\text{rt } \xi')$   $\wedge \text{rt } \xi \sqsubseteq_{dip} \text{rt } \xi'$ )
   $\wedge (\forall dip$ .  $sqn \ (\text{rt } \xi) \ dip \leq sqn \ (\text{rt } \xi') \ dip)$ "
```

**lemma quality\_increasesI [intro!]:**

```
assumes " $\wedge dip$ .  $dip \in kD(\text{rt } \xi) \implies dip \in kD(\text{rt } \xi')$ "
  and " $\wedge dip$ .  $\llbracket dip \in kD(\text{rt } \xi); dip \in kD(\text{rt } \xi') \rrbracket \implies \text{rt } \xi \sqsubseteq_{dip} \text{rt } \xi'$ "
  and " $\wedge dip$ .  $sqn \ (\text{rt } \xi) \ dip \leq sqn \ (\text{rt } \xi') \ dip$ "
shows "quality_increases  $\xi \ \xi'$ "
<proof>
```

**lemma quality\_increasesE [elim]:**

```
fixes dip
assumes "quality_increases  $\xi \ \xi'$ "
  and " $dip \in kD(\text{rt } \xi)$ "
  and " $\llbracket dip \in kD(\text{rt } \xi'); \text{rt } \xi \sqsubseteq_{dip} \text{rt } \xi'; sqn \ (\text{rt } \xi) \ dip \leq sqn \ (\text{rt } \xi') \ dip \rrbracket \implies R \ dip \ \xi \ \xi'$ "
shows " $R \ dip \ \xi \ \xi'$ "
<proof>
```

**lemma quality\_increases\_rt\_fresherD [dest]:**

```

    fixes ip
    assumes "quality_increases  $\xi \xi'$ "
      and "ip $\in$ kD(rt  $\xi$ )"
    shows "rt  $\xi \sqsubseteq_{ip}$  rt  $\xi'$ "
  <proof>

lemma quality_increases_sqnE [elim]:
  fixes dip
  assumes "quality_increases  $\xi \xi'$ "
    and "sqn (rt  $\xi$ ) dip  $\leq$  sqn (rt  $\xi'$ ) dip  $\implies$  R dip  $\xi \xi'$ "
  shows "R dip  $\xi \xi'$ "
  <proof>

lemma quality_increases_refl [intro, simp]: "quality_increases  $\xi \xi$ "
  <proof>

lemma strictly_fresher_quality_increases_right [elim]:
  fixes  $\sigma \sigma'$  dip
  assumes "rt ( $\sigma$  i)  $\sqsubseteq_{dip}$  rt ( $\sigma$  nhip)"
    and qinc: "quality_increases ( $\sigma$  nhip) ( $\sigma'$  nhip)"
    and "dip $\in$ kD(rt ( $\sigma$  nhip))"
  shows "rt ( $\sigma$  i)  $\sqsubseteq_{dip}$  rt ( $\sigma'$  nhip)"
  <proof>

lemma kD_quality_increases [elim]:
  assumes "i $\in$ kD(rt  $\xi$ )"
    and "quality_increases  $\xi \xi'$ "
  shows "i $\in$ kD(rt  $\xi')$ "
  <proof>

lemma kD_nsqn_quality_increases [elim]:
  assumes "i $\in$ kD(rt  $\xi$ )"
    and "quality_increases  $\xi \xi'$ "
  shows "i $\in$ kD(rt  $\xi')$   $\wedge$  nsqn (rt  $\xi$ ) i  $\leq$  nsqn (rt  $\xi')$  i"
  <proof>

lemma nsqn_quality_increases [elim]:
  assumes "i $\in$ kD(rt  $\xi$ )"
    and "quality_increases  $\xi \xi'$ "
  shows "nsqn (rt  $\xi$ ) i  $\leq$  nsqn (rt  $\xi')$  i"
  <proof>

lemma kD_nsqn_quality_increases_trans [elim]:
  assumes "i $\in$ kD(rt  $\xi$ )"
    and "s  $\leq$  nsqn (rt  $\xi$ ) i"
    and "quality_increases  $\xi \xi'$ "
  shows "i $\in$ kD(rt  $\xi')$   $\wedge$  s  $\leq$  nsqn (rt  $\xi')$  i"
  <proof>

lemma nsqn_quality_increases_nsqn_lt_lt [elim]:
  assumes "i $\in$ kD(rt  $\xi$ )"
    and "quality_increases  $\xi \xi'$ "
    and "s < nsqn (rt  $\xi$ ) i"
  shows "s < nsqn (rt  $\xi')$  i"
  <proof>

lemma nsqn_quality_increases_dhops [elim]:
  assumes "i $\in$ kD(rt  $\xi$ )"
    and "quality_increases  $\xi \xi'$ "
    and "nsqn (rt  $\xi$ ) i = nsqn (rt  $\xi')$  i"
  shows "the (dhops (rt  $\xi$ ) i)  $\geq$  the (dhops (rt  $\xi')$  i)"
  <proof>

lemma nsqn_quality_increases_nsqn_eq_le [elim]:

```

```

assumes "i∈kD(rt ξ)"
  and "quality_increases ξ ξ'"
  and "s = nsqn (rt ξ) i"
shows "s < nsqn (rt ξ') i ∨ (s = nsqn (rt ξ') i ∧ the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i))"
⟨proof⟩

```

lemma quality\_increases\_rreq\_rrep\_props [elim]:

```

fixes sn ip hops sip
assumes qinc: "quality_increases (σ sip) (σ' sip)"
  and "1 ≤ sn"
  and *: "ip∈kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
          ∧ (nsqn (rt (σ sip)) ip = sn
             → (the (dhops (rt (σ sip)) ip) ≤ hops
                  ∨ the (flag (rt (σ sip)) ip) = inv))"
shows "ip∈kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
       ∧ (nsqn (rt (σ' sip)) ip = sn
          → (the (dhops (rt (σ' sip)) ip) ≤ hops
              ∨ the (flag (rt (σ' sip)) ip) = inv))"
(is "_ ∧ ?nsqnafter")
⟨proof⟩

```

lemma quality\_increases\_rreq\_rrep\_props':

```

fixes sn ip hops sip
assumes "∀j. quality_increases (σ j) (σ' j)"
  and "1 ≤ sn"
  and *: "ip∈kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
          ∧ (nsqn (rt (σ sip)) ip = sn
             → (the (dhops (rt (σ sip)) ip) ≤ hops
                  ∨ the (flag (rt (σ sip)) ip) = inv))"
shows "ip∈kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
       ∧ (nsqn (rt (σ' sip)) ip = sn
          → (the (dhops (rt (σ' sip)) ip) ≤ hops
              ∨ the (flag (rt (σ' sip)) ip) = inv))"
⟨proof⟩

```

lemma rteq\_quality\_increases:

```

assumes "∀j. j ≠ i → quality_increases (σ j) (σ' j)"
  and "rt (σ' i) = rt (σ i)"
shows "∀j. quality_increases (σ j) (σ' j)"
⟨proof⟩

```

definition msg\_fresh :: "(ip ⇒ state) ⇒ msg ⇒ bool"

```

where "msg_fresh σ m ≡
  case m of Rreq hopsc _ _ _ oipc osnc sipc ⇒ osnc ≥ 1 ∧ (sipc ≠ oipc →
    oipc∈kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) oipc ≥ osnc
    ∧ (nsqn (rt (σ sipc)) oipc = osnc
       → (hopsc ≥ the (dhops (rt (σ sipc)) oipc)
          ∨ the (flag (rt (σ sipc)) oipc) = inv)))
  | Rrep hopsc dipc dsnc _ sipc ⇒ dsnc ≥ 1 ∧ (sipc ≠ dipc →
    dipc∈kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) dipc ≥ dsnc
    ∧ (nsqn (rt (σ sipc)) dipc = dsnc
       → (hopsc ≥ the (dhops (rt (σ sipc)) dipc)
          ∨ the (flag (rt (σ sipc)) dipc) = inv)))
  | Rerr destsc sipc ⇒ (∀ripc∈dom(destsc). (ripc∈kD(rt (σ sipc))
    ∧ the (destsc ripc) - 1 ≤ nsqn (rt (σ sipc)) ripc))
  | _ ⇒ True"

```

lemma msg\_fresh [simp]:

```

"∧hops rreqid dip dsn dsk oip osn sip.
  msg_fresh σ (Rreq hops rreqid dip dsn dsk oip osn sip) =
    (osn ≥ 1 ∧ (sip ≠ oip → oip∈kD(rt (σ sip))
      ∧ nsqn (rt (σ sip)) oip ≥ osn
      ∧ (nsqn (rt (σ sip)) oip = osn
         → (hops ≥ the (dhops (rt (σ sip)) oip)
            ∨ the (flag (rt (σ sip)) oip) = inv)))"

```

```

      ∨ the (flag (rt (σ sip)) oip) = inv))))"
"∧hops dip dsn oip sip. msg_fresh σ (Rrep hops dip dsn oip sip) =
  (dsn ≥ 1 ∧ (sip ≠ dip → dip ∈ kD(rt (σ sip))
    ∧ nsqn (rt (σ sip)) dip ≥ dsn
    ∧ (nsqn (rt (σ sip)) dip = dsn
      → (hops ≥ the (dhops (rt (σ sip)) dip))
        ∨ the (flag (rt (σ sip)) dip) = inv))))"
"∧dests sip.      msg_fresh σ (Rerr dests sip) =
  (∀rip ∈ dom(dests). (rip ∈ kD(rt (σ sip))
    ∧ the (dests rip) - 1 ≤ nsqn (rt (σ sip)) rip))"
"∧d dip.          msg_fresh σ (Newpkt d dip) = True"
"∧d dip sip.     msg_fresh σ (Pkt d dip sip) = True"
⟨proof⟩

```

```

lemma msg_fresh_inc_sn [simp, elim]:
  "msg_fresh σ m ⇒ rreq_rrep_sn m"
⟨proof⟩

```

```

lemma recv_msg_fresh_inc_sn [simp, elim]:
  "orecvmsg (msg_fresh) σ m ⇒ recvmmsg rreq_rrep_sn m"
⟨proof⟩

```

```

lemma rreq_nsqn_is_fresh [simp]:
  fixes σ msg hops rreqid dip dsn dsk oip osn sip
  assumes "rreq_rrep_fresh (rt (σ sip)) (Rreq hops rreqid dip dsn dsk oip osn sip)"
  and "rreq_rrep_sn (Rreq hops rreqid dip dsn dsk oip osn sip)"
  shows "msg_fresh σ (Rreq hops rreqid dip dsn dsk oip osn sip)"
  (is "msg_fresh σ ?msg")
⟨proof⟩

```

```

lemma rrep_nsqn_is_fresh [simp]:
  fixes σ msg hops dip dsn oip sip
  assumes "rreq_rrep_fresh (rt (σ sip)) (Rrep hops dip dsn oip sip)"
  and "rreq_rrep_sn (Rrep hops dip dsn oip sip)"
  shows "msg_fresh σ (Rrep hops dip dsn oip sip)"
  (is "msg_fresh σ ?msg")
⟨proof⟩

```

```

lemma rerr_nsqn_is_fresh [simp]:
  fixes σ msg dests sip
  assumes "rerr_invalid (rt (σ sip)) (Rerr dests sip)"
  shows "msg_fresh σ (Rerr dests sip)"
  (is "msg_fresh σ ?msg")
⟨proof⟩

```

```

lemma quality_increases_msg_fresh [elim]:
  assumes qinc: "∀j. quality_increases (σ j) (σ' j)"
  and "msg_fresh σ m"
  shows "msg_fresh σ' m"
⟨proof⟩

```

end

## 0.9 The ‘open’ AODV model

```

theory OAodv
imports Aodv "../AWN/OAWN_SOS_Labels" "../AWN/OAWN_Convert"
begin

```

Definitions for stating and proving global network properties over individual processes.

```

definition σAODV' :: "(ip ⇒ state) × ((state, msg, pseqp, pseqp label) seqp) set"
where "σAODV' ≡ {(λi. aodv_init i, ΓAODV PAodv)}"

```

```

abbreviation opaodv

```

```

:: "ip  $\Rightarrow$  ((ip  $\Rightarrow$  state)  $\times$  (state, msg, pseq, pseq label) seqp, msg seq_action) automaton"
where
  "opaadv i  $\equiv$  ( $\lfloor$  init =  $\sigma_{AODV}'$ , trans = oseqp_sos  $\Gamma_{AODV}$  i  $\rfloor$ )"

lemma initiali_aadv [intro!, simp]: "initiali i (init (opaadv i)) (init (paadv i))"
  <proof>

lemma oaadv_control_within [simp]: "control_within  $\Gamma_{AODV}$  (init (opaadv i))"
  <proof>

lemma  $\sigma_{AODV}'$ _labels [simp]: " $(\sigma, p) \in \sigma_{AODV}' \implies$  labels  $\Gamma_{AODV}$  p = {PAadv-:0}"
  <proof>

lemma oaadv_init_kD_empty [simp]:
  " $(\sigma, p) \in \sigma_{AODV}' \implies$  kD (rt ( $\sigma$  i)) = {}"
  <proof>

lemma oaadv_init_vD_empty [simp]:
  " $(\sigma, p) \in \sigma_{AODV}' \implies$  vD (rt ( $\sigma$  i)) = {}"
  <proof>

lemma oaadv_trans: "trans (opaadv i) = oseqp_sos  $\Gamma_{AODV}$  i"
  <proof>

declare
  oseq_invariant_ctermsI [OF aadv_wf oaadv_control_within aadv_simple_labels oaadv_trans, cterms_intros]
  oseq_step_invariant_ctermsI [OF aadv_wf oaadv_control_within aadv_simple_labels oaadv_trans, cterms_intros]

end

```

## 0.10 Global invariant proofs over sequential processes

```

theory Global_Invariants
imports Seq_Invariants
       Aadv_Predicates
       Fresher
       Quality_Increases
       "../AWN/DAWN_Convert"
       "OAadv"

begin

lemma other_quality_increases [elim]:
  assumes "other quality_increases I  $\sigma$   $\sigma'$ "
  shows " $\forall j$ . quality_increases ( $\sigma$  j) ( $\sigma'$  j)"
  <proof>

lemma weaken_otherwith [elim]:
  fixes m
  assumes *: "otherwith P I (orecvmsg Q)  $\sigma$   $\sigma'$  a"
  and weakenP: " $\bigwedge \sigma m$ . P  $\sigma$  m  $\implies$  P'  $\sigma$  m"
  and weakenQ: " $\bigwedge \sigma m$ . Q  $\sigma$  m  $\implies$  Q'  $\sigma$  m"
  shows "otherwith P' I (orecvmsg Q')  $\sigma$   $\sigma'$  a"
  <proof>

lemma oreceived_msg_inv:
  assumes other: " $\bigwedge \sigma \sigma' m$ . [ $P$   $\sigma$  m; other Q {i}  $\sigma$   $\sigma'$ ]  $\implies$  P  $\sigma'$  m"
  and local: " $\bigwedge \sigma m$ . P  $\sigma$  m  $\implies$  P ( $\sigma$ (i :=  $\sigma$  i (msg := m))) m"
  shows "opaadv i  $\models$  (otherwith Q {i} (orecvmsg P), other Q {i}  $\rightarrow$ )
        onl  $\Gamma_{AODV}$  ( $\lambda(\sigma, l)$ . l  $\in$  {PAadv-:1}  $\rightarrow$  P  $\sigma$  (msg ( $\sigma$  i)))"
  <proof>

```

(Equivalent to) Proposition 7.27

```

lemma local_quality_increases:
  "paadv i  $\models_A$  (rcvmsg rreq_rrep_sn  $\rightarrow$ ) onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_)$ ,  $\_$ , ( $\xi'$ ,  $\_$ )). quality_increases  $\xi$   $\xi'$ )"

```

*<proof>*

**lemmas** olocal\_quality\_increases =  
open\_seq\_step\_invariant [OF local\_quality\_increases initiali\_aadv oaadv\_trans aadv\_trans,  
simplified seqll\_onll\_swap]

**lemma** oquality\_increases:

"opaadv i  $\models_A$  (otherwith quality\_increases {i} (orecvmsg ( $\lambda\_.$  rreq\_rrep\_sn)),  
other quality\_increases {i}  $\rightarrow$ )  
onll  $\Gamma_{AODV}$  ( $\lambda((\sigma, \_), \_, (\sigma', \_)). \forall j. \text{quality\_increases } (\sigma j) (\sigma' j)$ )"  
(is " $\_ \models_A$  (?S,  $\_ \rightarrow$ )  $\_$ ")  
*<proof>*

**lemma** rreq\_rrep\_nsqn\_fresh\_any\_step\_invariant:

"opaadv i  $\models_A$  (act (recvmsg rreq\_rrep\_sn), other A {i}  $\rightarrow$ )  
onll  $\Gamma_{AODV}$  ( $\lambda((\sigma, \_), a, \_). \text{anycast } (\text{msg\_fresh } \sigma) a$ )"  
*<proof>*

**lemma** oreceived\_rreq\_rrep\_nsqn\_fresh\_inv:

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
other quality\_increases {i}  $\rightarrow$ )  
onll  $\Gamma_{AODV}$  ( $\lambda(\sigma, l). l \in \{\text{PAadv-:1}\} \rightarrow \text{msg\_fresh } \sigma (\text{msg } (\sigma i))$ )"  
*<proof>*

**lemma** oquality\_increases\_nsqn\_fresh:

"opaadv i  $\models_A$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
other quality\_increases {i}  $\rightarrow$ )  
onll  $\Gamma_{AODV}$  ( $\lambda((\sigma, \_), \_, (\sigma', \_)). \forall j. \text{quality\_increases } (\sigma j) (\sigma' j)$ )"  
*<proof>*

**lemma** oosn\_rreq:

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
other quality\_increases {i}  $\rightarrow$ )  
onll  $\Gamma_{AODV}$  (seq1 i ( $\lambda(\xi, l). l \in \{\text{PAadv-:4}, \text{PAadv-:5}\} \cup \{\text{PRreq-:n} \mid n. \text{True}\} \rightarrow 1 \leq \text{osn } \xi$ )"  
*<proof>*

**lemma** rreq\_sip:

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
other quality\_increases {i}  $\rightarrow$ )  
onll  $\Gamma_{AODV}$  ( $\lambda(\sigma, l).$   
( $l \in \{\text{PAadv-:4}, \text{PAadv-:5}, \text{PRreq-:0}, \text{PRreq-:2}\} \wedge \text{sip } (\sigma i) \neq \text{oip } (\sigma i)$   
 $\rightarrow \text{oip } (\sigma i) \in \text{kD}(\text{rt } (\sigma (\text{sip } (\sigma i))))$   
 $\wedge \text{nsqn } (\text{rt } (\sigma (\text{sip } (\sigma i)))) (\text{oip } (\sigma i)) \geq \text{osn } (\sigma i)$   
 $\wedge (\text{nsqn } (\text{rt } (\sigma (\text{sip } (\sigma i)))) (\text{oip } (\sigma i)) = \text{osn } (\sigma i)$   
 $\rightarrow (\text{hops } (\sigma i) \geq \text{the } (\text{dhops } (\text{rt } (\sigma (\text{sip } (\sigma i)))) (\text{oip } (\sigma i)))$   
 $\vee \text{the } (\text{flag } (\text{rt } (\sigma (\text{sip } (\sigma i)))) (\text{oip } (\sigma i))) = \text{inv}))$ )"  
(is " $\_ \models$  (?S, ?U  $\rightarrow$ )  $\_$ ")  
*<proof>*

**lemma** odsn\_rrep:

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
other quality\_increases {i}  $\rightarrow$ )  
onll  $\Gamma_{AODV}$  (seq1 i ( $\lambda(\xi, l). l \in \{\text{PAadv-:6}, \text{PAadv-:7}\} \cup \{\text{PRrep-:n} \mid n. \text{True}\} \rightarrow 1 \leq \text{dsn } \xi$ )"  
*<proof>*

**lemma** rrep\_sip:

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
other quality\_increases {i}  $\rightarrow$ )  
onll  $\Gamma_{AODV}$  ( $\lambda(\sigma, l).$   
( $l \in \{\text{PAadv-:6}, \text{PAadv-:7}, \text{PRrep-:0}, \text{PRrep-:1}\} \wedge \text{sip } (\sigma i) \neq \text{dip } (\sigma i)$   
 $\rightarrow \text{dip } (\sigma i) \in \text{kD}(\text{rt } (\sigma (\text{sip } (\sigma i))))$   
 $\wedge \text{nsqn } (\text{rt } (\sigma (\text{sip } (\sigma i)))) (\text{dip } (\sigma i)) \geq \text{dsn } (\sigma i)$   
 $\wedge (\text{nsqn } (\text{rt } (\sigma (\text{sip } (\sigma i)))) (\text{dip } (\sigma i)) = \text{dsn } (\sigma i)$   
 $\rightarrow (\text{hops } (\sigma i) \geq \text{the } (\text{dhops } (\text{rt } (\sigma (\text{sip } (\sigma i)))) (\text{dip } (\sigma i)))$ )

∨ the (flag (rt (σ (sip (σ i)))) (dip (σ i))) = inv)))"

(is "\_ ⊨ (?S, ?U →) \_")  
 ⟨proof⟩

**lemma rerr\_sip:**

"opaadv i ⊨ (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i} →)  
 onl Γ<sub>AODV</sub> (λ(σ, l).  
 l ∈ {PAadv-:8, PAadv-:9, PRerr-:0, PRerr-:1}  
 → (∀ ripc ∈ dom(dests (σ i)). ripc ∈ kD(rt (σ (sip (σ i)))) ∧  
 the (dests (σ i) ripc) - 1 ≤ nsqn (rt (σ (sip (σ i)))) ripc))"

(is "\_ ⊨ (?S, ?U →) \_")  
 ⟨proof⟩

**lemma prerr\_guard:** "paadv i ⊨

onl Γ<sub>AODV</sub> (λ(ξ, l). (l = PRerr-:1  
 → (∀ ip ∈ dom(dests ξ). ip ∈ vD(rt ξ)  
 ∧ the (nhop (rt ξ) ip) = sip ξ  
 ∧ sqn (rt ξ) ip < the (dests ξ ip))))"

⟨proof⟩

**lemmas oadpreRT\_welldefined =**

open\_seq\_invariant [OF addpreRT\_welldefined initiali\_aadv oaadv\_trans aadv\_trans,  
 simplified seq\_l\_onl\_swap,  
 THEN oinvariant\_anyact]

**lemmas odests\_vD\_inc\_sqn =**

open\_seq\_invariant [OF dests\_vD\_inc\_sqn initiali\_aadv oaadv\_trans aadv\_trans,  
 simplified seq\_l\_onl\_swap,  
 THEN oinvariant\_anyact]

**lemmas oprerr\_guard =**

open\_seq\_invariant [OF prerr\_guard initiali\_aadv oaadv\_trans aadv\_trans,  
 simplified seq\_l\_onl\_swap,  
 THEN oinvariant\_anyact]

Proposition 7.28

**lemma seq\_compare\_next\_hop':**

"opaadv i ⊨ (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i} →) onl Γ<sub>AODV</sub> (λ(σ, \_).  
 ∀ dip. let nhop = the (nhop (rt (σ i)) dip)  
 in dip ∈ kD(rt (σ i)) ∧ nhop ≠ dip →  
 dip ∈ kD(rt (σ nhop)) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhop)) dip)"

(is "\_ ⊨ (?S, ?U →) \_")  
 ⟨proof⟩

Proposition 7.30

**lemmas okD\_unk\_or\_atleast\_one =**

open\_seq\_invariant [OF kD\_unk\_or\_atleast\_one initiali\_aadv,  
 simplified seq\_l\_onl\_swap]

**lemmas ozero\_seq\_unk\_hops\_one =**

open\_seq\_invariant [OF zero\_seq\_unk\_hops\_one initiali\_aadv,  
 simplified seq\_l\_onl\_swap]

**lemma oreachable\_fresh\_okD\_unk\_or\_atleast\_one:**

fixes dip  
 assumes "(σ, p) ∈ oreachable (opaadv i)  
 (otherwith (op=) {i} (orecvmsg (λσ m. msg\_fresh σ m  
 ∧ msg\_zhops m)))  
 (other quality\_increases {i}))"  
 and "dip ∈ kD(rt (σ i))"  
 shows "π<sub>3</sub>(the (rt (σ i) dip)) = unk ∨ 1 ≤ π<sub>2</sub>(the (rt (σ i) dip))"  
 (is "?P dip")



*<proof>*

**lemma** *oreachable\_fresh\_ozero\_seq\_unk\_hops\_one*:

```
  fixes dip
  assumes "(σ, p) ∈ oreachable (opaadv i)
            (otherwith (op=) {i} (orecvmsg (λσ m. msg_fresh σ m
                                             ∧ msg_zhops m)))
            (other quality_increases {i})"
  and "dip ∈ kD(rt (σ i))"
  shows "sqn (rt (σ i)) dip = 0 →
        sqnf (rt (σ i)) dip = unk
        ∧ the (dhops (rt (σ i)) dip) = 1
        ∧ the (nhop (rt (σ i)) dip) = dip"
  (is "?P dip")
<proof>
```

**lemma** *seq\_nhop\_quality\_increases'*:

```
  shows "opaadv i ⊨ (otherwith (op=) {i}
                    (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
                    other quality_increases {i} →)
        onl ΓAODV (λ(σ, _). ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                   in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip))
                                   ∧ nhip ≠ dip
                                   → (rt (σ i)) ⊑dip (rt (σ nhip)))"
  (is "_ ⊨ (?S i, _ →) _")
<proof>
```

**lemma** *seq\_nhop\_quality\_increases*:

```
  shows "opaadv i ⊨ (otherwith (op=) {i}
                    (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
                    other quality_increases {i} →)
        global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                                   in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                   → (rt (σ i)) ⊑dip (rt (σ nhip)))"
  <proof>
```

**end**

## 0.11 Routing graphs and loop freedom

**theory** *Loop\_Freedom*

**imports** *Aadv\_Predicates Fresher*

**begin**

Define the central theorem that relates an invariant over network states to the absence of loops in the associated routing graph.

**definition**

```
  rt_graph :: "(ip ⇒ state) ⇒ ip ⇒ ip rel"
  where
    "rt_graph σ = (λdip.
      {(ip, ip') | ip ip' dsn dsk hops pre.
        ip ≠ dip ∧ rt (σ ip) dip = Some (dsn, dsk, val, hops, ip', pre)})"
```

Given the state of a network  $\sigma$ , a routing graph for a given destination  $dip$  abstracts the details of routing tables into nodes (ip addresses) and vertices (valid routes between ip addresses).

**lemma** *rt\_graphE [elim]*:

```
  fixes n dip ip ip'
  assumes "(ip, ip') ∈ rt_graph σ dip"
  shows "ip ≠ dip ∧ (∃r. rt (σ ip) = r
                    ∧ (∃dsn dsk hops pre. r dip = Some (dsn, dsk, val, hops, ip', pre)))"
  <proof>
```

**lemma** *rt\_graph\_vD [dest]*:

" $\bigwedge ip\ ip'\ \sigma\ dip.\ (ip,\ ip') \in rt\_graph\ \sigma\ dip \implies dip \in vD(rt\ (\sigma\ ip))$ "  
 <proof>

**lemma** *rt\_graph\_vD\_trans* [dest]:

" $\bigwedge ip\ ip'\ \sigma\ dip.\ (ip,\ ip') \in (rt\_graph\ \sigma\ dip)^+ \implies dip \in vD(rt\ (\sigma\ ip))$ "  
 <proof>

**lemma** *rt\_graph\_not\_dip* [dest]:

" $\bigwedge ip\ ip'\ \sigma\ dip.\ (ip,\ ip') \in rt\_graph\ \sigma\ dip \implies ip \neq dip$ "  
 <proof>

**lemma** *rt\_graph\_not\_dip\_trans* [dest]:

" $\bigwedge ip\ ip'\ \sigma\ dip.\ (ip,\ ip') \in (rt\_graph\ \sigma\ dip)^+ \implies ip \neq dip$ "  
 <proof>

NB: the property below cannot be lifted to the transitive closure

**lemma** *rt\_graph\_nhip\_is\_nhop* [dest]:

" $\bigwedge ip\ ip'\ \sigma\ dip.\ (ip,\ ip') \in rt\_graph\ \sigma\ dip \implies ip' = the\ (nhop\ (rt\ (\sigma\ ip))\ dip)$ "  
 <proof>

**theorem** *inv\_to\_loop\_freedom*:

assumes " $\forall i\ dip.\ let\ nhip = the\ (nhop\ (rt\ (\sigma\ i))\ dip)$   
           in  $dip \in vD\ (rt\ (\sigma\ i)) \cap vD\ (rt\ (\sigma\ nhip)) \wedge nhip \neq dip$   
            $\longrightarrow (rt\ (\sigma\ i)) \sqsubset_{dip}\ (rt\ (\sigma\ nhip))$ "  
 shows " $\forall dip.\ irrefl\ ((rt\_graph\ \sigma\ dip)^+)$ "  
 <proof>

end

## 0.12 Lift and transfer invariants to show loop freedom

**theory** *Aodv\_Loop\_Freedom*

imports "*../AWN/OClosed\_Transfer*" "*../AWN/Qmsg\_Lifting*" *Global\_Invariants* *Loop\_Freedom*  
 begin

### 0.12.1 Lift to parallel processes with queues

**lemma** *par\_step\_no\_change\_on\_send\_or\_receive*:

fixes  $\sigma\ s\ a\ \sigma'\ s'$   
 assumes " $((\sigma,\ s),\ a,\ (\sigma',\ s')) \in oparp\_sos\ i\ (oseqp\_sos\ \Gamma_{AODV}\ i)\ (seqp\_sos\ \Gamma_{QMSG})$ "  
 and " $a \neq \tau$ "  
 shows " $\sigma' i = \sigma i$ "  
 <proof>

**lemma** *par\_nhop\_quality\_increases*:

shows " $opaodv\ i\ \langle\langle i\ qmsg \models (otherwith\ (op=)\ \{i\}\ (orecvmsg\ (\lambda\sigma\ m.\ msg\_fresh\ \sigma\ m \wedge msg\_zhops\ m)),$   
            $other\ quality\_increases\ \{i\} \rightarrow)$   
       global  $(\lambda\sigma.\ \forall dip.\ let\ nhip = the\ (nhop\ (rt\ (\sigma\ i))\ dip)$   
           in  $dip \in vD\ (rt\ (\sigma\ i)) \cap vD\ (rt\ (\sigma\ nhip)) \wedge nhip \neq dip$   
            $\longrightarrow (rt\ (\sigma\ i)) \sqsubset_{dip}\ (rt\ (\sigma\ nhip))$ )"

<proof>

**lemma** *par\_rreq\_rrep\_sn\_quality\_increases*:

" $opaodv\ i\ \langle\langle i\ qmsg \models_A\ (\lambda\sigma\ \_.\ orecvmsg\ (\lambda\_.\ rreq\_rrep\_sn)\ \sigma,\ other\ (\lambda\_ \_.\ True)\ \{i\} \rightarrow)$   
            $globala\ (\lambda(\sigma,\ \_,\ \sigma').\ quality\_increases\ (\sigma\ i)\ (\sigma'\ i))$ "

<proof>

**lemma** *par\_rreq\_rrep\_nsqn\_fresh\_any\_step*:

shows " $opaodv\ i\ \langle\langle i\ qmsg \models_A\ (\lambda\sigma\ \_.\ orecvmsg\ (\lambda\_.\ rreq\_rrep\_sn)\ \sigma,$   
            $other\ (\lambda\_ \_.\ True)\ \{i\} \rightarrow)$   
        $globala\ (\lambda(\sigma,\ a,\ \sigma').\ anycast\ (msg\_fresh\ \sigma)\ a)$ "

<proof>

**lemma par\_anycast\_msg\_zhops:**

```
shows "opaadv i <<_i qmsg ⊢A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
      globala (λ(_, a, _). anycast msg_zhops a)"
⟨proof⟩
```

### 0.12.2 Lift to nodes

**lemma node\_step\_no\_change\_on\_send\_or\_receive:**

```
assumes "((σ, NodeS i P R), a, (σ', NodeS i' P' R')) ∈ onode_sos
        (oparp_sos i (oseqp_sos ΓAODV i) (seqp_sos ΓQMSG))"
and "a ≠ τ"
shows "σ' i = σ i"
⟨proof⟩
```

**lemma node\_nhop\_quality\_increases:**

```
shows "< i : opaadv i <<_i qmsg : R >_o ⊢
      (otherwith (op=) {i}
       (oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
      other quality_increases {i}
      →) global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
                          in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                          → (rt (σ i)) ⊆dip (rt (σ nhip)))"
⟨proof⟩
```

**lemma node\_quality\_increases:**

```
< i : opaadv i <<_i qmsg : R >_o ⊢A (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ,
                                     other (λ_ _. True) {i} →)
      globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
⟨proof⟩
```

**lemma node\_rreq\_rrep\_nsqn\_fresh\_any\_step:**

```
shows "< i : opaadv i <<_i qmsg : R >_o ⊢A
      (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
      globala (λ(σ, a, σ'). castmsg (msg_fresh σ) a)"
⟨proof⟩
```

**lemma node\_anycast\_msg\_zhops:**

```
shows "< i : opaadv i <<_i qmsg : R >_o ⊢A
      (λσ _. oarrivemsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
      globala (λ(_, a, _). castmsg msg_zhops a)"
⟨proof⟩
```

**lemma node\_silent\_change\_only:**

```
shows "< i : opaadv i <<_i qmsg : Ri >_o ⊢A (λσ _. oarrivemsg (λ_ _. True) σ,
                                             other (λ_ _. True) {i} →)
      globala (λ(σ, a, σ'). a ≠ τ → σ' i = σ i)"
⟨proof⟩
```

### 0.12.3 Lift to partial networks

**lemma arrive\_rreq\_rrep\_nsqn\_fresh\_inc\_sn [simp]:**

```
assumes "oarrivemsg (λσ m. msg_fresh σ m ∧ P σ m) σ m"
shows "oarrivemsg (λ_. rreq_rrep_sn) σ m"
⟨proof⟩
```

**lemma opnet\_nhop\_quality\_increases:**

```
shows "opnet (λi. opaadv i <<_i qmsg> p ⊢
          (otherwith (op=) (net_tree_ips p)
           (oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
          other quality_increases (net_tree_ips p) →)
      global (λσ. ∀i∈net_tree_ips p. ∀dip.
              let nhip = the (nhop (rt (σ i)) dip)
              in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip)
```

$\rightarrow (rt (\sigma i)) \sqsubseteq_{dip} (rt (\sigma nhip))$ )"

*<proof>*

#### 0.12.4 Lift to closed networks

**lemma** *onet\_nhop\_quality\_increases*:

**shows** "oclosed (opnet ( $\lambda i.$  opaadv i  $\langle\langle i \text{ qmsg} \rangle\rangle$  p)  
 $\models (\lambda \_ \_.$  True, other quality\_increases (net\_tree\_ips p)  $\rightarrow$ )  
 global ( $\lambda \sigma.$   $\forall i \in \text{net\_tree\_ips } p.$   $\forall dip.$   
     let nhip = the (nhop (rt ( $\sigma i$ )) dip)  
     in dip  $\in vD$  (rt ( $\sigma i$ ))  $\cap vD$  (rt ( $\sigma nhip$ ))  $\wedge$  nhip  $\neq$  dip  
      $\rightarrow (rt (\sigma i)) \sqsubseteq_{dip} (rt (\sigma nhip))$ )")  
 (is " $\_ \models (\_, ?U \rightarrow) ?inv$ ")  
*<proof>*

#### 0.12.5 Transfer into the standard model

**interpretation** *aadv\_openproc*: openproc paadv opaadv id

**rewrites** "aadv\_openproc.initmissing = initmissing"

*<proof>*

**interpretation** *aadv\_openproc\_par\_qmsg*: openproc\_parq paadv opaadv id qmsg

**rewrites** "aadv\_openproc\_par\_qmsg.netglobal = netglobal"

**and** "aadv\_openproc\_par\_qmsg.initmissing = initmissing"

*<proof>*

**lemma** *net\_nhop\_quality\_increases*:

**assumes** "wf\_net\_tree n"

**shows** "closed (pnet ( $\lambda i.$  paadv i  $\langle\langle \text{qmsg} \rangle\rangle$  n)  $\models$  netglobal  
 ( $\lambda \sigma.$   $\forall i$  dip. let nhip = the (nhop (rt ( $\sigma i$ )) dip)  
     in dip  $\in vD$  (rt ( $\sigma i$ ))  $\cap vD$  (rt ( $\sigma nhip$ ))  $\wedge$  nhip  $\neq$  dip  
      $\rightarrow (rt (\sigma i)) \sqsubseteq_{dip} (rt (\sigma nhip))$ )")

(is " $\_ \models$  netglobal ( $\lambda \sigma.$   $\forall i.$   $?inv \sigma i$ )")

*<proof>*

#### 0.12.6 Loop freedom of AODV

**theorem** *aadv\_loop\_freedom*:

**assumes** "wf\_net\_tree n"

**shows** "closed (pnet ( $\lambda i.$  paadv i  $\langle\langle \text{qmsg} \rangle\rangle$  n)  $\models$  netglobal ( $\lambda \sigma.$   $\forall dip.$  irrefl ((rt\_graph  $\sigma$  dip)<sup>+</sup>))"

*<proof>*

end

# Chapter 1

## Variant A: Skipping the RREQ ID

Explanation [4, §10.1]: AODV does not need the route request identifier. This number, in combination with the IP address of the originator, is used to identify every RREQ message in a unique way. This variant shows that the combination of the originator's IP address and its sequence number is just as suited to uniquely determine the route request to which the message belongs. Hence, the route request identifier field is not required. This can then reduce the size of the RREQ message.

### 1.1 Predicates and functions used in the AODV model

```
theory A_Aodv_Data
imports A_Norreqid
begin
```

#### 1.1.1 Sequence Numbers

Sequence numbers approximate the relative freshness of routing information.

```
definition inc :: "sqn  $\Rightarrow$  sqn"
  where "inc sn  $\equiv$  if sn = 0 then sn else sn + 1"
```

```
lemma less_than_inc [simp]: "x  $\leq$  inc x"
  <proof>
```

```
lemma inc_minus_suc_0 [simp]:
  "inc x - Suc 0 = x"
  <proof>
```

```
lemma inc_never_one' [simp, intro]: "inc x  $\neq$  Suc 0"
  <proof>
```

```
lemma inc_never_one [simp, intro]: "inc x  $\neq$  1"
  <proof>
```

#### 1.1.2 Modelling Routes

A route is a 6-tuple,  $(dsn, dsk, flag, hops, nhip, pre)$  where  $dsn$  is the 'destination sequence number',  $dsk$  is the 'destination-sequence-number status',  $flag$  is the route status,  $hops$  is the number of hops to the destination,  $nhip$  is the next hop toward the destination, and  $pre$  is the set of 'precursor nodes'—those interested in hearing about changes to the route.

```
type_synonym r = "sqn  $\times$  k  $\times$  f  $\times$  nat  $\times$  ip  $\times$  ip set"
```

```
definition proj2 :: "r  $\Rightarrow$  sqn" ("π2")
  where "π2  $\equiv$   $\lambda$ (dsn, _, _, _, _, _). dsn"
```

```
definition proj3 :: "r  $\Rightarrow$  k" ("π3")
  where "π3  $\equiv$   $\lambda$ (_, dsk, _, _, _, _). dsk"
```

```
definition proj4 :: "r  $\Rightarrow$  f" ("π4")
```

```

where "π4 ≡ λ(, , flag, , , ). flag"

definition proj5 :: "r ⇒ nat" ("π5")
  where "π5 ≡ λ(, , , hops, , ). hops"

definition proj6 :: "r ⇒ ip" ("π6")
  where "π6 ≡ λ(, , , , , nhip, ). nhip"

definition proj7 :: "r ⇒ ip set" ("π7")
  where "π7 ≡ λ(, , , , , , pre). pre"

lemma projs [simp]:
  "π2(dsn, dsk, flag, hops, nhip, pre) = dsn"
  "π3(dsn, dsk, flag, hops, nhip, pre) = dsk"
  "π4(dsn, dsk, flag, hops, nhip, pre) = flag"
  "π5(dsn, dsk, flag, hops, nhip, pre) = hops"
  "π6(dsn, dsk, flag, hops, nhip, pre) = nhip"
  "π7(dsn, dsk, flag, hops, nhip, pre) = pre"
  ⟨proof⟩

lemma proj3_pred [intro]: "[[ P kno; P unk ] ⇒ P (π3 x)]"
  ⟨proof⟩

lemma proj4_pred [intro]: "[[ P val; P inv ] ⇒ P (π4 x)]"
  ⟨proof⟩

lemma proj6_pair_snd [simp]:
  fixes dsn' r
  shows "π6 (dsn', snd (r)) = π6(r)"
  ⟨proof⟩



### 1.1.3 Routing Tables



Routing tables map ip addresses to route entries.

type_synonym rt = "ip → r"

syntax
  "_Sigma_route" :: "rt ⇒ ip → r" ("σroute'(, _)")

translations
  "σroute(rt, dip)" => "rt dip"

definition sqn :: "rt ⇒ ip ⇒ sqn"
  where "sqn rt dip ≡ case σroute(rt, dip) of Some r ⇒ π2(r) | None ⇒ 0"

definition sqnf :: "rt ⇒ ip ⇒ k"
  where "sqnf rt dip ≡ case σroute(rt, dip) of Some r ⇒ π3(r) | None ⇒ unk"

abbreviation flag :: "rt ⇒ ip → f"
  where "flag rt dip ≡ map_option π4 (σroute(rt, dip))"

abbreviation dhops :: "rt ⇒ ip → nat"
  where "dhops rt dip ≡ map_option π5 (σroute(rt, dip))"

abbreviation nhop :: "rt ⇒ ip → ip"
  where "nhop rt dip ≡ map_option π6 (σroute(rt, dip))"

abbreviation precs :: "rt ⇒ ip → ip set"
  where "precs rt dip ≡ map_option π7 (σroute(rt, dip))"

definition vD :: "rt ⇒ ip set"
  where "vD rt ≡ {dip. flag rt dip = Some val}"

definition iD :: "rt ⇒ ip set"

```

```

where "iD rt  $\equiv$  {dip. flag rt dip = Some inv}"

definition kD :: "rt  $\Rightarrow$  ip set"
  where "kD rt  $\equiv$  {dip. rt dip  $\neq$  None}"

lemma kD_is_vD_and_iD: "kD rt = vD rt  $\cup$  iD rt"
  <proof>

lemma vD_iD_gives_kD [simp]:
  " $\bigwedge$ ip rt. ip  $\in$  vD rt  $\implies$  ip  $\in$  kD rt"
  " $\bigwedge$ ip rt. ip  $\in$  iD rt  $\implies$  ip  $\in$  kD rt"
  <proof>

lemma kD_Some [dest]:
  fixes dip rt
  assumes "dip  $\in$  kD rt"
  shows " $\exists$  dsn dsk flag hops nhip pre.
     $\sigma_{route}(rt, dip) = \text{Some} (dsn, dsk, flag, hops, nhip, pre)$ "
  <proof>

lemma kD_None [dest]:
  fixes dip rt
  assumes "dip  $\notin$  kD rt"
  shows " $\sigma_{route}(rt, dip) = \text{None}$ "
  <proof>

lemma vD_Some [dest]:
  fixes dip rt
  assumes "dip  $\in$  vD rt"
  shows " $\exists$  dsn dsk hops nhip pre.
     $\sigma_{route}(rt, dip) = \text{Some} (dsn, dsk, val, hops, nhip, pre)$ "
  <proof>

lemma vD_empty [simp]: "vD Map.empty = {}"
  <proof>

lemma iD_Some [dest]:
  fixes dip rt
  assumes "dip  $\in$  iD rt"
  shows " $\exists$  dsn dsk hops nhip pre.
     $\sigma_{route}(rt, dip) = \text{Some} (dsn, dsk, inv, hops, nhip, pre)$ "
  <proof>

lemma val_is_vD [elim]:
  fixes ip rt
  assumes "ip  $\in$  kD(rt)"
  and "the (flag rt ip) = val"
  shows "ip  $\in$  vD(rt)"
  <proof>

lemma inv_is_iD [elim]:
  fixes ip rt
  assumes "ip  $\in$  kD(rt)"
  and "the (flag rt ip) = inv"
  shows "ip  $\in$  iD(rt)"
  <proof>

lemma iD_flag_is_inv [elim, simp]:
  fixes ip rt
  assumes "ip  $\in$  iD(rt)"
  shows "the (flag rt ip) = inv"
  <proof>

lemma kD_but_not_vD_is_iD [elim]:

```

```

    fixes ip rt
    assumes "ip ∈ kD(rt)"
      and "ip ∉ vD(rt)"
    shows "ip ∈ iD(rt)"
  ⟨proof⟩

lemma vD_or_iD [elim]:
  fixes ip rt
  assumes "ip ∈ kD(rt)"
    and "ip ∈ vD(rt) ⇒ P rt ip"
    and "ip ∈ iD(rt) ⇒ P rt ip"
  shows "P rt ip"
  ⟨proof⟩

lemma proj5_eq_dhops: "∧ dip rt. dip ∈ kD(rt) ⇒ π5(the (rt dip)) = the (dhops rt dip)"
  ⟨proof⟩

lemma proj4_eq_flag: "∧ dip rt. dip ∈ kD(rt) ⇒ π4(the (rt dip)) = the (flag rt dip)"
  ⟨proof⟩

lemma proj2_eq_sqn: "∧ dip rt. dip ∈ kD(rt) ⇒ π2(the (rt dip)) = sqn rt dip"
  ⟨proof⟩

lemma kD_sqnf_is_proj3 [simp]:
  "∧ ip rt. ip ∈ kD(rt) ⇒ sqnf rt ip = π3(the (rt ip))"
  ⟨proof⟩

lemma vD_flag_val [simp]:
  "∧ dip rt. dip ∈ vD (rt) ⇒ the (flag rt dip) = val"
  ⟨proof⟩

lemma kD_update [simp]:
  "∧ rt nip v. kD (rt(nip ↦ v)) = insert nip (kD rt)"
  ⟨proof⟩

lemma kD_empty [simp]: "kD Map.empty = {}"
  ⟨proof⟩

lemma ip_equal_or_known [elim]:
  fixes rt ip ip'
  assumes "ip = ip' ∨ ip ∈ kD(rt)"
    and "ip = ip' ⇒ P rt ip ip'"
    and "⌈ ip ≠ ip'; ip ∈ kD(rt) ⌋ ⇒ P rt ip ip'"
  shows "P rt ip ip'"
  ⟨proof⟩

```

### 1.1.4 Updating Routing Tables

Routing table entries are modified through explicit functions. The properties of these functions are important in invariant proofs.

#### Updating Precursor Lists

```

definition addpre :: "r ⇒ ip set ⇒ r"
  where "addpre r npre ≡ let (dsn, dsk, flag, hops, nhip, pre) = r in
    (dsn, dsk, flag, hops, nhip, pre ∪ npre)"

```

```

lemma proj2_addpre:
  fixes v pre
  shows "π2(addpre v pre) = π2(v)"
  ⟨proof⟩

```

```

lemma proj3_addpre:
  fixes v pre

```



```

shows "π3(addpre v pre) = π3(v)"
⟨proof⟩

lemma proj4_addpre:
  fixes v pre
  shows "π4(addpre v pre) = π4(v)"
  ⟨proof⟩

lemma proj5_addpre:
  fixes v pre
  shows "π5(addpre v pre) = π5(v)"
  ⟨proof⟩

lemma proj6_addpre:
  fixes dsn dsk flag hops nhip pre npre
  shows "π6(addpre v npre) = π6(v)"
  ⟨proof⟩

lemma proj7_addpre:
  fixes dsn dsk flag hops nhip pre npre
  shows "π7(addpre v npre) = π7(v) ∪ npre"
  ⟨proof⟩

lemma addpre_empty: "addpre r {} = r"
  ⟨proof⟩

lemma addpre_r:
  "addpre (dsn, dsk, fl, hops, nhip, pre) npre = (dsn, dsk, fl, hops, nhip, pre ∪ npre)"
  ⟨proof⟩

lemmas addpre_simps [simp] = proj2_addpre proj3_addpre proj4_addpre proj5_addpre
  proj6_addpre proj7_addpre addpre_empty addpre_r

definition addpreRT :: "rt ⇒ ip ⇒ ip set → rt"
  where "addpreRT rt dip npre ≡
    map_option (λs. rt (dip ↦ addpre s npre)) (σroute(rt, dip))"

lemma snd_addpre [simp]:
  "∧ dsn dsn' v pre. (dsn, snd(addpre (dsn', v) pre)) = addpre (dsn, v) pre"
  ⟨proof⟩

lemma proj2_addpreRT [simp]:
  fixes ip rt ip' npre
  assumes "ip ∈ kD rt"
  and "ip' ∈ kD rt"
  shows "π2(the (the (addpreRT rt ip' npre) ip)) = π2(the (rt ip))"
  ⟨proof⟩

lemma proj3_addpreRT [simp]:
  fixes ip rt ip' npre
  assumes "ip ∈ kD rt"
  and "ip' ∈ kD rt"
  shows "π3(the (the (addpreRT rt ip' npre) ip)) = π3(the (rt ip))"
  ⟨proof⟩

lemma proj5_addpreRT [simp]:
  "∧ rt dip ip npre. dip ∈ kD(rt) ⇒ π5(the (the (addpreRT rt dip npre) ip)) = π5(the (rt ip))"
  ⟨proof⟩

lemma flag_addpreRT [simp]:
  fixes rt pre ip dip
  assumes "dip ∈ kD rt"
  shows "flag (the (addpreRT rt dip pre)) ip = flag rt ip"
  ⟨proof⟩

```

```

lemma kD_addpreRT [simp]:
  fixes rt dip npre
  assumes "dip ∈ kD rt"
  shows "kD (the (addpreRT rt dip npre)) = kD rt"
  ⟨proof⟩

```

```

lemma vD_addpreRT [simp]:
  fixes rt dip npre
  assumes "dip ∈ kD rt"
  shows "vD (the (addpreRT rt dip npre)) = vD rt"
  ⟨proof⟩

```

```

lemma iD_addpreRT [simp]:
  fixes rt dip npre
  assumes "dip ∈ kD rt"
  shows "iD (the (addpreRT rt dip npre)) = iD rt"
  ⟨proof⟩

```

```

lemma nhop_addpreRT [simp]:
  fixes rt pre ip dip
  assumes "dip ∈ kD rt"
  shows "nhop (the (addpreRT rt dip pre)) ip = nhop rt ip"
  ⟨proof⟩

```

```

lemma sqn_addpreRT [simp]:
  fixes rt pre ip dip
  assumes "dip ∈ kD rt"
  shows "sqn (the (addpreRT rt dip pre)) ip = sqn rt ip"
  ⟨proof⟩

```

```

lemma dhops_addpreRT [simp]:
  fixes rt pre ip dip
  assumes "dip ∈ kD rt"
  shows "dhops (the (addpreRT rt dip pre)) ip = dhops rt ip"
  ⟨proof⟩

```

```

lemma sqnf_addpreRT [simp]:
  "∧ ip dip. ip ∈ kD(rt ξ) ⇒ sqnf (the (addpreRT (rt ξ) ip npre)) dip = sqnf (rt ξ) dip"
  ⟨proof⟩

```

## Updating route entries

```

lemma in_kD_case [simp]:
  fixes dip rt
  assumes "dip ∈ kD(rt)"
  shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = es (the (rt dip))"
  ⟨proof⟩

```

```

lemma not_in_kD_case [simp]:
  fixes dip rt
  assumes "dip ∉ kD(rt)"
  shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = en"
  ⟨proof⟩

```

```

lemma rt_Some_sqn [dest]:
  fixes rt and ip dsn dsk flag hops nhip pre
  assumes "rt ip = Some (dsn, dsk, flag, hops, nhip, pre)"
  shows "sqn rt ip = dsn"
  ⟨proof⟩

```

```

lemma not_kD_sqn [simp]:
  fixes dip rt
  assumes "dip ∉ kD(rt)"

```

```

  shows "sqn rt dip = 0"
  ⟨proof⟩

definition update_arg_wf :: "r ⇒ bool"
where "update_arg_wf r ≡ π4(r) = val ∧
      (π2(r) = 0) = (π3(r) = unk) ∧
      (π3(r) = unk → π5(r) = 1)"

lemma update_arg_wf_gives_cases:
  "∧r. update_arg_wf r ⇒ (π2(r) = 0) = (π3(r) = unk)"
  ⟨proof⟩

lemma update_arg_wf_tuples [simp]:
  "∧nhip pre. update_arg_wf (0, unk, val, Suc 0, nhip, pre)"
  "∧n hops nhip pre. update_arg_wf (Suc n, kno, val, hops, nhip, pre)"
  ⟨proof⟩

lemma update_arg_wf_tuples' [elim]:
  "∧n hops nhip pre. Suc 0 ≤ n ⇒ update_arg_wf (n, kno, val, hops, nhip, pre)"
  ⟨proof⟩

lemma wf_r_cases [intro]:
  fixes P r
  assumes "update_arg_wf r"
  and c1: "∧nhip pre. P (0, unk, val, Suc 0, nhip, pre)"
  and c2: "∧dsn hops nhip pre. dsn > 0 ⇒ P (dsn, kno, val, hops, nhip, pre)"
  shows "P r"
  ⟨proof⟩

definition update :: "rt ⇒ ip ⇒ r ⇒ rt"
where
  "update rt ip r ≡
  case σroute(rt, ip) of
  None ⇒ rt (ip ↦ r)
  | Some s ⇒
    if π2(s) < π2(r) then rt (ip ↦ addpre r (π7(s)))
    else if π2(s) = π2(r) ∧ (π5(s) > π5(r) ∨ π4(s) = inv)
      then rt (ip ↦ addpre r (π7(s)))
    else if π3(r) = unk
      then rt (ip ↦ (π2(s), snd (addpre r (π7(s))))
    else rt (ip ↦ addpre s (π7(r)))"

lemma update_simps [simp]:
  fixes r s nrt nr nr' ns rt ip
  defines "s ≡ the σroute(rt, ip)"
  and "nr ≡ addpre r (π7(s))"
  and "nr' ≡ (π2(s), π3(nr), π4(nr), π5(nr), π6(nr), π7(nr))"
  and "ns ≡ addpre s (π7(r))"
  shows
  "[ip ∉ kD(rt)] ⇒ update rt ip r = rt (ip ↦ r)"
  "[ip ∈ kD(rt); sqn rt ip < π2(r)] ⇒ update rt ip r = rt (ip ↦ nr)"
  "[ip ∈ kD(rt); sqn rt ip = π2(r);
  the (dhops rt ip) > π5(r)] ⇒ update rt ip r = rt (ip ↦ nr)"
  "[ip ∈ kD(rt); sqn rt ip = π2(r);
  flag rt ip = Some inv] ⇒ update rt ip r = rt (ip ↦ nr)"
  "[ip ∈ kD(rt); π3(r) = unk; (π2(r) = 0) = (π3(r) = unk)] ⇒ update rt ip r = rt (ip ↦ nr)"
  "[ip ∈ kD(rt); sqn rt ip ≥ π2(r); π3(r) = kno;
  sqn rt ip = π2(r) ⇒ the (dhops rt ip) ≤ π5(r) ∧ the (flag rt ip) = val ]
  ⇒ update rt ip r = rt (ip ↦ ns)"
  ⟨proof⟩

lemma update_cases [elim]:
  assumes "(π2(r) = 0) = (π3(r) = unk)"
  and c1: "[ip ∉ kD(rt)] ⇒ P (rt (ip ↦ r))"

```

```

and c2: "[[ip ∈ kD(rt); sqn rt ip < π2(r)]]
      ⇒ P (rt (ip ↦ addpre r (π7(the σroute(rt, ip)))))"
and c3: "[[ip ∈ kD(rt); sqn rt ip = π2(r); the (dhops rt ip) > π5(r)]]
      ⇒ P (rt (ip ↦ addpre r (π7(the σroute(rt, ip)))))"
and c4: "[[ip ∈ kD(rt); sqn rt ip = π2(r); the (flag rt ip) = inv]]
      ⇒ P (rt (ip ↦ addpre r (π7(the σroute(rt, ip)))))"
and c5: "[[ip ∈ kD(rt); π3(r) = unk]]
      ⇒ P (rt (ip ↦ (π2(the σroute(rt, ip)), π3(r),
                    π4(r), π5(r), π6(r), π7(addpre r (π7(the σroute(rt, ip)))))))"
and c6: "[[ip ∈ kD(rt); sqn rt ip ≥ π2(r); π3(r) = kno;
      sqn rt ip = π2(r) ⇒ the (dhops rt ip) ≤ π5(r) ∧ the (flag rt ip) = val]]
      ⇒ P (rt (ip ↦ addpre (the σroute(rt, ip)) (π7(r))))"
shows "(P (update rt ip r))"
⟨proof⟩

```

lemma update\_cases\_kD:

```

assumes "(π2(r) = 0) = (π3(r) = unk)"
and "ip ∈ kD(rt)"
and c2: "[[sqn rt ip < π2(r) ⇒ P (rt (ip ↦ addpre r (π7(the σroute(rt, ip))))]"
and c3: "[[sqn rt ip = π2(r); the (dhops rt ip) > π5(r)]]
      ⇒ P (rt (ip ↦ addpre r (π7(the σroute(rt, ip)))))"
and c4: "[[sqn rt ip = π2(r); the (flag rt ip) = inv]]
      ⇒ P (rt (ip ↦ addpre r (π7(the σroute(rt, ip)))))"
and c5: "[[π3(r) = unk ⇒ P (rt (ip ↦ (π2(the σroute(rt, ip)), π3(r),
                    π4(r), π5(r), π6(r),
                    π7(addpre r (π7(the σroute(rt, ip))))))]"
and c6: "[[sqn rt ip ≥ π2(r); π3(r) = kno;
      sqn rt ip = π2(r) ⇒ the (dhops rt ip) ≤ π5(r) ∧ the (flag rt ip) = val]]
      ⇒ P (rt (ip ↦ addpre (the σroute(rt, ip)) (π7(r))))"
shows "(P (update rt ip r))"
⟨proof⟩

```

lemma in\_kD\_after\_update [simp]:

```

fixes rt nip dsn dsk flag hops nhip pre
shows "kD (update rt nip (dsn, dsk, flag, hops, nhip, pre)) = insert nip (kD rt)"
⟨proof⟩

```

lemma nhop\_of\_update [simp]:

```

fixes rt dip dsn dsk flag hops nhip
assumes "rt ≠ update rt dip (dsn, dsk, flag, hops, nhip, {})"
shows "the (nhop (update rt dip (dsn, dsk, flag, hops, nhip, {})) dip) = nhip"
⟨proof⟩

```

lemma sqn\_if\_updated:

```

fixes rip v rt ip
shows "sqn (λx. if x = rip then Some v else rt x) ip
      = (if ip = rip then π2(v) else sqn rt ip)"
⟨proof⟩

```

lemma update\_sqn [simp]:

```

fixes rt dip rip dsn dsk hops nhip pre
assumes "(dsn = 0) = (dsk = unk)"
shows "sqn rt dip ≤ sqn (update rt rip (dsn, dsk, val, hops, nhip, pre)) dip"
⟨proof⟩

```

lemma sqn\_update\_bigger [simp]:

```

fixes rt ip ip' dsn dsk flag hops nhip pre
assumes "1 ≤ hops"
shows "sqn rt ip ≤ sqn (update rt ip' (dsn, dsk, flag, hops, nhip, pre)) ip"
⟨proof⟩

```

lemma dhops\_update [intro]:

```

fixes rt dsn dsk flag hops ip rip nhip pre

```

```

assumes ex: "∀ip∈kD rt. the (dhops rt ip) ≥ 1"
  and ip: "(ip = rip ∧ Suc 0 ≤ hops) ∨ (ip ≠ rip ∧ ip∈kD rt)"
  shows "Suc 0 ≤ the (dhops (update rt rip (dsn, dsk, flag, hops, nhip, pre)) ip)"
⟨proof⟩

lemma update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
  shows "(update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = rt ip"
⟨proof⟩

lemma nhop_update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
  shows "nhop (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = nhop rt ip"
⟨proof⟩

lemma dhops_update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
  shows "dhops (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = dhops rt ip"
⟨proof⟩

lemma sqn_update_same [simp]:
  "∧rt ip dsn dsk flag hops nhip pre. sqn (rt(ip ↦ v)) ip = π2(v)"
⟨proof⟩

lemma dhops_update_changed [simp]:
  fixes rt dip osn hops nhip
  assumes "rt ≠ update rt dip (osn, kno, val, hops, nhip, {})"
  shows "the (dhops (update rt dip (osn, kno, val, hops, nhip, {})) dip) = hops"
⟨proof⟩

lemma nhop_update_unk_val [simp]:
  "∧rt dip ip dsn hops npre.
  the (nhop (update rt dip (dsn, unk, val, hops, ip, npre)) dip) = ip"
⟨proof⟩

lemma nhop_update_changed [simp]:
  fixes rt dip dsn dsk flg hops sip
  assumes "update rt dip (dsn, dsk, flg, hops, sip, {}) ≠ rt"
  shows "the (nhop (update rt dip (dsn, dsk, flg, hops, sip, {})) dip) = sip"
⟨proof⟩

lemma update_rt_split_asm:
  "∧rt ip dsn dsk flag hops sip.
  P (update rt ip (dsn, dsk, flag, hops, sip, {}))
  =
  (¬(rt = update rt ip (dsn, dsk, flag, hops, sip, {})) ∧ ¬P rt
  ∨ rt ≠ update rt ip (dsn, dsk, flag, hops, sip, {})
  ∧ ¬P (update rt ip (dsn, dsk, flag, hops, sip, {})))"
⟨proof⟩

lemma sqn_update [simp]: "∧rt dip dsn flg hops sip.
rt ≠ update rt dip (dsn, kno, flg, hops, sip, {})
⇒ sqn (update rt dip (dsn, kno, flg, hops, sip, {})) dip = dsn"
⟨proof⟩

lemma sqnf_update [simp]: "∧rt dip dsn dsk flg hops sip.
rt ≠ update rt dip (dsn, dsk, flg, hops, sip, {})
⇒ sqnf (update rt dip (dsn, dsk, flg, hops, sip, {})) dip = dsk"
⟨proof⟩

lemma update_kno_dsn_greater_zero:

```

" $\wedge$ rt dip ip dsn hops npre.  $1 \leq dsn \implies 1 \leq (\text{sqn } (\text{update } \text{rt } \text{dip } (\text{dsn}, \text{kno}, \text{val}, \text{hops}, \text{ip}, \text{npre})) \text{dip})$ "  
 <proof>

lemma proj3\_update [simp]: " $\wedge$ rt dip dsn dsk flg hops sip.  
 rt  $\neq$  update rt dip (dsn, dsk, flg, hops, sip, { })  
 $\implies \pi_3(\text{the } (\text{update } \text{rt } \text{dip } (\text{dsn}, \text{dsk}, \text{flg}, \text{hops}, \text{sip}, \{ }) \text{dip})) = \text{dsk}$ "  
 <proof>

lemma nhop\_update\_changed\_kno\_val [simp]: " $\wedge$ rt ip dsn dsk hops nhip.  
 rt  $\neq$  update rt ip (dsn, kno, val, hops, nhip, { })  
 $\implies \text{the } (\text{nhop } (\text{update } \text{rt } \text{ip } (\text{dsn}, \text{kno}, \text{val}, \text{hops}, \text{nhip}, \{ }) \text{ip})) = \text{nhip}$ "  
 <proof>

lemma flag\_update [simp]: " $\wedge$ rt dip dsn flg hops sip.  
 rt  $\neq$  update rt dip (dsn, kno, flg, hops, sip, { })  
 $\implies \text{the } (\text{flag } (\text{update } \text{rt } \text{dip } (\text{dsn}, \text{kno}, \text{flg}, \text{hops}, \text{sip}, \{ }) \text{dip})) = \text{flg}$ "  
 <proof>

lemma the\_flag\_Some [dest!]:  
 fixes ip rt  
 assumes "the (flag rt ip) = x"  
 and "ip  $\in$  kD rt"  
 shows "flag rt ip = Some x"  
 <proof>

lemma kD\_update\_unchanged [dest]:  
 fixes rt dip dsn dsk flag hops nhip pre  
 assumes "rt = update rt dip (dsn, dsk, flag, hops, nhip, pre)"  
 shows "dip  $\in$  kD(rt)"  
 <proof>

lemma nhop\_update [simp]: " $\wedge$ rt dip dsn dsk flg hops sip.  
 rt  $\neq$  update rt dip (dsn, dsk, flg, hops, sip, { })  
 $\implies \text{the } (\text{nhop } (\text{update } \text{rt } \text{dip } (\text{dsn}, \text{dsk}, \text{flg}, \text{hops}, \text{sip}, \{ }) \text{dip})) = \text{sip}$ "  
 <proof>

lemma sqn\_update\_another [simp]:  
 fixes dip ip rt dsn dsk flag hops nhip pre  
 assumes "ip  $\neq$  dip"  
 shows "sqn (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = sqn rt ip"  
 <proof>

lemma sqnf\_update\_another [simp]:  
 fixes dip ip rt dsn dsk flag hops nhip pre  
 assumes "ip  $\neq$  dip"  
 shows "sqnf (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = sqnf rt ip"  
 <proof>

lemma vD\_update\_val [dest]:  
 " $\wedge$ dip rt dip' dsn dsk hops nhip pre.  
 dip  $\in$  vD(update rt dip' (dsn, dsk, val, hops, nhip, pre))  $\implies (\text{dip} \in \text{vD}(\text{rt}) \vee \text{dip} = \text{dip}'$ "  
 <proof>

## Invalidating route entries

definition invalidate :: "rt  $\Rightarrow$  (ip  $\rightarrow$  sqn)  $\Rightarrow$  rt"  
 where "invalidate rt dests  $\equiv$   
 $\lambda$ ip. case (rt ip, dests ip) of  
 (None, \_)  $\Rightarrow$  None  
 | (Some s, None)  $\Rightarrow$  Some s  
 | (Some (\_, dsk, \_, hops, nhip, pre), Some rsn)  $\Rightarrow$   
 Some (rsn, dsk, inv, hops, nhip, pre)"

lemma proj3\_invalidate [simp]:

" $\bigwedge \text{dip. } \pi_3(\text{the } ((\text{invalidate } \text{rt } \text{dests}) \text{ dip})) = \pi_3(\text{the } (\text{rt } \text{dip}))$ "  
<proof>

lemma proj5\_invalidate [simp]:  
" $\bigwedge \text{dip. } \pi_5(\text{the } ((\text{invalidate } \text{rt } \text{dests}) \text{ dip})) = \pi_5(\text{the } (\text{rt } \text{dip}))$ "  
<proof>

lemma proj6\_invalidate [simp]:  
" $\bigwedge \text{dip. } \pi_6(\text{the } ((\text{invalidate } \text{rt } \text{dests}) \text{ dip})) = \pi_6(\text{the } (\text{rt } \text{dip}))$ "  
<proof>

lemma proj7\_invalidate [simp]:  
" $\bigwedge \text{dip. } \pi_7(\text{the } ((\text{invalidate } \text{rt } \text{dests}) \text{ dip})) = \pi_7(\text{the } (\text{rt } \text{dip}))$ "  
<proof>

### 1.1.5 Route Requests

lemma invalidate\_kD\_inv [simp]:  
" $\bigwedge \text{rt } \text{dests. } \text{kD } (\text{invalidate } \text{rt } \text{dests}) = \text{kD } \text{rt}$ "  
<proof>

lemma invalidate\_sqn:  
fixes rt dip dests  
assumes " $\forall \text{rsn. } \text{dests } \text{dip} = \text{Some } \text{rsn} \longrightarrow \text{sqn } \text{rt } \text{dip} \leq \text{rsn}$ "  
shows " $\text{sqn } \text{rt } \text{dip} \leq \text{sqn } (\text{invalidate } \text{rt } \text{dests}) \text{ dip}$ "  
<proof>

lemma sqn\_invalidate\_in\_dests [simp]:  
fixes dests ipa rsn rt  
assumes " $\text{dests } \text{ipa} = \text{Some } \text{rsn}$ "  
and " $\text{ipa} \in \text{kD}(\text{rt})$ "  
shows " $\text{sqn } (\text{invalidate } \text{rt } \text{dests}) \text{ ipa} = \text{rsn}$ "  
<proof>

lemma dhops\_invalidate [simp]:  
" $\bigwedge \text{dip. } \text{the } (\text{dhops } (\text{invalidate } \text{rt } \text{dests}) \text{ dip}) = \text{the } (\text{dhops } \text{rt } \text{dip})$ "  
<proof>

lemma sqnf\_invalidate [simp]:  
" $\bigwedge \text{dip. } \text{sqnf } (\text{invalidate } (\text{rt } \xi) (\text{dests } \xi)) \text{ dip} = \text{sqnf } (\text{rt } \xi) \text{ dip}$ "  
<proof>

lemma nhop\_invalidate [simp]:  
" $\bigwedge \text{dip. } \text{the } (\text{nhop } (\text{invalidate } (\text{rt } \xi) (\text{dests } \xi)) \text{ dip}) = \text{the } (\text{nhop } (\text{rt } \xi) \text{ dip})$ "  
<proof>

lemma invalidate\_other [simp]:  
fixes rt dests dip  
assumes " $\text{dip} \notin \text{dom}(\text{dests})$ "  
shows " $\text{invalidate } \text{rt } \text{dests } \text{dip} = \text{rt } \text{dip}$ "  
<proof>

lemma invalidate\_none [simp]:  
fixes rt dests dip  
assumes " $\text{dip} \notin \text{kD}(\text{rt})$ "  
shows " $\text{invalidate } \text{rt } \text{dests } \text{dip} = \text{None}$ "  
<proof>

lemma vD\_invalidate\_vD\_not\_dests:  
" $\bigwedge \text{dip } \text{rt } \text{dests. } \text{dip} \in \text{vD}(\text{invalidate } \text{rt } \text{dests}) \implies \text{dip} \in \text{vD}(\text{rt}) \wedge \text{dests } \text{dip} = \text{None}$ "  
<proof>

lemma sqn\_invalidate\_not\_in\_dests [simp]:  
fixes dests dip rt

```

assumes "dip $\notin$ dom(dests)"
shows "sqn (invalidate rt dests) dip = sqn rt dip"
<proof>

```

lemma invalidate\_changes:

```

fixes rt dests dip dsn dsk flag hops nhip pre
assumes "invalidate rt dests dip = Some (dsn, dsk, flag, hops, nhip, pre)"
shows " dsn = (case dests dip of None  $\Rightarrow$   $\pi_2$ (the (rt dip)) | Some rsn  $\Rightarrow$  rsn)
 $\wedge$  dsk =  $\pi_3$ (the (rt dip))
 $\wedge$  flag = (if dests dip = None then  $\pi_4$ (the (rt dip)) else inv)
 $\wedge$  hops =  $\pi_5$ (the (rt dip))
 $\wedge$  nhip =  $\pi_6$ (the (rt dip))
 $\wedge$  pre =  $\pi_7$ (the (rt dip))"
<proof>

```

```

lemma proj3_inv: " $\wedge$ dip rt dests. dip $\in$ kD (rt)
 $\Rightarrow$   $\pi_3$ (the (invalidate rt dests dip)) =  $\pi_3$ (the (rt dip))"
<proof>

```

lemma dests\_iD\_invalidate [simp]:

```

assumes "dests ip = Some rsn"
and "ip $\in$ kD(rt)"
shows "ip $\in$ iD(invalidate rt dests)"
<proof>

```

### 1.1.6 Queued Packets

Functions for sending data packets.

```

type_synonym store = "ip  $\rightarrow$  (p  $\times$  data list)"

```

```

definition sigma_queue :: "store  $\Rightarrow$  ip  $\Rightarrow$  data list" (" $\sigma_{\text{queue}}'(\_, \_)$ ")
where " $\sigma_{\text{queue}}$ (store, dip)  $\equiv$  case store dip of None  $\Rightarrow$  [] | Some (p, q)  $\Rightarrow$  q"

```

```

definition qD :: "store  $\Rightarrow$  ip set"
where "qD  $\equiv$  dom"

```

```

definition add :: "data  $\Rightarrow$  ip  $\Rightarrow$  store  $\Rightarrow$  store"
where "add d dip store  $\equiv$  case store dip of
None  $\Rightarrow$  store (dip  $\mapsto$  (req, [d]))
| Some (p, q)  $\Rightarrow$  store (dip  $\mapsto$  (p, q @ [d]))"

```

```

lemma qD_add [simp]:
fixes d dip store
shows "qD(add d dip store) = insert dip (qD store)"
<proof>

```

```

definition drop :: "ip  $\Rightarrow$  store  $\rightarrow$  store"
where "drop dip store  $\equiv$ 
map_option ( $\lambda$ (p, q). if tl q = [] then store (dip := None)
else store (dip  $\mapsto$  (p, tl q))) (store dip)"

```

```

definition sigma_p_flag :: "store  $\Rightarrow$  ip  $\rightarrow$  p" (" $\sigma_{\text{p-flag}}'(\_, \_)$ ")
where " $\sigma_{\text{p-flag}}$ (store, dip)  $\equiv$  map_option fst (store dip)"

```

```

definition unsetRRF :: "store  $\Rightarrow$  ip  $\Rightarrow$  store"
where "unsetRRF store dip  $\equiv$  case store dip of
None  $\Rightarrow$  store
| Some (p, q)  $\Rightarrow$  store (dip  $\mapsto$  (noreq, q))"

```

```

definition setRRF :: "store  $\Rightarrow$  (ip  $\rightarrow$  sqn)  $\Rightarrow$  store"
where "setRRF store dests  $\equiv$   $\lambda$ dip. if dests dip = None then store dip
else map_option ( $\lambda$ (_, q). (req, q)) (store dip)"

```



### 1.1.7 Comparison with the original technical report

The major differences with the AODV technical report of Fehnker et al are:

1. *nhop* is partial, thus a ‘*the*’ is needed, similarly for *dhops* and *addpreRT*.
2. *precs* is partial.
3.  $\sigma_{p\text{-flag}}(\textit{store}, \textit{dip})$  is partial.
4. The routing table (*rt*) is modelled as a map ( $\textit{ip} \Rightarrow \textit{r option}$ ) rather than a set of 7-tuples, likewise, the *r* is a 6-tuple rather than a 7-tuple, i.e., the destination ip-address (*dip*) is taken from the argument to the function, rather than a part of the result. Well-definedness then follows from the structure of the type and more related facts are available automatically, rather than having to be acquired through tedious proofs.
5. Similar remarks hold for the *dests* mapping passed to *invalidate*, and *store*.

end

## 1.2 AODV protocol messages

```
theory A_Aodv_Message
imports A_Norreqid
begin
```

```
datatype msg =
  Rreq nat ip sqn k ip sqn ip
  | Rrep nat ip sqn ip ip
  | Rerr "ip  $\rightarrow$  sqn" ip
  | Newpkt data ip
  | Pkt data ip ip
```

```
instantiation msg :: msg
begin
```

```
  definition newpkt_def [simp]: "newpkt  $\equiv$   $\lambda(d, dip). \text{Newpkt } d \text{ dip}"$ 
  definition eq_newpkt_def: "eq_newpkt m  $\equiv$  case m of Newpkt d dip  $\Rightarrow$  True | _  $\Rightarrow$  False"
```

```
  instance <proof>
```

```
end
```

The *msg* type models the different messages used within AODV. The instantiation as a *msg* is a technicality due to the special treatment of *newpkt* messages in the AWN SOS rules. This use of classes allows a clean separation of the AWN-specific definitions and these AODV-specific definitions.

```
definition rreq :: "nat  $\times$  ip  $\times$  sqn  $\times$  k  $\times$  ip  $\times$  sqn  $\times$  ip  $\Rightarrow$  msg"
  where "rreq  $\equiv$   $\lambda(\textit{hops}, \textit{dip}, \textit{dsn}, \textit{dsk}, \textit{oip}, \textit{osn}, \textit{sip}). \text{Rreq } \textit{hops} \textit{ dip } \textit{dsn} \textit{ dsk } \textit{oip} \textit{ osn} \textit{ sip}"$ 
```

```
lemma rreq_simp [simp]:
```

```
  "rreq(hops, dip, dsn, dsk, oip, osn, sip) = Rreq hops dip dsn dsk oip osn sip"
  <proof>
```

```
definition rrep :: "nat  $\times$  ip  $\times$  sqn  $\times$  ip  $\times$  ip  $\Rightarrow$  msg"
```

```
  where "rrep  $\equiv$   $\lambda(\textit{hops}, \textit{dip}, \textit{dsn}, \textit{oip}, \textit{sip}). \text{Rrep } \textit{hops} \textit{ dip } \textit{dsn} \textit{ oip} \textit{ sip}"$ 
```

```
lemma rrep_simp [simp]:
```

```
  "rrep(hops, dip, dsn, oip, sip) = Rrep hops dip dsn oip sip"
  <proof>
```

```
definition rerr :: "(ip  $\rightarrow$  sqn)  $\times$  ip  $\Rightarrow$  msg"
```

```
  where "rerr  $\equiv$   $\lambda(\textit{dests}, \textit{sip}). \text{Rerr } \textit{dests} \textit{ sip}"$ 
```

```
lemma rerr_simp [simp]:
```

```
  "rerr(dests, sip) = Rerr dests sip"
```

```

⟨proof⟩

lemma not_eq_newpkt_rreq [simp]: "¬eq_newpkt (Rreq hops dip dsn dsk oip osn sip)"
  ⟨proof⟩

lemma not_eq_newpkt_rrep [simp]: "¬eq_newpkt (Rrep hops dip dsn oip sip)"
  ⟨proof⟩

lemma not_eq_newpkt_rerr [simp]: "¬eq_newpkt (Rerr dests sip)"
  ⟨proof⟩

lemma not_eq_newpkt_pkt [simp]: "¬eq_newpkt (Pkt d dip sip)"
  ⟨proof⟩

definition pkt :: "data × ip × ip ⇒ msg"
  where "pkt ≡ λ(d, dip, sip). Pkt d dip sip"

lemma pkt_simp [simp]:
  "pkt(d, dip, sip) = Pkt d dip sip"
  ⟨proof⟩

end

```

## 1.3 The AODV protocol

```

theory A_Aodv
imports A_Aodv_Data A_Aodv_Message
  "../.../AWN/AWN_SOS_Labels" "../.../AWN/AWN_Invariants"
begin

```

### 1.3.1 Data state

```

record state =
  ip      :: "ip"
  sn      :: "sqn"
  rt      :: "rt"
  rreqs   :: "(ip × sqn) set"
  store   :: "store"

  msg     :: "msg"
  data    :: "data"
  dests   :: "ip → sqn"
  pre     :: "ip set"
  dip     :: "ip"
  oip     :: "ip"
  hops    :: "nat"
  dsn     :: "sqn"
  dsk     :: "k"
  osn     :: "sqn"
  sip     :: "ip"

abbreviation aodv_init :: "ip ⇒ state"
where "aodv_init i ≡ (|
  ip = i,
  sn = 1,
  rt = empty,
  rreqs = {},
  store = empty,

  msg = (SOME x. True),
  data = (SOME x. True),
  dests = (SOME x. True),
  pre = (SOME x. True),
  dip = (SOME x. True),

```

```

    oip    = (SOME x. True),
    hops   = (SOME x. True),
    dsn    = (SOME x. True),
    dsk    = (SOME x. True),
    osn    = (SOME x. True),
    sip    = (SOME x. x ≠ i)
  )"

```

**lemma** `some_neq_not_eq [simp]: "¬((SOME x :: nat. x ≠ i) = i)"`  
 ⟨*proof*⟩

**definition** `clear_locals :: "state ⇒ state"`

```

where "clear_locals ξ = ξ (|
  msg      := (SOME x. True),
  data     := (SOME x. True),
  dests    := (SOME x. True),
  pre      := (SOME x. True),
  dip      := (SOME x. True),
  oip      := (SOME x. True),
  hops     := (SOME x. True),
  dsn      := (SOME x. True),
  dsk      := (SOME x. True),
  osn      := (SOME x. True),
  sip      := (SOME x. x ≠ ip ξ)
)"

```

**lemma** `clear_locals_sip_not_ip [simp]: "¬(sip (clear_locals ξ) = ip ξ)"`  
 ⟨*proof*⟩

**lemma** `clear_locals_but_not_globals [simp]:`

```

"ip (clear_locals ξ) = ip ξ"
"sn (clear_locals ξ) = sn ξ"
"rt (clear_locals ξ) = rt ξ"
"rreqs (clear_locals ξ) = rreqs ξ"
"store (clear_locals ξ) = store ξ"
⟨proof⟩

```

### 1.3.2 Auxilliary message handling definitions

**definition** `is_newpkt`

```

where "is_newpkt ξ ≡ case msg ξ of
  Newpkt data' dip' ⇒ { ξ(|data := data', dip := dip'|) }
  | _ ⇒ {}"

```

**definition** `is_pkt`

```

where "is_pkt ξ ≡ case msg ξ of
  Pkt data' dip' oip' ⇒ { ξ(|data := data', dip := dip', oip := oip'|) }
  | _ ⇒ {}"

```

**definition** `is_rreq`

```

where "is_rreq ξ ≡ case msg ξ of
  Rreq hops' dip' dsn' dsk' oip' osn' sip' ⇒
    { ξ(|hops := hops', dip := dip', dsn := dsn',
        dsk := dsk', oip := oip', osn := osn', sip := sip'|) }
  | _ ⇒ {}"

```

**lemma** `is_rreq_asm [dest!]:`

```

assumes "ξ' ∈ is_rreq ξ"
shows "(∃hops' rreqid' dip' dsn' dsk' oip' osn' sip'.
  msg ξ = Rreq hops' dip' dsn' dsk' oip' osn' sip' ∧
  ξ' = ξ(|hops := hops', dip := dip', dsn := dsn',
        dsk := dsk', oip := oip', osn := osn', sip := sip'|))"

```

⟨*proof*⟩

```

definition is_rrep
where "is_rrep  $\xi \equiv$  case msg  $\xi$  of
      Rrep hops' dip' dsn' oip' sip'  $\Rightarrow$ 
        {  $\xi(\mid$  hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip'  $\mid)$  }
      | _  $\Rightarrow$  {}"

```

```

lemma is_rrep_asm [dest!]:
  assumes " $\xi' \in$  is_rrep  $\xi$ "
  shows " $(\exists$  hops' dip' dsn' oip' sip'.
        msg  $\xi =$  Rrep hops' dip' dsn' oip' sip'  $\wedge$ 
         $\xi' = \xi(\mid$  hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip'  $\mid)$ )"
  <proof>

```

```

definition is_rerr
where "is_rerr  $\xi \equiv$  case msg  $\xi$  of
      Rerr dests' sip'  $\Rightarrow$  {  $\xi(\mid$  dests := dests', sip := sip'  $\mid)$  }
      | _  $\Rightarrow$  {}"

```

```

lemma is_rerr_asm [dest!]:
  assumes " $\xi' \in$  is_rerr  $\xi$ "
  shows " $(\exists$  dests' sip'.
        msg  $\xi =$  Rerr dests' sip'  $\wedge$ 
         $\xi' = \xi(\mid$  dests := dests', sip := sip'  $\mid)$ )"
  <proof>

```

```

lemmas is_msg_defs =
  is_rerr_def is_rrep_def is_rreq_def is_pkt_def is_newpkt_def

```

```

lemma is_msg_inv_ip [simp]:
  " $\xi' \in$  is_rerr  $\xi \implies$  ip  $\xi' =$  ip  $\xi$ "
  " $\xi' \in$  is_rrep  $\xi \implies$  ip  $\xi' =$  ip  $\xi$ "
  " $\xi' \in$  is_rreq  $\xi \implies$  ip  $\xi' =$  ip  $\xi$ "
  " $\xi' \in$  is_pkt  $\xi \implies$  ip  $\xi' =$  ip  $\xi$ "
  " $\xi' \in$  is_newpkt  $\xi \implies$  ip  $\xi' =$  ip  $\xi$ "
  <proof>

```

```

lemma is_msg_inv_sn [simp]:
  " $\xi' \in$  is_rerr  $\xi \implies$  sn  $\xi' =$  sn  $\xi$ "
  " $\xi' \in$  is_rrep  $\xi \implies$  sn  $\xi' =$  sn  $\xi$ "
  " $\xi' \in$  is_rreq  $\xi \implies$  sn  $\xi' =$  sn  $\xi$ "
  " $\xi' \in$  is_pkt  $\xi \implies$  sn  $\xi' =$  sn  $\xi$ "
  " $\xi' \in$  is_newpkt  $\xi \implies$  sn  $\xi' =$  sn  $\xi$ "
  <proof>

```

```

lemma is_msg_inv_rt [simp]:
  " $\xi' \in$  is_rerr  $\xi \implies$  rt  $\xi' =$  rt  $\xi$ "
  " $\xi' \in$  is_rrep  $\xi \implies$  rt  $\xi' =$  rt  $\xi$ "
  " $\xi' \in$  is_rreq  $\xi \implies$  rt  $\xi' =$  rt  $\xi$ "
  " $\xi' \in$  is_pkt  $\xi \implies$  rt  $\xi' =$  rt  $\xi$ "
  " $\xi' \in$  is_newpkt  $\xi \implies$  rt  $\xi' =$  rt  $\xi$ "
  <proof>

```

```

lemma is_msg_inv_rreqs [simp]:
  " $\xi' \in$  is_rerr  $\xi \implies$  rreqs  $\xi' =$  rreqs  $\xi$ "
  " $\xi' \in$  is_rrep  $\xi \implies$  rreqs  $\xi' =$  rreqs  $\xi$ "
  " $\xi' \in$  is_rreq  $\xi \implies$  rreqs  $\xi' =$  rreqs  $\xi$ "
  " $\xi' \in$  is_pkt  $\xi \implies$  rreqs  $\xi' =$  rreqs  $\xi$ "
  " $\xi' \in$  is_newpkt  $\xi \implies$  rreqs  $\xi' =$  rreqs  $\xi$ "
  <proof>

```

```

lemma is_msg_inv_store [simp]:
  " $\xi' \in$  is_rerr  $\xi \implies$  store  $\xi' =$  store  $\xi$ "
  " $\xi' \in$  is_rrep  $\xi \implies$  store  $\xi' =$  store  $\xi$ "
  " $\xi' \in$  is_rreq  $\xi \implies$  store  $\xi' =$  store  $\xi$ "

```

```

"ξ' ∈ is_pkt ξ    ⇒ store ξ' = store ξ"
"ξ' ∈ is_newpkt ξ ⇒ store ξ' = store ξ"
⟨proof⟩

```

**lemma** *is\_msg\_inv\_sip* [simp]:

```

"ξ' ∈ is_pkt ξ    ⇒ sip ξ' = sip ξ"
"ξ' ∈ is_newpkt ξ ⇒ sip ξ' = sip ξ"
⟨proof⟩

```

### 1.3.3 The protocol process

**datatype** *pseqp* =

```

  PAadv
| PNewPkt
| PPkt
| PRreq
| PRrep
| PRerr

```

**fun** *nat\_of\_seqp* :: "pseqp ⇒ nat"

**where**

```

  "nat_of_seqp PAadv = 1"
| "nat_of_seqp PPkt = 2"
| "nat_of_seqp PNewPkt = 3"
| "nat_of_seqp PRreq = 4"
| "nat_of_seqp PRrep = 5"
| "nat_of_seqp PRerr = 6"

```

**instantiation** "pseqp" :: ord

**begin**

**definition** *less\_eq\_seqp* [iff]: "l1 ≤ l2 = (nat\_of\_seqp l1 ≤ nat\_of\_seqp l2)"

**definition** *less\_seqp* [iff]: "l1 < l2 = (nat\_of\_seqp l1 < nat\_of\_seqp l2)"

**instance** ⟨proof⟩

**end**

**abbreviation** *AADV*

**where**

```

"AADV ≡ λ_. [[clear_locals]] call(PAadv)"

```

**abbreviation** *PKT*

**where**

```

"PKT args ≡

```

```

  [[ξ. let (data, dip, oip) = args ξ in
    (clear_locals ξ) (| data := data, dip := dip, oip := oip |)]
  call(PPkt)"

```

**abbreviation** *NEWPKT*

**where**

```

"NEWPKT args ≡
  [[ξ. let (data, dip) = args ξ in
    (clear_locals ξ) (| data := data, dip := dip |)]
  call(PNewPkt)"

```

**abbreviation** *RREQ*

**where**

```

"RREQ args ≡
  [[ξ. let (hops, dip, dsn, dsk, oip, osn, sip) = args ξ in
    (clear_locals ξ) (| hops := hops, dip := dip,
                      dsn := dsn, dsk := dsk, oip := oip,
                      osn := osn, sip := sip |)]
  call(PRreq)"

```

**abbreviation** *RREP*

**where**

```

"RREP args ≡
  [[ξ. let (hops, dip, dsn, oip, sip) = args ξ in
    (clear_locals ξ) (| hops := hops, dip := dip, dsn := dsn,
      oip := oip, sip := sip |)]
  call(PRrep)"

```

abbreviation RERR

where

```

"RERR args ≡
  [[ξ. let (dests, sip) = args ξ in
    (clear_locals ξ) (| dests := dests, sip := sip |)]
  call(PRerr)"

```

fun  $\Gamma_{AODV} :: "(state, msg, pseqp, pseqp label) seqp\_env"$

where

```

"ΓAODV PAodv = labelled PAodv (
  receive(λmsg' ξ. ξ (| msg := msg' |)).
  (
    ⟨is_newpkt⟩ NEWPKT(λξ. (data ξ, ip ξ))
    ⊕ ⟨is_pkt⟩ PKT(λξ. (data ξ, dip ξ, oip ξ))
    ⊕ ⟨is_rreq⟩
      [[ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, { }) |)]
      RREQ(λξ. (hops ξ, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, sip ξ))
    ⊕ ⟨is_rrep⟩
      [[ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, { }) |)]
      RREP(λξ. (hops ξ, dip ξ, dsn ξ, oip ξ, sip ξ))
    ⊕ ⟨is_rerr⟩
      [[ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, { }) |)]
      RERR(λξ. (dests ξ, sip ξ))
  )
  ⊕ ⟨λξ. { ξ(| dip := dip |) | dip. dip ∈ qD(store ξ) ∩ vD(rt ξ) }⟩
    [[ξ. ξ (| data := hd(σqueue(store ξ, dip ξ)) |)]
    unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, ip ξ)).
    [[ξ. ξ (| store := the (drop (dip ξ) (store ξ)) |)]
    AODV()
    ▷ [[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
      then Some (inc (sqn (rt ξ) rip)) else None) |)]
      [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)]
      [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |)]
      [[ξ. ξ (| pre := ⋃ { the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } |)]
      [[ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ { })
        then (dests ξ) rip else None) |)]
      groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)). AODV()
  ⊕ ⟨λξ. { ξ(| dip := dip |)
    | dip. dip ∈ qD(store ξ) - vD(rt ξ) ∧ the (σp-flag(store ξ, dip)) = req }⟩
    [[ξ. ξ (| store := unsetRRF (store ξ) (dip ξ) |)]
    [[ξ. ξ (| sn := inc (sn ξ) |)]
    [[ξ. ξ (| rreqs := rreqs ξ ∪ {(ip ξ, sn ξ)} |)]
    broadcast(λξ. rreq(0, dip ξ, sqn (rt ξ) (dip ξ), sqnf (rt ξ) (dip ξ),
      ip ξ, sn ξ, ip ξ)). AODV()"

```

```

| "ΓAODV PNewPkt = labelled PNewPkt (
  ⟨ξ. dip ξ = ip ξ⟩
  deliver(λξ. data ξ).AODV()
  ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
  [[ξ. ξ (| store := add (data ξ) (dip ξ) (store ξ) |)]
  AODV()"

```

```

| "ΓAODV PPkt = labelled PPkt (
  ⟨ξ. dip ξ = ip ξ⟩
  deliver(λξ. data ξ).AODV()
  ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
  (
    ⟨ξ. dip ξ ∈ vD (rt ξ)⟩
    unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, oip ξ)).AODV()
  )

```

```

▷
[[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
then Some (inc (sqn (rt ξ) rip)) else None) |])]
[[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)]]
[[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |)]]
[[ξ. ξ (| pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } |)]]
[[ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ { })
then (dests ξ) rip else None) |)]]
groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
⊕ ⟨ξ. dip ξ ∉ vD (rt ξ)⟩
(
⟨ξ. dip ξ ∈ iD (rt ξ)⟩
groupcast(λξ. the (precs (rt ξ) (dip ξ)),
λξ. rerr([dip ξ ↦ sqn (rt ξ) (dip ξ)], ip ξ)). AODV()
⊕ ⟨ξ. dip ξ ∉ iD (rt ξ)⟩
AODV()
)
))"

| "ΓAODV PRreq = labelled PRreq (
⟨ξ. (oip ξ, osn ξ) ∈ rreqs ξ⟩
AODV()
⊕ ⟨ξ. (oip ξ, osn ξ) ∉ rreqs ξ⟩
[[ξ. ξ (| rt := update (rt ξ) (oip ξ) (osn ξ, kno, val, hops ξ + 1, sip ξ, { }) |)]]
[[ξ. ξ (| rreqs := rreqs ξ ∪ {(oip ξ, osn ξ)} |)]]
(
⟨ξ. dip ξ = ip ξ⟩
[[ξ. ξ (| sn := max (sn ξ) (dsn ξ) |)]]
unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(0, dip ξ, sn ξ, oip ξ, ip ξ)).AODV()
▷
[[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
then Some (inc (sqn (rt ξ) rip)) else None) |])]
[[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)]]
[[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |)]]
[[ξ. ξ (| pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } |)]]
[[ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ { })
then (dests ξ) rip else None) |)]]
groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
(
⟨ξ. dip ξ ∈ vD (rt ξ) ∧ dsn ξ ≤ sqn (rt ξ) (dip ξ) ∧ sqnf (rt ξ) (dip ξ) = kno⟩
[[ξ. ξ (| rt := the (addpreRT (rt ξ) (dip ξ) {sip ξ}) |)]]
[[ξ. ξ (| rt := the (addpreRT (rt ξ) (oip ξ) {the (nhop (rt ξ) (dip ξ))}) |)]]
unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(the (dhops (rt ξ) (dip ξ)), dip ξ,
sqn (rt ξ) (dip ξ), oip ξ, ip ξ)).
AODV()
▷
[[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
then Some (inc (sqn (rt ξ) rip)) else None) |])]
[[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)]]
[[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |)]]
[[ξ. ξ (| pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } |)]]
[[ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ { })
then (dests ξ) rip else None) |)]]
groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
⊕ ⟨ξ. dip ξ ∉ vD (rt ξ) ∨ sqn (rt ξ) (dip ξ) < dsn ξ ∨ sqnf (rt ξ) (dip ξ) = unk⟩
broadcast(λξ. rreq(hops ξ + 1, dip ξ, max (sqn (rt ξ) (dip ξ)) (dsn ξ),
dsk ξ, oip ξ, osn ξ, ip ξ)).
AODV()
)
))"

| "ΓAODV PRrep = labelled PRrep (
⟨ξ. rt ξ ≠ update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, { })⟩

```

```

(
  [[ξ. ξ (| rt := update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, { }) |) ]]
  (
    (ξ. oip ξ = ip ξ )
    AODV()
    ⊕ (ξ. oip ξ ≠ ip ξ )
    (
      (ξ. oip ξ ∈ vD (rt ξ))
      [[ξ. ξ (| rt := the (addpreRT (rt ξ) (dip ξ) {the (nhop (rt ξ) (oip ξ))}) |) ]]
      [[ξ. ξ (| rt := the (addpreRT (rt ξ) (the (nhop (rt ξ) (dip ξ)))
        {the (nhop (rt ξ) (oip ξ))}) |) ]]
      unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(hops ξ + 1, dip ξ, dsn ξ, oip ξ, ip ξ)).
      AODV()
    ▷
      [[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
        then Some (inc (sqn (rt ξ) rip)) else None) |) ]]
      [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |) ]]
      [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |) ]]
      [[ξ. ξ (| pre := ⋃ { the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } |) ]]
      [[ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ { })
        then (dests ξ) rip else None) |) ]]
      groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
    ⊕ (ξ. oip ξ ∉ vD (rt ξ))
    AODV()
  )
)
)
)
⊕ (ξ. rt ξ = update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, { }) )
AODV()
)"

/ "ΓAODV PRerr = labelled PRerr (
  [[ξ. ξ (| dests := (λrip. case (dests ξ) rip of None ⇒ None
    | Some rsn ⇒ if rip ∈ vD (rt ξ) ∧ the (nhop (rt ξ) rip) = sip ξ
      ∧ sqn (rt ξ) rip < rsn then Some rsn else None) |) ]]
  [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |) ]]
  [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |) ]]
  [[ξ. ξ (| pre := ⋃ { the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } |) ]]
  [[ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ { })
    then (dests ξ) rip else None) |) ]]
  groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)). AODV()"

declare ΓAODV.simps [simp del, code del]
lemmas ΓAODV.simps [simp, code] = ΓAODV.simps [simplified]

fun ΓAODV.skeleton
where
  "ΓAODV.skeleton PAodv = seqp_skeleton (ΓAODV PAodv)"
  / "ΓAODV.skeleton PNewPkt = seqp_skeleton (ΓAODV PNewPkt)"
  / "ΓAODV.skeleton PPkt = seqp_skeleton (ΓAODV PPkt)"
  / "ΓAODV.skeleton PRreq = seqp_skeleton (ΓAODV PRreq)"
  / "ΓAODV.skeleton PRrep = seqp_skeleton (ΓAODV PRrep)"
  / "ΓAODV.skeleton PRerr = seqp_skeleton (ΓAODV PRerr)"

lemma ΓAODV.skeleton_wf [simp]:
  "wellformed ΓAODV.skeleton"
  <proof>

declare ΓAODV.skeleton.simps [simp del, code del]
lemmas ΓAODV.skeleton.simps [simp, code]
  = ΓAODV.skeleton.simps [simplified ΓAODV.simps seqp_skeleton.simps]

lemma aodv_proc_cases [dest]:
  fixes p pn

```



shows "p ∈ cterms1 (Γ<sub>AODV</sub> pn) ⇒  
 (p ∈ cterms1 (Γ<sub>AODV</sub> PAadv) ∨  
 p ∈ cterms1 (Γ<sub>AODV</sub> PNewPkt) ∨  
 p ∈ cterms1 (Γ<sub>AODV</sub> PPkt) ∨  
 p ∈ cterms1 (Γ<sub>AODV</sub> PRreq) ∨  
 p ∈ cterms1 (Γ<sub>AODV</sub> PRrep) ∨  
 p ∈ cterms1 (Γ<sub>AODV</sub> PRerr))"  
 ⟨proof⟩

**definition** σ<sub>AODV</sub> :: "ip ⇒ (state × (state, msg, pseq, pseq label) seq) set"  
 where "σ<sub>AODV</sub> i ≡ {(aadv\_init i, Γ<sub>AODV</sub> PAadv)}"

**abbreviation** paadv  
 :: "ip ⇒ (state × (state, msg, pseq, pseq label) seq, msg seq\_action) automaton"  
 where  
 "paadv i ≡ (| init = σ<sub>AODV</sub> i, trans = seqp\_sos Γ<sub>AODV</sub> |)"

**lemma** aadv\_trans: "trans (paadv i) = seqp\_sos Γ<sub>AODV</sub>"  
 ⟨proof⟩

**lemma** aadv\_control\_within [simp]: "control\_within Γ<sub>AODV</sub> (init (paadv i))"  
 ⟨proof⟩

**lemma** aadv\_wf [simp]:  
 "wellformed Γ<sub>AODV</sub>"  
 ⟨proof⟩

**lemmas** aadv\_labels\_not\_empty [simp] = labels\_not\_empty [OF aadv\_wf]

**lemma** aadv\_ex\_label [intro]: "∃l. l ∈ labels Γ<sub>AODV</sub> p"  
 ⟨proof⟩

**lemma** aadv\_ex\_labelE [elim]:  
 assumes "∀l ∈ labels Γ<sub>AODV</sub> p. P l p"  
 and "∃p l. P l p ⇒ Q"  
 shows "Q"  
 ⟨proof⟩

**lemma** aadv\_simple\_labels [simp]: "simple\_labels Γ<sub>AODV</sub>"  
 ⟨proof⟩

**lemma** σ<sub>AODV</sub>\_labels [simp]: "(ξ, p) ∈ σ<sub>AODV</sub> i ⇒ labels Γ<sub>AODV</sub> p = {PAadv-.:0}"  
 ⟨proof⟩

**lemma** aadv\_init\_kD\_empty [simp]:  
 "(ξ, p) ∈ σ<sub>AODV</sub> i ⇒ kD (rt ξ) = {}"  
 ⟨proof⟩

**lemma** aadv\_init\_sip\_not\_ip [simp]: "¬(sip (aadv\_init i) = i)" ⟨proof⟩

**lemma** aadv\_init\_sip\_not\_ip' [simp]:  
 assumes "(ξ, p) ∈ σ<sub>AODV</sub> i"  
 shows "sip ξ ≠ ip ξ"  
 ⟨proof⟩

**lemma** aadv\_init\_sip\_not\_i [simp]:  
 assumes "(ξ, p) ∈ σ<sub>AODV</sub> i"  
 shows "sip ξ ≠ i"  
 ⟨proof⟩

**lemma** clear\_locals\_sip\_not\_ip':  
 assumes "ip ξ = i"  
 shows "¬(sip (clear\_locals ξ) = i)"  
 ⟨proof⟩

Stop the simplifier from descending into process terms.

```
declare seqp_congs [cong]
```

Configure the main invariant tactic for AODV.

```
declare
```

```
  ΓAODV_simps [ctermenv]
  aodv_proc_cases [ctermenv_cases]
  seq_invariant_ctermenvI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
                           ctermenv_intros]
  seq_step_invariant_ctermenvI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
                                ctermenv_intros]
```

```
end
```

## 1.4 Invariant assumptions and properties

```
theory A_Aodv_Predicates
```

```
imports A_Aodv
```

```
begin
```

Definitions for expression assumptions on incoming messages and properties of outgoing messages.

```
abbreviation not_Pkt :: "msg ⇒ bool"
```

```
where "not_Pkt m ≡ case m of Pkt _ _ _ ⇒ False | _ ⇒ True"
```

```
definition msg_sender :: "msg ⇒ ipc"
```

```
where "msg_sender m ≡ case m of Rreq _ _ _ _ _ ipc ⇒ ipc
    | Rrep _ _ _ _ ipc ⇒ ipc
    | Rerr _ ipc ⇒ ipc
    | Pkt _ _ ipc ⇒ ipc"
```

```
lemma msg_sender_simps [simp]:
```

```
"∧hops dip dsn dsk oip osn sip.
  msg_sender (Rreq hops dip dsn dsk oip osn sip) = sip"
"∧hops dip dsn oip sip. msg_sender (Rrep hops dip dsn oip sip) = sip"
"∧dests sip.          msg_sender (Rerr dests sip) = sip"
"∧dip sip.            msg_sender (Pkt d dip sip) = sip"
⟨proof⟩
```

```
definition msg_zhops :: "msg ⇒ bool"
```

```
where "msg_zhops m ≡ case m of
    Rreq hopsc dipc _ _ oipcc _ sipc ⇒ hopsc = 0 → oipcc = sipc
  | Rrep hopsc dipc _ _ sipc ⇒ hopsc = 0 → dipc = sipc
  | _ ⇒ True"
```

```
lemma msg_zhops_simps [simp]:
```

```
"∧hops dip dsn dsk oip osn sip.
  msg_zhops (Rreq hops dip dsn dsk oip osn sip) = (hops = 0 → oip = sip)"
"∧hops dip dsn oip sip. msg_zhops (Rrep hops dip dsn oip sip) = (hops = 0 → dip = sip)"
"∧dests sip.          msg_zhops (Rerr dests sip) = True"
"∧dip.                msg_zhops (Newpkt d dip) = True"
"∧dip sip.           msg_zhops (Pkt d dip sip) = True"
⟨proof⟩
```

```
definition rreq_rrep_sn :: "msg ⇒ bool"
```

```
where "rreq_rrep_sn m ≡ case m of Rreq _ _ _ _ _ osnc _ ⇒ osnc ≥ 1
    | Rrep _ _ dsnc _ _ ⇒ dsnc ≥ 1
    | _ ⇒ True"
```

```
lemma rreq_rrep_sn_simps [simp]:
```

```
"∧hops dip dsn dsk oip osn sip.
  rreq_rrep_sn (Rreq hops dip dsn dsk oip osn sip) = (osn ≥ 1)"
"∧hops dip dsn oip sip. rreq_rrep_sn (Rrep hops dip dsn oip sip) = (dsn ≥ 1)"
"∧dests sip.          rreq_rrep_sn (Rerr dests sip) = True"
```

```

"∧d dip.                rreq_rrep_sn (Newpkt d dip) = True"
"∧d dip sip.           rreq_rrep_sn (Pkt d dip sip) = True"
⟨proof⟩

```

**definition** `rreq_rrep_fresh` :: "rt ⇒ msg ⇒ bool"

```

where "rreq_rrep_fresh crt m ≡ case m of Rreq hopsc _ _ _ oipc osnc ipcc ⇒ (ipcc ≠ oipc →
    oipc∈kD(crt) ∧ (sqn crt oipc > osnc
        ∨ (sqn crt oipc = osnc
            ∧ the (dhops crt oipc) ≤ hopsc
            ∧ the (flag crt oipc) = val)))
    | Rrep hopsc dipc dsnc _ ipcc ⇒ (ipcc ≠ dipc →
        dipc∈kD(crt)
        ∧ sqn crt dipc = dsnc
        ∧ the (dhops crt dipc) = hopsc
        ∧ the (flag crt dipc) = val)
    | _ ⇒ True"

```

**lemma** `rreq_rrep_fresh [simp]`:

```

"∧hops dip dsn dsk oip osn sip.
    rreq_rrep_fresh crt (Rreq hops dip dsn dsk oip osn sip) =
    (sip ≠ oip → oip∈kD(crt)
        ∧ (sqn crt oip > osn
            ∨ (sqn crt oip = osn
                ∧ the (dhops crt oip) ≤ hops
                ∧ the (flag crt oip) = val)))"
"∧hops dip dsn oip sip. rreq_rrep_fresh crt (Rrep hops dip dsn oip sip) =
    (sip ≠ dip → dip∈kD(crt)
        ∧ sqn crt dip = dsn
        ∧ the (dhops crt dip) = hops
        ∧ the (flag crt dip) = val)"
"∧dests sip.          rreq_rrep_fresh crt (Rerr dests sip) = True"
"∧d dip.              rreq_rrep_fresh crt (Newpkt d dip) = True"
"∧d dip sip.         rreq_rrep_fresh crt (Pkt d dip sip) = True"
⟨proof⟩

```

**definition** `rerr_invalid` :: "rt ⇒ msg ⇒ bool"

```

where "rerr_invalid crt m ≡ case m of Rerr destsc _ ⇒ (∀ripc∈dom(destsc).
    (ripc∈iD(crt) ∧ the (destsc ripc) = sqn crt ripc))
    | _ ⇒ True"

```

**lemma** `rerr_invalid [simp]`:

```

"∧hops dip dsn dsk oip osn sip.
    rerr_invalid crt (Rreq hops dip dsn dsk oip osn sip) = True"
"∧hops dip dsn oip sip. rerr_invalid crt (Rrep hops dip dsn oip sip) = True"
"∧dests sip.          rerr_invalid crt (Rerr dests sip) = (∀ripc∈dom(dests).
    ripc∈iD(crt) ∧ the (dests ripc) = sqn crt ripc)"
"∧d dip.              rerr_invalid crt (Newpkt d dip) = True"
"∧d dip sip.         rerr_invalid crt (Pkt d dip sip) = True"
⟨proof⟩

```

**definition**

```

initmissing :: "(nat ⇒ state option) × 'a ⇒ (nat ⇒ state) × 'a"

```

**where**

```

"initmissing σ = (λi. case (fst σ) i of None ⇒ aadv_init i | Some s ⇒ s, snd σ)"

```

**lemma** `not_in_net_ips_fst_init_missing [simp]`:

```

assumes "i ∉ net_ips σ"
shows "fst (initmissing (netgmap fst σ)) i = aadv_init i"
⟨proof⟩

```

**lemma** `fst_initmissing_netgmap_pair_fst [simp]`:

```

"fst (initmissing (netgmap (λ(p, q). (fst (id p), snd (id p), q)) s))
    = fst (initmissing (netgmap fst s))"
⟨proof⟩

```

We introduce a streamlined alternative to `initmissing` with `netgmap` to simplify invariant statements and thus facilitate their comprehension and presentation.

```
lemma fst_initmissing_netgmap_default_aodv_init_netlift:
  "fst (initmissing (netgmap fst s)) = default aodv_init (netlift fst s)"
  ⟨proof⟩
```

**definition**

```
netglobal :: "(nat ⇒ state) ⇒ bool ⇒ ((state × 'b) × 'c) net_state ⇒ bool"
where
  "netglobal P ≡ (λs. P (default aodv_init (netlift fst s)))"
```

end

## 1.5 Quality relations between routes

```
theory A_Fresher
imports A_Aodv_Data
begin
```

### 1.5.1 Net sequence numbers

**On individual routes**

**definition**

```
nsqnr :: "r ⇒ sqn"
where
  "nsqnr r ≡ if π4(r) = val ∨ π2(r) = 0 then π2(r) else (π2(r) - 1)"
```

**lemma nsqnr\_def'**:

```
"nsqnr r = (if π4(r) = inv then π2(r) - 1 else π2(r))"
⟨proof⟩
```

**lemma nsqn<sub>r</sub>\_zero [simp]**:

```
"∧ dsn dsk flag hops nhip pre. nsqnr (0, dsk, flag, hops, nhip, pre) = 0"
⟨proof⟩
```

**lemma nsqn<sub>r</sub>\_val [simp]**:

```
"∧ dsn dsk hops nhip pre. nsqnr (dsn, dsk, val, hops, nhip, pre) = dsn"
⟨proof⟩
```

**lemma nsqn<sub>r</sub>\_inv [simp]**:

```
"∧ dsn dsk hops nhip pre. nsqnr (dsn, dsk, inv, hops, nhip, pre) = dsn - 1"
⟨proof⟩
```

**lemma nsqn<sub>r</sub>\_lte\_dsn [simp]**:

```
"∧ dsn dsk flag hops nhip pre. nsqnr (dsn, dsk, flag, hops, nhip, pre) ≤ dsn"
⟨proof⟩
```

**On routes in routing tables**

**definition**

```
nsqn :: "rt ⇒ ip ⇒ sqn"
where
  "nsqn ≡ λrt dip. case σroute(rt, dip) of None ⇒ 0 | Some r ⇒ nsqnr(r)"
```

**lemma nsqn\_sqn\_def**:

```
"∧ rt dip. nsqn rt dip = (if flag rt dip = Some val ∨ sqn rt dip = 0
  then sqn rt dip else sqn rt dip - 1)"
⟨proof⟩
```

**lemma not\_in\_kD\_nsqn [simp]**:

```
assumes "dip ∉ kD(rt)"
shows "nsqn rt dip = 0"
⟨proof⟩
```

```

lemma kD_nsqn:
  assumes "dip ∈ kD(rt)"
  shows "nsqn rt dip = nsqnr(the (σroute(rt, dip)))"
  ⟨proof⟩

lemma nsqnr_r_flag_pred [simp, intro]:
  fixes dsn dsk flag hops nhip pre
  assumes "P (nsqnr (dsn, dsk, val, hops, nhip, pre))"
  and "P (nsqnr (dsn, dsk, inv, hops, nhip, pre))"
  shows "P (nsqnr (dsn, dsk, flag, hops, nhip, pre))"
  ⟨proof⟩

lemma nsqnr_addpreRT_inv [simp]:
  "∧rt dip npre dip'. dip ∈ kD(rt) ⇒
  nsqnr (the (the (addpreRT rt dip npre) dip')) = nsqnr (the (rt dip'))"
  ⟨proof⟩

lemma sqn_nsqn:
  "∧rt dip. sqn rt dip - 1 ≤ nsqn rt dip"
  ⟨proof⟩

lemma nsqn_sqn: "nsqn rt dip ≤ sqn rt dip"
  ⟨proof⟩

lemma val_nsqn_sqn [elim, simp]:
  assumes "ip ∈ kD(rt)"
  and "the (flag rt ip) = val"
  shows "nsqn rt ip = sqn rt ip"
  ⟨proof⟩

lemma vD_nsqn_sqn [elim, simp]:
  assumes "ip ∈ vD(rt)"
  shows "nsqn rt ip = sqn rt ip"
  ⟨proof⟩

lemma inv_nsqn_sqn [elim, simp]:
  assumes "ip ∈ kD(rt)"
  and "the (flag rt ip) = inv"
  shows "nsqn rt ip = sqn rt ip - 1"
  ⟨proof⟩

lemma iD_nsqn_sqn [elim, simp]:
  assumes "ip ∈ iD(rt)"
  shows "nsqn rt ip = sqn rt ip - 1"
  ⟨proof⟩

lemma nsqn_update_changed_kno_val [simp]: "∧rt ip dsn dsk hops nhip.
  rt ≠ update rt ip (dsn, kno, val, hops, nhip, { })
  ⇒ nsqn (update rt ip (dsn, kno, val, hops, nhip, { })) ip = dsn"
  ⟨proof⟩

lemma nsqn_addpreRT_inv [simp]:
  "∧rt dip npre dip'. dip ∈ kD(rt) ⇒
  nsqn (the (addpreRT rt dip npre)) dip' = nsqn rt dip'"
  ⟨proof⟩

lemma nsqn_update_other [simp]:
  fixes dsn dsk flag hops dip nhip pre rt ip
  assumes "dip ≠ ip"
  shows "nsqn (update rt ip (dsn, dsk, flag, hops, nhip, pre)) dip = nsqn rt dip"
  ⟨proof⟩

lemma nsqn_invalidate_eq:

```

```

assumes "dip ∈ kD(rt)"
  and "dests dip = Some rsn"
  shows "nsqn (invalidate rt dests) dip = rsn - 1"
⟨proof⟩

```

```

lemma nsqn_invalidate_other [simp]:
  assumes "dip ∈ kD(rt)"
    and "dip ∉ dom dests"
  shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
⟨proof⟩

```

## 1.5.2 Comparing routes

definition

```

fresher :: "r ⇒ r ⇒ bool" ("(_/ ⊆ _)" [51, 51] 50)

```

where

```

"fresher r r' ≡ ((nsqnr r < nsqnr r') ∨ (nsqnr r = nsqnr r' ∧ π5(r) ≥ π5(r')))"

```

```

lemma fresherI1 [intro]:
  assumes "nsqnr r < nsqnr r'"
  shows "r ⊆ r'"
⟨proof⟩

```

```

lemma fresherI2 [intro]:
  assumes "nsqnr r = nsqnr r'"
    and "π5(r) ≥ π5(r')"
  shows "r ⊆ r'"
⟨proof⟩

```

```

lemma fresherI [intro]:
  assumes "(nsqnr r < nsqnr r') ∨ (nsqnr r = nsqnr r' ∧ π5(r) ≥ π5(r'))"
  shows "r ⊆ r'"
⟨proof⟩

```

```

lemma fresherE [elim]:
  assumes "r ⊆ r'"
    and "nsqnr r < nsqnr r' ⇒ P r r'"
    and "nsqnr r = nsqnr r' ∧ π5(r) ≥ π5(r') ⇒ P r r'"
  shows "P r r'"
⟨proof⟩

```

```

lemma fresher_refl [simp]: "r ⊆ r"
⟨proof⟩

```

```

lemma fresher_trans [elim, trans]:
  "[ x ⊆ y; y ⊆ z ] ⇒ x ⊆ z"
⟨proof⟩

```

```

lemma not_fresher_trans [elim, trans]:
  "[ ¬(x ⊆ y); ¬(z ⊆ x) ] ⇒ ¬(z ⊆ y)"
⟨proof⟩

```

```

lemma fresher_dsn_flag_hops_const [simp]:
  fixes dsn dsk dsk' flag hops nhip nhip' pre pre'
  shows "(dsn, dsk, flag, hops, nhip, pre) ⊆ (dsn, dsk', flag, hops, nhip', pre')"
⟨proof⟩

```

```

lemma addpre_fresher [simp]: "∧r npre. r ⊆ (addpre r npre)"
⟨proof⟩

```

## 1.5.3 Comparing routing tables

definition

```

rt_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"

```

where

"rt\_fresher  $\equiv \lambda \text{dip } rt \text{ } rt'. (\text{the } (\sigma_{\text{route}}(rt, \text{dip}))) \sqsubseteq (\text{the } (\sigma_{\text{route}}(rt', \text{dip})))$ "

abbreviation

rt\_fresher\_syn :: "rt  $\Rightarrow$  ip  $\Rightarrow$  rt  $\Rightarrow$  bool" ("(\_/  $\sqsubseteq$  \_)" [51, 999, 51] 50)

where

"rt1  $\sqsubseteq_i$  rt2  $\equiv$  rt\_fresher i rt1 rt2"

lemma rt\_fresher\_def':

"(rt1  $\sqsubseteq_i$  rt2) = (nsqn<sub>r</sub> (the (rt1 i)) < nsqn<sub>r</sub> (the (rt2 i))  $\vee$   
nsqn<sub>r</sub> (the (rt1 i)) = nsqn<sub>r</sub> (the (rt2 i))  $\wedge$   $\pi_5$  (the (rt2 i))  $\leq$   $\pi_5$  (the (rt1 i)))"

$\langle$ proof $\rangle$

lemma single\_rt\_fresher [intro]:

assumes "the (rt1 ip)  $\sqsubseteq$  the (rt2 ip)"

shows "rt1  $\sqsubseteq_{\text{ip}}$  rt2"

$\langle$ proof $\rangle$

lemma rt\_fresher\_single [intro]:

assumes "rt1  $\sqsubseteq_{\text{ip}}$  rt2"

shows "the (rt1 ip)  $\sqsubseteq$  the (rt2 ip)"

$\langle$ proof $\rangle$

lemma rt\_fresher\_def2:

assumes "dip  $\in$  kD(rt1)"

and "dip  $\in$  kD(rt2)"

shows "(rt1  $\sqsubseteq_{\text{dip}}$  rt2) = (nsqn rt1 dip < nsqn rt2 dip  
 $\vee$  (nsqn rt1 dip = nsqn rt2 dip  
 $\wedge$  the (dhops rt1 dip)  $\geq$  the (dhops rt2 dip)))"

$\langle$ proof $\rangle$

lemma rt\_fresherI1 [intro]:

assumes "dip  $\in$  kD(rt1)"

and "dip  $\in$  kD(rt2)"

and "nsqn rt1 dip < nsqn rt2 dip"

shows "rt1  $\sqsubseteq_{\text{dip}}$  rt2"

$\langle$ proof $\rangle$

lemma rt\_fresherI2 [intro]:

assumes "dip  $\in$  kD(rt1)"

and "dip  $\in$  kD(rt2)"

and "nsqn rt1 dip = nsqn rt2 dip"

and "the (dhops rt1 dip)  $\geq$  the (dhops rt2 dip)"

shows "rt1  $\sqsubseteq_{\text{dip}}$  rt2"

$\langle$ proof $\rangle$

lemma rt\_fresherE [elim]:

assumes "rt1  $\sqsubseteq_{\text{dip}}$  rt2"

and "dip  $\in$  kD(rt1)"

and "dip  $\in$  kD(rt2)"

and "[ nsqn rt1 dip < nsqn rt2 dip ]  $\implies$  P rt1 rt2 dip"

and "[ nsqn rt1 dip = nsqn rt2 dip;  
the (dhops rt1 dip)  $\geq$  the (dhops rt2 dip) ]  $\implies$  P rt1 rt2 dip"

shows "P rt1 rt2 dip"

$\langle$ proof $\rangle$

lemma rt\_fresher\_refl [simp]: "rt  $\sqsubseteq_{\text{dip}}$  rt"

$\langle$ proof $\rangle$

lemma rt\_fresher\_trans [elim, trans]:

assumes "rt1  $\sqsubseteq_{\text{dip}}$  rt2"

and "rt2  $\sqsubseteq_{\text{dip}}$  rt3"

shows "rt1  $\sqsubseteq_{\text{dip}}$  rt3"

$\langle$ proof $\rangle$

lemma *rt\_fresher\_if\_Some* [intro!]:

assumes "the (rt dip)  $\sqsubseteq$  r"  
shows "rt  $\sqsubseteq_{\text{dip}}$  ( $\lambda ip. \text{if } ip = \text{dip} \text{ then Some } r \text{ else } rt \text{ ip}$ )"  
<proof>

definition *rt\_fresh\_as* :: "ip  $\Rightarrow$  rt  $\Rightarrow$  rt  $\Rightarrow$  bool"

where

"rt\_fresh\_as  $\equiv \lambda \text{dip } rt1 \ rt2. (rt1 \sqsubseteq_{\text{dip}} rt2) \wedge (rt2 \sqsubseteq_{\text{dip}} rt1)$ "

abbreviation

*rt\_fresh\_as\_syn* :: "rt  $\Rightarrow$  ip  $\Rightarrow$  rt  $\Rightarrow$  bool" ("(\_/  $\approx_{\text{dip}}$  \_)" [51, 999, 51] 50)

where

"rt1  $\approx_i$  rt2  $\equiv$  rt\_fresh\_as i rt1 rt2"

lemma *rt\_fresh\_as\_refl* [simp]: " $\wedge rt \text{ dip}. rt \approx_{\text{dip}} rt$ "

<proof>

lemma *rt\_fresh\_as\_trans* [simp, intro, trans]:

" $\wedge rt1 \ rt2 \ rt3 \text{ dip}. [\![ \ rt1 \approx_{\text{dip}} rt2; \ rt2 \approx_{\text{dip}} rt3 \ ]\!] \Longrightarrow rt1 \approx_{\text{dip}} rt3$ "

<proof>

lemma *rt\_fresh\_asI* [intro!]:

assumes "rt1  $\sqsubseteq_{\text{dip}}$  rt2"

and "rt2  $\sqsubseteq_{\text{dip}}$  rt1"

shows "rt1  $\approx_{\text{dip}}$  rt2"

<proof>

lemma *rt\_fresh\_as\_fresherI* [intro]:

assumes "dip  $\in kD(rt1)$ "

and "dip  $\in kD(rt2)$ "

and "the (rt1 dip)  $\sqsubseteq$  the (rt2 dip)"

and "the (rt2 dip)  $\sqsubseteq$  the (rt1 dip)"

shows "rt1  $\approx_{\text{dip}}$  rt2"

<proof>

lemma *nsqn\_rt\_fresh\_asI*:

assumes "dip  $\in kD(rt)$ "

and "dip  $\in kD(rt')$ "

and "nsqn rt dip = nsqn rt' dip"

and " $\pi_5(\text{the } (rt \text{ dip})) = \pi_5(\text{the } (rt' \text{ dip}))$ "

shows "rt  $\approx_{\text{dip}}$  rt'"

<proof>

lemma *rt\_fresh\_asE* [elim]:

assumes "rt1  $\approx_{\text{dip}}$  rt2"

and " $[\![ \ rt1 \sqsubseteq_{\text{dip}} rt2; \ rt2 \sqsubseteq_{\text{dip}} rt1 \ ]\!] \Longrightarrow P \text{ rt1 } rt2 \text{ dip}$ "

shows "P rt1 rt2 dip"

<proof>

lemma *rt\_fresh\_asD1* [dest]:

assumes "rt1  $\approx_{\text{dip}}$  rt2"

shows "rt1  $\sqsubseteq_{\text{dip}}$  rt2"

<proof>

lemma *rt\_fresh\_asD2* [dest]:

assumes "rt1  $\approx_{\text{dip}}$  rt2"

shows "rt2  $\sqsubseteq_{\text{dip}}$  rt1"

<proof>

lemma *rt\_fresh\_as\_sym*:

assumes "rt1  $\approx_{\text{dip}}$  rt2"

shows "rt2  $\approx_{\text{dip}}$  rt1"

<proof>



```

lemma not_rt_fresh_asI1 [intro]:
  assumes "¬ (rt1  $\sqsubseteq_{dip}$  rt2)"
  shows "¬ (rt1  $\approx_{dip}$  rt2)"
  <proof>

lemma not_rt_fresh_asI2 [intro]:
  assumes "¬ (rt2  $\sqsubseteq_{dip}$  rt1)"
  shows "¬ (rt1  $\approx_{dip}$  rt2)"
  <proof>

lemma not_single_rt_fresher [elim]:
  assumes "¬(the (rt1 ip)  $\sqsubseteq$  the (rt2 ip))"
  shows "¬(rt1  $\sqsubseteq_{ip}$  rt2)"
  <proof>

lemmas not_single_rt_fresh_asI1 [intro] = not_rt_fresh_asI1 [OF not_single_rt_fresher]
lemmas not_single_rt_fresh_asI2 [intro] = not_rt_fresh_asI2 [OF not_single_rt_fresher]

lemma not_rt_fresher_single [elim]:
  assumes "¬(rt1  $\sqsubseteq_{ip}$  rt2)"
  shows "¬(the (rt1 ip)  $\sqsubseteq$  the (rt2 ip))"
  <proof>

lemma rt_fresh_as_nsqr:
  assumes "dip  $\in$  kD(rt1)"
  and "dip  $\in$  kD(rt2)"
  and "rt1  $\approx_{dip}$  rt2"
  shows "nsqrr (the (rt2 dip)) = nsqrr (the (rt1 dip))"
  <proof>

lemma rt_fresher_mapupd [intro!]:
  assumes "dip $\in$ kD(rt)"
  and "the (rt dip)  $\sqsubseteq$  r"
  shows "rt  $\sqsubseteq_{dip}$  rt(dip  $\mapsto$  r)"
  <proof>

lemma rt_fresher_map_update_other [intro!]:
  assumes "dip $\in$ kD(rt)"
  and "dip  $\neq$  ip"
  shows "rt  $\sqsubseteq_{dip}$  rt(ip  $\mapsto$  r)"
  <proof>

lemma rt_fresher_update_other [simp]:
  assumes inkD: "dip $\in$ kD(rt)"
  and "dip  $\neq$  ip"
  shows "rt  $\sqsubseteq_{dip}$  update rt ip r"
  <proof>

theorem rt_fresher_update [simp]:
  assumes "dip $\in$ kD(rt)"
  and "the (dhops rt dip)  $\geq$  1"
  and "update_arg_wf r"
  shows "rt  $\sqsubseteq_{dip}$  update rt ip r"
  <proof>

theorem rt_fresher_invalidate [simp]:
  assumes "dip $\in$ kD(rt)"
  and indests: " $\forall rip \in \text{dom}(\text{dests}). rip \in vD(rt) \wedge \text{sqn } rt \text{ } rip < \text{the } (\text{dests } rip)$ "
  shows "rt  $\sqsubseteq_{dip}$  invalidate rt dests"
  <proof>

lemma nsqrr_invalidate [simp]:
  assumes "dip $\in$ kD(rt)"

```

`and "dip ∈ dom(dests)"`  
`shows "nsqnr (the (invalidate rt dests dip)) = the (dests dip) - 1"`  
`<proof>`

**lemma** `rt_fresh_as_inc_invalidate [simp]:`  
`assumes "dip ∈ kD(rt)"`  
`and "∀ rip ∈ dom(dests). rip ∈ vD(rt) ∧ the (dests rip) = inc (sqn rt rip)"`  
`shows "rt ≈dip invalidate rt dests"`  
`<proof>`

**lemmas** `rt_fresher_inc_invalidate [simp] = rt_fresh_as_inc_invalidate [THEN rt_fresh_asD1]`

**lemma** `rt_fresh_as_addpreRT [simp]:`  
`assumes "ip ∈ kD(rt)"`  
`shows "rt ≈dip the (addpreRT rt ip npre)"`  
`<proof>`

**lemmas** `rt_fresher_addpreRT [simp] = rt_fresh_as_addpreRT [THEN rt_fresh_asD1]`

#### 1.5.4 Strictly comparing routing tables

**definition** `rt_strictly_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"`

where

`"rt_strictly_fresher ≡ λdip rt1 rt2. (rt1 ⊑dip rt2) ∧ ¬(rt1 ≈dip rt2)"`

abbreviation

`rt_strictly_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" ("(_/ ⊑_)" [51, 999, 51] 50)`

where

`"rt1 ⊑i rt2 ≡ rt_strictly_fresher i rt1 rt2"`

**lemma** `rt_strictly_fresher_def'':`  
`"rt1 ⊑i rt2 = ((rt1 ⊑i rt2) ∧ ¬(rt2 ⊑i rt1))"`  
`<proof>`

**lemma** `rt_strictly_fresherI' [intro]:`  
`assumes "rt1 ⊑i rt2"`  
`and "¬(rt2 ⊑i rt1)"`  
`shows "rt1 ⊑i rt2"`  
`<proof>`

**lemma** `rt_strictly_fresherE' [elim]:`  
`assumes "rt1 ⊑i rt2"`  
`and "[| rt1 ⊑i rt2; ¬(rt2 ⊑i rt1) |] ⇒ P rt1 rt2 i"`  
`shows "P rt1 rt2 i"`  
`<proof>`

**lemma** `rt_strictly_fresherI [intro]:`  
`assumes "rt1 ⊑i rt2"`  
`and "¬(rt1 ≈i rt2)"`  
`shows "rt1 ⊑i rt2"`  
`<proof>`

**lemmas** `rt_strictly_fresher_singleI [elim] = rt_strictly_fresherI [OF single_rt_fresher]`

**lemma** `rt_strictly_fresherE [elim]:`  
`assumes "rt1 ⊑i rt2"`  
`and "[| rt1 ⊑i rt2; ¬(rt1 ≈i rt2) |] ⇒ P rt1 rt2 i"`  
`shows "P rt1 rt2 i"`  
`<proof>`

**lemma** `rt_strictly_fresher_def':`  
`"rt1 ⊑i rt2 =`  
`(nsqnr (the (rt1 i)) < nsqnr (the (rt2 i)))`  
`∨ (nsqnr (the (rt1 i)) = nsqnr (the (rt2 i)) ∧ π5(the (rt1 i)) > π5(the (rt2 i)))"`

```

⟨proof⟩

lemma rt_strictly_fresher_fresherD [dest]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
  shows "the (rt1 dip)  $\sqsubseteq$  the (rt2 dip)"
⟨proof⟩

lemma rt_strictly_fresher_not_fresh_asD [dest]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
  shows " $\neg$  rt1  $\approx_{dip}$  rt2"
⟨proof⟩

lemma rt_strictly_fresher_trans [elim, trans]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
  and "rt2  $\sqsubseteq_{dip}$  rt3"
  shows "rt1  $\sqsubseteq_{dip}$  rt3"
⟨proof⟩

lemma rt_strictly_fresher_irefl [simp]: " $\neg$  (rt  $\sqsubseteq_{dip}$  rt)"
⟨proof⟩

lemma rt_fresher_trans_rt_strictly_fresher [elim, trans]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
  and "rt2  $\sqsubseteq_{dip}$  rt3"
  shows "rt1  $\sqsubseteq_{dip}$  rt3"
⟨proof⟩

lemma rt_fresher_trans_rt_strictly_fresher' [elim, trans]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
  and "rt2  $\sqsubseteq_{dip}$  rt3"
  shows "rt1  $\sqsubseteq_{dip}$  rt3"
⟨proof⟩

lemma rt_fresher_imp_nsqn_le:
  assumes "rt1  $\sqsubseteq_{ip}$  rt2"
  and "ip  $\in$  kD rt1"
  and "ip  $\in$  kD rt2"
  shows "nsqn rt1 ip  $\leq$  nsqn rt2 ip"
⟨proof⟩

lemma rt_strictly_fresher_ltI [intro]:
  assumes "dip  $\in$  kD(rt1)"
  and "dip  $\in$  kD(rt2)"
  and "nsqn rt1 dip < nsqn rt2 dip"
  shows "rt1  $\sqsubseteq_{dip}$  rt2"
⟨proof⟩

lemma rt_strictly_fresher_eqI [intro]:
  assumes "i  $\in$  kD(rt1)"
  and "i  $\in$  kD(rt2)"
  and "nsqn rt1 i = nsqn rt2 i"
  and " $\pi_5$ (the (rt2 i)) <  $\pi_5$ (the (rt1 i))"
  shows "rt1  $\sqsubseteq_i$  rt2"
⟨proof⟩

lemma invalidate_rtsf_left [simp]:
  " $\bigwedge$  dests dip rt rt'. dests dip = None  $\implies$  (invalidate rt dests  $\sqsubseteq_{dip}$  rt') = (rt  $\sqsubseteq_{dip}$  rt')"
⟨proof⟩

lemma vD_invalidate_rt_strictly_fresher [simp]:
  assumes "dip  $\in$  vD(invalidate rt1 dests)"
  shows "(invalidate rt1 dests  $\sqsubseteq_{dip}$  rt2) = (rt1  $\sqsubseteq_{dip}$  rt2)"
⟨proof⟩

```

```

lemma rt_strictly_fresher_update_other [elim!]:
  "∧ dip ip rt r rt'. [ dip ≠ ip; rt ⊆dip rt' ] ⇒ update rt ip r ⊆dip rt'"
  ⟨proof⟩

lemma addpreRT_strictly_fresher [simp]:
  assumes "dip ∈ kD(rt)"
  shows "(the (addpreRT rt dip npre) ⊆ip rt2) = (rt ⊆ip rt2)"
  ⟨proof⟩

lemma lt_sqn_imp_update_strictly_fresher:
  assumes "dip ∈ vD (rt2 nhip)"
  and *: "osn < sqn (rt2 nhip) dip"
  and **: "rt ≠ update rt dip (osn, kno, val, hops, nhip, {})"
  shows "update rt dip (osn, kno, val, hops, nhip, {}) ⊆dip rt2 nhip"
  ⟨proof⟩

lemma dhops_le_hops_imp_update_strictly_fresher:
  assumes "dip ∈ vD(rt2 nhip)"
  and sqn: "sqn (rt2 nhip) dip = osn"
  and hop: "the (dhops (rt2 nhip) dip) ≤ hops"
  and **: "rt ≠ update rt dip (osn, kno, val, Suc hops, nhip, {})"
  shows "update rt dip (osn, kno, val, Suc hops, nhip, {}) ⊆dip rt2 nhip"
  ⟨proof⟩

lemma nsqn_invalidate:
  assumes "dip ∈ kD(rt)"
  and "∀ ip ∈ dom(dests). ip ∈ vD(rt) ∧ the (dests ip) = inc (sqn rt ip)"
  shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
  ⟨proof⟩

end

```

## 1.6 Invariant proofs on individual processes

```

theory A_Seq_Invariants
imports "../.../AWN/Invariants" A_Aodv A_Aodv_Data A_Aodv_Predicates A_Fresher

begin

```

The proposition numbers are taken from the December 2013 version of the Fehnker et al technical report.

Proposition 7.2

```

lemma sequence_number_increases:
  "paodv i ⊨A onll ΓAODV (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
  ⟨proof⟩

```

```

lemma sequence_number_one_or_bigger:
  "paodv i ⊨ onl ΓAODV (λ(ξ, _). 1 ≤ sn ξ)"
  ⟨proof⟩

```

We can get rid of the onl/onll if desired...

```

lemma sequence_number_increases':
  "paodv i ⊨A (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
  ⟨proof⟩

```

```

lemma sequence_number_one_or_bigger':
  "paodv i ⊨ (λ(ξ, _). 1 ≤ sn ξ)"
  ⟨proof⟩

```

```

lemma sip_in_kD:
  "paodv i ⊨ onl ΓAODV (λ(ξ, 1). 1 ∈ ({PAodv-:7} ∪ {PAodv-:5} ∪ {PRrep-:0..PRrep-:1}
    ∪ {PRreq-:0..PRreq-:3}) → sip ξ ∈ kD (rt ξ))"
  ⟨proof⟩

```

**lemma rrep\_1\_update\_changes:**

"paadv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). (l = PRrep-1 \rightarrow$   
 $rt \xi \neq update (rt \xi) (dip \xi) (dsn \xi, kno, val, hops \xi + 1, sip \xi, \{\}))$ )"

*<proof>*

**lemma addpreRT\_partly\_welldefined:**

"paadv i  $\models$   
onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). (l \in \{PRreq-:16..PRreq-:18\} \cup \{PRrep-:2..PRrep-:6\} \rightarrow dip \xi \in kD (rt \xi))$   
 $\wedge (l \in \{PRreq-:3..PRreq-:17\} \rightarrow oip \xi \in kD (rt \xi))$ )"

*<proof>*

Proposition 7.38

**lemma includes\_nhip:**

"paadv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). \forall dip \in kD(rt \xi). the (nhop (rt \xi) dip) \in kD(rt \xi)$ )"

*<proof>*

Proposition 7.22: needed in Proposition 7.4

**lemma addpreRT\_welldefined:**

"paadv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). (l \in \{PRreq-:16..PRreq-:18\} \rightarrow dip \xi \in kD (rt \xi)) \wedge$   
 $(l = PRreq-:17 \rightarrow oip \xi \in kD (rt \xi)) \wedge$   
 $(l = PRrep-:5 \rightarrow dip \xi \in kD (rt \xi)) \wedge$   
 $(l = PRrep-:6 \rightarrow (the (nhop (rt \xi) (dip \xi))) \in kD (rt \xi))$ )"

(is " $\_ \models$  onl  $\Gamma_{AODV}$  ?P")

*<proof>*

Proposition 7.4

**lemma known\_destinations\_increase:**

"paadv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), \_, (\xi', \_)). kD (rt \xi) \subseteq kD (rt \xi')$ )"

*<proof>*

Proposition 7.5

**lemma rreqs\_increase:**

"paadv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), \_, (\xi', \_)). rreqs \xi \subseteq rreqs \xi'$ )"

*<proof>*

**lemma dests\_bigger\_than\_sqn:**

"paadv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). l \in \{PAadv-:15..PAadv-:19\}$   
 $\cup \{PPkt-:7..PPkt-:11\}$   
 $\cup \{PRreq-:9..PRreq-:13\}$   
 $\cup \{PRreq-:21..PRreq-:25\}$   
 $\cup \{PRrep-:10..PRrep-:14\}$   
 $\cup \{PRerr-:1..PRerr-:5\}$   
 $\rightarrow (\forall ip \in dom(dests \xi). ip \in kD(rt \xi) \wedge sqn (rt \xi) ip \leq the (dests \xi ip))$ )"

*<proof>*

Proposition 7.6

**lemma sqns\_increase:**

"paadv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), \_, (\xi', \_)). \forall ip. sqn (rt \xi) ip \leq sqn (rt \xi') ip$ )"

*<proof>*

Proposition 7.7

**lemma ip\_constant:**

"paadv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_). ip \xi = i$ )"

*<proof>*

Proposition 7.8

**lemma sender\_ip\_valid':**

"paadv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), a, \_). anycast (\lambda m. not\_Pkt m \rightarrow msg\_sender m = ip \xi) a$ )"

*<proof>*

**lemma sender\_ip\_valid:**

"paadv i  $\models_A$  onll  $\Gamma_{AODV} (\lambda((\xi, \_), a, \_). \text{anycast } (\lambda m. \text{not\_Pkt } m \longrightarrow \text{msg\_sender } m = i) a)$ "  
 <proof>

**lemma received\_msg\_inv:**

"paadv i  $\models$  (recvmsg P  $\rightarrow$ ) onl  $\Gamma_{AODV} (\lambda(\xi, l). l \in \{\text{PAodv-:1}\} \longrightarrow P (\text{msg } \xi))$ "  
 <proof>

Proposition 7.9

**lemma sip\_not\_ip':**

"paadv i  $\models$  (recvmsg  $(\lambda m. \text{not\_Pkt } m \longrightarrow \text{msg\_sender } m \neq i) \rightarrow$ ) onl  $\Gamma_{AODV} (\lambda(\xi, \_). \text{sip } \xi \neq \text{ip } \xi)$ "  
 <proof>

**lemma sip\_not\_ip:**

"paadv i  $\models$  (recvmsg  $(\lambda m. \text{not\_Pkt } m \longrightarrow \text{msg\_sender } m \neq i) \rightarrow$ ) onl  $\Gamma_{AODV} (\lambda(\xi, \_). \text{sip } \xi \neq i)$ "  
 <proof>

Neither sip\_not\_ip' nor sip\_not\_ip is needed to show loop freedom.

Proposition 7.10

**lemma hop\_count\_positive:**

"paadv i  $\models$  onl  $\Gamma_{AODV} (\lambda(\xi, \_). \forall \text{ip} \in \text{kD} (\text{rt } \xi). \text{the } (\text{dhops } (\text{rt } \xi) \text{ ip}) \geq 1)$ "  
 <proof>

**lemma rreq\_dip\_in\_vD\_dip\_eq\_ip:**

"paadv i  $\models$  onl  $\Gamma_{AODV} (\lambda(\xi, l). (l \in \{\text{PRreq-:16..PRreq-:18}\} \longrightarrow \text{dip } \xi \in \text{vD}(\text{rt } \xi))$   
 $\wedge (l \in \{\text{PRreq-:5, PRreq-:6}\} \longrightarrow \text{dip } \xi = \text{ip } \xi)$   
 $\wedge (l \in \{\text{PRreq-:15..PRreq-:18}\} \longrightarrow \text{dip } \xi \neq \text{ip } \xi))$ "

<proof>

Proposition 7.11

**lemma anycast\_msg\_zhops:**

" $\bigwedge \text{rreqid dip dsn dsk oip osn sip}.$   
 paadv i  $\models_A$  onll  $\Gamma_{AODV} (\lambda(\_, a, \_). \text{anycast } \text{msg\_zhops } a)$ "  
 <proof>

**lemma hop\_count\_zero\_oip\_dip\_sip:**

"paadv i  $\models$  (recvmsg msg\_zhops  $\rightarrow$ ) onl  $\Gamma_{AODV} (\lambda(\xi, l).$   
 $(l \in \{\text{PAodv-:4..PAodv-:5}\} \cup \{\text{PRreq-:n/n. True}\} \longrightarrow$   
 $(\text{hops } \xi = 0 \longrightarrow \text{oip } \xi = \text{sip } \xi))$   
 $\wedge$   
 $((l \in \{\text{PAodv-:6..PAodv-:7}\} \cup \{\text{PRrep-:n/n. True}\} \longrightarrow$   
 $(\text{hops } \xi = 0 \longrightarrow \text{dip } \xi = \text{sip } \xi))))$ "

<proof>

**lemma osn\_rreq:**

"paadv i  $\models$  (recvmsg rreq\_rrep\_sn  $\rightarrow$ ) onl  $\Gamma_{AODV} (\lambda(\xi, l).$   
 $l \in \{\text{PAodv-:4, PAodv-:5}\} \cup \{\text{PRreq-:n/n. True}\} \longrightarrow 1 \leq \text{osn } \xi)$ "  
 <proof>

**lemma osn\_rreq':**

"paadv i  $\models$  (recvmsg  $(\lambda m. \text{rreq\_rrep\_sn } m \wedge \text{msg\_zhops } m) \rightarrow$ ) onl  $\Gamma_{AODV} (\lambda(\xi, l).$   
 $l \in \{\text{PAodv-:4, PAodv-:5}\} \cup \{\text{PRreq-:n/n. True}\} \longrightarrow 1 \leq \text{osn } \xi)$ "  
 <proof>

**lemma dsn\_rrep:**

"paadv i  $\models$  (recvmsg rreq\_rrep\_sn  $\rightarrow$ ) onl  $\Gamma_{AODV} (\lambda(\xi, l).$   
 $l \in \{\text{PAodv-:6, PAodv-:7}\} \cup \{\text{PRrep-:n/n. True}\} \longrightarrow 1 \leq \text{dsn } \xi)$ "  
 <proof>

**lemma dsn\_rrep':**

"paadv i  $\models$  (recvmsg  $(\lambda m. \text{rreq\_rrep\_sn } m \wedge \text{msg\_zhops } m) \rightarrow$ ) onl  $\Gamma_{AODV} (\lambda(\xi, l).$   
 $l \in \{\text{PAodv-:6, PAodv-:7}\} \cup \{\text{PRrep-:n/n. True}\} \longrightarrow 1 \leq \text{dsn } \xi)$ "  
 <proof>

**lemma hop\_count\_zero\_oip\_dip\_sip':**

```
"paadv i  $\models$  (recvmsg ( $\lambda m. rreq_rrep\_sn\ m \wedge msg\_zhops\ m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l).$ 
  ( $l \in \{PAadv-:4..PAadv-:5\} \cup \{PRreq-:n/n.\ True\}$   $\rightarrow$ 
    (hops  $\xi = 0 \rightarrow oip\ \xi = sip\ \xi$ ))
   $\wedge$ 
  (( $l \in \{PAadv-:6..PAadv-:7\} \cup \{PRrep-:n/n.\ True\}$   $\rightarrow$ 
    (hops  $\xi = 0 \rightarrow dip\ \xi = sip\ \xi$ ))))"
```

*<proof>*

Proposition 7.12

**lemma zero\_seq\_unk\_hops\_one':**

```
"paadv i  $\models$  (recvmsg ( $\lambda m. rreq_rrep\_sn\ m \wedge msg\_zhops\ m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_).$ 
   $\forall dip \in kD(rt\ \xi). (sqn\ (rt\ \xi)\ dip = 0 \rightarrow sqnf\ (rt\ \xi)\ dip = unk)$ 
   $\wedge (sqnf\ (rt\ \xi)\ dip = unk \rightarrow the\ (dhops\ (rt\ \xi)\ dip) = 1)$ 
   $\wedge (the\ (dhops\ (rt\ \xi)\ dip) = 1 \rightarrow the\ (nhop\ (rt\ \xi)\ dip) = dip))"$ 
```

*<proof>*

**lemma zero\_seq\_unk\_hops\_one:**

```
"paadv i  $\models$  (recvmsg ( $\lambda m. rreq_rrep\_sn\ m \wedge msg\_zhops\ m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_).$ 
   $\forall dip \in kD(rt\ \xi). (sqn\ (rt\ \xi)\ dip = 0 \rightarrow (sqnf\ (rt\ \xi)\ dip = unk$ 
     $\wedge the\ (dhops\ (rt\ \xi)\ dip) = 1$ 
     $\wedge the\ (nhop\ (rt\ \xi)\ dip) = dip))"$ 
```

*<proof>*

**lemma kD\_unk\_or\_atleast\_one:**

```
"paadv i  $\models$  (recvmsg rreq_rrep_sn  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l).$ 
   $\forall dip \in kD(rt\ \xi). \pi_3(the\ (rt\ \xi\ dip)) = unk \vee 1 \leq \pi_2(the\ (rt\ \xi\ dip))"$ 
```

*<proof>*

Proposition 7.13

**lemma rreq\_rrep\_sn\_any\_step\_invariant:**

```
"paadv i  $\models_A$  (recvmsg rreq_rrep_sn  $\rightarrow$ ) onll  $\Gamma_{AODV}$  ( $\lambda(\_, a, \_).$  anycast rreq_rrep_sn a)"
```

*<proof>*

Proposition 7.14

**lemma rreq\_rrep\_fresh\_any\_step\_invariant:**

```
"paadv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), a, \_).$  anycast (rreq_rrep_fresh (rt  $\xi$ )) a)"
```

*<proof>*

Proposition 7.15

**lemma rerr\_invalid\_any\_step\_invariant:**

```
"paadv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), a, \_).$  anycast (rerr_invalid (rt  $\xi$ )) a)"
```

*<proof>*

Proposition 7.16

Some well-definedness obligations are irrelevant for the Isabelle development:

1. In each routing table there is at most one entry for each destination: guaranteed by type.
2. In each store of queued data packets there is at most one data queue for each destination: guaranteed by structure.
3. Whenever a set of pairs  $(rip, rsn)$  is assigned to the variable  $dests$  of type  $ip \rightarrow sqn$ , or to the first argument of the function  $rerr$ , this set is a partial function, i.e., there is at most one entry  $(rip, rsn)$  for each destination  $rip$ : guaranteed by type.

**lemma dests\_vD\_inc\_sqn:**

```
"paadv i  $\models$ 
  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l).$  ( $l \in \{PAadv-:15, PPkt-:7, PRreq-:9, PRreq-:21, PRrep-:10\}$ 
     $\rightarrow (\forall ip \in dom(dests\ \xi). ip \in vD(rt\ \xi) \wedge the\ (dests\ \xi\ ip) = inc\ (sqn\ (rt\ \xi)\ ip))$ )
   $\wedge (l = PRerr-:1$ 
     $\rightarrow (\forall ip \in dom(dests\ \xi). ip \in vD(rt\ \xi) \wedge the\ (dests\ \xi\ ip) > sqn\ (rt\ \xi)\ ip))"$ 
```

*<proof>*

Proposition 7.27

lemma route\_tables\_fresher:

"paadv i  $\models_A$  (recvmmsg rreq\_rrep\_sn  $\rightarrow$ ) onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), \_, (\xi', \_))$ .  
 $\forall dip \in kD(rt \xi). rt \xi \sqsubseteq_{dip} rt \xi'$ )"

*<proof>*

end

## 1.7 The quality increases predicate

theory A\_Quality\_Increases

imports A\_Aodv\_Predicates A\_Fresher

begin

definition quality\_increases :: "state  $\Rightarrow$  state  $\Rightarrow$  bool"

where "quality\_increases  $\xi \xi'$   $\equiv$  ( $\forall dip \in kD(rt \xi). dip \in kD(rt \xi') \wedge rt \xi \sqsubseteq_{dip} rt \xi'$ )  
 $\wedge$  ( $\forall dip. sqn (rt \xi) dip \leq sqn (rt \xi') dip$ )"

lemma quality\_increasesI [intro!]:

assumes " $\bigwedge dip. dip \in kD(rt \xi) \implies dip \in kD(rt \xi')$ "  
and " $\bigwedge dip. \llbracket dip \in kD(rt \xi); dip \in kD(rt \xi') \rrbracket \implies rt \xi \sqsubseteq_{dip} rt \xi'$ "  
and " $\bigwedge dip. sqn (rt \xi) dip \leq sqn (rt \xi') dip$ "  
shows "quality\_increases  $\xi \xi'$ "

*<proof>*

lemma quality\_increasesE [elim]:

fixes dip  
assumes "quality\_increases  $\xi \xi'$ "  
and " $dip \in kD(rt \xi)$ "  
and " $\llbracket dip \in kD(rt \xi'); rt \xi \sqsubseteq_{dip} rt \xi'; sqn (rt \xi) dip \leq sqn (rt \xi') dip \rrbracket \implies R dip \xi \xi'$ "  
shows " $R dip \xi \xi'$ "

*<proof>*

lemma quality\_increases\_rt\_fresherD [dest]:

fixes ip  
assumes "quality\_increases  $\xi \xi'$ "  
and " $ip \in kD(rt \xi)$ "  
shows " $rt \xi \sqsubseteq_{ip} rt \xi'$ "

*<proof>*

lemma quality\_increases\_sqnE [elim]:

fixes dip  
assumes "quality\_increases  $\xi \xi'$ "  
and " $sqn (rt \xi) dip \leq sqn (rt \xi') dip \implies R dip \xi \xi'$ "  
shows " $R dip \xi \xi'$ "

*<proof>*

lemma quality\_increases\_refl [intro, simp]: "quality\_increases  $\xi \xi$ "

*<proof>*

lemma strictly\_fresher\_quality\_increases\_right [elim]:

fixes  $\sigma \sigma'$  dip  
assumes " $rt (\sigma i) \sqsubseteq_{dip} rt (\sigma nhip)$ "  
and qinc: "quality\_increases  $(\sigma nhip) (\sigma' nhip)$ "  
and " $dip \in kD(rt (\sigma nhip))$ "  
shows " $rt (\sigma i) \sqsubseteq_{dip} rt (\sigma' nhip)$ "

*<proof>*

lemma kD\_quality\_increases [elim]:

assumes " $i \in kD(rt \xi)$ "  
and "quality\_increases  $\xi \xi'$ "  
shows " $i \in kD(rt \xi')$ "



*<proof>*

```
lemma kD_nsqn_quality_increases [elim]:  
  assumes "i ∈ kD(rt ξ)"  
    and "quality_increases ξ ξ'"  
  shows "i ∈ kD(rt ξ') ∧ nsqn (rt ξ) i ≤ nsqn (rt ξ') i"  
<proof>
```

```
lemma nsqn_quality_increases [elim]:  
  assumes "i ∈ kD(rt ξ)"  
    and "quality_increases ξ ξ'"  
  shows "nsqn (rt ξ) i ≤ nsqn (rt ξ') i"  
<proof>
```

```
lemma kD_nsqn_quality_increases_trans [elim]:  
  assumes "i ∈ kD(rt ξ)"  
    and "s ≤ nsqn (rt ξ) i"  
    and "quality_increases ξ ξ'"  
  shows "i ∈ kD(rt ξ') ∧ s ≤ nsqn (rt ξ') i"  
<proof>
```

```
lemma nsqn_quality_increases_nsqn_lt_lt [elim]:  
  assumes "i ∈ kD(rt ξ)"  
    and "quality_increases ξ ξ'"  
    and "s < nsqn (rt ξ) i"  
  shows "s < nsqn (rt ξ') i"  
<proof>
```

```
lemma nsqn_quality_increases_dhops [elim]:  
  assumes "i ∈ kD(rt ξ)"  
    and "quality_increases ξ ξ'"  
    and "nsqn (rt ξ) i = nsqn (rt ξ') i"  
  shows "the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i)"  
<proof>
```

```
lemma nsqn_quality_increases_nsqn_eq_le [elim]:  
  assumes "i ∈ kD(rt ξ)"  
    and "quality_increases ξ ξ'"  
    and "s = nsqn (rt ξ) i"  
  shows "s < nsqn (rt ξ') i ∨ (s = nsqn (rt ξ') i ∧ the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i))"  
<proof>
```

```
lemma quality_increases_rreq_rrep_props [elim]:  
  fixes sn ip hops sip  
  assumes qinc: "quality_increases (σ sip) (σ' sip)"  
    and "1 ≤ sn"  
    and *: "ip ∈ kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip  
           ∧ (nsqn (rt (σ sip)) ip = sn  
             → (the (dhops (rt (σ sip)) ip) ≤ hops  
                ∨ the (flag (rt (σ sip)) ip) = inv))"  
  shows "ip ∈ kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip  
        ∧ (nsqn (rt (σ' sip)) ip = sn  
          → (the (dhops (rt (σ' sip)) ip) ≤ hops  
             ∨ the (flag (rt (σ' sip)) ip) = inv))"  
    (is "_ ∧ ?nsqnafter")  
<proof>
```

```
lemma quality_increases_rreq_rrep_props':  
  fixes sn ip hops sip  
  assumes "∀ j. quality_increases (σ j) (σ' j)"  
    and "1 ≤ sn"  
    and *: "ip ∈ kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip  
           ∧ (nsqn (rt (σ sip)) ip = sn  
             → (the (dhops (rt (σ sip)) ip) ≤ hops
```

$\vee \text{the } (\text{flag } (\text{rt } (\sigma \text{ sip})) \text{ ip}) = \text{inv})"$

shows " $\text{ip} \in \text{kD}(\text{rt } (\sigma' \text{ sip})) \wedge \text{sn} \leq \text{nsqn } (\text{rt } (\sigma' \text{ sip})) \text{ ip}$   
 $\wedge (\text{nsqn } (\text{rt } (\sigma' \text{ sip})) \text{ ip} = \text{sn})$   
 $\rightarrow (\text{the } (\text{dhops } (\text{rt } (\sigma' \text{ sip})) \text{ ip}) \leq \text{hops}$   
 $\vee \text{the } (\text{flag } (\text{rt } (\sigma' \text{ sip})) \text{ ip}) = \text{inv})"$

$\langle \text{proof} \rangle$

**lemma** *rteq\_quality\_increases*:

assumes " $\forall j. j \neq i \rightarrow \text{quality\_increases } (\sigma j) (\sigma' j)"$   
and " $\text{rt } (\sigma' i) = \text{rt } (\sigma i)"$   
shows " $\forall j. \text{quality\_increases } (\sigma j) (\sigma' j)"$

$\langle \text{proof} \rangle$

**definition** *msg\_fresh* :: " $(\text{ip} \Rightarrow \text{state}) \Rightarrow \text{msg} \Rightarrow \text{bool}"$

where "*msg\_fresh*  $\sigma m \equiv$

case *m* of *Rreq* *hops* *oipc* *osnc* *sipc*  $\Rightarrow \text{osnc} \geq 1 \wedge (\text{sipc} \neq \text{oipc} \rightarrow$   
 $\text{oipc} \in \text{kD}(\text{rt } (\sigma \text{ sipc})) \wedge \text{nsqn } (\text{rt } (\sigma \text{ sipc})) \text{ oipc} \geq \text{osnc}$   
 $\wedge (\text{nsqn } (\text{rt } (\sigma \text{ sipc})) \text{ oipc} = \text{osnc})$   
 $\rightarrow (\text{hops} \geq \text{the } (\text{dhops } (\text{rt } (\sigma \text{ sipc})) \text{ oipc})$   
 $\vee \text{the } (\text{flag } (\text{rt } (\sigma \text{ sipc})) \text{ oipc}) = \text{inv}))$   
| *Rrep* *hops* *dipc* *dsnc* *sipc*  $\Rightarrow \text{dsnc} \geq 1 \wedge (\text{sipc} \neq \text{dipc} \rightarrow$   
 $\text{dipc} \in \text{kD}(\text{rt } (\sigma \text{ sipc})) \wedge \text{nsqn } (\text{rt } (\sigma \text{ sipc})) \text{ dipc} \geq \text{dsnc}$   
 $\wedge (\text{nsqn } (\text{rt } (\sigma \text{ sipc})) \text{ dipc} = \text{dsnc})$   
 $\rightarrow (\text{hops} \geq \text{the } (\text{dhops } (\text{rt } (\sigma \text{ sipc})) \text{ dipc})$   
 $\vee \text{the } (\text{flag } (\text{rt } (\sigma \text{ sipc})) \text{ dipc}) = \text{inv}))$   
| *Rerr* *destsc* *sipc*  $\Rightarrow (\forall \text{ripc} \in \text{dom}(\text{destsc}). (\text{ripc} \in \text{kD}(\text{rt } (\sigma \text{ sipc}))$   
 $\wedge \text{the } (\text{destsc } \text{ripc}) - 1 \leq \text{nsqn } (\text{rt } (\sigma \text{ sipc})) \text{ ripc}))$   
|  $\_ \Rightarrow \text{True}"$

**lemma** *msg\_fresh [simp]*:

" $\wedge \text{hops } \text{dip } \text{dsn } \text{dsk } \text{oip } \text{osn } \text{sip}.$   
*msg\_fresh*  $\sigma (\text{Rreq } \text{hops } \text{dip } \text{dsn } \text{dsk } \text{oip } \text{osn } \text{sip}) =$   
 $(\text{osn} \geq 1 \wedge (\text{sip} \neq \text{oip} \rightarrow \text{oip} \in \text{kD}(\text{rt } (\sigma \text{ sip})))$   
 $\wedge \text{nsqn } (\text{rt } (\sigma \text{ sip})) \text{ oip} \geq \text{osn}$   
 $\wedge (\text{nsqn } (\text{rt } (\sigma \text{ sip})) \text{ oip} = \text{osn})$   
 $\rightarrow (\text{hops} \geq \text{the } (\text{dhops } (\text{rt } (\sigma \text{ sip})) \text{ oip})$   
 $\vee \text{the } (\text{flag } (\text{rt } (\sigma \text{ sip})) \text{ oip}) = \text{inv}))"$

" $\wedge \text{hops } \text{dip } \text{dsn } \text{oip } \text{sip}.$  *msg\_fresh*  $\sigma (\text{Rrep } \text{hops } \text{dip } \text{dsn } \text{oip } \text{sip}) =$   
 $(\text{dsn} \geq 1 \wedge (\text{sip} \neq \text{dip} \rightarrow \text{dip} \in \text{kD}(\text{rt } (\sigma \text{ sip})))$   
 $\wedge \text{nsqn } (\text{rt } (\sigma \text{ sip})) \text{ dip} \geq \text{dsn}$   
 $\wedge (\text{nsqn } (\text{rt } (\sigma \text{ sip})) \text{ dip} = \text{dsn})$   
 $\rightarrow (\text{hops} \geq \text{the } (\text{dhops } (\text{rt } (\sigma \text{ sip})) \text{ dip})$   
 $\vee \text{the } (\text{flag } (\text{rt } (\sigma \text{ sip})) \text{ dip}) = \text{inv}))"$

" $\wedge \text{dests } \text{sip}.$  *msg\_fresh*  $\sigma (\text{Rerr } \text{dests } \text{sip}) =$   
 $(\forall \text{ripc} \in \text{dom}(\text{dests}). (\text{ripc} \in \text{kD}(\text{rt } (\sigma \text{ sip})))$   
 $\wedge \text{the } (\text{dests } \text{ripc}) - 1 \leq \text{nsqn } (\text{rt } (\sigma \text{ sip})) \text{ ripc}))"$

" $\wedge \text{d } \text{dip}.$  *msg\_fresh*  $\sigma (\text{Newpkt } \text{d } \text{dip}) = \text{True}"$   
" $\wedge \text{d } \text{dip } \text{sip}.$  *msg\_fresh*  $\sigma (\text{Pkt } \text{d } \text{dip } \text{sip}) = \text{True}"$   
 $\langle \text{proof} \rangle$

**lemma** *msg\_fresh\_inc\_sn [simp, elim]*:

"*msg\_fresh*  $\sigma m \Rightarrow \text{rreq\_rrep\_sn } m"$   
 $\langle \text{proof} \rangle$

**lemma** *recv\_msg\_fresh\_inc\_sn [simp, elim]*:

"*orecvmsg* (*msg\_fresh*)  $\sigma m \Rightarrow \text{recvmsg } \text{rreq\_rrep\_sn } m"$   
 $\langle \text{proof} \rangle$

**lemma** *rreq\_nsqn\_is\_fresh [simp]*:

fixes  $\sigma \text{ msg } \text{hops } \text{dip } \text{dsn } \text{dsk } \text{oip } \text{osn } \text{sip}$   
assumes "*rreq\_rrep\_fresh* ( $\text{rt } (\sigma \text{ sip})) (\text{Rreq } \text{hops } \text{dip } \text{dsn } \text{dsk } \text{oip } \text{osn } \text{sip})"$   
and "*rreq\_rrep\_sn* ( $\text{Rreq } \text{hops } \text{dip } \text{dsn } \text{dsk } \text{oip } \text{osn } \text{sip})"$   
shows "*msg\_fresh*  $\sigma (\text{Rreq } \text{hops } \text{dip } \text{dsn } \text{dsk } \text{oip } \text{osn } \text{sip})"$   
(is "*msg\_fresh*  $\sigma \text{ ?msg}$ ")

*<proof>*

```
lemma rrep_nsqn_is_fresh [simp]:  
  fixes  $\sigma$  msg hops dip dsn oip sip  
  assumes "rreq_rrep_fresh (rt ( $\sigma$  sip)) (Rrep hops dip dsn oip sip)"  
    and "rreq_rrep_sn (Rrep hops dip dsn oip sip)"  
  shows "msg_fresh  $\sigma$  (Rrep hops dip dsn oip sip)"  
    (is "msg_fresh  $\sigma$  ?msg")  
<proof>
```

```
lemma rerr_nsqn_is_fresh [simp]:  
  fixes  $\sigma$  msg dests sip  
  assumes "rerr_invalid (rt ( $\sigma$  sip)) (Rerr dests sip)"  
  shows "msg_fresh  $\sigma$  (Rerr dests sip)"  
    (is "msg_fresh  $\sigma$  ?msg")  
<proof>
```

```
lemma quality_increases_msg_fresh [elim]:  
  assumes qinc: " $\forall j$ . quality_increases ( $\sigma$  j) ( $\sigma'$  j)"  
    and "msg_fresh  $\sigma$  m"  
  shows "msg_fresh  $\sigma'$  m"  
<proof>
```

end

## 1.8 The ‘open’ AODV model

```
theory A_OAodv  
imports A_Aodv "../..../AWN/OAWN_SOS_Labels" "../..../AWN/OAWN_Convert"  
begin
```

Definitions for stating and proving global network properties over individual processes.

```
definition  $\sigma_{AODV}'$  :: " $((ip \Rightarrow state) \times ((state, msg, pseqp, pseqp label) seqp)) set$ "  
where " $\sigma_{AODV}' \equiv \{(\lambda i. aodv\_init\ i, \Gamma_{AODV}\ PAodv)\}$ "
```

```
abbreviation opaodv  
  :: " $ip \Rightarrow ((ip \Rightarrow state) \times (state, msg, pseqp, pseqp label) seqp, msg seq\_action) automaton$ "  
where  
  " $opaodv\ i \equiv (\mid\ init = \sigma_{AODV}', trans = oseqp\_sos\ \Gamma_{AODV}\ i\ \mid)$ "
```

```
lemma initiali_aodv [intro!, simp]: "initiali i (init (opaodv i)) (init (paodv i))"  
<proof>
```

```
lemma oaodv_control_within [simp]: "control_within  $\Gamma_{AODV}$  (init (opaodv i))"  
<proof>
```

```
lemma  $\sigma_{AODV}'$ _labels [simp]: " $(\sigma, p) \in \sigma_{AODV}' \implies labels\ \Gamma_{AODV}\ p = \{PAodv-:0\}$ "  
<proof>
```

```
lemma oaodv_init_kD_empty [simp]:  
  " $(\sigma, p) \in \sigma_{AODV}' \implies kD\ (rt\ (\sigma\ i)) = \{\}$ "  
<proof>
```

```
lemma oaodv_init_vD_empty [simp]:  
  " $(\sigma, p) \in \sigma_{AODV}' \implies vD\ (rt\ (\sigma\ i)) = \{\}$ "  
<proof>
```

```
lemma oaodv_trans: "trans (opaodv i) = oseqp_sos  $\Gamma_{AODV}$  i"  
<proof>
```

declare

```
oseq_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros]  
oseq_step_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros]
```

end

## 1.9 Global invariant proofs over sequential processes

```

theory A_Global_Invariants
imports A_Seq_Invariants
        A_Aadv_Predicates
        A_Fresher
        A_Quality_Increases
        "../.../AWN/OAWN_Convert"
        A_OAadv

begin

lemma other_quality_increases [elim]:
  assumes "other quality_increases I  $\sigma$   $\sigma'$ "
  shows " $\forall j$ . quality_increases ( $\sigma$  j) ( $\sigma'$  j)"
  <proof>

lemma weaken_otherwith [elim]:
  fixes m
  assumes *: "otherwith P I (orecvmsg Q)  $\sigma$   $\sigma'$  a"
  and weakenP: " $\bigwedge \sigma$  m. P  $\sigma$  m  $\implies$  P'  $\sigma$  m"
  and weakenQ: " $\bigwedge \sigma$  m. Q  $\sigma$  m  $\implies$  Q'  $\sigma$  m"
  shows "otherwith P' I (orecvmsg Q')  $\sigma$   $\sigma'$  a"
  <proof>

lemma oreceived_msg_inv:
  assumes other: " $\bigwedge \sigma$   $\sigma'$  m.  $\llbracket$  P  $\sigma$  m; other Q {i}  $\sigma$   $\sigma'$   $\rrbracket \implies$  P  $\sigma'$  m"
  and local: " $\bigwedge \sigma$  m. P  $\sigma$  m  $\implies$  P ( $\sigma$ (i :=  $\sigma$  i(msg := m))) m"
  shows "opaadv i  $\models$  (otherwith Q {i} (orecvmsg P), other Q {i}  $\rightarrow$ )
        onl  $\Gamma_{AODV}$  ( $\lambda(\sigma, l)$ .  $l \in \{PAadv-:1\} \longrightarrow$  P  $\sigma$  (msg ( $\sigma$  i)))"
  <proof>

(Equivalent to) Proposition 7.27

lemma local_quality_increases:
  "paadv i  $\models_A$  (recvmsg rreq_rrep_sn  $\rightarrow$ ) onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_)$ ,  $\_$ , ( $\xi'$ ,  $\_$ )). quality_increases  $\xi$   $\xi'$ )"
  <proof>

lemmas olocal_quality_increases =
  open_seq_step_invariant [OF local_quality_increases initiali_aadv oaadv_trans aadv_trans,
    simplified seqll_onll_swap]

lemma oquality_increases:
  "opaadv i  $\models_A$  (otherwith quality_increases {i} (orecvmsg ( $\lambda$ _. rreq_rrep_sn)),
    other quality_increases {i}  $\rightarrow$ )
    onll  $\Gamma_{AODV}$  ( $\lambda((\sigma, \_)$ ,  $\_$ , ( $\sigma'$ ,  $\_$ )).  $\forall j$ . quality_increases ( $\sigma$  j) ( $\sigma'$  j))"
  (is " $\_ \models_A$  (?S,  $\_ \rightarrow$ )  $\_$ ")
  <proof>

lemma rreq_rrep_nsqn_fresh_any_step_invariant:
  "opaadv i  $\models_A$  (act (recvmsg rreq_rrep_sn), other A {i}  $\rightarrow$ )
    onll  $\Gamma_{AODV}$  ( $\lambda((\sigma, \_)$ , a,  $\_$ ). anycast (msg_fresh  $\sigma$ ) a)"
  <proof>

lemma oreceived_rreq_rrep_nsqn_fresh_inv:
  "opaadv i  $\models$  (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i}  $\rightarrow$ )
    onl  $\Gamma_{AODV}$  ( $\lambda(\sigma, l)$ .  $l \in \{PAadv-:1\} \longrightarrow$  msg_fresh  $\sigma$  (msg ( $\sigma$  i)))"
  <proof>

lemma oquality_increases_nsqn_fresh:
  "opaadv i  $\models_A$  (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i}  $\rightarrow$ )
    onll  $\Gamma_{AODV}$  ( $\lambda((\sigma, \_)$ ,  $\_$ , ( $\sigma'$ ,  $\_$ )).  $\forall j$ . quality_increases ( $\sigma$  j) ( $\sigma'$  j))"

```

$\langle proof \rangle$

**lemma oosn\_rreq:**

```
"opaadv i  $\models$  (otherwith quality_increases {i} (orecvmsg msg_fresh),
  other quality_increases {i}  $\rightarrow$ )
  onl  $\Gamma_{AODV}$  (seq1 i ( $\lambda(\xi, l). l \in \{PAadv:-4, PAadv:-5\} \cup \{PRreq:-n \mid n. True\} \rightarrow 1 \leq osn \xi$ ))"
 $\langle proof \rangle$ 
```

**lemma rreq\_sip:**

```
"opaadv i  $\models$  (otherwith quality_increases {i} (orecvmsg msg_fresh),
  other quality_increases {i}  $\rightarrow$ )
  onl  $\Gamma_{AODV}$  ( $\lambda(\sigma, l).$ 
    ( $l \in \{PAadv:-4, PAadv:-5, PRreq:-0, PRreq:-2\} \wedge sip(\sigma i) \neq oip(\sigma i)$ 
       $\rightarrow oip(\sigma i) \in kD(rt(\sigma(sip(\sigma i))))$ 
         $\wedge nsqn(rt(\sigma(sip(\sigma i))))(oip(\sigma i)) \geq osn(\sigma i)$ 
         $\wedge (nsqn(rt(\sigma(sip(\sigma i))))(oip(\sigma i)) = osn(\sigma i)$ 
           $\rightarrow (hops(\sigma i) \geq the(dhops(rt(\sigma(sip(\sigma i))))(oip(\sigma i)))$ 
             $\vee the(flag(rt(\sigma(sip(\sigma i))))(oip(\sigma i))) = inv$ ))"
  (is " $\_ \models (?S, ?U \rightarrow) \_$ "
 $\langle proof \rangle$ 
```

**lemma odsn\_rrep:**

```
"opaadv i  $\models$  (otherwith quality_increases {i} (orecvmsg msg_fresh),
  other quality_increases {i}  $\rightarrow$ )
  onl  $\Gamma_{AODV}$  (seq1 i ( $\lambda(\xi, l). l \in \{PAadv:-6, PAadv:-7\} \cup \{PRrep:-n \mid n. True\} \rightarrow 1 \leq dsn \xi$ ))"
 $\langle proof \rangle$ 
```

**lemma rrep\_sip:**

```
"opaadv i  $\models$  (otherwith quality_increases {i} (orecvmsg msg_fresh),
  other quality_increases {i}  $\rightarrow$ )
  onl  $\Gamma_{AODV}$  ( $\lambda(\sigma, l).$ 
    ( $l \in \{PAadv:-6, PAadv:-7, PRrep:-0, PRrep:-1\} \wedge sip(\sigma i) \neq dip(\sigma i)$ 
       $\rightarrow dip(\sigma i) \in kD(rt(\sigma(sip(\sigma i))))$ 
         $\wedge nsqn(rt(\sigma(sip(\sigma i))))(dip(\sigma i)) \geq dsn(\sigma i)$ 
         $\wedge (nsqn(rt(\sigma(sip(\sigma i))))(dip(\sigma i)) = dsn(\sigma i)$ 
           $\rightarrow (hops(\sigma i) \geq the(dhops(rt(\sigma(sip(\sigma i))))(dip(\sigma i)))$ 
             $\vee the(flag(rt(\sigma(sip(\sigma i))))(dip(\sigma i))) = inv$ ))"
  (is " $\_ \models (?S, ?U \rightarrow) \_$ "
 $\langle proof \rangle$ 
```

**lemma rerr\_sip:**

```
"opaadv i  $\models$  (otherwith quality_increases {i} (orecvmsg msg_fresh),
  other quality_increases {i}  $\rightarrow$ )
  onl  $\Gamma_{AODV}$  ( $\lambda(\sigma, l).$ 
    ( $l \in \{PAadv:-8, PAadv:-9, PRerr:-0, PRerr:-1\}$ 
       $\rightarrow (\forall ripc \in dom(dests(\sigma i)). ripc \in kD(rt(\sigma(sip(\sigma i)))) \wedge$ 
         $the(dests(\sigma i) ripc) - 1 \leq nsqn(rt(\sigma(sip(\sigma i))))(ripc)$ ))"
  (is " $\_ \models (?S, ?U \rightarrow) \_$ "
 $\langle proof \rangle$ 
```

**lemma prerr\_guard:** "paadv i  $\models$

```
  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). (l = PRerr:-1$ 
     $\rightarrow (\forall ip \in dom(dests \xi). ip \in vD(rt \xi)$ 
       $\wedge the(nhop(rt \xi) ip) = sip \xi$ 
       $\wedge sqn(rt \xi) ip < the(dests \xi ip))))$ "
```

$\langle proof \rangle$

**lemmas oadpreRT\_welldefined =**

```
  open_seq_invariant [OF addpreRT_welldefined initiali_aadv oaadv_trans aadv_trans,
    simplified seq1_onl_swap,
    THEN oinvariant_anyact]
```

**lemmas odests\_vD\_inc\_sqn =**

```
  open_seq_invariant [OF dests_vD_inc_sqn initiali_aadv oaadv_trans aadv_trans,
```

```
simplified seq1_onl_swap,
THEN oinvariant_anyact]
```

```
lemmas oprerr_guard =
  open_seq_invariant [OF prerr_guard initiali_aodv oaodv_trans aodv_trans,
    simplified seq1_onl_swap,
    THEN oinvariant_anyact]
```

Proposition 7.28

```
lemma seq_compare_next_hop':
  "opaodv i  $\models$  (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i}  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\sigma, \_)$ .
     $\forall$ dip. let nhip = the (nhop (rt ( $\sigma$  i)) dip)
      in dip  $\in$  kD(rt ( $\sigma$  i))  $\wedge$  nhip  $\neq$  dip  $\rightarrow$ 
        dip  $\in$  kD(rt ( $\sigma$  nhip))  $\wedge$  nsqn (rt ( $\sigma$  i)) dip  $\leq$  nsqn (rt ( $\sigma$  nhip)) dip)"
  (is "_  $\models$  (?S, ?U  $\rightarrow$ ) _")
  <proof>
```

Proposition 7.30

```
lemmas okD_unk_or_atleast_one =
  open_seq_invariant [OF kD_unk_or_atleast_one initiali_aodv,
    simplified seq1_onl_swap]
```

```
lemmas ozero_seq_unk_hops_one =
  open_seq_invariant [OF zero_seq_unk_hops_one initiali_aodv,
    simplified seq1_onl_swap]
```

```
lemma oreachable_fresh_okD_unk_or_atleast_one:
  fixes dip
  assumes "( $\sigma, p$ )  $\in$  oreachable (opaodv i)
    (otherwith (op=) {i} (orecvmsg ( $\lambda\sigma$  m. msg_fresh  $\sigma$  m
       $\wedge$  msg_zhops m)))
    (other quality_increases {i})"
  and "dip  $\in$  kD(rt ( $\sigma$  i))"
  shows " $\pi_3$ (the (rt ( $\sigma$  i) dip)) = unk  $\vee$  1  $\leq$   $\pi_2$ (the (rt ( $\sigma$  i) dip))"
  (is "?P dip")
  <proof>
```

```
lemma oreachable_fresh_ozero_seq_unk_hops_one:
  fixes dip
  assumes "( $\sigma, p$ )  $\in$  oreachable (opaodv i)
    (otherwith (op=) {i} (orecvmsg ( $\lambda\sigma$  m. msg_fresh  $\sigma$  m
       $\wedge$  msg_zhops m)))
    (other quality_increases {i})"
  and "dip  $\in$  kD(rt ( $\sigma$  i))"
  shows "sqn (rt ( $\sigma$  i)) dip = 0  $\rightarrow$ 
    sqnf (rt ( $\sigma$  i)) dip = unk
     $\wedge$  the (dhops (rt ( $\sigma$  i)) dip) = 1
     $\wedge$  the (nhop (rt ( $\sigma$  i)) dip) = dip"
  (is "?P dip")
  <proof>
```

```
lemma seq_nhop_quality_increases':
  shows "opaodv i  $\models$  (otherwith (op=) {i}
    (orecvmsg ( $\lambda\sigma$  m. msg_fresh  $\sigma$  m  $\wedge$  msg_zhops m)),
    other quality_increases {i}  $\rightarrow$ )
    onl  $\Gamma_{AODV}$  ( $\lambda(\sigma, \_)$ .  $\forall$ dip. let nhip = the (nhop (rt ( $\sigma$  i)) dip)
      in dip  $\in$  vD (rt ( $\sigma$  i))  $\cap$  vD (rt ( $\sigma$  nhip))
         $\wedge$  nhip  $\neq$  dip
         $\rightarrow$  (rt ( $\sigma$  i))  $\sqsubset_{dip}$  (rt ( $\sigma$  nhip)))"
  (is "_  $\models$  (?S i, _  $\rightarrow$ ) _")
  <proof>
```

```
lemma seq_compare_next_hop:
```

```

fixes w
shows "opaadv i  $\models$  (otherwith (op=) {i} (orecvmsg msg_fresh),
  other quality_increases {i}  $\rightarrow$ )
  global ( $\lambda\sigma. \forall dip. \text{let } nhip = \text{the } (nhop \text{ (rt } (\sigma \text{ i})) \text{ dip})$ 
    in  $dip \in kD(\text{rt } (\sigma \text{ i})) \wedge nhip \neq dip \rightarrow$ 
       $dip \in kD(\text{rt } (\sigma \text{ nhip}))$ 
       $\wedge \text{nsqn } (\text{rt } (\sigma \text{ i})) \text{ dip} \leq \text{nsqn } (\text{rt } (\sigma \text{ nhip})) \text{ dip}$ )"
<proof>

```

```

lemma seq_nhop_quality_increases:
shows "opaadv i  $\models$  (otherwith (op=) {i}
  (orecvmsg ( $\lambda\sigma m. \text{msg\_fresh } \sigma m \wedge \text{msg\_zhops } m$ ),
  other quality_increases {i}  $\rightarrow$ )
  global ( $\lambda\sigma. \forall dip. \text{let } nhip = \text{the } (nhop \text{ (rt } (\sigma \text{ i})) \text{ dip})$ 
    in  $dip \in vD \text{ (rt } (\sigma \text{ i})) \cap vD \text{ (rt } (\sigma \text{ nhip})) \wedge nhip \neq dip$ 
       $\rightarrow (\text{rt } (\sigma \text{ i})) \sqsubset_{dip} (\text{rt } (\sigma \text{ nhip}))$ )"
<proof>

```

end

## 1.10 Routing graphs and loop freedom

```

theory A_Loop_Freedom
imports A_Aodv_Predicates A_Fresher
begin

```

Define the central theorem that relates an invariant over network states to the absence of loops in the associate routing graph.

**definition**

```

rt_graph :: "(ip  $\Rightarrow$  state)  $\Rightarrow$  ip  $\Rightarrow$  ip rel"
where
  "rt_graph  $\sigma = (\lambda dip.
    \{(ip, ip') \mid ip \text{ ip}' \text{ dsn dsk hops pre.}
      ip \neq dip \wedge \text{rt } (\sigma \text{ ip}) \text{ dip} = \text{Some } (dsn, dsk, val, hops, ip', pre)\})"$ 

```

Given the state of a network  $\sigma$ , a routing graph for a given destination  $dip$  abstracts the details of routing tables into nodes (ip addresses) and vertices (valid routes between ip addresses).

```

lemma rt_graphE [elim]:
  fixes n dip ip ip'
  assumes "(ip, ip')  $\in$  rt_graph  $\sigma$  dip"
  shows "ip  $\neq$  dip  $\wedge$  ( $\exists r. \text{rt } (\sigma \text{ ip}) = r$ 
     $\wedge$  ( $\exists dsn \text{ dsk hops pre. } r \text{ dip} = \text{Some } (dsn, dsk, val, hops, ip', pre)$ ))"
<proof>

```

```

lemma rt_graph_vD [dest]:
  " $\bigwedge ip \text{ ip}' \sigma \text{ dip. } (ip, ip') \in \text{rt\_graph } \sigma \text{ dip} \implies dip \in vD(\text{rt } (\sigma \text{ ip}))"$ 
<proof>

```

```

lemma rt_graph_vD_trans [dest]:
  " $\bigwedge ip \text{ ip}' \sigma \text{ dip. } (ip, ip') \in (\text{rt\_graph } \sigma \text{ dip})^+ \implies dip \in vD(\text{rt } (\sigma \text{ ip}))"$ 
<proof>

```

```

lemma rt_graph_not_dip [dest]:
  " $\bigwedge ip \text{ ip}' \sigma \text{ dip. } (ip, ip') \in \text{rt\_graph } \sigma \text{ dip} \implies ip \neq dip"$ 
<proof>

```

```

lemma rt_graph_not_dip_trans [dest]:
  " $\bigwedge ip \text{ ip}' \sigma \text{ dip. } (ip, ip') \in (\text{rt\_graph } \sigma \text{ dip})^+ \implies ip \neq dip"$ 
<proof>

```

NB: the property below cannot be lifted to the transitive closure

```

lemma rt_graph_nhip_is_nhop [dest]:
  " $\bigwedge ip \text{ ip}' \sigma \text{ dip. } (ip, ip') \in \text{rt\_graph } \sigma \text{ dip} \implies ip' = \text{the } (nhop \text{ (rt } (\sigma \text{ ip})) \text{ dip})"$ 

```

*<proof>*

**theorem** *inv\_to\_loop\_freedom*:

```
assumes "∀ i dip. let nhip = the (nhop (rt (σ i)) dip)
          in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
          → (rt (σ i)) ⊆dip (rt (σ nhip))"
shows "∀ dip. irrefl ((rt_graph σ dip)+)"
```

*<proof>*

end

## 1.11 Lift and transfer invariants to show loop freedom

**theory** *A\_Aodv\_Loop\_Freedom*

```
imports "../..../AWN/OClosed_Transfer" "../..../AWN/Qmsg_Lifting" A_Global_Invariants A_Loop_Freedom
begin
```

lift to parallel processes with queues

**lemma** *par\_step\_no\_change\_on\_send\_or\_receive*:

```
fixes σ s a σ' s'
assumes "((σ, s), a, (σ', s')) ∈ oparp_sos i (oseqp_sos ΓAODV i) (seqp_sos ΓQMSG)"
and "a ≠ τ"
shows "σ' i = σ i"
```

*<proof>*

**lemma** *par\_nhop\_quality\_increases*:

```
shows "opaodv i ⟨⟨i qmsg ⊢ (otherwith (op=) {i} (orecvmsg (λσ m.
                    msg_fresh σ m ∧ msg_zhops m)),
                    other quality_increases {i} →)
        global (λσ. ∀ dip. let nhip = the (nhop (rt (σ i)) dip)
                    in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                    → (rt (σ i)) ⊆dip (rt (σ nhip)))⟩"
```

*<proof>*

**lemma** *par\_rreq\_rrep\_sn\_quality\_increases*:

```
"opaodv i ⟨⟨i qmsg ⊢A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
        globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))⟩"
```

*<proof>*

**lemma** *par\_rreq\_rrep\_nsqn\_fresh\_any\_step*:

```
shows "opaodv i ⟨⟨i qmsg ⊢A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ,
                    other (λ_ _. True) {i} →)
        globala (λ(σ, a, σ'). anycast (msg_fresh σ) a)⟩"
```

*<proof>*

**lemma** *par\_anycast\_msg\_zhops*:

```
shows "opaodv i ⟨⟨i qmsg ⊢A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
        globala (λ(_, a, _). anycast msg_zhops a)⟩"
```

*<proof>*

### 1.11.1 Lift to nodes

**lemma** *node\_step\_no\_change\_on\_send\_or\_receive*:

```
assumes "((σ, NodeS i P R), a, (σ', NodeS i' P' R')) ∈ onode_sos
          (oparp_sos i (oseqp_sos ΓAODV i) (seqp_sos ΓQMSG))"
and "a ≠ τ"
shows "σ' i = σ i"
```

*<proof>*

**lemma** *node\_nhop\_quality\_increases*:

```
shows "⟨ i : opaodv i ⟨⟨i qmsg : R ⟩o ⊢
        (otherwith (op=) {i}
        (oarrivmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
```



```

    other quality_increases {i}
  →) global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
    in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
    → (rt (σ i)) ⊆dip (rt (σ nhip)))"

```

⟨proof⟩

**lemma node\_quality\_increases:**

```

"⟨ i : opaadv i ⟨⟨i qmsg : R ⟩o ⟩A (λσ _ . oarrivemsg (λ_ . rreq_rrep_sn) σ,
  other (λ_ _ . True) {i} →)
  globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"

```

⟨proof⟩

**lemma node\_rreq\_rrep\_nsqn\_fresh\_any\_step:**

```

shows "⟨ i : opaadv i ⟨⟨i qmsg : R ⟩o ⟩A
  (λσ _ . oarrivemsg (λ_ . rreq_rrep_sn) σ, other (λ_ _ . True) {i} →)
  globala (λ(σ, a, σ'). castmsg (msg_fresh σ) a)"

```

⟨proof⟩

**lemma node\_anycast\_msg\_zhops:**

```

shows "⟨ i : opaadv i ⟨⟨i qmsg : R ⟩o ⟩A
  (λσ _ . oarrivemsg (λ_ . rreq_rrep_sn) σ, other (λ_ _ . True) {i} →)
  globala (λ(_, a, _). castmsg msg_zhops a)"

```

⟨proof⟩

**lemma node\_silent\_change\_only:**

```

shows "⟨ i : opaadv i ⟨⟨i qmsg : Ri ⟩o ⟩A (λσ _ . oarrivemsg (λ_ _ . True) σ,
  other (λ_ _ . True) {i} →)
  globala (λ(σ, a, σ'). a ≠ τ → σ' i = σ i)"

```

⟨proof⟩

### 1.11.2 Lift to partial networks

**lemma arrive\_rreq\_rrep\_nsqn\_fresh\_inc\_sn [simp]:**

```

assumes "oarrivemsg (λσ m. msg_fresh σ m ∧ P σ m) σ m"
shows "oarrivemsg (λ_ . rreq_rrep_sn) σ m"

```

⟨proof⟩

**lemma opnet\_nhop\_quality\_increases:**

```

shows "opnet (λi. opaadv i ⟨⟨i qmsg ⟩ p ⟩
  (otherwith (op=) (net_tree_ips p)
    (oarrivemsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
  other quality_increases (net_tree_ips p) →)
  global (λσ. ∀i∈net_tree_ips p. ∀dip.
    let nhip = the (nhop (rt (σ i)) dip)
    in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
    → (rt (σ i)) ⊆dip (rt (σ nhip)))"

```

⟨proof⟩

### 1.11.3 Lift to closed networks

**lemma onet\_nhop\_quality\_increases:**

```

shows "oclosed (opnet (λi. opaadv i ⟨⟨i qmsg ⟩ p ⟩
  ⟩ (λ_ _ _ . True, other quality_increases (net_tree_ips p) →)
  global (λσ. ∀i∈net_tree_ips p. ∀dip.
    let nhip = the (nhop (rt (σ i)) dip)
    in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
    → (rt (σ i)) ⊆dip (rt (σ nhip)))"

```

(is "\_ ⟩ (=, ?U →) ?inv")

⟨proof⟩

### 1.11.4 Transfer into the standard model

**interpretation aadv\_openproc: openproc paadv opaadv id**

```

rewrites "aadv_openproc.initmissing = initmissing"

```

⟨proof⟩

```

interpretation aadv_openproc_par_qmsg: openproc_parq paadv opaadv id qmsg
  rewrites "aadv_openproc_par_qmsg.netglobal = netglobal"
  and "aadv_openproc_par_qmsg.initmissing = initmissing"
  ⟨proof⟩

```

**lemma** net\_nhop\_quality\_increases:

```

  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paadv i ⟨⟨ qmsg⟩ n) ≡ netglobal
    (λσ. ∀i dip. let nhip = the (nhop (rt (σ i)) dip)
      in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
      → (rt (σ i)) ⊆dip (rt (σ nhip))))"
  (is "_ ≡ netglobal (λσ. ∀i. ?inv σ i)")
  ⟨proof⟩

```

### 1.11.5 Loop freedom of AODV

**theorem** aadv\_loop\_freedom:

```

  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paadv i ⟨⟨ qmsg⟩ n) ≡ netglobal (λσ. ∀dip. irrefl ((rt_graph σ dip)+))"
  ⟨proof⟩

```

**end**

# Chapter 2

## Variant B: Forwarding the Route Reply

Explanation [4, §10.2]: In AODV's route discovery process, a RREP message from the destination node is unicast back along a route towards the originator of the RREQ message. Every intermediate node on the selected route will process the RREP message and, in most cases, forward it towards the originator node. However, there is a possibility that the RREP message is discarded at an intermediate node, which results in the originator node not receiving a reply. The discarding of the RREP message is due to the RFC specification of AODV [6] stating that an intermediate node only forwards the RREP message if it is not the originator node and it has created or updated a routing table entry to the destination node described in the RREP message. The latter requirement means that if a valid routing table entry to the destination node already exists, and is not updated when processing the RREP message, then the intermediate node will not forward the message. A solution to this problem is to require intermediate nodes to forward all RREP messages that they receive.

### 2.1 Predicates and functions used in the AODV model

```
theory B_Aodv_Data
imports B_FwdrrEPS
begin
```

#### 2.1.1 Sequence Numbers

Sequence numbers approximate the relative freshness of routing information.

```
definition inc :: "sqn  $\Rightarrow$  sqn"
  where "inc sn  $\equiv$  if sn = 0 then sn else sn + 1"
```

```
lemma less_than_inc [simp]: "x  $\leq$  inc x"
  <proof>
```

```
lemma inc_minus_suc_0 [simp]:
  "inc x - Suc 0 = x"
  <proof>
```

```
lemma inc_never_one' [simp, intro]: "inc x  $\neq$  Suc 0"
  <proof>
```

```
lemma inc_never_one [simp, intro]: "inc x  $\neq$  1"
  <proof>
```

#### 2.1.2 Modelling Routes

A route is a 6-tuple,  $(dsn, dsk, flag, hops, nhop, pre)$  where  $dsn$  is the 'destination sequence number',  $dsk$  is the 'destination-sequence-number status',  $flag$  is the route status,  $hops$  is the number of hops to the destination,  $nhop$  is the next hop toward the destination, and  $pre$  is the set of 'precursor nodes'—those interested in hearing about changes to the route.

```
type_synonym r = "sqn  $\times$  k  $\times$  f  $\times$  nat  $\times$  ip  $\times$  ip set"
```

```
definition proj2 :: "r  $\Rightarrow$  sqn" (" $\pi_2$ ")
```

```

where "π2 ≡ λ(dsn, _, _, _, _). dsn"

definition proj3 :: "r ⇒ k" ("π3")
  where "π3 ≡ λ(_, dsk, _, _, _). dsk"

definition proj4 :: "r ⇒ f" ("π4")
  where "π4 ≡ λ(_, _, flag, _, _). flag"

definition proj5 :: "r ⇒ nat" ("π5")
  where "π5 ≡ λ(_, _, _, hops, _, _). hops"

definition proj6 :: "r ⇒ ip" ("π6")
  where "π6 ≡ λ(_, _, _, _, nhip, _). nhip"

definition proj7 :: "r ⇒ ip set" ("π7")
  where "π7 ≡ λ(_, _, _, _, _, pre). pre"

lemma projs [simp]:
  "π2(dsn, dsk, flag, hops, nhip, pre) = dsn"
  "π3(dsn, dsk, flag, hops, nhip, pre) = dsk"
  "π4(dsn, dsk, flag, hops, nhip, pre) = flag"
  "π5(dsn, dsk, flag, hops, nhip, pre) = hops"
  "π6(dsn, dsk, flag, hops, nhip, pre) = nhip"
  "π7(dsn, dsk, flag, hops, nhip, pre) = pre"
  ⟨proof⟩

lemma proj3_pred [intro]: "[[ P kno; P unk ] ⇒ P (π3 x)]"
  ⟨proof⟩

lemma proj4_pred [intro]: "[[ P val; P inv ] ⇒ P (π4 x)]"
  ⟨proof⟩

lemma proj6_pair_snd [simp]:
  fixes dsn' r
  shows "π6(dsn', snd (r)) = π6(r)"
  ⟨proof⟩

```

### 2.1.3 Routing Tables

Routing tables map ip addresses to route entries.

```

type_synonym rt = "ip → r"

syntax
  "_Sigma_route" :: "rt ⇒ ip → r" ("σroute'(_, _)'")

translations
  "σroute(rt, dip)" => "rt dip"

definition sqn :: "rt ⇒ ip ⇒ sqn"
  where "sqn rt dip ≡ case σroute(rt, dip) of Some r ⇒ π2(r) | None ⇒ 0"

definition sqnf :: "rt ⇒ ip ⇒ k"
  where "sqnf rt dip ≡ case σroute(rt, dip) of Some r ⇒ π3(r) | None ⇒ unk"

abbreviation flag :: "rt ⇒ ip → f"
  where "flag rt dip ≡ map_option π4 (σroute(rt, dip))"

abbreviation dhops :: "rt ⇒ ip → nat"
  where "dhops rt dip ≡ map_option π5 (σroute(rt, dip))"

abbreviation nhip :: "rt ⇒ ip → ip"
  where "nhip rt dip ≡ map_option π6 (σroute(rt, dip))"

abbreviation precs :: "rt ⇒ ip → ip set"

```

```

where "precs rt dip  $\equiv$  map_option  $\pi_7$  ( $\sigma_{route}(rt, dip)$ )"

definition vD :: "rt  $\Rightarrow$  ip set"
  where "vD rt  $\equiv$  {dip. flag rt dip = Some val}"

definition iD :: "rt  $\Rightarrow$  ip set"
  where "iD rt  $\equiv$  {dip. flag rt dip = Some inv}"

definition kD :: "rt  $\Rightarrow$  ip set"
  where "kD rt  $\equiv$  {dip. rt dip  $\neq$  None}"

lemma kD_is_vD_and_iD: "kD rt = vD rt  $\cup$  iD rt"
  <proof>

lemma vD_iD_gives_kD [simp]:
  " $\bigwedge ip$  rt. ip  $\in$  vD rt  $\implies$  ip  $\in$  kD rt"
  " $\bigwedge ip$  rt. ip  $\in$  iD rt  $\implies$  ip  $\in$  kD rt"
  <proof>

lemma kD_Some [dest]:
  fixes dip rt
  assumes "dip  $\in$  kD rt"
  shows " $\exists$  dsn dsk flag hops nhip pre.
     $\sigma_{route}(rt, dip) = \text{Some } (dsn, dsk, flag, hops, nhip, pre)$ "
  <proof>

lemma kD_None [dest]:
  fixes dip rt
  assumes "dip  $\notin$  kD rt"
  shows " $\sigma_{route}(rt, dip) = \text{None}$ "
  <proof>

lemma vD_Some [dest]:
  fixes dip rt
  assumes "dip  $\in$  vD rt"
  shows " $\exists$  dsn dsk hops nhip pre.
     $\sigma_{route}(rt, dip) = \text{Some } (dsn, dsk, val, hops, nhip, pre)$ "
  <proof>

lemma vD_empty [simp]: "vD Map.empty = {}"
  <proof>

lemma iD_Some [dest]:
  fixes dip rt
  assumes "dip  $\in$  iD rt"
  shows " $\exists$  dsn dsk hops nhip pre.
     $\sigma_{route}(rt, dip) = \text{Some } (dsn, dsk, inv, hops, nhip, pre)$ "
  <proof>

lemma val_is_vD [elim]:
  fixes ip rt
  assumes "ip  $\in$  kD(rt)"
  and "the (flag rt ip) = val"
  shows "ip  $\in$  vD(rt)"
  <proof>

lemma inv_is_iD [elim]:
  fixes ip rt
  assumes "ip  $\in$  kD(rt)"
  and "the (flag rt ip) = inv"
  shows "ip  $\in$  iD(rt)"
  <proof>

lemma iD_flag_is_inv [elim, simp]:

```

```

    fixes ip rt
    assumes "ip∈iD(rt)"
    shows "the (flag rt ip) = inv"
  ⟨proof⟩

lemma kD_but_not_vD_is_iD [elim]:
  fixes ip rt
  assumes "ip∈kD(rt)"
    and "ip∉vD(rt)"
  shows "ip∈iD(rt)"
  ⟨proof⟩

lemma vD_or_iD [elim]:
  fixes ip rt
  assumes "ip∈kD(rt)"
    and "ip∈vD(rt) ⇒ P rt ip"
    and "ip∈iD(rt) ⇒ P rt ip"
  shows "P rt ip"
  ⟨proof⟩

lemma proj5_eq_dhops: "∧dip rt. dip∈kD(rt) ⇒ π5(the (rt dip)) = the (dhops rt dip)"
  ⟨proof⟩

lemma proj4_eq_flag: "∧dip rt. dip∈kD(rt) ⇒ π4(the (rt dip)) = the (flag rt dip)"
  ⟨proof⟩

lemma proj2_eq_sqn: "∧dip rt. dip∈kD(rt) ⇒ π2(the (rt dip)) = sqn rt dip"
  ⟨proof⟩

lemma kD_sqnf_is_proj3 [simp]:
  "∧ip rt. ip∈kD(rt) ⇒ sqnf rt ip = π3(the (rt ip))"
  ⟨proof⟩

lemma vD_flag_val [simp]:
  "∧dip rt. dip ∈ vD (rt) ⇒ the (flag rt dip) = val"
  ⟨proof⟩

lemma kD_update [simp]:
  "∧rt nip v. kD (rt(nip ↦ v)) = insert nip (kD rt)"
  ⟨proof⟩

lemma kD_empty [simp]: "kD Map.empty = {}"
  ⟨proof⟩

lemma ip_equal_or_known [elim]:
  fixes rt ip ip'
  assumes "ip = ip' ∨ ip∈kD(rt)"
    and "ip = ip' ⇒ P rt ip ip'"
    and "[[ ip ≠ ip'; ip∈kD(rt) ] ] ⇒ P rt ip ip'"
  shows "P rt ip ip'"
  ⟨proof⟩

```

## 2.1.4 Updating Routing Tables

Routing table entries are modified through explicit functions. The properties of these functions are important in invariant proofs.

### Updating Precursor Lists

```

definition addpre :: "r ⇒ ip set ⇒ r"
  where "addpre r npre ≡ let (dsn, dsk, flag, hops, nhip, pre) = r in
    (dsn, dsk, flag, hops, nhip, pre ∪ npre)"

```

```

lemma proj2_addpre:

```

```

fixes v pre
shows " $\pi_2(\text{addpre } v \text{ pre}) = \pi_2(v)$ "
<proof>

lemma proj3_addpre:
fixes v pre
shows " $\pi_3(\text{addpre } v \text{ pre}) = \pi_3(v)$ "
<proof>

lemma proj4_addpre:
fixes v pre
shows " $\pi_4(\text{addpre } v \text{ pre}) = \pi_4(v)$ "
<proof>

lemma proj5_addpre:
fixes v pre
shows " $\pi_5(\text{addpre } v \text{ pre}) = \pi_5(v)$ "
<proof>

lemma proj6_addpre:
fixes dsn dsk flag hops nhip pre npre
shows " $\pi_6(\text{addpre } v \text{ npre}) = \pi_6(v)$ "
<proof>

lemma proj7_addpre:
fixes dsn dsk flag hops nhip pre npre
shows " $\pi_7(\text{addpre } v \text{ npre}) = \pi_7(v) \cup \text{npre}$ "
<proof>

lemma addpre_empty: "addpre r {} = r"
<proof>

lemma addpre_r:
"addpre (dsn, dsk, fl, hops, nhip, pre) npre = (dsn, dsk, fl, hops, nhip, pre  $\cup$  npre)"
<proof>

lemmas addpre_simps [simp] = proj2_addpre proj3_addpre proj4_addpre proj5_addpre
proj6_addpre proj7_addpre addpre_empty addpre_r

definition addpreRT :: "rt  $\Rightarrow$  ip  $\Rightarrow$  ip set  $\rightarrow$  rt"
where "addpreRT rt dip npre  $\equiv$ 
map_option ( $\lambda s. \text{rt } (\text{dip } \mapsto \text{addpre } s \text{ npre})$ ) ( $\sigma_{\text{route}}(\text{rt}, \text{dip})$ )"

lemma snd_addpre [simp]:
" $\bigwedge \text{dsn dsn}' v \text{ pre}. (\text{dsn}, \text{snd}(\text{addpre } (\text{dsn}', v) \text{ pre})) = \text{addpre } (\text{dsn}, v) \text{ pre}$ "
<proof>

lemma proj2_addpreRT [simp]:
fixes ip rt ip' npre
assumes "ip  $\in$  kD rt"
and "ip'  $\in$  kD rt"
shows " $\pi_2(\text{the } (\text{the } (\text{addpreRT } \text{rt } \text{ip}' \text{ npre}) \text{ ip})) = \pi_2(\text{the } (\text{rt } \text{ip}))$ "
<proof>

lemma proj3_addpreRT [simp]:
fixes ip rt ip' npre
assumes "ip  $\in$  kD rt"
and "ip'  $\in$  kD rt"
shows " $\pi_3(\text{the } (\text{the } (\text{addpreRT } \text{rt } \text{ip}' \text{ npre}) \text{ ip})) = \pi_3(\text{the } (\text{rt } \text{ip}))$ "
<proof>

lemma proj5_addpreRT [simp]:
" $\bigwedge \text{rt dip ip npre}. \text{dip} \in \text{kD}(\text{rt}) \implies \pi_5(\text{the } (\text{the } (\text{addpreRT } \text{rt } \text{dip } \text{npre}) \text{ ip})) = \pi_5(\text{the } (\text{rt } \text{ip}))$ "
<proof>

```

```

lemma flag_addpreRT [simp]:
  fixes rt pre ip dip
  assumes "dip ∈ kD rt"
  shows "flag (the (addpreRT rt dip pre)) ip = flag rt ip"
  ⟨proof⟩

lemma kD_addpreRT [simp]:
  fixes rt dip npre
  assumes "dip ∈ kD rt"
  shows "kD (the (addpreRT rt dip npre)) = kD rt"
  ⟨proof⟩

lemma vD_addpreRT [simp]:
  fixes rt dip npre
  assumes "dip ∈ kD rt"
  shows "vD (the (addpreRT rt dip npre)) = vD rt"
  ⟨proof⟩

lemma iD_addpreRT [simp]:
  fixes rt dip npre
  assumes "dip ∈ kD rt"
  shows "iD (the (addpreRT rt dip npre)) = iD rt"
  ⟨proof⟩

lemma nhop_addpreRT [simp]:
  fixes rt pre ip dip
  assumes "dip ∈ kD rt"
  shows "nhop (the (addpreRT rt dip pre)) ip = nhop rt ip"
  ⟨proof⟩

lemma sqn_addpreRT [simp]:
  fixes rt pre ip dip
  assumes "dip ∈ kD rt"
  shows "sqn (the (addpreRT rt dip pre)) ip = sqn rt ip"
  ⟨proof⟩

lemma dhops_addpreRT [simp]:
  fixes rt pre ip dip
  assumes "dip ∈ kD rt"
  shows "dhops (the (addpreRT rt dip pre)) ip = dhops rt ip"
  ⟨proof⟩

lemma sqnf_addpreRT [simp]:
  "∧ip dip. ip ∈ kD(rt ξ) ⇒ sqnf (the (addpreRT (rt ξ) ip npre)) dip = sqnf (rt ξ) dip"
  ⟨proof⟩

Updating route entries

lemma in_kD_case [simp]:
  fixes dip rt
  assumes "dip ∈ kD(rt)"
  shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = es (the (rt dip))"
  ⟨proof⟩

lemma not_in_kD_case [simp]:
  fixes dip rt
  assumes "dip ∉ kD(rt)"
  shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = en"
  ⟨proof⟩

lemma rt_Some_sqn [dest]:
  fixes rt and ip dsn dsk flag hops nhip pre
  assumes "rt ip = Some (dsn, dsk, flag, hops, nhip, pre)"

```



```

  shows "sqn rt ip = dsn"
  ⟨proof⟩

lemma not_kD_sqn [simp]:
  fixes dip rt
  assumes "dip ∉ kD(rt)"
  shows "sqn rt dip = 0"
  ⟨proof⟩

definition update_arg_wf :: "r ⇒ bool"
where "update_arg_wf r ≡ π4(r) = val ∧
      (π2(r) = 0) = (π3(r) = unk) ∧
      (π3(r) = unk ⟶ π5(r) = 1)"

lemma update_arg_wf_gives_cases:
  "∧r. update_arg_wf r ⟹ (π2(r) = 0) = (π3(r) = unk)"
  ⟨proof⟩

lemma update_arg_wf_tuples [simp]:
  "∧nhip pre. update_arg_wf (0, unk, val, Suc 0, nhip, pre)"
  "∧n hops nhip pre. update_arg_wf (Suc n, kno, val, hops, nhip, pre)"
  ⟨proof⟩

lemma update_arg_wf_tuples' [elim]:
  "∧n hops nhip pre. Suc 0 ≤ n ⟹ update_arg_wf (n, kno, val, hops, nhip, pre)"
  ⟨proof⟩

lemma wf_r_cases [intro]:
  fixes P r
  assumes "update_arg_wf r"
  and c1: "∧nhip pre. P (0, unk, val, Suc 0, nhip, pre)"
  and c2: "∧dsn hops nhip pre. dsn > 0 ⟹ P (dsn, kno, val, hops, nhip, pre)"
  shows "P r"
  ⟨proof⟩

definition update :: "rt ⇒ ip ⇒ r ⇒ rt"
where
  "update rt ip r ≡
  case σroute(rt, ip) of
  None ⇒ rt (ip ↦ r)
  | Some s ⇒
    if π2(s) < π2(r) then rt (ip ↦ addpre r (π7(s)))
    else if π2(s) = π2(r) ∧ (π5(s) > π5(r) ∨ π4(s) = inv)
    then rt (ip ↦ addpre r (π7(s)))
    else if π3(r) = unk
    then rt (ip ↦ (π2(s), snd (addpre r (π7(s))))
    else rt (ip ↦ addpre s (π7(r)))"

lemma update_simps [simp]:
  fixes r s nrt nr nr' ns rt ip
  defines "s ≡ the σroute(rt, ip)"
  and "nr ≡ addpre r (π7(s))"
  and "nr' ≡ (π2(s), π3(nr), π4(nr), π5(nr), π6(nr), π7(nr))"
  and "ns ≡ addpre s (π7(r))"
  shows
  "[ip ∉ kD(rt)] ⟹ update rt ip r = rt (ip ↦ r)"
  "[ip ∈ kD(rt); sqn rt ip < π2(r)] ⟹ update rt ip r = rt (ip ↦ nr)"
  "[ip ∈ kD(rt); sqn rt ip = π2(r);
  the (dhops rt ip) > π5(r)] ⟹ update rt ip r = rt (ip ↦ nr)"
  "[ip ∈ kD(rt); sqn rt ip = π2(r);
  flag rt ip = Some inv] ⟹ update rt ip r = rt (ip ↦ nr)"
  "[ip ∈ kD(rt); π3(r) = unk; (π2(r) = 0) = (π3(r) = unk)] ⟹ update rt ip r = rt (ip ↦ nr)"
  "[ip ∈ kD(rt); sqn rt ip ≥ π2(r); π3(r) = kno;
  sqn rt ip = π2(r) ⟹ the (dhops rt ip) ≤ π5(r) ∧ the (flag rt ip) = val ]"

```

$\implies \text{update rt ip r} = \text{rt (ip } \mapsto \text{ns)}$ "

*<proof>*

**lemma update\_cases [elim]:**

assumes " $(\pi_2(r) = 0) = (\pi_3(r) = \text{unk})$ "

and c1: " $\llbracket \text{ip} \notin \text{kD}(\text{rt}) \rrbracket \implies P(\text{rt (ip } \mapsto \text{r)})$ "

and c2: " $\llbracket \text{ip} \in \text{kD}(\text{rt}); \text{sqn rt ip} < \pi_2(r) \rrbracket$   
 $\implies P(\text{rt (ip } \mapsto \text{addpre r } (\pi_7(\text{the } \sigma_{\text{route}}(\text{rt}, \text{ip}))))$ "

and c3: " $\llbracket \text{ip} \in \text{kD}(\text{rt}); \text{sqn rt ip} = \pi_2(r); \text{the (dhops rt ip)} > \pi_5(r) \rrbracket$   
 $\implies P(\text{rt (ip } \mapsto \text{addpre r } (\pi_7(\text{the } \sigma_{\text{route}}(\text{rt}, \text{ip}))))$ "

and c4: " $\llbracket \text{ip} \in \text{kD}(\text{rt}); \text{sqn rt ip} = \pi_2(r); \text{the (flag rt ip)} = \text{inv} \rrbracket$   
 $\implies P(\text{rt (ip } \mapsto \text{addpre r } (\pi_7(\text{the } \sigma_{\text{route}}(\text{rt}, \text{ip}))))$ "

and c5: " $\llbracket \text{ip} \in \text{kD}(\text{rt}); \pi_3(r) = \text{unk} \rrbracket$   
 $\implies P(\text{rt (ip } \mapsto (\pi_2(\text{the } \sigma_{\text{route}}(\text{rt}, \text{ip})), \pi_3(r),$   
 $\pi_4(r), \pi_5(r), \pi_6(r), \pi_7(\text{addpre r } (\pi_7(\text{the } \sigma_{\text{route}}(\text{rt}, \text{ip}))))))$ "

and c6: " $\llbracket \text{ip} \in \text{kD}(\text{rt}); \text{sqn rt ip} \geq \pi_2(r); \pi_3(r) = \text{kno};$   
 $\text{sqn rt ip} = \pi_2(r) \implies \text{the (dhops rt ip)} \leq \pi_5(r) \wedge \text{the (flag rt ip)} = \text{val} \rrbracket$   
 $\implies P(\text{rt (ip } \mapsto \text{addpre (the } \sigma_{\text{route}}(\text{rt}, \text{ip})) (\pi_7(r))))$ "

shows " $P(\text{update rt ip r})$ "

*<proof>*

**lemma update\_cases\_kD:**

assumes " $(\pi_2(r) = 0) = (\pi_3(r) = \text{unk})$ "

and " $\text{ip} \in \text{kD}(\text{rt})$ "

and c2: " $\llbracket \text{sqn rt ip} < \pi_2(r) \rrbracket \implies P(\text{rt (ip } \mapsto \text{addpre r } (\pi_7(\text{the } \sigma_{\text{route}}(\text{rt}, \text{ip}))))$ "

and c3: " $\llbracket \llbracket \text{sqn rt ip} = \pi_2(r); \text{the (dhops rt ip)} > \pi_5(r) \rrbracket$   
 $\implies P(\text{rt (ip } \mapsto \text{addpre r } (\pi_7(\text{the } \sigma_{\text{route}}(\text{rt}, \text{ip}))))$ "

and c4: " $\llbracket \llbracket \text{sqn rt ip} = \pi_2(r); \text{the (flag rt ip)} = \text{inv} \rrbracket$   
 $\implies P(\text{rt (ip } \mapsto \text{addpre r } (\pi_7(\text{the } \sigma_{\text{route}}(\text{rt}, \text{ip}))))$ "

and c5: " $\llbracket \pi_3(r) = \text{unk} \rrbracket \implies P(\text{rt (ip } \mapsto (\pi_2(\text{the } \sigma_{\text{route}}(\text{rt}, \text{ip})), \pi_3(r),$   
 $\pi_4(r), \pi_5(r), \pi_6(r),$   
 $\pi_7(\text{addpre r } (\pi_7(\text{the } \sigma_{\text{route}}(\text{rt}, \text{ip}))))))$ "

and c6: " $\llbracket \llbracket \text{sqn rt ip} \geq \pi_2(r); \pi_3(r) = \text{kno};$   
 $\text{sqn rt ip} = \pi_2(r) \implies \text{the (dhops rt ip)} \leq \pi_5(r) \wedge \text{the (flag rt ip)} = \text{val} \rrbracket$   
 $\implies P(\text{rt (ip } \mapsto \text{addpre (the } \sigma_{\text{route}}(\text{rt}, \text{ip})) (\pi_7(r))))$ "

shows " $P(\text{update rt ip r})$ "

*<proof>*

**lemma in\_kD\_after\_update [simp]:**

fixes rt nip dsn dsk flag hops nhip pre

shows " $\text{kD}(\text{update rt nip (dsn, dsk, flag, hops, nhip, pre)}) = \text{insert nip (kD rt)}$ "

*<proof>*

**lemma nhop\_of\_update [simp]:**

fixes rt dip dsn dsk flag hops nhip

assumes " $\text{rt} \neq \text{update rt dip (dsn, dsk, flag, hops, nhip, \{ })}$ "

shows " $\text{the (nhop (update rt dip (dsn, dsk, flag, hops, nhip, \{ }))) dip} = \text{nhip}$ "

*<proof>*

**lemma sqn\_if\_updated:**

fixes rip v rt ip

shows " $\text{sqn } (\lambda x. \text{if } x = \text{rip} \text{ then Some } v \text{ else rt } x) \text{ ip}$   
 $= (\text{if } \text{ip} = \text{rip} \text{ then } \pi_2(v) \text{ else sqn rt ip})$ "

*<proof>*

**lemma update\_sqn [simp]:**

fixes rt dip rip dsn dsk hops nhip pre

assumes " $(\text{dsn} = 0) = (\text{dsk} = \text{unk})$ "

shows " $\text{sqn rt dip} \leq \text{sqn (update rt rip (dsn, dsk, val, hops, nhip, pre)) dip}$ "

*<proof>*

**lemma sqn\_update\_bigger [simp]:**

fixes rt ip ip' dsn dsk flag hops nhip pre

```

assumes "1 ≤ hops"
  shows "sqn rt ip ≤ sqn (update rt ip' (dsn, dsk, flag, hops, nhip, pre)) ip"
⟨proof⟩

lemma dhops_update [intro]:
  fixes rt dsn dsk flag hops ip rip nhip pre
  assumes ex: "∀ ip ∈ kD rt. the (dhops rt ip) ≥ 1"
  and ip: "(ip = rip ∧ Suc 0 ≤ hops) ∨ (ip ≠ rip ∧ ip ∈ kD rt)"
  shows "Suc 0 ≤ the (dhops (update rt rip (dsn, dsk, flag, hops, nhip, pre)) ip)"
⟨proof⟩

lemma update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
  shows "(update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = rt ip"
⟨proof⟩

lemma nhop_update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
  shows "nhop (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = nhop rt ip"
⟨proof⟩

lemma dhops_update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
  shows "dhops (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = dhops rt ip"
⟨proof⟩

lemma sqn_update_same [simp]:
  "∧ rt ip dsn dsk flag hops nhip pre. sqn (rt(ip ↦ v)) ip = π2(v)"
⟨proof⟩

lemma dhops_update_changed [simp]:
  fixes rt dip osn hops nhip
  assumes "rt ≠ update rt dip (osn, kno, val, hops, nhip, {})"
  shows "the (dhops (update rt dip (osn, kno, val, hops, nhip, {})) dip) = hops"
⟨proof⟩

lemma nhop_update_unk_val [simp]:
  "∧ rt dip ip dsn hops npre.
  the (nhop (update rt dip (dsn, unk, val, hops, ip, npre)) dip) = ip"
⟨proof⟩

lemma nhop_update_changed [simp]:
  fixes rt dip dsn dsk flg hops sip
  assumes "update rt dip (dsn, dsk, flg, hops, sip, {}) ≠ rt"
  shows "the (nhop (update rt dip (dsn, dsk, flg, hops, sip, {})) dip) = sip"
⟨proof⟩

lemma update_rt_split_asm:
  "∧ rt ip dsn dsk flag hops sip.
  P (update rt ip (dsn, dsk, flag, hops, sip, {}))
  =
  (¬(rt = update rt ip (dsn, dsk, flag, hops, sip, {})) ∧ ¬P rt
  ∨ rt ≠ update rt ip (dsn, dsk, flag, hops, sip, {})
  ∧ ¬P (update rt ip (dsn, dsk, flag, hops, sip, {})))"
⟨proof⟩

lemma sqn_update [simp]: "∧ rt dip dsn flg hops sip.
rt ≠ update rt dip (dsn, kno, flg, hops, sip, {})
⇒ sqn (update rt dip (dsn, kno, flg, hops, sip, {})) dip = dsn"
⟨proof⟩

```

**lemma** *sqnf\_update [simp]*: " $\bigwedge$ rt dip dsn dsk flg hops sip.  
 rt  $\neq$  update rt dip (dsn, dsk, flg, hops, sip, { })  
 $\implies$  sqnf (update rt dip (dsn, dsk, flg, hops, sip, { })) dip = dsk"  
 <proof>

**lemma** *update\_kno\_dsn\_greater\_zero*:  
" $\bigwedge$ rt dip ip dsn hops npre.  $1 \leq$  dsn  $\implies$   $1 \leq$  (sqn (update rt dip (dsn, kno, val, hops, ip, npre)) dip)"  
 <proof>

**lemma** *proj3\_update [simp]*: " $\bigwedge$ rt dip dsn dsk flg hops sip.  
 rt  $\neq$  update rt dip (dsn, dsk, flg, hops, sip, { })  
 $\implies$   $\pi_3$ (the (update rt dip (dsn, dsk, flg, hops, sip, { }) dip)) = dsk"  
 <proof>

**lemma** *nhop\_update\_changed\_kno\_val [simp]*: " $\bigwedge$ rt ip dsn dsk hops nhip.  
 rt  $\neq$  update rt ip (dsn, kno, val, hops, nhip, { })  
 $\implies$  the (nhop (update rt ip (dsn, kno, val, hops, nhip, { })) ip) = nhip"  
 <proof>

**lemma** *flag\_update [simp]*: " $\bigwedge$ rt dip dsn flg hops sip.  
 rt  $\neq$  update rt dip (dsn, kno, flg, hops, sip, { })  
 $\implies$  the (flag (update rt dip (dsn, kno, flg, hops, sip, { })) dip) = flg"  
 <proof>

**lemma** *the\_flag\_Some [dest!]*:  
 fixes ip rt  
 assumes "the (flag rt ip) = x"  
 and "ip  $\in$  kD rt"  
 shows "flag rt ip = Some x"  
 <proof>

**lemma** *kD\_update\_unchanged [dest]*:  
 fixes rt dip dsn dsk flag hops nhip pre  
 assumes "rt = update rt dip (dsn, dsk, flag, hops, nhip, pre)"  
 shows "dip  $\in$  kD(rt)"  
 <proof>

**lemma** *nhop\_update [simp]*: " $\bigwedge$ rt dip dsn dsk flg hops sip.  
 rt  $\neq$  update rt dip (dsn, dsk, flg, hops, sip, { })  
 $\implies$  the (nhop (update rt dip (dsn, dsk, flg, hops, sip, { })) dip) = sip"  
 <proof>

**lemma** *sqn\_update\_another [simp]*:  
 fixes dip ip rt dsn dsk flag hops nhip pre  
 assumes "ip  $\neq$  dip"  
 shows "sqn (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = sqn rt ip"  
 <proof>

**lemma** *sqnf\_update\_another [simp]*:  
 fixes dip ip rt dsn dsk flag hops nhip pre  
 assumes "ip  $\neq$  dip"  
 shows "sqnf (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = sqnf rt ip"  
 <proof>

**lemma** *vD\_update\_val [dest]*:  
" $\bigwedge$ dip rt dip' dsn dsk hops nhip pre.  
 dip  $\in$  vD(update rt dip' (dsn, dsk, val, hops, nhip, pre))  $\implies$  (dip  $\in$  vD(rt)  $\vee$  dip=dip')"  
 <proof>

## Invalidating route entries

**definition** *invalidate* :: "rt  $\Rightarrow$  (ip  $\rightarrow$  sqn)  $\Rightarrow$  rt"  
 where "invalidate rt dests  $\equiv$   
 $\lambda$ ip. case (rt ip, dests ip) of

```

  (None, _) ⇒ None
| (Some s, None) ⇒ Some s
| (Some (_, dsk, _, hops, nhop, pre), Some rsn) ⇒
  Some (rsn, dsk, inv, hops, nhop, pre)"

```

```

lemma proj3_invalidate [simp]:
  "∧dip. π3(the ((invalidate rt dests) dip)) = π3(the (rt dip))"
  ⟨proof⟩

```

```

lemma proj5_invalidate [simp]:
  "∧dip. π5(the ((invalidate rt dests) dip)) = π5(the (rt dip))"
  ⟨proof⟩

```

```

lemma proj6_invalidate [simp]:
  "∧dip. π6(the ((invalidate rt dests) dip)) = π6(the (rt dip))"
  ⟨proof⟩

```

```

lemma proj7_invalidate [simp]:
  "∧dip. π7(the ((invalidate rt dests) dip)) = π7(the (rt dip))"
  ⟨proof⟩

```

```

lemma invalidate_kD_inv [simp]:
  "∧rt dests. kD (invalidate rt dests) = kD rt"
  ⟨proof⟩

```

```

lemma invalidate_sqn:
  fixes rt dip dests
  assumes "∀rsn. dests dip = Some rsn → sqn rt dip ≤ rsn"
  shows "sqn rt dip ≤ sqn (invalidate rt dests) dip"
  ⟨proof⟩

```

```

lemma sqn_invalidate_in_dests [simp]:
  fixes dests ipa rsn rt
  assumes "dests ipa = Some rsn"
  and "ipa ∈ kD(rt)"
  shows "sqn (invalidate rt dests) ipa = rsn"
  ⟨proof⟩

```

```

lemma dhops_invalidate [simp]:
  "∧dip. the (dhops (invalidate rt dests) dip) = the (dhops rt dip)"
  ⟨proof⟩

```

```

lemma sqnf_invalidate [simp]:
  "∧dip. sqnf (invalidate (rt ξ) (dests ξ)) dip = sqnf (rt ξ) dip"
  ⟨proof⟩

```

```

lemma nhop_invalidate [simp]:
  "∧dip. the (nhop (invalidate (rt ξ) (dests ξ)) dip) = the (nhop (rt ξ) dip)"
  ⟨proof⟩

```

```

lemma invalidate_other [simp]:
  fixes rt dests dip
  assumes "dip ∉ dom(dests)"
  shows "invalidate rt dests dip = rt dip"
  ⟨proof⟩

```

```

lemma invalidate_none [simp]:
  fixes rt dests dip
  assumes "dip ∉ kD(rt)"
  shows "invalidate rt dests dip = None"
  ⟨proof⟩

```

```

lemma vD_invalidate_vD_not_dests:
  "∧dip rt dests. dip ∈ vD (invalidate rt dests) ⇒ dip ∈ vD(rt) ∧ dests dip = None"

```

*<proof>*

```
lemma sqn_invalidate_not_in_dests [simp]:  
  fixes dests dip rt  
  assumes "dip ∉ dom(dests)"  
  shows "sqn (invalidate rt dests) dip = sqn rt dip"  
  <proof>
```

```
lemma invalidate_changes:  
  fixes rt dests dip dsn dsk flag hops nhip pre  
  assumes "invalidate rt dests dip = Some (dsn, dsk, flag, hops, nhip, pre)"  
  shows " dsn = (case dests dip of None ⇒  $\pi_2$ (the (rt dip)) | Some rsn ⇒ rsn)  
    ∧ dsk =  $\pi_3$ (the (rt dip))  
    ∧ flag = (if dests dip = None then  $\pi_4$ (the (rt dip)) else inv)  
    ∧ hops =  $\pi_5$ (the (rt dip))  
    ∧ nhip =  $\pi_6$ (the (rt dip))  
    ∧ pre =  $\pi_7$ (the (rt dip))"  
  <proof>
```

```
lemma proj3_inv: " $\bigwedge$ dip rt dests. dip ∈ kD (rt)  
  ⇒  $\pi_3$ (the (invalidate rt dests dip)) =  $\pi_3$ (the (rt dip))"  
  <proof>
```

```
lemma dests_iD_invalidate [simp]:  
  assumes "dests ip = Some rsn"  
    and "ip ∈ kD(rt)"  
  shows "ip ∈ iD(invalidate rt dests)"  
  <proof>
```

## 2.1.5 Route Requests

Generate a fresh route request identifier.

```
definition nrreqid :: "(ip × rreqid) set ⇒ ip ⇒ rreqid"  
  where "nrreqid rreqs ip ≡ Max ({n. (ip, n) ∈ rreqs} ∪ {0}) + 1"
```

## 2.1.6 Queued Packets

Functions for sending data packets.

```
type_synonym store = "ip → (p × data list)"
```

```
definition sigma_queue :: "store ⇒ ip ⇒ data list" ("σqueue'(_, _)'")  
  where "σqueue(store, dip) ≡ case store dip of None ⇒ [] | Some (p, q) ⇒ q"
```

```
definition qD :: "store ⇒ ip set"  
  where "qD ≡ dom"
```

```
definition add :: "data ⇒ ip ⇒ store ⇒ store"  
  where "add d dip store ≡ case store dip of  
    None ⇒ store (dip ↦ (req, [d]))  
    | Some (p, q) ⇒ store (dip ↦ (p, q @ [d]))"
```

```
lemma qD_add [simp]:  
  fixes d dip store  
  shows "qD(add d dip store) = insert dip (qD store)"  
  <proof>
```

```
definition drop :: "ip ⇒ store → store"  
  where "drop dip store ≡  
    map_option (λ(p, q). if tl q = [] then store (dip := None)  
      else store (dip ↦ (p, tl q))) (store dip)"
```

```
definition sigma_p_flag :: "store ⇒ ip → p" ("σp-flag'(_, _)'")
```

where " $\sigma_{p\text{-flag}}(\text{store}, \text{dip}) \equiv \text{map\_option fst (store dip)}$ "

```
definition unsetRRF :: "store  $\Rightarrow$  ip  $\Rightarrow$  store"
  where "unsetRRF store dip  $\equiv$  case store dip of
        None  $\Rightarrow$  store
        | Some (p, q)  $\Rightarrow$  store (dip  $\mapsto$  (noreq, q))"
```

```
definition setRRF :: "store  $\Rightarrow$  (ip  $\rightarrow$  sqn)  $\Rightarrow$  store"
  where "setRRF store dests  $\equiv$   $\lambda$ dip. if dests dip = None then store dip
        else map_option ( $\lambda$ (_, q). (req, q)) (store dip)"
```

### 2.1.7 Comparison with the original technical report

The major differences with the AODV technical report of Fehnker et al are:

1. *nhop* is partial, thus a ‘*the*’ is needed, similarly for *dhops* and *addpreRT*.
2. *precs* is partial.
3.  $\sigma_{p\text{-flag}}(\text{store}, \text{dip})$  is partial.
4. The routing table (*rt*) is modelled as a map ( $\text{ip} \Rightarrow r \text{ option}$ ) rather than a set of 7-tuples, likewise, the *r* is a 6-tuple rather than a 7-tuple, i.e., the destination ip-address (*dip*) is taken from the argument to the function, rather than a part of the result. Well-definedness then follows from the structure of the type and more related facts are available automatically, rather than having to be acquired through tedious proofs.
5. Similar remarks hold for the *dests* mapping passed to *invalidate*, and *store*.

end

## 2.2 AODV protocol messages

```
theory B_Aodv_Message
imports B_Fwdrreps
begin
```

```
datatype msg =
  Rreq nat rreqid ip sqn k ip sqn ip
  | Rrep nat ip sqn ip ip
  | Rerr "ip  $\rightarrow$  sqn" ip
  | Newpkt data ip
  | Pkt data ip ip
```

```
instantiation msg :: msg
```

```
begin
```

```
  definition newpkt_def [simp]: "newpkt  $\equiv$   $\lambda$ (d, dip). Newpkt d dip"
```

```
  definition eq_newpkt_def: "eq_newpkt m  $\equiv$  case m of Newpkt d dip  $\Rightarrow$  True | _  $\Rightarrow$  False"
```

```
  instance <proof>
```

```
end
```

The *msg* type models the different messages used within AODV. The instantiation as a *msg* is a technicality due to the special treatment of *newpkt* messages in the AWN SOS rules. This use of classes allows a clean separation of the AWN-specific definitions and these AODV-specific definitions.

```
definition rreq :: "nat  $\times$  rreqid  $\times$  ip  $\times$  sqn  $\times$  k  $\times$  ip  $\times$  sqn  $\times$  ip  $\Rightarrow$  msg"
  where "rreq  $\equiv$   $\lambda$ (hops, rreqid, dip, dsn, dsk, oip, osn, sip).
        Rreq hops rreqid dip dsn dsk oip osn sip"
```

```
lemma rreq_simp [simp]:
```

```
  "rreq(hops, rreqid, dip, dsn, dsk, oip, osn, sip) = Rreq hops rreqid dip dsn dsk oip osn sip"
  <proof>
```

```
definition rrep :: "nat  $\times$  ip  $\times$  sqn  $\times$  ip  $\times$  ip  $\Rightarrow$  msg"
```

```

where "rrep ≡ λ(hops, dip, dsn, oip, sip). Rrep hops dip dsn oip sip"

lemma rrep_simp [simp]:
  "rrep(hops, dip, dsn, oip, sip) = Rrep hops dip dsn oip sip"
  ⟨proof⟩

definition rerr :: "(ip → sqn) × ip ⇒ msg"
  where "rerr ≡ λ(dests, sip). Rerr dests sip"

lemma rerr_simp [simp]:
  "rerr(dests, sip) = Rerr dests sip"
  ⟨proof⟩

lemma not_eq_newpkt_rreq [simp]: "¬eq_newpkt (Rreq hops rreqid dip dsn dsk oip osn sip)"
  ⟨proof⟩

lemma not_eq_newpkt_rrep [simp]: "¬eq_newpkt (Rrep hops dip dsn oip sip)"
  ⟨proof⟩

lemma not_eq_newpkt_rerr [simp]: "¬eq_newpkt (Rerr dests sip)"
  ⟨proof⟩

lemma not_eq_newpkt_pkt [simp]: "¬eq_newpkt (Pkt d dip sip)"
  ⟨proof⟩

definition pkt :: "data × ip × ip ⇒ msg"
  where "pkt ≡ λ(d, dip, sip). Pkt d dip sip"

lemma pkt_simp [simp]:
  "pkt(d, dip, sip) = Pkt d dip sip"
  ⟨proof⟩

```

end

## 2.3 The AODV protocol

```

theory B_Aodv
imports B_Aodv_Data B_Aodv_Message
  "../.../AWN/AWN_SOS_Labels" "../.../AWN/AWN_Invariants"
begin

```

### 2.3.1 Data state

```

record state =
  ip      :: "ip"
  sn      :: "sqn"
  rt      :: "rt"
  rreqs   :: "(ip × rreqid) set"
  store   :: "store"

  msg     :: "msg"
  data    :: "data"
  dests   :: "ip → sqn"
  pre     :: "ip set"
  rreqid  :: "rreqid"
  dip     :: "ip"
  oip     :: "ip"
  hops    :: "nat"
  dsn     :: "sqn"
  dsk     :: "k"
  osn     :: "sqn"
  sip     :: "ip"

abbreviation aodv_init :: "ip ⇒ state"

```



```

where "aadv_init i ≡ (
  ip = i,
  sn = 1,
  rt = empty,
  rreqs = {},
  store = empty,

  msg    = (SOME x. True),
  data   = (SOME x. True),
  dests  = (SOME x. True),
  pre    = (SOME x. True),
  rreqid = (SOME x. True),
  dip    = (SOME x. True),
  oip    = (SOME x. True),
  hops   = (SOME x. True),
  dsn    = (SOME x. True),
  dsk    = (SOME x. True),
  osn    = (SOME x. True),
  sip    = (SOME x. x ≠ i)
)"

```

```

lemma some_neq_not_eq [simp]: "¬((SOME x :: nat. x ≠ i) = i)"
⟨proof⟩

```

```

definition clear_locals :: "state ⇒ state"

```

```

where "clear_locals ξ = ξ (
  msg    := (SOME x. True),
  data   := (SOME x. True),
  dests  := (SOME x. True),
  pre    := (SOME x. True),
  rreqid := (SOME x. True),
  dip    := (SOME x. True),
  oip    := (SOME x. True),
  hops   := (SOME x. True),
  dsn    := (SOME x. True),
  dsk    := (SOME x. True),
  osn    := (SOME x. True),
  sip    := (SOME x. x ≠ ip ξ)
)"

```

```

lemma clear_locals_sip_not_ip [simp]: "¬(sip (clear_locals ξ) = ip ξ)"
⟨proof⟩

```

```

lemma clear_locals_but_not_globals [simp]:

```

```

"ip (clear_locals ξ) = ip ξ"
"sn (clear_locals ξ) = sn ξ"
"rt (clear_locals ξ) = rt ξ"
"rreqs (clear_locals ξ) = rreqs ξ"
"store (clear_locals ξ) = store ξ"
⟨proof⟩

```

### 2.3.2 Auxilliary message handling definitions

```

definition is_newpkt

```

```

where "is_newpkt ξ ≡ case msg ξ of
  Newpkt data' dip' ⇒ { ξ(|data := data', dip := dip'|) }
  | _ ⇒ {}"

```

```

definition is_pkt

```

```

where "is_pkt ξ ≡ case msg ξ of
  Pkt data' dip' oip' ⇒ { ξ(|data := data', dip := dip', oip := oip'|) }
  | _ ⇒ {}"

```

```

definition is_rreq

```

where "is\_rreq  $\xi \equiv$  case msg  $\xi$  of  
     Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip'  $\Rightarrow$   
     {  $\xi$ (| hops := hops', rreqid := rreqid', dip := dip', dsn := dsn',  
        dsk := dsk', oip := oip', osn := osn', sip := sip' |) }  
 | \_  $\Rightarrow$  {}"

lemma is\_rreq\_asm [dest!]:  
 assumes " $\xi' \in$  is\_rreq  $\xi$ "  
 shows " $(\exists$  hops' rreqid' dip' dsn' dsk' oip' osn' sip'.  
     msg  $\xi =$  Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip'  $\wedge$   
      $\xi' = \xi$ (| hops := hops', rreqid := rreqid', dip := dip', dsn := dsn',  
        dsk := dsk', oip := oip', osn := osn', sip := sip' |))"

<proof>

definition is\_rrep  
 where "is\_rrep  $\xi \equiv$  case msg  $\xi$  of  
     Rrep hops' dip' dsn' oip' sip'  $\Rightarrow$   
     {  $\xi$ (| hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' |) }  
 | \_  $\Rightarrow$  {}"

lemma is\_rrep\_asm [dest!]:  
 assumes " $\xi' \in$  is\_rrep  $\xi$ "  
 shows " $(\exists$  hops' dip' dsn' oip' sip'.  
     msg  $\xi =$  Rrep hops' dip' dsn' oip' sip'  $\wedge$   
      $\xi' = \xi$ (| hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' |))"

<proof>

definition is\_rerr  
 where "is\_rerr  $\xi \equiv$  case msg  $\xi$  of  
     Rerr dests' sip'  $\Rightarrow$  {  $\xi$ (| dests := dests', sip := sip' |) }  
 | \_  $\Rightarrow$  {}"

lemma is\_rerr\_asm [dest!]:  
 assumes " $\xi' \in$  is\_rerr  $\xi$ "  
 shows " $(\exists$  dests' sip'.  
     msg  $\xi =$  Rerr dests' sip'  $\wedge$   
      $\xi' = \xi$ (| dests := dests', sip := sip' |))"

<proof>

lemmas is\_msg\_defs =  
 is\_rerr\_def is\_rrep\_def is\_rreq\_def is\_pkt\_def is\_newpkt\_def

lemma is\_msg\_inv\_ip [simp]:  
 " $\xi' \in$  is\_rerr  $\xi \implies$  ip  $\xi' =$  ip  $\xi$ "  
 " $\xi' \in$  is\_rrep  $\xi \implies$  ip  $\xi' =$  ip  $\xi$ "  
 " $\xi' \in$  is\_rreq  $\xi \implies$  ip  $\xi' =$  ip  $\xi$ "  
 " $\xi' \in$  is\_pkt  $\xi \implies$  ip  $\xi' =$  ip  $\xi$ "  
 " $\xi' \in$  is\_newpkt  $\xi \implies$  ip  $\xi' =$  ip  $\xi$ "  
 <proof>

lemma is\_msg\_inv\_sn [simp]:  
 " $\xi' \in$  is\_rerr  $\xi \implies$  sn  $\xi' =$  sn  $\xi$ "  
 " $\xi' \in$  is\_rrep  $\xi \implies$  sn  $\xi' =$  sn  $\xi$ "  
 " $\xi' \in$  is\_rreq  $\xi \implies$  sn  $\xi' =$  sn  $\xi$ "  
 " $\xi' \in$  is\_pkt  $\xi \implies$  sn  $\xi' =$  sn  $\xi$ "  
 " $\xi' \in$  is\_newpkt  $\xi \implies$  sn  $\xi' =$  sn  $\xi$ "  
 <proof>

lemma is\_msg\_inv\_rt [simp]:  
 " $\xi' \in$  is\_rerr  $\xi \implies$  rt  $\xi' =$  rt  $\xi$ "  
 " $\xi' \in$  is\_rrep  $\xi \implies$  rt  $\xi' =$  rt  $\xi$ "  
 " $\xi' \in$  is\_rreq  $\xi \implies$  rt  $\xi' =$  rt  $\xi$ "  
 " $\xi' \in$  is\_pkt  $\xi \implies$  rt  $\xi' =$  rt  $\xi$ "  
 " $\xi' \in$  is\_newpkt  $\xi \implies$  rt  $\xi' =$  rt  $\xi$ "

*<proof>*

```
lemma is_msg_inv_rreqs [simp]:  
  "ξ' ∈ is_rerr ξ  ⇒ rreqs ξ' = rreqs ξ"  
  "ξ' ∈ is_rrep ξ  ⇒ rreqs ξ' = rreqs ξ"  
  "ξ' ∈ is_rreq ξ  ⇒ rreqs ξ' = rreqs ξ"  
  "ξ' ∈ is_pkt ξ   ⇒ rreqs ξ' = rreqs ξ"  
  "ξ' ∈ is_newpkt ξ ⇒ rreqs ξ' = rreqs ξ"  
<proof>
```

```
lemma is_msg_inv_store [simp]:  
  "ξ' ∈ is_rerr ξ  ⇒ store ξ' = store ξ"  
  "ξ' ∈ is_rrep ξ  ⇒ store ξ' = store ξ"  
  "ξ' ∈ is_rreq ξ  ⇒ store ξ' = store ξ"  
  "ξ' ∈ is_pkt ξ   ⇒ store ξ' = store ξ"  
  "ξ' ∈ is_newpkt ξ ⇒ store ξ' = store ξ"  
<proof>
```

```
lemma is_msg_inv_sip [simp]:  
  "ξ' ∈ is_pkt ξ   ⇒ sip ξ' = sip ξ"  
  "ξ' ∈ is_newpkt ξ ⇒ sip ξ' = sip ξ"  
<proof>
```

### 2.3.3 The protocol process

```
datatype pseqp =  
  PAadv  
  | PNewPkt  
  | PPkt  
  | PRreq  
  | PRrep  
  | PRerr
```

```
fun nat_of_seqp :: "pseqp ⇒ nat"  
where  
  "nat_of_seqp PAadv = 1"  
| "nat_of_seqp PPkt  = 2"  
| "nat_of_seqp PNewPkt = 3"  
| "nat_of_seqp PRreq = 4"  
| "nat_of_seqp PRrep = 5"  
| "nat_of_seqp PRerr = 6"
```

```
instantiation "pseqp" :: ord  
begin  
definition less_eq_seqp [iff]: "l1 ≤ l2 = (nat_of_seqp l1 ≤ nat_of_seqp l2)"  
definition less_seqp [iff]:   "l1 < l2 = (nat_of_seqp l1 < nat_of_seqp l2)"  
instance <proof>  
end
```

```
abbreviation AODV  
where  
  "AODV ≡ λ_. [[clear_locals]] call(PAadv)"
```

```
abbreviation PKT  
where  
  "PKT args ≡  
  
  [[ξ. let (data, dip, oip) = args ξ in  
    (clear_locals ξ) (| data := data, dip := dip, oip := oip |)]  
  call(PPkt)]"
```

```
abbreviation NEWPKT  
where  
  "NEWPKT args ≡  
  [[ξ. let (data, dip) = args ξ in
```

```

(clear_locals ξ) (| data := data, dip := dip |)
call(PNewPkt)"

```

abbreviation RREQ

where

```

"RREQ args ≡
[[ξ. let (hops, rreqid, dip, dsn, dsk, oip, osn, sip) = args ξ in
(clear_locals ξ) (| hops := hops, rreqid := rreqid, dip := dip,
dsn := dsn, dsk := dsk, oip := oip,
osn := osn, sip := sip |)]]

call(PRreq)"

```

abbreviation RREP

where

```

"RREP args ≡
[[ξ. let (hops, dip, dsn, oip, sip) = args ξ in
(clear_locals ξ) (| hops := hops, dip := dip, dsn := dsn,
oip := oip, sip := sip |)]]

call(PRrep)"

```

abbreviation RERR

where

```

"RERR args ≡
[[ξ. let (dests, sip) = args ξ in
(clear_locals ξ) (| dests := dests, sip := sip |)]]

call(PRerr)"

```

fun  $\Gamma_{AODV} :: "(state, msg, pseqp, pseqp label) seqp\_env"$

where

```

" $\Gamma_{AODV}$  PAodv = labelled PAodv (
receive( $\lambda$ msg' ξ. ξ (| msg := msg' |)).
(
<is_newpkt> NEWPKT( $\lambda$ ξ. (data ξ, ip ξ))
⊕ <is_pkt> PKT( $\lambda$ ξ. (data ξ, dip ξ, oip ξ))
⊕ <is_rreq>
[[ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {f}) |)]]
RREQ( $\lambda$ ξ. (hops ξ, rreqid ξ, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, sip ξ))
⊕ <is_rrep>
[[ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {f}) |)]]
RREP( $\lambda$ ξ. (hops ξ, dip ξ, dsn ξ, oip ξ, sip ξ))
⊕ <is_rerr>
[[ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {f}) |)]]
RERR( $\lambda$ ξ. (dests ξ, sip ξ))
)
⊕ < $\lambda$ ξ. { ξ (| dip := dip |) | dip. dip ∈ qD(store ξ) ∩ vD(rt ξ) }>
[[ξ. ξ (| data := hd( $\sigma$ queue(store ξ, dip ξ)) |)]]
unicast( $\lambda$ ξ. the (nhop (rt ξ) (dip ξ)),  $\lambda$ ξ. pkt(data ξ, dip ξ, ip ξ)).
[[ξ. ξ (| store := the (drop (dip ξ) (store ξ)) |)]]
AODV()
▷ [[ξ. ξ (| dests := ( $\lambda$ rip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
then Some (inc (sqn (rt ξ) rip)) else None) |)]]
[[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)]]
[[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |)]]
[[ξ. ξ (| pre :=  $\bigcup$  { the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } |)]]
[[ξ. ξ (| dests := ( $\lambda$ rip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {f})
then (dests ξ) rip else None) |)]]
groupcast( $\lambda$ ξ. pre ξ,  $\lambda$ ξ. rerr(dests ξ, ip ξ)). AODV()
⊕ < $\lambda$ ξ. { ξ (| dip := dip |)
| dip. dip ∈ qD(store ξ) - vD(rt ξ) ∧ the ( $\sigma_{p\_flag}$ (store ξ, dip)) = req }>
[[ξ. ξ (| store := unsetRRF (store ξ) (dip ξ) |)]]
[[ξ. ξ (| sn := inc (sn ξ) |)]]
[[ξ. ξ (| rreqid := nrreqid (rreqs ξ) (ip ξ) |)]]
[[ξ. ξ (| rreqs := rreqs ξ ∪ {(ip ξ, rreqid ξ)} |)]]
broadcast( $\lambda$ ξ. rreq(0, rreqid ξ, dip ξ, sqn (rt ξ) (dip ξ), sqnf (rt ξ) (dip ξ),
ip ξ, sn ξ, ip ξ)). AODV()"
```

```

/ "ΓAODV PNewPkt = labelled PNewPkt (
  ⟨ξ. dip ξ = ip ξ⟩
  deliver(λξ. data ξ).AODV()
  ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
  [[ξ. ξ (| store := add (data ξ) (dip ξ) (store ξ) |)]
  AODV()]"

/ "ΓAODV PPkt = labelled PPkt (
  ⟨ξ. dip ξ = ip ξ⟩
  deliver(λξ. data ξ).AODV()
  ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
  (
    ⟨ξ. dip ξ ∈ vD (rt ξ)⟩
    unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, oip ξ)).AODV()
    ▷
    [[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
      then Some (inc (sqn (rt ξ) rip)) else None) |)]
    [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)]
    [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |)]
    [[ξ. ξ (| pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } |)]
    [[ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ { })
      then (dests ξ) rip else None) |)]
    groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
  ⊕ ⟨ξ. dip ξ ∉ vD (rt ξ)⟩
  (
    ⟨ξ. dip ξ ∈ iD (rt ξ)⟩
    groupcast(λξ. the (precs (rt ξ) (dip ξ)),
      λξ. rerr([dip ξ ↦ sqn (rt ξ) (dip ξ)], ip ξ)). AODV()
  ⊕ ⟨ξ. dip ξ ∉ iD (rt ξ)⟩
    AODV()
  )
  )
  )"

/ "ΓAODV PRreq = labelled PRreq (
  ⟨ξ. (oip ξ, rreqid ξ) ∈ rreqs ξ⟩
  AODV()
  ⊕ ⟨ξ. (oip ξ, rreqid ξ) ∉ rreqs ξ⟩
  [[ξ. ξ (| rt := update (rt ξ) (oip ξ) (osn ξ, kno, val, hops ξ + 1, sip ξ, { }) |)]
  [[ξ. ξ (| rreqs := rreqs ξ ∪ {(oip ξ, rreqid ξ)} |)]
  (
    ⟨ξ. dip ξ = ip ξ⟩
    [[ξ. ξ (| sn := max (sn ξ) (dsn ξ) |)]
    unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(0, dip ξ, sn ξ, oip ξ, ip ξ)).AODV()
    ▷
    [[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
      then Some (inc (sqn (rt ξ) rip)) else None) |)]
    [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)]
    [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |)]
    [[ξ. ξ (| pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } |)]
    [[ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ { })
      then (dests ξ) rip else None) |)]
    groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
  ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
  (
    ⟨ξ. dip ξ ∈ vD (rt ξ) ∧ dsn ξ ≤ sqn (rt ξ) (dip ξ) ∧ sqnf (rt ξ) (dip ξ) = kno⟩
    [[ξ. ξ (| rt := the (addpreRT (rt ξ) (dip ξ) {sip ξ}) |)]
    [[ξ. ξ (| rt := the (addpreRT (rt ξ) (oip ξ) {the (nhop (rt ξ) (dip ξ)}) |)]
    unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(the (dhops (rt ξ) (dip ξ)), dip ξ,
      sqn (rt ξ) (dip ξ), oip ξ, ip ξ)).
    AODV()
    ▷
    [[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
      then Some (inc (sqn (rt ξ) rip)) else None) |)]
  )
  )
  )"

```

```

    [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)]
    [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |)]
    [[ξ. ξ (| pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } |)]
    [[ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
        then (dests ξ) rip else None) |)]
    groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
⊕ ⟨ξ. dip ξ ∉ vD (rt ξ) ∨ sqn (rt ξ) (dip ξ) < dsn ξ ∨ sqnf (rt ξ) (dip ξ) = unk⟩
    broadcast(λξ. rreq(hops ξ + 1, rreqid ξ, dip ξ, max (sqn (rt ξ) (dip ξ)) (dsn ξ),
        dsk ξ, oip ξ, osn ξ, ip ξ)).
    AODV()
  )
)
)"

/ "ΓAODV PRrep = labelled PRrep (
  [[ξ. ξ (| rt := update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, {}) |)]
  (
    ⟨ξ. oip ξ = ip ξ ⟩
    AODV()
  ⊕ ⟨ξ. oip ξ ≠ ip ξ ⟩
  (
    ⟨ξ. oip ξ ∈ vD (rt ξ) ∧ dip ξ ∈ vD (rt ξ) ⟩
    [[ξ. ξ (| rt := the (addpreRT (rt ξ) (dip ξ)
        {the (nhop (rt ξ) (oip ξ))}) |)]
    [[ξ. ξ (| rt := the (addpreRT (rt ξ) (the (nhop (rt ξ) (dip ξ))) {the (nhop (rt ξ) (oip ξ))}) |)]
  )]
  unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(the (dhops (rt ξ) (dip ξ)), dip ξ,
    sqn (rt ξ) (dip ξ), oip ξ, ip ξ)).
  AODV()
  ▷
  [[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
    then Some (inc (sqn (rt ξ) rip)) else None) |)]
  [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)]
  [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |)]
  [[ξ. ξ (| pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } |)]
  [[ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
    then (dests ξ) rip else None) |)]
  groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
  ⊕ ⟨ξ. oip ξ ∉ vD (rt ξ) ∨ dip ξ ∉ vD (rt ξ) ⟩
  AODV()
  )
)
)"

/ "ΓAODV PRerr = labelled PRerr (
  [[ξ. ξ (| dests := (λrip. case (dests ξ) rip of None ⇒ None
    | Some rsn ⇒ if rip ∈ vD (rt ξ) ∧ the (nhop (rt ξ) rip) = sip ξ
        ∧ sqn (rt ξ) rip < rsn then Some rsn else None) |)]
  [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)]
  [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |)]
  [[ξ. ξ (| pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } |)]
  [[ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
    then (dests ξ) rip else None) |)]
  groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)). AODV()"

declare ΓAODV.simps [simp del, code del]
lemmas ΓAODV.simps [simp, code] = ΓAODV.simps [simplified]

fun ΓAODV.skeleton
where
  "ΓAODV.skeleton PAodv = seqp_skeleton (ΓAODV PAodv)"
  / "ΓAODV.skeleton PNewPkt = seqp_skeleton (ΓAODV PNewPkt)"
  / "ΓAODV.skeleton PPkt = seqp_skeleton (ΓAODV PPkt)"
  / "ΓAODV.skeleton PRreq = seqp_skeleton (ΓAODV PRreq)"
  / "ΓAODV.skeleton PRrep = seqp_skeleton (ΓAODV PRrep)"

```

```

| " $\Gamma_{AODV\_skeleton} PRerr = seqp\_skeleton (\Gamma_{AODV} PRerr)$ "

lemma  $\Gamma_{AODV\_skeleton\_wf}$  [simp]:
  "wellformed  $\Gamma_{AODV\_skeleton}$ "
  <proof>

declare  $\Gamma_{AODV\_skeleton.simps}$  [simp del, code del]
lemmas  $\Gamma_{AODV\_skeleton.simps}$  [simp, code]
      =  $\Gamma_{AODV\_skeleton.simps}$  [simplified  $\Gamma_{AODV.simps} seqp\_skeleton.simps$ ]

lemma aodv_proc_cases [dest]:
  fixes p pn
  shows "p  $\in$  ctermsl ( $\Gamma_{AODV} pn$ )  $\implies$ 
        (p  $\in$  ctermsl ( $\Gamma_{AODV} PAodv$ )  $\vee$ 
         p  $\in$  ctermsl ( $\Gamma_{AODV} PNewPkt$ )  $\vee$ 
         p  $\in$  ctermsl ( $\Gamma_{AODV} PPkt$ )  $\vee$ 
         p  $\in$  ctermsl ( $\Gamma_{AODV} PRreq$ )  $\vee$ 
         p  $\in$  ctermsl ( $\Gamma_{AODV} PRrep$ )  $\vee$ 
         p  $\in$  ctermsl ( $\Gamma_{AODV} PRerr$ ))"

  <proof>

definition  $\sigma_{AODV} :: "ip \implies (state \times (state, msg, pseqp, pseqp label) seqp) set"$ 
where " $\sigma_{AODV} i \equiv \{(aodv\_init\ i, \Gamma_{AODV} PAodv)\}$ "

abbreviation paodv
  :: "ip  $\implies (state \times (state, msg, pseqp, pseqp label) seqp, msg seq\_action) automaton"$ 
where
  " $paodv\ i \equiv (\{ init = \sigma_{AODV} i, trans = seqp\_sos\ \Gamma_{AODV} \})$ "

lemma aodv_trans: "trans (paodv i) = seqp\_sos  $\Gamma_{AODV}$ "
  <proof>

lemma aodv_control_within [simp]: "control_within  $\Gamma_{AODV}$  (init (paodv i))"
  <proof>

lemma aodv_wf [simp]:
  "wellformed  $\Gamma_{AODV}$ "
  <proof>

lemmas aodv_labels_not_empty [simp] = labels_not_empty [OF aodv_wf]

lemma aodv_ex_label [intro]: " $\exists l. l \in labels\ \Gamma_{AODV}\ p$ "
  <proof>

lemma aodv_ex_labelE [elim]:
  assumes " $\forall l \in labels\ \Gamma_{AODV}\ p. P\ l\ p$ "
  and " $\exists p\ l. P\ l\ p \implies Q$ "
  shows "Q"
  <proof>

lemma aodv_simple_labels [simp]: "simple_labels  $\Gamma_{AODV}$ "
  <proof>

lemma  $\sigma_{AODV\_labels}$  [simp]: " $(\xi, p) \in \sigma_{AODV}\ i \implies labels\ \Gamma_{AODV}\ p = \{PAodv-:0\}$ "
  <proof>

lemma aodv_init_kD_empty [simp]:
  " $(\xi, p) \in \sigma_{AODV}\ i \implies kD\ (rt\ \xi) = \{\}$ "
  <proof>

lemma aodv_init_sip_not_ip [simp]: " $\neg (sip\ (aodv\_init\ i) = i)$ " <proof>

lemma aodv_init_sip_not_ip' [simp]:
  assumes " $(\xi, p) \in \sigma_{AODV}\ i$ "

```

```

  shows "sip  $\xi \neq ip \xi$ "
  <proof>

```

```

lemma aadv_init_sip_not_i [simp]:
  assumes " $(\xi, p) \in \sigma_{AODV} i$ "
  shows "sip  $\xi \neq i$ "
  <proof>

```

```

lemma clear_locals_sip_not_ip':
  assumes "ip  $\xi = i$ "
  shows " $\neg(\text{sip}(\text{clear\_locals } \xi) = i)$ "
  <proof>

```

Stop the simplifier from descending into process terms.

```

declare seqp_congs [cong]

```

Configure the main invariant tactic for AODV.

```

declare
   $\Gamma_{AODV\_simps}$  [ctermenv]
  aadv_proc_cases [ctermst_cases]
  seq_invariant_ctermstI [OF aadv_wf aadv_control_within aadv_simple_labels aadv_trans,
                           ctermst_intros]
  seq_step_invariant_ctermstI [OF aadv_wf aadv_control_within aadv_simple_labels aadv_trans,
                                ctermst_intros]

```

end

## 2.4 Invariant assumptions and properties

```

theory B_Aadv_Predicates
imports B_Aadv
begin

```

Definitions for expression assumptions on incoming messages and properties of outgoing messages.

```

abbreviation not_Pkt :: "msg  $\Rightarrow$  bool"
where "not_Pkt m  $\equiv$  case m of Pkt _ _ _  $\Rightarrow$  False | _  $\Rightarrow$  True"

```

```

definition msg_sender :: "msg  $\Rightarrow$  ip"
where "msg_sender m  $\equiv$  case m of Rreq _ _ _ _ _ ipc  $\Rightarrow$  ipc
      | Rrep _ _ _ _ ipc  $\Rightarrow$  ipc
      | Rerr _ ipc  $\Rightarrow$  ipc
      | Pkt _ _ ipc  $\Rightarrow$  ipc"

```

```

lemma msg_sender_simps [simp]:
  " $\bigwedge$ hops rreqid dip dsn dsk oip osn sip.
    msg_sender (Rreq hops rreqid dip dsn dsk oip osn sip) = sip"
  " $\bigwedge$ hops dip dsn oip sip. msg_sender (Rrep hops dip dsn oip sip) = sip"
  " $\bigwedge$ destds sip. msg_sender (Rerr destds sip) = sip"
  " $\bigwedge$ d dip sip. msg_sender (Pkt d dip sip) = sip"
  <proof>

```

```

definition msg_zhops :: "msg  $\Rightarrow$  bool"
where "msg_zhops m  $\equiv$  case m of
      Rreq hopsdip _ _ oipc _ sipc  $\Rightarrow$  hopsdip = 0  $\longrightarrow$  oipc = sipc
      | Rrep hopsdip dip _ _ sipc  $\Rightarrow$  hopsdip = 0  $\longrightarrow$  dip = sipc
      | _  $\Rightarrow$  True"

```

```

lemma msg_zhops_simps [simp]:
  " $\bigwedge$ hops rreqid dip dsn dsk oip osn sip.
    msg_zhops (Rreq hops rreqid dip dsn dsk oip osn sip) = (hops = 0  $\longrightarrow$  oip = sip)"
  " $\bigwedge$ hops dip dsn oip sip. msg_zhops (Rrep hops dip dsn oip sip) = (hops = 0  $\longrightarrow$  dip = sip)"
  " $\bigwedge$ destds sip. msg_zhops (Rerr destds sip) = True"
  " $\bigwedge$ dip. msg_zhops (Newpkt d dip) = True"

```



" $\wedge$ d dip sip.                   msg\_zhops (Pkt d dip sip)                   = True"  
 <proof>

**definition** rreq\_rrep\_sn :: "msg  $\Rightarrow$  bool"  
 where "rreq\_rrep\_sn m  $\equiv$  case m of Rreq \_ \_ \_ \_ osnc \_  $\Rightarrow$  osnc  $\geq$  1  
           | Rrep \_ \_ dsnc \_ \_  $\Rightarrow$  dsnc  $\geq$  1  
           | \_  $\Rightarrow$  True"

**lemma** rreq\_rrep\_sn\_simps [simp]:  
 " $\wedge$ hops rreqid dip dsn dsk oip osn sip.  
       rreq\_rrep\_sn (Rreq hops rreqid dip dsn dsk oip osn sip) = (osn  $\geq$  1)"  
 " $\wedge$ hops dip dsn oip sip. rreq\_rrep\_sn (Rrep hops dip dsn oip sip) = (dsn  $\geq$  1)"  
 " $\wedge$ dests sip.            rreq\_rrep\_sn (Rerr dests sip) = True"  
 " $\wedge$ d dip.                rreq\_rrep\_sn (Newpkt d dip)    = True"  
 " $\wedge$ d dip sip.            rreq\_rrep\_sn (Pkt d dip sip)    = True"  
 <proof>

**definition** rreq\_rrep\_fresh :: "rt  $\Rightarrow$  msg  $\Rightarrow$  bool"  
 where "rreq\_rrep\_fresh crt m  $\equiv$  case m of Rreq hopsc \_ \_ \_ oipc osnc ipcc  $\Rightarrow$  (ipcc  $\neq$  oipc  $\longrightarrow$   
   oipc  $\in$  kD(crt)  $\wedge$  (sqn crt oipc > osnc  
    $\vee$  (sqn crt oipc = osnc  
    $\wedge$  the (dhops crt oipc)  $\leq$  hopsc  
    $\wedge$  the (flag crt oipc) = val)))  
       | Rrep hopsc dipc dsnc \_ ipcc  $\Rightarrow$  (ipcc  $\neq$  dipc  $\longrightarrow$   
   dipc  $\in$  kD(crt)  
    $\wedge$  sqn crt dipc = dsnc  
    $\wedge$  the (dhops crt dipc) = hopsc  
    $\wedge$  the (flag crt dipc) = val)  
       | \_  $\Rightarrow$  True"

**lemma** rreq\_rrep\_fresh [simp]:  
 " $\wedge$ hops rreqid dip dsn dsk oip osn sip.  
       rreq\_rrep\_fresh crt (Rreq hops rreqid dip dsn dsk oip osn sip) =  
       (sip  $\neq$  oip  $\longrightarrow$  oip  $\in$  kD(crt)  
        $\wedge$  (sqn crt oip > osn  
        $\vee$  (sqn crt oip = osn  
        $\wedge$  the (dhops crt oip)  $\leq$  hops  
        $\wedge$  the (flag crt oip) = val)))"  
 " $\wedge$ hops dip dsn oip sip. rreq\_rrep\_fresh crt (Rrep hops dip dsn oip sip) =  
       (sip  $\neq$  dip  $\longrightarrow$  dip  $\in$  kD(crt)  
        $\wedge$  sqn crt dip = dsn  
        $\wedge$  the (dhops crt dip) = hops  
        $\wedge$  the (flag crt dip) = val)"  
 " $\wedge$ dests sip.            rreq\_rrep\_fresh crt (Rerr dests sip) = True"  
 " $\wedge$ d dip.                rreq\_rrep\_fresh crt (Newpkt d dip)    = True"  
 " $\wedge$ d dip sip.            rreq\_rrep\_fresh crt (Pkt d dip sip)    = True"  
 <proof>

**definition** rerr\_invalid :: "rt  $\Rightarrow$  msg  $\Rightarrow$  bool"  
 where "rerr\_invalid crt m  $\equiv$  case m of Rerr destsc \_  $\Rightarrow$  ( $\forall$ ripc  $\in$  dom(destsc).  
   (ripc  $\in$  iD(crt)  $\wedge$  the (destsc ripc) = sqn crt ripc))  
       | \_  $\Rightarrow$  True"

**lemma** rerr\_invalid [simp]:  
 " $\wedge$ hops rreqid dip dsn dsk oip osn sip.  
       rerr\_invalid crt (Rreq hops rreqid dip dsn dsk oip osn sip) = True"  
 " $\wedge$ hops dip dsn oip sip. rerr\_invalid crt (Rrep hops dip dsn oip sip) = True"  
 " $\wedge$ dests sip.            rerr\_invalid crt (Rerr dests sip) = ( $\forall$ ripc  $\in$  dom(dests).  
   ripc  $\in$  iD(crt)  $\wedge$  the (dests ripc) = sqn crt ripc)"  
 " $\wedge$ d dip.                rerr\_invalid crt (Newpkt d dip)    = True"  
 " $\wedge$ d dip sip.            rerr\_invalid crt (Pkt d dip sip)    = True"  
 <proof>

**definition**

```

  initmissing :: "(nat  $\Rightarrow$  state option)  $\times$  'a  $\Rightarrow$  (nat  $\Rightarrow$  state)  $\times$  'a"
where
  "initmissing  $\sigma$  = ( $\lambda$ i. case (fst  $\sigma$ ) i of None  $\Rightarrow$  aadv_init i | Some s  $\Rightarrow$  s, snd  $\sigma$ )"

lemma not_in_net_ips_fst_init_missing [simp]:
  assumes "i  $\notin$  net_ips  $\sigma$ "
  shows "fst (initmissing (netgmap fst  $\sigma$ )) i = aadv_init i"
  <proof>

lemma fst_initmissing_netgmap_pair_fst [simp]:
  "fst (initmissing (netgmap ( $\lambda$ (p, q). (fst (id p), snd (id p), q)) s))
    = fst (initmissing (netgmap fst s))"
  <proof>

```

We introduce a streamlined alternative to `initmissing` with `netgmap` to simplify invariant statements and thus facilitate their comprehension and presentation.

```

lemma fst_initmissing_netgmap_default_aadv_init_netlift:
  "fst (initmissing (netgmap fst s)) = default aadv_init (netlift fst s)"
  <proof>

```

```

definition
  netglobal :: "(nat  $\Rightarrow$  state)  $\Rightarrow$  bool  $\Rightarrow$  ((state  $\times$  'b)  $\times$  'c) net_state  $\Rightarrow$  bool"
where
  "netglobal P  $\equiv$  ( $\lambda$ s. P (default aadv_init (netlift fst s)))"

end

```

## 2.5 Quality relations between routes

```

theory B_Fresher
imports B_Aadv_Data
begin

```

### 2.5.1 Net sequence numbers

On individual routes

```

definition
  nsqnr :: "r  $\Rightarrow$  sqn"
where
  "nsqnr r  $\equiv$  if  $\pi_4$ (r) = val  $\vee$   $\pi_2$ (r) = 0 then  $\pi_2$ (r) else ( $\pi_2$ (r) - 1)"

```

```

lemma nsqnr_def':
  "nsqnr r = (if  $\pi_4$ (r) = inv then  $\pi_2$ (r) - 1 else  $\pi_2$ (r))"
  <proof>

```

```

lemma nsqnr_zero [simp]:
  " $\bigwedge$ dsn dsk flag hops nhip pre. nsqnr (0, dsk, flag, hops, nhip, pre) = 0"
  <proof>

```

```

lemma nsqnr_val [simp]:
  " $\bigwedge$ dsn dsk hops nhip pre. nsqnr (dsn, dsk, val, hops, nhip, pre) = dsn"
  <proof>

```

```

lemma nsqnr_inv [simp]:
  " $\bigwedge$ dsn dsk hops nhip pre. nsqnr (dsn, dsk, inv, hops, nhip, pre) = dsn - 1"
  <proof>

```

```

lemma nsqnr_lte_dsn [simp]:
  " $\bigwedge$ dsn dsk flag hops nhip pre. nsqnr (dsn, dsk, flag, hops, nhip, pre)  $\leq$  dsn"
  <proof>

```

## On routes in routing tables

### definition

$nsqn :: "rt \Rightarrow ip \Rightarrow sqn"$

### where

$"nsqn \equiv \lambda rt\ dip. \text{ case } \sigma_{route}(rt, dip) \text{ of None} \Rightarrow 0 \mid \text{Some } r \Rightarrow nsqn_r(r)"$

### lemma nsqn\_sqn\_def:

$"\bigwedge rt\ dip. nsqn\ rt\ dip = (\text{if flag } rt\ dip = \text{Some } val \vee sqn\ rt\ dip = 0$   
 $\text{then } sqn\ rt\ dip \text{ else } sqn\ rt\ dip - 1)"$

$\langle proof \rangle$

### lemma not\_in\_kD\_nsqn [simp]:

assumes  $"dip \notin kD(rt)"$

shows  $"nsqn\ rt\ dip = 0"$

$\langle proof \rangle$

### lemma kD\_nsqn:

assumes  $"dip \in kD(rt)"$

shows  $"nsqn\ rt\ dip = nsqn_r(\text{the } (\sigma_{route}(rt, dip)))"$

$\langle proof \rangle$

### lemma nsqnr\_r\_flag\_pred [simp, intro]:

fixes  $dsn\ dsk\ flag\ hops\ nhip\ pre$

assumes  $"P (nsqn_r (dsn, dsk, val, hops, nhip, pre))"$

and  $"P (nsqn_r (dsn, dsk, inv, hops, nhip, pre))"$

shows  $"P (nsqn_r (dsn, dsk, flag, hops, nhip, pre))"$

$\langle proof \rangle$

### lemma nsqn\_r\_addpreRT\_inv [simp]:

$"\bigwedge rt\ dip\ npre\ dip'. dip \in kD(rt) \implies$

$nsqn_r (\text{the } (\text{the } (\text{addpreRT } rt\ dip\ npre) dip')) = nsqn_r (\text{the } (rt\ dip'))"$

$\langle proof \rangle$

### lemma sqn\_nsqn:

$"\bigwedge rt\ dip. sqn\ rt\ dip - 1 \leq nsqn\ rt\ dip"$

$\langle proof \rangle$

### lemma nsqn\_sqn: $"nsqn\ rt\ dip \leq sqn\ rt\ dip"$

$\langle proof \rangle$

### lemma val\_nsqn\_sqn [elim, simp]:

assumes  $"ip \in kD(rt)"$

and  $"\text{the } (flag\ rt\ ip) = val"$

shows  $"nsqn\ rt\ ip = sqn\ rt\ ip"$

$\langle proof \rangle$

### lemma vD\_nsqn\_sqn [elim, simp]:

assumes  $"ip \in vD(rt)"$

shows  $"nsqn\ rt\ ip = sqn\ rt\ ip"$

$\langle proof \rangle$

### lemma inv\_nsqn\_sqn [elim, simp]:

assumes  $"ip \in kD(rt)"$

and  $"\text{the } (flag\ rt\ ip) = inv"$

shows  $"nsqn\ rt\ ip = sqn\ rt\ ip - 1"$

$\langle proof \rangle$

### lemma iD\_nsqn\_sqn [elim, simp]:

assumes  $"ip \in iD(rt)"$

shows  $"nsqn\ rt\ ip = sqn\ rt\ ip - 1"$

$\langle proof \rangle$

### lemma nsqn\_update\_changed\_kno\_val [simp]: $"\bigwedge rt\ ip\ dsn\ dsk\ hops\ nhip.$

$rt \neq \text{update } rt\ ip\ (dsn, kno, val, hops, nhip, \{ })"$

$\implies \text{nsqn } (\text{update } rt \text{ ip } (dsn, kno, val, hops, nhip, \{\})) \text{ ip} = dsn$   
 $\langle \text{proof} \rangle$

**lemma** *nsqn\_addpreRT\_inv [simp]*:  
 $\text{"}\bigwedge rt \text{ dip npre dip}'. \text{ dip} \in kD(rt) \implies$   
 $\text{nsqn } (\text{the } (\text{addpreRT } rt \text{ dip npre})) \text{ dip}' = \text{nsqn } rt \text{ dip}'$   
 $\langle \text{proof} \rangle$

**lemma** *nsqn\_update\_other [simp]*:  
 fixes *dsn dsk flag hops dip nhip pre rt ip*  
 assumes  $\text{"dip} \neq \text{ip}"$   
 shows  $\text{"nsqn } (\text{update } rt \text{ ip } (dsn, dsk, flag, hops, nhip, pre)) \text{ dip} = \text{nsqn } rt \text{ dip}"$   
 $\langle \text{proof} \rangle$

**lemma** *nsqn\_invalidate\_eq*:  
 assumes  $\text{"dip} \in kD(rt)"$   
 and  $\text{"dests } dip = \text{Some } rsn"$   
 shows  $\text{"nsqn } (\text{invalidate } rt \text{ dests}) \text{ dip} = rsn - 1"$   
 $\langle \text{proof} \rangle$

**lemma** *nsqn\_invalidate\_other [simp]*:  
 assumes  $\text{"dip} \in kD(rt)"$   
 and  $\text{"dip} \notin \text{dom } \text{dests}"$   
 shows  $\text{"nsqn } (\text{invalidate } rt \text{ dests}) \text{ dip} = \text{nsqn } rt \text{ dip}"$   
 $\langle \text{proof} \rangle$

## 2.5.2 Comparing routes

**definition**

*fresher* ::  $\text{"}r \Rightarrow r' \Rightarrow \text{bool}"$  ( $\text{"}(\_ / \sqsubseteq \_)"$  [51, 51] 50)

where

$\text{"fresher } r \text{ } r' \equiv ((\text{nsqn}_r \text{ } r < \text{nsqn}_r \text{ } r') \vee (\text{nsqn}_r \text{ } r = \text{nsqn}_r \text{ } r' \wedge \pi_5(r) \geq \pi_5(r')))"$

**lemma** *fresherI1 [intro]*:  
 assumes  $\text{"nsqn}_r \text{ } r < \text{nsqn}_r \text{ } r'"$   
 shows  $\text{"}r \sqsubseteq r'"$   
 $\langle \text{proof} \rangle$

**lemma** *fresherI2 [intro]*:  
 assumes  $\text{"nsqn}_r \text{ } r = \text{nsqn}_r \text{ } r'"$   
 and  $\text{"}\pi_5(r) \geq \pi_5(r'"$   
 shows  $\text{"}r \sqsubseteq r'"$   
 $\langle \text{proof} \rangle$

**lemma** *fresherI [intro]*:  
 assumes  $\text{"}(\text{nsqn}_r \text{ } r < \text{nsqn}_r \text{ } r') \vee (\text{nsqn}_r \text{ } r = \text{nsqn}_r \text{ } r' \wedge \pi_5(r) \geq \pi_5(r'))"$   
 shows  $\text{"}r \sqsubseteq r'"$   
 $\langle \text{proof} \rangle$

**lemma** *fresherE [elim]*:  
 assumes  $\text{"}r \sqsubseteq r'"$   
 and  $\text{"nsqn}_r \text{ } r < \text{nsqn}_r \text{ } r' \implies P \text{ } r \text{ } r'"$   
 and  $\text{"nsqn}_r \text{ } r = \text{nsqn}_r \text{ } r' \wedge \pi_5(r) \geq \pi_5(r') \implies P \text{ } r \text{ } r'"$   
 shows  $\text{"}P \text{ } r \text{ } r'"$   
 $\langle \text{proof} \rangle$

**lemma** *fresher\_refl [simp]*:  $\text{"}r \sqsubseteq r"$   
 $\langle \text{proof} \rangle$

**lemma** *fresher\_trans [elim, trans]*:  
 $\text{"}[ x \sqsubseteq y; y \sqsubseteq z ] \implies x \sqsubseteq z"$   
 $\langle \text{proof} \rangle$

**lemma** *not\_fresher\_trans [elim, trans]*:

" $\llbracket \neg(x \sqsubseteq y); \neg(z \sqsubseteq x) \rrbracket \implies \neg(z \sqsubseteq y)$ "  
 <proof>

lemma fresher\_dsn\_flag\_hops\_const [simp]:  
 fixes dsn dsk dsk' flag hops nhip nhip' pre pre'  
 shows "(dsn, dsk, flag, hops, nhip, pre)  $\sqsubseteq$  (dsn, dsk', flag, hops, nhip', pre)"  
 <proof>

lemma addpre\_fresher [simp]: " $\bigwedge r \text{ npre}. r \sqsubseteq (\text{addpre } r \text{ npre})$ "  
 <proof>

### 2.5.3 Comparing routing tables

definition

rt\_fresher :: "ip  $\Rightarrow$  rt  $\Rightarrow$  rt  $\Rightarrow$  bool"

where

"rt\_fresher  $\equiv$   $\lambda$ dip rt rt'. (the ( $\sigma_{\text{route}}(\text{rt}, \text{dip})$ ))  $\sqsubseteq$  (the ( $\sigma_{\text{route}}(\text{rt}', \text{dip})$ ))"

abbreviation

rt\_fresher\_syn :: "rt  $\Rightarrow$  ip  $\Rightarrow$  rt  $\Rightarrow$  bool" (" $\_ / \sqsubseteq \_$ " [51, 999, 51] 50)

where

"rt1  $\sqsubseteq_i$  rt2  $\equiv$  rt\_fresher i rt1 rt2"

lemma rt\_fresher\_def':

"(rt1  $\sqsubseteq_i$  rt2) = (nsqn<sub>r</sub> (the (rt1 i)) < nsqn<sub>r</sub> (the (rt2 i)))  $\vee$   
 nsqn<sub>r</sub> (the (rt1 i)) = nsqn<sub>r</sub> (the (rt2 i))  $\wedge$   $\pi_5$  (the (rt2 i))  $\leq$   $\pi_5$  (the (rt1 i)))"

<proof>

lemma single\_rt\_fresher [intro]:

assumes "the (rt1 ip)  $\sqsubseteq$  the (rt2 ip)"

shows "rt1  $\sqsubseteq_{\text{ip}}$  rt2"

<proof>

lemma rt\_fresher\_single [intro]:

assumes "rt1  $\sqsubseteq_{\text{ip}}$  rt2"

shows "the (rt1 ip)  $\sqsubseteq$  the (rt2 ip)"

<proof>

lemma rt\_fresher\_def2:

assumes "dip  $\in$  kD(rt1)"

and "dip  $\in$  kD(rt2)"

shows "(rt1  $\sqsubseteq_{\text{dip}}$  rt2) = (nsqn rt1 dip < nsqn rt2 dip  
 $\vee$  (nsqn rt1 dip = nsqn rt2 dip  
 $\wedge$  the (dhops rt1 dip)  $\geq$  the (dhops rt2 dip)))"

<proof>

lemma rt\_fresherI1 [intro]:

assumes "dip  $\in$  kD(rt1)"

and "dip  $\in$  kD(rt2)"

and "nsqn rt1 dip < nsqn rt2 dip"

shows "rt1  $\sqsubseteq_{\text{dip}}$  rt2"

<proof>

lemma rt\_fresherI2 [intro]:

assumes "dip  $\in$  kD(rt1)"

and "dip  $\in$  kD(rt2)"

and "nsqn rt1 dip = nsqn rt2 dip"

and "the (dhops rt1 dip)  $\geq$  the (dhops rt2 dip)"

shows "rt1  $\sqsubseteq_{\text{dip}}$  rt2"

<proof>

lemma rt\_fresherE [elim]:

assumes "rt1  $\sqsubseteq_{\text{dip}}$  rt2"

and "dip  $\in$  kD(rt1)"

```

    and "dip ∈ kD(rt2)"
    and "[[ nsqn rt1 dip < nsqn rt2 dip ]] ⇒ P rt1 rt2 dip"
    and "[[ nsqn rt1 dip = nsqn rt2 dip;
      the (dhops rt1 dip) ≥ the (dhops rt2 dip) ]] ⇒ P rt1 rt2 dip"
  shows "P rt1 rt2 dip"
⟨proof⟩

lemma rt_fresher_refl [simp]: "rt ⊆dip rt"
⟨proof⟩

lemma rt_fresher_trans [elim, trans]:
  assumes "rt1 ⊆dip rt2"
    and "rt2 ⊆dip rt3"
  shows "rt1 ⊆dip rt3"
⟨proof⟩

lemma rt_fresher_if_Some [intro!]:
  assumes "the (rt dip) ⊆ r"
  shows "rt ⊆dip (λip. if ip = dip then Some r else rt ip)"
⟨proof⟩

definition rt_fresh_as :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_fresh_as ≡ λdip rt1 rt2. (rt1 ⊆dip rt2) ∧ (rt2 ⊆dip rt1)"

abbreviation
  rt_fresh_as_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" ("(_/ ≈ip _)" [51, 999, 51] 50)
where
  "rt1 ≈ip rt2 ≡ rt_fresh_as ip rt1 rt2"

lemma rt_fresh_as_refl [simp]: "∧rt dip. rt ≈dip rt"
⟨proof⟩

lemma rt_fresh_as_trans [simp, intro, trans]:
  "∧rt1 rt2 rt3 dip. [[ rt1 ≈dip rt2; rt2 ≈dip rt3 ]] ⇒ rt1 ≈dip rt3"
⟨proof⟩

lemma rt_fresh_asI [intro!]:
  assumes "rt1 ⊆dip rt2"
    and "rt2 ⊆dip rt1"
  shows "rt1 ≈dip rt2"
⟨proof⟩

lemma rt_fresh_as_fresherI [intro]:
  assumes "dip ∈ kD(rt1)"
    and "dip ∈ kD(rt2)"
    and "the (rt1 dip) ⊆ the (rt2 dip)"
    and "the (rt2 dip) ⊆ the (rt1 dip)"
  shows "rt1 ≈dip rt2"
⟨proof⟩

lemma nsqn_rt_fresh_asI:
  assumes "dip ∈ kD(rt)"
    and "dip ∈ kD(rt′)"
    and "nsqn rt dip = nsqn rt′ dip"
    and "π5(the (rt dip)) = π5(the (rt′ dip))"
  shows "rt ≈dip rt′"
⟨proof⟩

lemma rt_fresh_asE [elim]:
  assumes "rt1 ≈dip rt2"
    and "[[ rt1 ⊆dip rt2; rt2 ⊆dip rt1 ]] ⇒ P rt1 rt2 dip"
  shows "P rt1 rt2 dip"
⟨proof⟩

```

```

lemma rt_fresh_asD1 [dest]:
  assumes "rt1  $\approx_{dip}$  rt2"
  shows "rt1  $\sqsubseteq_{dip}$  rt2"
  <proof>

lemma rt_fresh_asD2 [dest]:
  assumes "rt1  $\approx_{dip}$  rt2"
  shows "rt2  $\sqsubseteq_{dip}$  rt1"
  <proof>

lemma rt_fresh_as_sym:
  assumes "rt1  $\approx_{dip}$  rt2"
  shows "rt2  $\approx_{dip}$  rt1"
  <proof>

lemma not_rt_fresh_asI1 [intro]:
  assumes " $\neg$  (rt1  $\sqsubseteq_{dip}$  rt2)"
  shows " $\neg$  (rt1  $\approx_{dip}$  rt2)"
  <proof>

lemma not_rt_fresh_asI2 [intro]:
  assumes " $\neg$  (rt2  $\sqsubseteq_{dip}$  rt1)"
  shows " $\neg$  (rt1  $\approx_{dip}$  rt2)"
  <proof>

lemma not_single_rt_fresher [elim]:
  assumes " $\neg$ (the (rt1 ip)  $\sqsubseteq$  the (rt2 ip))"
  shows " $\neg$ (rt1  $\sqsubseteq_{ip}$  rt2)"
  <proof>

lemmas not_single_rt_fresh_asI1 [intro] = not_rt_fresh_asI1 [OF not_single_rt_fresher]
lemmas not_single_rt_fresh_asI2 [intro] = not_rt_fresh_asI2 [OF not_single_rt_fresher]

lemma not_rt_fresher_single [elim]:
  assumes " $\neg$ (rt1  $\sqsubseteq_{ip}$  rt2)"
  shows " $\neg$ (the (rt1 ip)  $\sqsubseteq$  the (rt2 ip))"
  <proof>

lemma rt_fresh_as_nsqr:
  assumes "dip  $\in$  kD(rt1)"
  and "dip  $\in$  kD(rt2)"
  and "rt1  $\approx_{dip}$  rt2"
  shows "nsqrr (the (rt2 dip)) = nsqrr (the (rt1 dip))"
  <proof>

lemma rt_fresher_mapupd [intro!]:
  assumes "dip  $\in$  kD(rt)"
  and "the (rt dip)  $\sqsubseteq$  r"
  shows "rt  $\sqsubseteq_{dip}$  rt(dip  $\mapsto$  r)"
  <proof>

lemma rt_fresher_map_update_other [intro!]:
  assumes "dip  $\in$  kD(rt)"
  and "dip  $\neq$  ip"
  shows "rt  $\sqsubseteq_{dip}$  rt(ip  $\mapsto$  r)"
  <proof>

lemma rt_fresher_update_other [simp]:
  assumes inkD: "dip  $\in$  kD(rt)"
  and "dip  $\neq$  ip"
  shows "rt  $\sqsubseteq_{dip}$  update rt ip r"
  <proof>

```

**theorem** *rt\_fresher\_update* [simp]:

assumes "dip ∈ kD(rt)"  
and "the (dhops rt dip) ≥ 1"  
and "update\_arg\_wf r"  
shows "rt ⊆<sub>dip</sub> update rt ip r"  
<proof>

**theorem** *rt\_fresher\_invalidate* [simp]:

assumes "dip ∈ kD(rt)"  
and indests: "∀ rip ∈ dom(dests). rip ∈ vD(rt) ∧ sqn rt rip < the (dests rip)"  
shows "rt ⊆<sub>dip</sub> invalidate rt dests"  
<proof>

**lemma** *nsqn<sub>r</sub>\_invalidate* [simp]:

assumes "dip ∈ kD(rt)"  
and "dip ∈ dom(dests)"  
shows "nsqn<sub>r</sub> (the (invalidate rt dests dip)) = the (dests dip) - 1"  
<proof>

**lemma** *rt\_fresh\_as\_inc\_invalidate* [simp]:

assumes "dip ∈ kD(rt)"  
and "∀ rip ∈ dom(dests). rip ∈ vD(rt) ∧ the (dests rip) = inc (sqn rt rip)"  
shows "rt ≈<sub>dip</sub> invalidate rt dests"  
<proof>

**lemmas** *rt\_fresher\_inc\_invalidate* [simp] = *rt\_fresh\_as\_inc\_invalidate* [THEN *rt\_fresh\_asD1*]

**lemma** *rt\_fresh\_as\_addpreRT* [simp]:

assumes "ip ∈ kD(rt)"  
shows "rt ≈<sub>dip</sub> the (addpreRT rt ip npre)"  
<proof>

**lemmas** *rt\_fresher\_addpreRT* [simp] = *rt\_fresh\_as\_addpreRT* [THEN *rt\_fresh\_asD1*]

## 2.5.4 Strictly comparing routing tables

**definition** *rt\_strictly\_fresher* :: "ip ⇒ rt ⇒ rt ⇒ bool"

where

"rt\_strictly\_fresher ≡ λdip rt1 rt2. (rt1 ⊆<sub>dip</sub> rt2) ∧ ¬(rt1 ≈<sub>dip</sub> rt2)"

**abbreviation**

*rt\_strictly\_fresher\_syn* :: "rt ⇒ ip ⇒ rt ⇒ bool" ("(\_/ ⊆<sub>\_</sub> \_)" [51, 999, 51] 50)

where

"rt1 ⊆<sub>i</sub> rt2 ≡ rt\_strictly\_fresher i rt1 rt2"

**lemma** *rt\_strictly\_fresher\_def''*:

"rt1 ⊆<sub>i</sub> rt2 = ((rt1 ⊆<sub>i</sub> rt2) ∧ ¬(rt2 ⊆<sub>i</sub> rt1))"  
<proof>

**lemma** *rt\_strictly\_fresherI'* [intro]:

assumes "rt1 ⊆<sub>i</sub> rt2"  
and "¬(rt2 ⊆<sub>i</sub> rt1)"  
shows "rt1 ⊆<sub>i</sub> rt2"  
<proof>

**lemma** *rt\_strictly\_fresherE'* [elim]:

assumes "rt1 ⊆<sub>i</sub> rt2"  
and "[| rt1 ⊆<sub>i</sub> rt2; ¬(rt2 ⊆<sub>i</sub> rt1) |] ⇒ P rt1 rt2 i"  
shows "P rt1 rt2 i"  
<proof>

**lemma** *rt\_strictly\_fresherI* [intro]:

assumes "rt1 ⊆<sub>i</sub> rt2"  
and "¬(rt1 ≈<sub>i</sub> rt2)"



```

  shows "rt1  $\sqsubseteq_i$  rt2"
  <proof>

lemmas rt_strictly_fresher_singleI [elim] = rt_strictly_fresherI [OF single_rt_fresher]

lemma rt_strictly_fresherE [elim]:
  assumes "rt1  $\sqsubseteq_i$  rt2"
    and "[[ rt1  $\sqsubseteq_i$  rt2;  $\neg$ (rt1  $\approx_i$  rt2) ]  $\implies$  P rt1 rt2 i"
  shows "P rt1 rt2 i"
  <proof>

lemma rt_strictly_fresher_def':
  "rt1  $\sqsubseteq_i$  rt2 =
  (nsqnr (the (rt1 i)) < nsqnr (the (rt2 i))
   $\vee$  (nsqnr (the (rt1 i)) = nsqnr (the (rt2 i))  $\wedge$   $\pi_5$ (the (rt1 i)) >  $\pi_5$ (the (rt2 i))))"
  <proof>

lemma rt_strictly_fresher_fresherD [dest]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
  shows "the (rt1 dip)  $\sqsubseteq$  the (rt2 dip)"
  <proof>

lemma rt_strictly_fresher_not_fresh_asD [dest]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
  shows " $\neg$  rt1  $\approx_{dip}$  rt2"
  <proof>

lemma rt_strictly_fresher_trans [elim, trans]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
    and "rt2  $\sqsubseteq_{dip}$  rt3"
  shows "rt1  $\sqsubseteq_{dip}$  rt3"
  <proof>

lemma rt_strictly_fresher_irefl [simp]: " $\neg$  (rt  $\sqsubseteq_{dip}$  rt)"
  <proof>

lemma rt_fresher_trans_rt_strictly_fresher [elim, trans]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
    and "rt2  $\sqsubseteq_{dip}$  rt3"
  shows "rt1  $\sqsubseteq_{dip}$  rt3"
  <proof>

lemma rt_fresher_trans_rt_strictly_fresher' [elim, trans]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
    and "rt2  $\sqsubseteq_{dip}$  rt3"
  shows "rt1  $\sqsubseteq_{dip}$  rt3"
  <proof>

lemma rt_fresher_imp_nsqn_le:
  assumes "rt1  $\sqsubseteq_{ip}$  rt2"
    and "ip  $\in$  kD rt1"
    and "ip  $\in$  kD rt2"
  shows "nsqn rt1 ip  $\leq$  nsqn rt2 ip"
  <proof>

lemma rt_strictly_fresher_ltI [intro]:
  assumes "dip  $\in$  kD(rt1)"
    and "dip  $\in$  kD(rt2)"
    and "nsqn rt1 dip < nsqn rt2 dip"
  shows "rt1  $\sqsubseteq_{dip}$  rt2"
  <proof>

lemma rt_strictly_fresher_eqI [intro]:
  assumes "i  $\in$  kD(rt1)"

```

```

    and "i ∈ kD(rt2)"
    and "nsqn rt1 i = nsqn rt2 i"
    and "π5(the (rt2 i)) < π5(the (rt1 i))"
  shows "rt1 ⊑i rt2"
⟨proof⟩

```

```

lemma invalidate_rtsf_left [simp]:
  "∧ dests dip rt rt'. dests dip = None ⇒ (invalidate rt dests ⊑dip rt') = (rt ⊑dip rt')"
⟨proof⟩

```

```

lemma vD_invalidate_rt_strictly_fresher [simp]:
  assumes "dip ∈ vD(invalidate rt1 dests)"
  shows "(invalidate rt1 dests ⊑dip rt2) = (rt1 ⊑dip rt2)"
⟨proof⟩

```

```

lemma rt_strictly_fresher_update_other [elim!]:
  "∧ dip ip rt r rt'. [ dip ≠ ip; rt ⊑dip rt' ] ⇒ update rt ip r ⊑dip rt'"
⟨proof⟩

```

```

lemma addpreRT_strictly_fresher [simp]:
  assumes "dip ∈ kD(rt)"
  shows "(the (addpreRT rt dip npre) ⊑ip rt2) = (rt ⊑ip rt2)"
⟨proof⟩

```

```

lemma lt_sqn_imp_update_strictly_fresher:
  assumes "dip ∈ vD (rt2 nhip)"
  and *: "osn < sqn (rt2 nhip) dip"
  and **: "rt ≠ update rt dip (osn, kno, val, hops, nhip, {})"
  shows "update rt dip (osn, kno, val, hops, nhip, {}) ⊑dip rt2 nhip"
⟨proof⟩

```

```

lemma dhops_le_hops_imp_update_strictly_fresher:
  assumes "dip ∈ vD(rt2 nhip)"
  and sqn: "sqn (rt2 nhip) dip = osn"
  and hop: "the (dhops (rt2 nhip) dip) ≤ hops"
  and **: "rt ≠ update rt dip (osn, kno, val, Suc hops, nhip, {})"
  shows "update rt dip (osn, kno, val, Suc hops, nhip, {}) ⊑dip rt2 nhip"
⟨proof⟩

```

```

lemma nsqn_invalidate:
  assumes "dip ∈ kD(rt)"
  and "∀ ip ∈ dom(dests). ip ∈ vD(rt) ∧ the (dests ip) = inc (sqn rt ip)"
  shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
⟨proof⟩

```

end

## 2.6 Invariant proofs on individual processes

```

theory B_Seq_Invariants
imports "../..../AWN/Invariants" B_Aodv B_Aodv_Data B_Aodv_Predicates B_Fresher

```

begin

The proposition numbers are taken from the December 2013 version of the Fehnker et al technical report.

Proposition 7.2

```

lemma sequence_number_increases:
  "paodv i ⊨A onll ΓAODV (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
⟨proof⟩

```

```

lemma sequence_number_one_or_bigger:
  "paodv i ⊨ onl ΓAODV (λ(ξ, _). 1 ≤ sn ξ)"
⟨proof⟩

```

We can get rid of the onl/onll if desired...

**lemma sequence\_number\_increases':**

"paadv i  $\models_A (\lambda((\xi, \_), \_, (\xi', \_)). \text{sn } \xi \leq \text{sn } \xi')$ "  
 ⟨proof⟩

**lemma sequence\_number\_one\_or\_bigger':**

"paadv i  $\models (\lambda(\xi, \_). 1 \leq \text{sn } \xi)$ "  
 ⟨proof⟩

**lemma sip\_in\_kD:**

"paadv i  $\models \text{onl } \Gamma_{AODV} (\lambda(\xi, l). l \in (\{\text{PAodv-:7}\} \cup \{\text{PAodv-:5}\} \cup \{\text{PRrep-:0..PRrep-:4}\} \cup \{\text{PRreq-:0..PRreq-:3}\}) \rightarrow \text{sip } \xi \in \text{kD } (rt \ \xi))$ "  
 ⟨proof⟩

**lemma addpreRT\_partly\_welldefined:**

"paadv i  $\models$   
 $\text{onl } \Gamma_{AODV} (\lambda(\xi, l). (l \in \{\text{PRreq-:16..PRreq-:18}\} \cup \{\text{PRrep-:1..PRrep-:5}\} \rightarrow \text{dip } \xi \in \text{kD } (rt \ \xi))$   
 $\wedge (l \in \{\text{PRreq-:3..PRreq-:17}\} \rightarrow \text{oip } \xi \in \text{kD } (rt \ \xi)))$ "  
 ⟨proof⟩

Proposition 7.38

**lemma includes\_nhip:**

"paadv i  $\models \text{onl } \Gamma_{AODV} (\lambda(\xi, l). \forall \text{dip} \in \text{kD}(rt \ \xi). \text{the } (\text{nhop } (rt \ \xi) \ \text{dip}) \in \text{kD}(rt \ \xi))$ "  
 ⟨proof⟩

Proposition 7.22: needed in Proposition 7.4

**lemma addpreRT\_welldefined:**

"paadv i  $\models \text{onl } \Gamma_{AODV} (\lambda(\xi, l). (l \in \{\text{PRreq-:16..PRreq-:18}\} \rightarrow \text{dip } \xi \in \text{kD } (rt \ \xi)) \wedge$   
 $(l = \text{PRreq-:17} \rightarrow \text{oip } \xi \in \text{kD } (rt \ \xi)) \wedge$   
 $(l = \text{PRrep-:4} \rightarrow \text{dip } \xi \in \text{kD } (rt \ \xi)) \wedge$   
 $(l = \text{PRrep-:5} \rightarrow (\text{the } (\text{nhop } (rt \ \xi) \ (\text{dip } \xi))) \in \text{kD } (rt \ \xi)))$ "  
 (is " $\_ \models \text{onl } \Gamma_{AODV} \ ?P$ ")  
 ⟨proof⟩

Proposition 7.4

**lemma known\_destinations\_increase:**

"paadv i  $\models_A \text{onll } \Gamma_{AODV} (\lambda((\xi, \_), \_, (\xi', \_)). \text{kD } (rt \ \xi) \subseteq \text{kD } (rt \ \xi'))$ "  
 ⟨proof⟩

Proposition 7.5

**lemma rreqs\_increase:**

"paadv i  $\models_A \text{onll } \Gamma_{AODV} (\lambda((\xi, \_), \_, (\xi', \_)). \text{rreqs } \xi \subseteq \text{rreqs } \xi')$ "  
 ⟨proof⟩

**lemma dests\_bigger\_than\_sqn:**

"paadv i  $\models \text{onl } \Gamma_{AODV} (\lambda(\xi, l). l \in \{\text{PAodv-:15..PAodv-:19}\}$   
 $\cup \{\text{PPkt-:7..PPkt-:11}\}$   
 $\cup \{\text{PRreq-:9..PRreq-:13}\}$   
 $\cup \{\text{PRreq-:21..PRreq-:25}\}$   
 $\cup \{\text{PRrep-:9..PRrep-:13}\}$   
 $\cup \{\text{PRerr-:1..PRerr-:5}\}$   
 $\rightarrow (\forall \text{ip} \in \text{dom}(\text{dests } \xi). \text{ip} \in \text{kD}(rt \ \xi) \wedge \text{sqn } (rt \ \xi) \ \text{ip} \leq \text{the } (\text{dests } \xi \ \text{ip}))$ "  
 ⟨proof⟩

Proposition 7.6

**lemma sqns\_increase:**

"paadv i  $\models_A \text{onll } \Gamma_{AODV} (\lambda((\xi, \_), \_, (\xi', \_)). \forall \text{ip}. \text{sqn } (rt \ \xi) \ \text{ip} \leq \text{sqn } (rt \ \xi') \ \text{ip})$ "  
 ⟨proof⟩

Proposition 7.7

**lemma ip\_constant:**

"paadv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_). ip \xi = i$ )"  
 <proof>

Proposition 7.8

**lemma sender\_ip\_valid':**

"paadv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), a, \_). anycast (\lambda m. not\_Pkt m \longrightarrow msg\_sender m = ip \xi) a$ )"  
 <proof>

**lemma sender\_ip\_valid:**

"paadv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), a, \_). anycast (\lambda m. not\_Pkt m \longrightarrow msg\_sender m = i) a$ )"  
 <proof>

**lemma received\_msg\_inv:**

"paadv i  $\models$  (recvmsg P  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). l \in \{PAadv-:1\} \longrightarrow P (msg \xi)$ )"  
 <proof>

Proposition 7.9

**lemma sip\_not\_ip':**

"paadv i  $\models$  (recvmsg ( $\lambda m. not\_Pkt m \longrightarrow msg\_sender m \neq i$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_). sip \xi \neq ip \xi$ )"  
 <proof>

**lemma sip\_not\_ip:**

"paadv i  $\models$  (recvmsg ( $\lambda m. not\_Pkt m \longrightarrow msg\_sender m \neq i$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_). sip \xi \neq i$ )"  
 <proof>

Neither *sip\_not\_ip'* nor *sip\_not\_ip* is needed to show loop freedom.

Proposition 7.10

**lemma hop\_count\_positive:**

"paadv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_). \forall ip \in kD (rt \xi). the (dhops (rt \xi) ip) \geq 1$ )"  
 <proof>

**lemma rreq\_dip\_in\_vD\_dip\_eq\_ip:**

"paadv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). (l \in \{PRreq-:16..PRreq-:18\} \longrightarrow dip \xi \in vD(rt \xi))$   
 $\wedge (l \in \{PRreq-:5, PRreq-:6\} \longrightarrow dip \xi = ip \xi)$   
 $\wedge (l \in \{PRreq-:15..PRreq-:18\} \longrightarrow dip \xi \neq ip \xi)$ )"  
 <proof>

**lemma rrep\_dip\_in\_vD:**

"paadv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). (l \in \{PRrep-:4..PRrep-:6\} \longrightarrow dip \xi \in vD(rt \xi))$ )"  
 <proof>

Proposition 7.11

**lemma anycast\_msg\_zhops:**

" $\bigwedge rreqid dip dsn dsk oip osn sip.$   
 paadv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda(\_, a, \_). anycast msg\_zhops a$ )"  
 <proof>

**lemma hop\_count\_zero\_oip\_dip\_sip:**

"paadv i  $\models$  (recvmsg msg\_zhops  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l).$   
 $(l \in \{PAadv-:4..PAadv-:5\} \cup \{PRreq-:n/n. True\} \longrightarrow$   
 $(hops \xi = 0 \longrightarrow oip \xi = sip \xi))$   
 $\wedge$   
 $((l \in \{PAadv-:6..PAadv-:7\} \cup \{PRrep-:n/n. True\} \longrightarrow$   
 $(hops \xi = 0 \longrightarrow dip \xi = sip \xi))))$ "  
 <proof>

**lemma osn\_rreq:**

"paadv i  $\models$  (recvmsg rreq\_rrep\_sn  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l).$   
 $l \in \{PAadv-:4, PAadv-:5\} \cup \{PRreq-:n/n. True\} \longrightarrow 1 \leq osn \xi$ )"  
 <proof>

**lemma osn\_rreq':**

"paadv i  $\models$  (recvmsg ( $\lambda m. rreq\_rrep\_sn\ m \wedge msg\_zhops\ m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, 1)$ ).  
 $1 \in \{PAadv-:4, PAadv-:5\} \cup \{PRreq-:n/n. True\} \rightarrow 1 \leq osn\ \xi$ )"  
 <proof>

**lemma dsn\_rrep:**

"paadv i  $\models$  (recvmsg rreq\_rrep\_sn  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, 1)$ .  
 $1 \in \{PAadv-:6, PAadv-:7\} \cup \{PRrep-:n/n. True\} \rightarrow 1 \leq dsn\ \xi$ )"  
 <proof>

**lemma dsn\_rrep':**

"paadv i  $\models$  (recvmsg ( $\lambda m. rreq\_rrep\_sn\ m \wedge msg\_zhops\ m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, 1)$ .  
 $1 \in \{PAadv-:6, PAadv-:7\} \cup \{PRrep-:n/n. True\} \rightarrow 1 \leq dsn\ \xi$ )"  
 <proof>

**lemma hop\_count\_zero\_oip\_dip\_sip':**

"paadv i  $\models$  (recvmsg ( $\lambda m. rreq\_rrep\_sn\ m \wedge msg\_zhops\ m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, 1)$ .  
 $(1 \in \{PAadv-:4..PAadv-:5\} \cup \{PRreq-:n/n. True\} \rightarrow$   
 $(hops\ \xi = 0 \rightarrow oip\ \xi = sip\ \xi))$   
 $\wedge$   
 $((1 \in \{PAadv-:6..PAadv-:7\} \cup \{PRrep-:n/n. True\} \rightarrow$   
 $(hops\ \xi = 0 \rightarrow dip\ \xi = sip\ \xi))))$ "  
 <proof>

Proposition 7.12

**lemma zero\_seq\_unk\_hops\_one':**

"paadv i  $\models$  (recvmsg ( $\lambda m. rreq\_rrep\_sn\ m \wedge msg\_zhops\ m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_)$ .  
 $\forall dip \in kD(rt\ \xi). (sqn\ (rt\ \xi)\ dip = 0 \rightarrow sqnf\ (rt\ \xi)\ dip = unk)$   
 $\wedge (sqnf\ (rt\ \xi)\ dip = unk \rightarrow the\ (dhops\ (rt\ \xi)\ dip) = 1)$   
 $\wedge (the\ (dhops\ (rt\ \xi)\ dip) = 1 \rightarrow the\ (nhop\ (rt\ \xi)\ dip) = dip))$ "  
 <proof>

**lemma zero\_seq\_unk\_hops\_one:**

"paadv i  $\models$  (recvmsg ( $\lambda m. rreq\_rrep\_sn\ m \wedge msg\_zhops\ m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_)$ .  
 $\forall dip \in kD(rt\ \xi). (sqn\ (rt\ \xi)\ dip = 0 \rightarrow (sqnf\ (rt\ \xi)\ dip = unk$   
 $\wedge the\ (dhops\ (rt\ \xi)\ dip) = 1$   
 $\wedge the\ (nhop\ (rt\ \xi)\ dip) = dip))$ "  
 <proof>

**lemma kD\_unk\_or\_atleast\_one:**

"paadv i  $\models$  (recvmsg rreq\_rrep\_sn  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, 1)$ .  
 $\forall dip \in kD(rt\ \xi). \pi_3(the\ (rt\ \xi)\ dip) = unk \vee 1 \leq \pi_2(the\ (rt\ \xi)\ dip))$ "  
 <proof>

Proposition 7.13

**lemma rreq\_rrep\_sn\_any\_step\_invariant:**

"paadv i  $\models_A$  (recvmsg rreq\_rrep\_sn  $\rightarrow$ ) onll  $\Gamma_{AODV}$  ( $\lambda(\_, a, \_)$ . anycast rreq\_rrep\_sn a)"  
 <proof>

Proposition 7.14

**lemma rreq\_rrep\_fresh\_any\_step\_invariant:**

"paadv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), a, \_)$ . anycast (rreq\_rrep\_fresh (rt  $\xi$ )) a)"  
 <proof>

Proposition 7.15

**lemma rerr\_invalid\_any\_step\_invariant:**

"paadv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), a, \_)$ . anycast (rerr\_invalid (rt  $\xi$ )) a)"  
 <proof>

Proposition 7.16

Some well-definedness obligations are irrelevant for the Isabelle development:

1. In each routing table there is at most one entry for each destination: guaranteed by type.

2. In each store of queued data packets there is at most one data queue for each destination: guaranteed by structure.
3. Whenever a set of pairs  $(rip, rsn)$  is assigned to the variable  $dests$  of type  $ip \rightarrow sqn$ , or to the first argument of the function  $rerr$ , this set is a partial function, i.e., there is at most one entry  $(rip, rsn)$  for each destination  $rip$ : guaranteed by type.

**lemma** *dests\_vD\_inc\_sqn*:

```
"paadv i  $\models$ 
  on1  $\Gamma_{AODV} (\lambda(\xi, l). (l \in \{PAadv-:15, PPkt-:7, PRreq-:9, PRreq-:21, PRrep-:9\}$ 
     $\rightarrow (\forall ip \in \text{dom}(dests \ \xi). ip \in vD(rt \ \xi) \wedge \text{the}(dests \ \xi \ ip) = \text{inc}(sqn(rt \ \xi) \ ip)))$ 
   $\wedge (l = PRerr-:1$ 
     $\rightarrow (\forall ip \in \text{dom}(dests \ \xi). ip \in vD(rt \ \xi) \wedge \text{the}(dests \ \xi \ ip) > sqn(rt \ \xi) \ ip)))"$ 
<proof>
```

Proposition 7.27

**lemma** *route\_tables\_fresher*:

```
"paadv i  $\models_A (\text{recvm}sg \ rreq\_rrep\_sn \rightarrow) \text{onll} \ \Gamma_{AODV} (\lambda((\xi, \_), \_, (\xi', \_)).$ 
 $\forall dip \in kD(rt \ \xi). rt \ \xi \sqsubseteq_{dip} rt \ \xi')$ "
<proof>
```

end

## 2.7 The quality increases predicate

**theory** *B\_Quality\_Increases*

**imports** *B\_Aadv\_Predicates B\_Fresher*

**begin**

**definition** *quality\_increases* :: "state  $\Rightarrow$  state  $\Rightarrow$  bool"

```
where "quality_increases  $\xi \ \xi' \equiv (\forall dip \in kD(rt \ \xi). dip \in kD(rt \ \xi') \wedge rt \ \xi \sqsubseteq_{dip} rt \ \xi')$ 
 $\wedge (\forall dip. sqn(rt \ \xi) \ dip \leq sqn(rt \ \xi') \ dip)"$ 
```

**lemma** *quality\_increasesI [intro!]*:

```
assumes " $\wedge dip. dip \in kD(rt \ \xi) \implies dip \in kD(rt \ \xi')$ "
  and " $\wedge dip. \llbracket dip \in kD(rt \ \xi); dip \in kD(rt \ \xi') \rrbracket \implies rt \ \xi \sqsubseteq_{dip} rt \ \xi'$ "
  and " $\wedge dip. sqn(rt \ \xi) \ dip \leq sqn(rt \ \xi') \ dip"$ 
shows "quality_increases  $\xi \ \xi'$ "
<proof>
```

**lemma** *quality\_increasesE [elim]*:

```
fixes dip
assumes "quality_increases  $\xi \ \xi'$ "
  and "dip  $\in kD(rt \ \xi)$ "
  and " $\llbracket dip \in kD(rt \ \xi'); rt \ \xi \sqsubseteq_{dip} rt \ \xi'; sqn(rt \ \xi) \ dip \leq sqn(rt \ \xi') \ dip \rrbracket \implies R \ dip \ \xi \ \xi'$ "
shows "R dip  $\xi \ \xi'$ "
<proof>
```

**lemma** *quality\_increases\_rt\_fresherD [dest]*:

```
fixes ip
assumes "quality_increases  $\xi \ \xi'$ "
  and "ip  $\in kD(rt \ \xi)$ "
shows "rt  $\xi \sqsubseteq_{ip} rt \ \xi'$ "
<proof>
```

**lemma** *quality\_increases\_sqnE [elim]*:

```
fixes dip
assumes "quality_increases  $\xi \ \xi'$ "
  and "sqn(rt  $\xi$ ) dip  $\leq sqn(rt \ \xi') \ dip \implies R \ dip \ \xi \ \xi'$ "
shows "R dip  $\xi \ \xi'$ "
<proof>
```

**lemma** *quality\_increases\_refl [intro, simp]*: "quality\_increases  $\xi \ \xi$ "

*<proof>*

```
lemma strictly_fresher_quality_increases_right [elim]:  
  fixes  $\sigma$   $\sigma'$  dip  
  assumes "rt ( $\sigma$  i)  $\sqsubset_{\text{dip}}$  rt ( $\sigma$  nhip)"  
    and qinc: "quality_increases ( $\sigma$  nhip) ( $\sigma'$  nhip)"  
    and "dip  $\in$  kD(rt ( $\sigma$  nhip))"  
  shows "rt ( $\sigma$  i)  $\sqsubset_{\text{dip}}$  rt ( $\sigma'$  nhip)"  
<proof>
```

```
lemma kD_quality_increases [elim]:  
  assumes "i  $\in$  kD(rt  $\xi$ )"  
    and "quality_increases  $\xi$   $\xi'$ "  
  shows "i  $\in$  kD(rt  $\xi')$ "  
<proof>
```

```
lemma kD_nsqn_quality_increases [elim]:  
  assumes "i  $\in$  kD(rt  $\xi$ )"  
    and "quality_increases  $\xi$   $\xi'$ "  
  shows "i  $\in$  kD(rt  $\xi')$   $\wedge$  nsqn (rt  $\xi$ ) i  $\leq$  nsqn (rt  $\xi')$  i"  
<proof>
```

```
lemma nsqn_quality_increases [elim]:  
  assumes "i  $\in$  kD(rt  $\xi$ )"  
    and "quality_increases  $\xi$   $\xi'$ "  
  shows "nsqn (rt  $\xi$ ) i  $\leq$  nsqn (rt  $\xi')$  i"  
<proof>
```

```
lemma kD_nsqn_quality_increases_trans [elim]:  
  assumes "i  $\in$  kD(rt  $\xi$ )"  
    and "s  $\leq$  nsqn (rt  $\xi$ ) i"  
    and "quality_increases  $\xi$   $\xi'$ "  
  shows "i  $\in$  kD(rt  $\xi')$   $\wedge$  s  $\leq$  nsqn (rt  $\xi')$  i"  
<proof>
```

```
lemma nsqn_quality_increases_nsqn_lt_lt [elim]:  
  assumes "i  $\in$  kD(rt  $\xi$ )"  
    and "quality_increases  $\xi$   $\xi'$ "  
    and "s < nsqn (rt  $\xi$ ) i"  
  shows "s < nsqn (rt  $\xi')$  i"  
<proof>
```

```
lemma nsqn_quality_increases_dhops [elim]:  
  assumes "i  $\in$  kD(rt  $\xi$ )"  
    and "quality_increases  $\xi$   $\xi'$ "  
    and "nsqn (rt  $\xi$ ) i = nsqn (rt  $\xi')$  i"  
  shows "the (dhops (rt  $\xi$ ) i)  $\geq$  the (dhops (rt  $\xi')$  i)"  
<proof>
```

```
lemma nsqn_quality_increases_nsqn_eq_le [elim]:  
  assumes "i  $\in$  kD(rt  $\xi$ )"  
    and "quality_increases  $\xi$   $\xi'$ "  
    and "s = nsqn (rt  $\xi$ ) i"  
  shows "s < nsqn (rt  $\xi')$  i  $\vee$  (s = nsqn (rt  $\xi')$  i  $\wedge$  the (dhops (rt  $\xi$ ) i)  $\geq$  the (dhops (rt  $\xi')$  i))"  
<proof>
```

```
lemma quality_increases_rreq_rrep_props [elim]:  
  fixes sn ip hops sip  
  assumes qinc: "quality_increases ( $\sigma$  sip) ( $\sigma'$  sip)"  
    and "1  $\leq$  sn"  
    and *: "ip  $\in$  kD(rt ( $\sigma$  sip))  $\wedge$  sn  $\leq$  nsqn (rt ( $\sigma$  sip)) ip  
       $\wedge$  (nsqn (rt ( $\sigma$  sip)) ip = sn  
         $\rightarrow$  (the (dhops (rt ( $\sigma$  sip)) ip)  $\leq$  hops  
           $\vee$  the (flag (rt ( $\sigma$  sip)) ip) = inv))"
```

```

shows "ip∈kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
      ∧ (nsqn (rt (σ' sip)) ip = sn
        → (the (dhops (rt (σ' sip)) ip) ≤ hops
            ∨ the (flag (rt (σ' sip)) ip) = inv))"
  (is "_ ∧ ?nsqnafter")
⟨proof⟩

```

lemma quality\_increases\_rreq\_rrep\_props':

```

fixes sn ip hops sip
assumes "∀j. quality_increases (σ j) (σ' j)"
and "1 ≤ sn"
and *: "ip∈kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
       ∧ (nsqn (rt (σ sip)) ip = sn
         → (the (dhops (rt (σ sip)) ip) ≤ hops
             ∨ the (flag (rt (σ sip)) ip) = inv))"
shows "ip∈kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
      ∧ (nsqn (rt (σ' sip)) ip = sn
        → (the (dhops (rt (σ' sip)) ip) ≤ hops
            ∨ the (flag (rt (σ' sip)) ip) = inv))"
⟨proof⟩

```

lemma rteq\_quality\_increases:

```

assumes "∀j. j ≠ i → quality_increases (σ j) (σ' j)"
and "rt (σ' i) = rt (σ i)"
shows "∀j. quality_increases (σ j) (σ' j)"
⟨proof⟩

```

definition msg\_fresh :: "(ip ⇒ state) ⇒ msg ⇒ bool"

```

where "msg_fresh σ m ≡
  case m of Rreq hopsc _ _ _ oipc osnc sipc ⇒ osnc ≥ 1 ∧ (sipc ≠ oipc →
    oipc∈kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) oipc ≥ osnc
    ∧ (nsqn (rt (σ sipc)) oipc = osnc
      → (hopsc ≥ the (dhops (rt (σ sipc)) oipc)
          ∨ the (flag (rt (σ sipc)) oipc) = inv)))
  | Rrep hopsc dipc dsnc _ sipc ⇒ dsnc ≥ 1 ∧ (sipc ≠ dipc →
    dipc∈kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) dipc ≥ dsnc
    ∧ (nsqn (rt (σ sipc)) dipc = dsnc
      → (hopsc ≥ the (dhops (rt (σ sipc)) dipc)
          ∨ the (flag (rt (σ sipc)) dipc) = inv)))
  | Rerr destsc sipc ⇒ (∀ripc∈dom(destsc). (ripc∈kD(rt (σ sipc))
    ∧ the (destsc ripc) - 1 ≤ nsqn (rt (σ sipc)) ripc))
  | _ ⇒ True"

```

lemma msg\_fresh [simp]:

```

"∧hops rreqid dip dsn dsk oip osn sip.
  msg_fresh σ (Rreq hops rreqid dip dsn dsk oip osn sip) =
    (osn ≥ 1 ∧ (sip ≠ oip → oip∈kD(rt (σ sip))
      ∧ nsqn (rt (σ sip)) oip ≥ osn
      ∧ (nsqn (rt (σ sip)) oip = osn
        → (hops ≥ the (dhops (rt (σ sip)) oip)
            ∨ the (flag (rt (σ sip)) oip) = inv))))))"
"∧hops dip dsn oip sip. msg_fresh σ (Rrep hops dip dsn oip sip) =
  (dsn ≥ 1 ∧ (sip ≠ dip → dip∈kD(rt (σ sip))
    ∧ nsqn (rt (σ sip)) dip ≥ dsn
    ∧ (nsqn (rt (σ sip)) dip = dsn
      → (hops ≥ the (dhops (rt (σ sip)) dip)
          ∨ the (flag (rt (σ sip)) dip) = inv))))))"
"∧dests sip. msg_fresh σ (Rerr dests sip) =
  (∀ripc∈dom(dests). (ripc∈kD(rt (σ sip))
    ∧ the (dests ripc) - 1 ≤ nsqn (rt (σ sip)) ripc))"
"∧d dip. msg_fresh σ (Newpkt d dip) = True"
"∧d dip sip. msg_fresh σ (Pkt d dip sip) = True"
⟨proof⟩

```



**lemma** *msg\_fresh\_inc\_sn* [*simp*, *elim*]:

"*msg\_fresh*  $\sigma$   $m \implies$  *rreq\_rrep\_sn*  $m$ "  
<*proof*>

**lemma** *recv\_msg\_fresh\_inc\_sn* [*simp*, *elim*]:

"*orecvmsg* (*msg\_fresh*)  $\sigma$   $m \implies$  *recvmsg* *rreq\_rrep\_sn*  $m$ "  
<*proof*>

**lemma** *rreq\_nsqn\_is\_fresh* [*simp*]:

*fixes*  $\sigma$  *msg* *hops* *rreqid* *dip* *dsn* *dsk* *oip* *osn* *sip*  
*assumes* "*rreq\_rrep\_fresh* (*rt* ( $\sigma$  *sip*)) (*Rreq* *hops* *rreqid* *dip* *dsn* *dsk* *oip* *osn* *sip*)"  
  *and* "*rreq\_rrep\_sn* (*Rreq* *hops* *rreqid* *dip* *dsn* *dsk* *oip* *osn* *sip*)"  
*shows* "*msg\_fresh*  $\sigma$  (*Rreq* *hops* *rreqid* *dip* *dsn* *dsk* *oip* *osn* *sip*)"  
  (*is* "*msg\_fresh*  $\sigma$  ?*msg*")  
<*proof*>

**lemma** *rrep\_nsqn\_is\_fresh* [*simp*]:

*fixes*  $\sigma$  *msg* *hops* *dip* *dsn* *oip* *sip*  
*assumes* "*rreq\_rrep\_fresh* (*rt* ( $\sigma$  *sip*)) (*Rrep* *hops* *dip* *dsn* *oip* *sip*)"  
  *and* "*rreq\_rrep\_sn* (*Rrep* *hops* *dip* *dsn* *oip* *sip*)"  
*shows* "*msg\_fresh*  $\sigma$  (*Rrep* *hops* *dip* *dsn* *oip* *sip*)"  
  (*is* "*msg\_fresh*  $\sigma$  ?*msg*")  
<*proof*>

**lemma** *rerr\_nsqn\_is\_fresh* [*simp*]:

*fixes*  $\sigma$  *msg* *dests* *sip*  
*assumes* "*rerr\_invalid* (*rt* ( $\sigma$  *sip*)) (*Rerr* *dests* *sip*)"  
*shows* "*msg\_fresh*  $\sigma$  (*Rerr* *dests* *sip*)"  
  (*is* "*msg\_fresh*  $\sigma$  ?*msg*")  
<*proof*>

**lemma** *quality\_increases\_msg\_fresh* [*elim*]:

*assumes* *qinc*: " $\forall j. \text{quality\_increases } (\sigma \ j) (\sigma' \ j)$ "  
  *and* "*msg\_fresh*  $\sigma \ m$ "  
*shows* "*msg\_fresh*  $\sigma' \ m$ "  
<*proof*>

**end**

## 2.8 The 'open' AODV model

**theory** *B\_OAodv*

**imports** *B\_Aodv* "../.../AWN/OAWN\_SOS\_Labels" "../.../AWN/OAWN\_Convert"

**begin**

Definitions for stating and proving global network properties over individual processes.

**definition**  $\sigma_{AODV}'$  :: " $((ip \Rightarrow state) \times ((state, msg, pseqp, pseqp \ label) \ seqp)) \ set$ "  
**where** " $\sigma_{AODV}' \equiv \{(\lambda i. \text{aodv\_init } i, \Gamma_{AODV} \ PAodv)\}$ "

**abbreviation** *opaodv*

:: " $ip \Rightarrow ((ip \Rightarrow state) \times (state, msg, pseqp, pseqp \ label) \ seqp, msg \ seq\_action) \ automaton$ "

**where**

"*opaodv*  $i \equiv \{ \mid \text{init} = \sigma_{AODV}', \text{trans} = \text{oseqp\_sos } \Gamma_{AODV} \ i \}$ "

**lemma** *initiali\_aodv* [*intro!*, *simp*]: "*initiali*  $i$  (*init* (*opaodv*  $i$ )) (*init* (*paodv*  $i$ ))"

<*proof*>

**lemma** *oaodv\_control\_within* [*simp*]: "*control\_within*  $\Gamma_{AODV}$  (*init* (*opaodv*  $i$ ))"

<*proof*>

**lemma**  $\sigma_{AODV}'\_labels$  [*simp*]: " $(\sigma, p) \in \sigma_{AODV}' \implies \text{labels } \Gamma_{AODV} \ p = \{PAodv-:0\}$ "

<*proof*>

**lemma** *oaodv\_init\_kD\_empty* [*simp*]:

```
"( $\sigma$ , p)  $\in$   $\sigma_{AODV}$ '  $\implies$  kD (rt ( $\sigma$  i)) = {}"
⟨proof⟩
```

```
lemma oaodv_init_vD_empty [simp]:
```

```
"( $\sigma$ , p)  $\in$   $\sigma_{AODV}$ '  $\implies$  vD (rt ( $\sigma$  i)) = {}"
⟨proof⟩
```

```
lemma oaodv_trans: "trans (opaodv i) = oseqp_sos  $\Gamma_{AODV}$  i"
```

```
⟨proof⟩
```

```
declare
```

```
oseq_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros]
oseq_step_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros]
```

```
end
```

## 2.9 Global invariant proofs over sequential processes

```
theory B_Global_Invariants
```

```
imports B_Seq_Invariants
```

```
  B_Aodv_Predicates
```

```
  B_Fresher
```

```
  B_Quality_Increases
```

```
  "../.../AWN/OAWN_Convert"
```

```
  B_OAodv
```

```
begin
```

```
lemma other_quality_increases [elim]:
```

```
  assumes "other quality_increases I  $\sigma$   $\sigma$ '"
  shows " $\forall j$ . quality_increases ( $\sigma$  j) ( $\sigma$ ' j)"
⟨proof⟩
```

```
lemma weaken_otherwith [elim]:
```

```
  fixes m
  assumes *: "otherwith P I (orecvmsg Q)  $\sigma$   $\sigma$ ' a"
  and weakenP: " $\bigwedge \sigma m$ . P  $\sigma$  m  $\implies$  P'  $\sigma$  m"
  and weakenQ: " $\bigwedge \sigma m$ . Q  $\sigma$  m  $\implies$  Q'  $\sigma$  m"
  shows "otherwith P' I (orecvmsg Q')  $\sigma$   $\sigma$ ' a"
⟨proof⟩
```

```
lemma oreceived_msg_inv:
```

```
  assumes other: " $\bigwedge \sigma \sigma' m$ . [ $P \sigma m$ ; other Q {i}  $\sigma \sigma'$ ]  $\implies$  P  $\sigma' m$ "
  and local: " $\bigwedge \sigma m$ . P  $\sigma$  m  $\implies$  P ( $\sigma$ (i :=  $\sigma$  i(msg := m))) m"
  shows "opaodv i  $\models$  (otherwith Q {i} (orecvmsg P), other Q {i}  $\rightarrow$ )
  onl  $\Gamma_{AODV}$  ( $\lambda(\sigma, l)$ . l  $\in$  {PAodv-:1}  $\rightarrow$  P  $\sigma$  (msg ( $\sigma$  i)))"
⟨proof⟩
```

(Equivalent to) Proposition 7.27

```
lemma local_quality_increases:
```

```
"paodv i  $\models_A$  (rcvmsg rreq_rrep_sn  $\rightarrow$ ) onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_)$ ,  $\_$ , ( $\xi'$ ,  $\_$ )). quality_increases  $\xi$   $\xi'$ )"
⟨proof⟩
```

```
lemmas olocal_quality_increases =
```

```
  open_seq_step_invariant [OF local_quality_increases initiali_aodv oaodv_trans aodv_trans,
  simplified seql_onll_swap]
```

```
lemma oquality_increases:
```

```
"opaodv i  $\models_A$  (otherwith quality_increases {i} (orecvmsg ( $\lambda \_$ . rreq_rrep_sn)),
  other quality_increases {i}  $\rightarrow$ )
  onll  $\Gamma_{AODV}$  ( $\lambda((\sigma, \_)$ ,  $\_$ , ( $\sigma'$ ,  $\_$ )).  $\forall j$ . quality_increases ( $\sigma$  j) ( $\sigma'$  j))"
(is " $\_ \models_A$  (?S,  $\_ \rightarrow$ )  $\_$ ")
⟨proof⟩
```

```
lemma rreq_rrep_nsqn_fresh_any_step_invariant:
```

"opaadv i  $\models_A$  (act (recvmsg rreq\_rrep\_sn), other A {i}  $\rightarrow$ )  
 onll  $\Gamma_{AODV}$  ( $\lambda((\sigma, \_), a, \_)$ . anycast (msg\_fresh  $\sigma$ ) a)"  
 <proof>

**lemma oreceived\_rreq\_rrep\_nsqn\_fresh\_inv:**

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onll  $\Gamma_{AODV}$  ( $\lambda(\sigma, l)$ .  $l \in \{PAodv-:1\} \rightarrow$  msg\_fresh  $\sigma$  (msg ( $\sigma$  i)))"  
 <proof>

**lemma oquality\_increases\_nsqn\_fresh:**

"opaadv i  $\models_A$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onll  $\Gamma_{AODV}$  ( $\lambda((\sigma, \_), \_, (\sigma', \_))$ .  $\forall j$ . quality\_increases ( $\sigma$  j) ( $\sigma'$  j))"  
 <proof>

**lemma oosn\_rreq:**

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onll  $\Gamma_{AODV}$  (seq1 i ( $\lambda(\xi, l)$ .  $l \in \{PAodv-:4, PAodv-:5\} \cup \{PRreq-:n \mid n. \text{True}\} \rightarrow 1 \leq$  osn  $\xi$ ))"  
 <proof>

**lemma rreq\_sip:**

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onll  $\Gamma_{AODV}$  ( $\lambda(\sigma, l)$ .  
 ( $l \in \{PAodv-:4, PAodv-:5, PRreq-:0, PRreq-:2\} \wedge$  sip ( $\sigma$  i)  $\neq$  oip ( $\sigma$  i))  
 $\rightarrow$  oip ( $\sigma$  i)  $\in$  kD(rt ( $\sigma$  (sip ( $\sigma$  i))))  
 $\wedge$  nsqn (rt ( $\sigma$  (sip ( $\sigma$  i)))) (oip ( $\sigma$  i))  $\geq$  osn ( $\sigma$  i)  
 $\wedge$  (nsqn (rt ( $\sigma$  (sip ( $\sigma$  i)))) (oip ( $\sigma$  i)) = osn ( $\sigma$  i))  
 $\rightarrow$  (hops ( $\sigma$  i)  $\geq$  the (dhops (rt ( $\sigma$  (sip ( $\sigma$  i)))) (oip ( $\sigma$  i))))  
 $\vee$  the (flag (rt ( $\sigma$  (sip ( $\sigma$  i)))) (oip ( $\sigma$  i))) = inv)))"  
 (is "\_  $\models$  (?S, ?U  $\rightarrow$ ) \_")  
 <proof>

**lemma odsn\_rrep:**

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onll  $\Gamma_{AODV}$  (seq1 i ( $\lambda(\xi, l)$ .  $l \in \{PAodv-:6, PAodv-:7\} \cup \{PRrep-:n \mid n. \text{True}\} \rightarrow 1 \leq$  dsn  $\xi$ ))"  
 <proof>

**lemma rrep\_sip:**

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onll  $\Gamma_{AODV}$  ( $\lambda(\sigma, l)$ .  
 ( $l \in \{PAodv-:6, PAodv-:7, PRrep-:0, PRrep-:1\} \wedge$  sip ( $\sigma$  i)  $\neq$  dip ( $\sigma$  i))  
 $\rightarrow$  dip ( $\sigma$  i)  $\in$  kD(rt ( $\sigma$  (sip ( $\sigma$  i))))  
 $\wedge$  nsqn (rt ( $\sigma$  (sip ( $\sigma$  i)))) (dip ( $\sigma$  i))  $\geq$  dsn ( $\sigma$  i)  
 $\wedge$  (nsqn (rt ( $\sigma$  (sip ( $\sigma$  i)))) (dip ( $\sigma$  i)) = dsn ( $\sigma$  i))  
 $\rightarrow$  (hops ( $\sigma$  i)  $\geq$  the (dhops (rt ( $\sigma$  (sip ( $\sigma$  i)))) (dip ( $\sigma$  i))))  
 $\vee$  the (flag (rt ( $\sigma$  (sip ( $\sigma$  i)))) (dip ( $\sigma$  i))) = inv)))"  
 (is "\_  $\models$  (?S, ?U  $\rightarrow$ ) \_")  
 <proof>

**lemma rerr\_sip:**

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onll  $\Gamma_{AODV}$  ( $\lambda(\sigma, l)$ .  
 $l \in \{PAodv-:8, PAodv-:9, PRerr-:0, PRerr-:1\}$   
 $\rightarrow$  ( $\forall \text{ripc} \in \text{dom}(\text{dests } (\sigma \text{ i})). \text{ripc} \in \text{kD}(\text{rt } (\sigma \text{ (sip } (\sigma \text{ i)))) \wedge$   
 the (dests ( $\sigma$  i) ripc) - 1  $\leq$  nsqn (rt ( $\sigma$  (sip ( $\sigma$  i)))) ripc))"  
 (is "\_  $\models$  (?S, ?U  $\rightarrow$ ) \_")  
 <proof>

```

lemma prerr_guard: "paadv i  $\models$ 
  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l)$ . ( $l = PRerr-:1$ 
     $\rightarrow (\forall ip \in \text{dom}(\text{dests } \xi)$ .  $ip \in vD(\text{rt } \xi)$ 
       $\wedge \text{the } (\text{nhop } (\text{rt } \xi) \text{ ip}) = sip \ \xi$ 
       $\wedge \text{sqn } (\text{rt } \xi) \text{ ip} < \text{the } (\text{dests } \xi \text{ ip}))$ ))"
<proof>

```

```

lemmas oaddpreRT_welldefined =
  open_seq_invariant [OF addpreRT_welldefined initiali_aadv oaadv_trans aadv_trans,
    simplified seq1_onl_swap,
    THEN oinvariant_anyact]

```

```

lemmas odests_vD_inc_sqn =
  open_seq_invariant [OF dests_vD_inc_sqn initiali_aadv oaadv_trans aadv_trans,
    simplified seq1_onl_swap,
    THEN oinvariant_anyact]

```

```

lemmas oprerr_guard =
  open_seq_invariant [OF prerr_guard initiali_aadv oaadv_trans aadv_trans,
    simplified seq1_onl_swap,
    THEN oinvariant_anyact]

```

Proposition 7.28

```

lemma seq_compare_next_hop':
  "opaadv i  $\models$  (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i}  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\sigma, \_)$ .
     $\forall dip$ . let nhop = the (nhop (rt ( $\sigma$  i)) dip)
      in  $dip \in kD(\text{rt } (\sigma \text{ i})) \wedge \text{nhop} \neq dip \rightarrow$ 
         $dip \in kD(\text{rt } (\sigma \text{ nhop})) \wedge \text{nsqn } (\text{rt } (\sigma \text{ i})) \text{ dip} \leq \text{nsqn } (\text{rt } (\sigma \text{ nhop})) \text{ dip}$ )"
  (is "_  $\models$  (?S, ?U  $\rightarrow$ ) _")
<proof>

```

Proposition 7.30

```

lemmas okD_unk_or_atleast_one =
  open_seq_invariant [OF kD_unk_or_atleast_one initiali_aadv,
    simplified seq1_onl_swap]

```

```

lemmas ozero_seq_unk_hops_one =
  open_seq_invariant [OF zero_seq_unk_hops_one initiali_aadv,
    simplified seq1_onl_swap]

```

```

lemma oreachable_fresh_okD_unk_or_atleast_one:
  fixes dip
  assumes "( $\sigma, p$ )  $\in$  oreachable (opaadv i)
    (otherwith (op=) {i} (orecvmsg ( $\lambda\sigma \text{ m}$ . msg_fresh  $\sigma \text{ m}$ 
       $\wedge \text{msg\_zhops } \text{m}$ )))
    (other quality_increases {i}))"
  and "dip  $\in kD(\text{rt } (\sigma \text{ i}))"$ 
  shows " $\pi_3(\text{the } (\text{rt } (\sigma \text{ i}) \text{ dip})) = \text{unk} \vee 1 \leq \pi_2(\text{the } (\text{rt } (\sigma \text{ i}) \text{ dip}))"$ 
    (is "?P dip")
  <proof>

```

```

lemma oreachable_fresh_ozero_seq_unk_hops_one:
  fixes dip
  assumes "( $\sigma, p$ )  $\in$  oreachable (opaadv i)
    (otherwith (op=) {i} (orecvmsg ( $\lambda\sigma \text{ m}$ . msg_fresh  $\sigma \text{ m}$ 
       $\wedge \text{msg\_zhops } \text{m}$ )))
    (other quality_increases {i}))"
  and "dip  $\in kD(\text{rt } (\sigma \text{ i}))"$ 
  shows "sqn (rt ( $\sigma$  i)) dip = 0  $\rightarrow$ 
    sqnf (rt ( $\sigma$  i)) dip = unk
     $\wedge \text{the } (\text{dhops } (\text{rt } (\sigma \text{ i})) \text{ dip}) = 1$ 
     $\wedge \text{the } (\text{nhop } (\text{rt } (\sigma \text{ i})) \text{ dip}) = \text{dip}"$ 
    (is "?P dip")

```

*<proof>*

**lemma seq\_nhop\_quality\_increases':**

```
shows "opaadv i  $\models$  (otherwith (op=) {i}
      (orecvmsg ( $\lambda\sigma$  m. msg_fresh  $\sigma$  m  $\wedge$  msg_zhops m)),
      other quality_increases {i}  $\rightarrow$ )
      onl  $\Gamma_{AODV}$  ( $\lambda(\sigma, \_)$ .  $\forall$ dip. let nhip = the (nhop (rt ( $\sigma$  i)) dip)
      in dip  $\in$  vD (rt ( $\sigma$  i))  $\cap$  vD (rt ( $\sigma$  nhip))
       $\wedge$  nhip  $\neq$  dip
       $\rightarrow$  (rt ( $\sigma$  i))  $\sqsubset_{dip}$  (rt ( $\sigma$  nhip)))"
```

(is " $\_ \models$  (?S i,  $\_ \rightarrow$ )  $\_$ ")

*<proof>*

**lemma seq\_compare\_next\_hop:**

fixes w

```
shows "opaadv i  $\models$  (otherwith (op=) {i} (orecvmsg msg_fresh),
      other quality_increases {i}  $\rightarrow$ )
      global ( $\lambda\sigma$ .  $\forall$ dip. let nhip = the (nhop (rt ( $\sigma$  i)) dip)
      in dip  $\in$  kD(rt ( $\sigma$  i))  $\wedge$  nhip  $\neq$  dip  $\rightarrow$ 
      dip  $\in$  kD(rt ( $\sigma$  nhip))
       $\wedge$  nsqn (rt ( $\sigma$  i)) dip  $\leq$  nsqn (rt ( $\sigma$  nhip)) dip)"
```

*<proof>*

**lemma seq\_nhop\_quality\_increases:**

```
shows "opaadv i  $\models$  (otherwith (op=) {i}
      (orecvmsg ( $\lambda\sigma$  m. msg_fresh  $\sigma$  m  $\wedge$  msg_zhops m)),
      other quality_increases {i}  $\rightarrow$ )
      global ( $\lambda\sigma$ .  $\forall$ dip. let nhip = the (nhop (rt ( $\sigma$  i)) dip)
      in dip  $\in$  vD (rt ( $\sigma$  i))  $\cap$  vD (rt ( $\sigma$  nhip))  $\wedge$  nhip  $\neq$  dip
       $\rightarrow$  (rt ( $\sigma$  i))  $\sqsubset_{dip}$  (rt ( $\sigma$  nhip)))"
```

*<proof>*

end

## 2.10 Routing graphs and loop freedom

**theory B\_Loop\_Freedom**

**imports B\_Aadv\_Predicates B\_Fresher**

**begin**

Define the central theorem that relates an invariant over network states to the absence of loops in the associated routing graph.

**definition**

```
rt_graph :: "(ip  $\Rightarrow$  state)  $\Rightarrow$  ip  $\Rightarrow$  ip rel"
```

where

```
"rt_graph  $\sigma$  = ( $\lambda$ dip.
  {(ip, ip') | ip ip' dsn dsk hops pre.
   ip  $\neq$  dip  $\wedge$  rt ( $\sigma$  ip) dip = Some (dsn, dsk, val, hops, ip', pre)})"
```

Given the state of a network  $\sigma$ , a routing graph for a given destination ip address  $dip$  abstracts the details of routing tables into nodes (ip addresses) and vertices (valid routes between ip addresses).

**lemma rt\_graphE [elim]:**

```
fixes n dip ip ip'
assumes "(ip, ip')  $\in$  rt_graph  $\sigma$  dip"
shows "ip  $\neq$  dip  $\wedge$  ( $\exists$ r. rt ( $\sigma$  ip) = r
       $\wedge$  ( $\exists$ dsn dsk hops pre. r dip = Some (dsn, dsk, val, hops, ip', pre)))"
```

*<proof>*

**lemma rt\_graph\_vD [dest]:**

```
" $\bigwedge$ ip ip'  $\sigma$  dip. (ip, ip')  $\in$  rt_graph  $\sigma$  dip  $\implies$  dip  $\in$  vD(rt ( $\sigma$  ip))"
```

*<proof>*

**lemma rt\_graph\_vD\_trans [dest]:**

```
" $\bigwedge ip\ ip'\ \sigma\ dip.\ (ip,\ ip') \in (rt\_graph\ \sigma\ dip)^+ \implies dip \in vD(rt\ (\sigma\ ip))"$ 
<proof>
```

```
lemma rt_graph_not_dip [dest]:
```

```
" $\bigwedge ip\ ip'\ \sigma\ dip.\ (ip,\ ip') \in rt\_graph\ \sigma\ dip \implies ip \neq dip"$ 
<proof>
```

```
lemma rt_graph_not_dip_trans [dest]:
```

```
" $\bigwedge ip\ ip'\ \sigma\ dip.\ (ip,\ ip') \in (rt\_graph\ \sigma\ dip)^+ \implies ip \neq dip"$ 
<proof>
```

NB: the property below cannot be lifted to the transitive closure

```
lemma rt_graph_nhip_is_nhop [dest]:
```

```
" $\bigwedge ip\ ip'\ \sigma\ dip.\ (ip,\ ip') \in rt\_graph\ \sigma\ dip \implies ip' = the\ (nhop\ (rt\ (\sigma\ ip))\ dip)"$ 
<proof>
```

```
theorem inv_to_loop_freedom:
```

```
assumes " $\forall i\ dip.\ let\ nhip = the\ (nhop\ (rt\ (\sigma\ i))\ dip)$ 
in  $dip \in vD\ (rt\ (\sigma\ i)) \cap vD\ (rt\ (\sigma\ nhip)) \wedge nhip \neq dip$ 
 $\longrightarrow (rt\ (\sigma\ i)) \sqsubseteq_{dip}\ (rt\ (\sigma\ nhip))"$ 
shows " $\forall dip.\ irrefl\ ((rt\_graph\ \sigma\ dip)^+)"$ 
<proof>
```

end

## 2.11 Lift and transfer invariants to show loop freedom

```
theory B_Aodv_Loop_Freedom
```

```
imports "../..../AWN/OClosed_Transfer" "../..../AWN/Qmsg_Lifting" B_Global_Invariants B_Loop_Freedom
begin
```

### 2.11.1 Lift to parallel processes with queues

```
lemma par_step_no_change_on_send_or_receive:
```

```
fixes  $\sigma\ s\ a\ \sigma'\ s'$ 
assumes " $((\sigma,\ s),\ a,\ (\sigma',\ s')) \in oparp\_sos\ i\ (oseqp\_sos\ \Gamma_{AODV}\ i)\ (seqp\_sos\ \Gamma_{QMSG})"$ 
and " $a \neq \tau"$ 
shows " $\sigma' i = \sigma i"$ 
<proof>
```

```
lemma par_nhop_quality_increases:
```

```
shows " $opaodv\ i\ \langle\langle i\ qmsg \models (otherwith\ (op=)\ \{i\}\ (orecvmsg\ (\lambda\ \sigma\ m.\$ 
 $msg\_fresh\ \sigma\ m \wedge msg\_zhops\ m)),$ 
 $other\ quality\_increases\ \{i\} \rightarrow)$ 
 $global\ (\lambda\ \sigma.\ \forall dip.\ let\ nhip = the\ (nhop\ (rt\ (\sigma\ i))\ dip)$ 
in  $dip \in vD\ (rt\ (\sigma\ i)) \cap vD\ (rt\ (\sigma\ nhip)) \wedge nhip \neq dip$ 
 $\longrightarrow (rt\ (\sigma\ i)) \sqsubseteq_{dip}\ (rt\ (\sigma\ nhip))\rangle\rangle"$ 
<proof>
```

```
lemma par_rreq_rrep_sn_quality_increases:
```

```
" $opaodv\ i\ \langle\langle i\ qmsg \models_A\ (\lambda\ \sigma\ \_.\ orecvmsg\ (\lambda\ \_.\ rreq\_rrep\_sn)\ \sigma,\ other\ (\lambda\ \_.\ True)\ \{i\} \rightarrow)$ 
 $globala\ (\lambda(\sigma,\ \_,\ \sigma').\ quality\_increases\ (\sigma\ i)\ (\sigma'\ i))\rangle\rangle"$ 
<proof>
```

```
lemma par_rreq_rrep_nsqn_fresh_any_step:
```

```
shows " $opaodv\ i\ \langle\langle i\ qmsg \models_A\ (\lambda\ \sigma\ \_.\ orecvmsg\ (\lambda\ \_.\ rreq\_rrep\_sn)\ \sigma,$ 
 $other\ (\lambda\ \_.\ True)\ \{i\} \rightarrow)$ 
 $globala\ (\lambda(\sigma,\ a,\ \sigma').\ anycast\ (msg\_fresh\ \sigma)\ a)\rangle\rangle"$ 
<proof>
```

```
lemma par_anycast_msg_zhops:
```

```
shows " $opaodv\ i\ \langle\langle i\ qmsg \models_A\ (\lambda\ \sigma\ \_.\ orecvmsg\ (\lambda\ \_.\ rreq\_rrep\_sn)\ \sigma,\ other\ (\lambda\ \_.\ True)\ \{i\} \rightarrow)$ 
 $globala\ (\lambda(\_,\ a,\ \_).\ anycast\ msg\_zhops\ a)\rangle\rangle"$ 
<proof>
```

## 2.11.2 Lift to nodes

**lemma** `node_step_no_change_on_send_or_receive`:

```

assumes "( $(\sigma, \text{NodeS } i \text{ } P \text{ } R)$ ,  $a$ ,  $(\sigma', \text{NodeS } i' \text{ } P' \text{ } R')$ )  $\in$  onode_sos
        (oparp_sos  $i$  (oseqp_sos  $\Gamma_{AODV}$   $i$ ) (seqp_sos  $\Gamma_{QMSG}$ ))"
and "a  $\neq \tau$ "
shows " $\sigma' \text{ } i = \sigma \text{ } i$ "
<proof>

```

**lemma** `node_nhop_quality_increases`:

```

shows "<math>i : \text{opaadv } i \langle\langle_i \text{ qmsg} : R \rangle_o \models</math>
      (otherwith (op=)  $\{i\}$ 
        (oarrivemsg  $(\lambda\sigma \text{ } m. \text{msg\_fresh } \sigma \text{ } m \wedge \text{msg\_zhops } m)$ ),
        other quality_increases  $\{i\}$ 
         $\rightarrow$  global  $(\lambda\sigma. \forall \text{dip. let } \text{nhop} = \text{the } (\text{nhop } (\text{rt } (\sigma \text{ } i)) \text{ dip})$ 
           $\text{in } \text{dip} \in \text{vD } (\text{rt } (\sigma \text{ } i)) \cap \text{vD } (\text{rt } (\sigma \text{ } \text{nhop})) \wedge \text{nhop} \neq \text{dip}$ 
           $\rightarrow (\text{rt } (\sigma \text{ } i)) \sqsubset_{\text{dip}} (\text{rt } (\sigma \text{ } \text{nhop}))$ )"
<proof>

```

**lemma** `node_quality_increases`:

```

"<math>i : \text{opaadv } i \langle\langle_i \text{ qmsg} : R \rangle_o \models_A (\lambda\sigma \text{ } \_ . \text{oarrivemsg } (\lambda\_ \_ . \text{rreq\_rrep\_sn}) \sigma,</math>
      other  $(\lambda\_ \_ . \text{True}) \{i\} \rightarrow$ 
      globala  $(\lambda(\sigma, \_, \sigma'). \text{quality\_increases } (\sigma \text{ } i) (\sigma' \text{ } i))$ "
<proof>

```

**lemma** `node_rreq_rrep_nsqn_fresh_any_step`:

```

shows "<math>i : \text{opaadv } i \langle\langle_i \text{ qmsg} : R \rangle_o \models_A</math>
       $(\lambda\sigma \text{ } \_ . \text{oarrivemsg } (\lambda\_ \_ . \text{rreq\_rrep\_sn}) \sigma, \text{other } (\lambda\_ \_ . \text{True}) \{i\} \rightarrow)$ 
      globala  $(\lambda(\sigma, a, \sigma'). \text{castmsg } (\text{msg\_fresh } \sigma) a)$ "
<proof>

```

**lemma** `node_anycast_msg_zhops`:

```

shows "<math>i : \text{opaadv } i \langle\langle_i \text{ qmsg} : R \rangle_o \models_A</math>
       $(\lambda\sigma \text{ } \_ . \text{oarrivemsg } (\lambda\_ \_ . \text{rreq\_rrep\_sn}) \sigma, \text{other } (\lambda\_ \_ . \text{True}) \{i\} \rightarrow)$ 
      globala  $(\lambda(\_, a, \_). \text{castmsg } \text{msg\_zhops } a)$ "
<proof>

```

**lemma** `node_silent_change_only`:

```

shows "<math>i : \text{opaadv } i \langle\langle_i \text{ qmsg} : R_i \rangle_o \models_A (\lambda\sigma \text{ } \_ . \text{oarrivemsg } (\lambda\_ \_ . \text{True}) \sigma,</math>
      other  $(\lambda\_ \_ . \text{True}) \{i\} \rightarrow$ 
      globala  $(\lambda(\sigma, a, \sigma'). a \neq \tau \rightarrow \sigma' \text{ } i = \sigma \text{ } i)$ "
<proof>

```

## 2.11.3 Lift to partial networks

**lemma** `arrive_rreq_rrep_nsqn_fresh_inc_sn [simp]`:

```

assumes "oarrivemsg  $(\lambda\sigma \text{ } m. \text{msg\_fresh } \sigma \text{ } m \wedge P \text{ } \sigma \text{ } m) \sigma \text{ } m$ "
shows "oarrivemsg  $(\lambda\_ \_ . \text{rreq\_rrep\_sn}) \sigma \text{ } m$ "
<proof>

```

**lemma** `opnet_nhop_quality_increases`:

```

shows "opnet  $(\lambda i. \text{opaadv } i \langle\langle_i \text{ qmsg} \rangle_p \models$ 
      (otherwith (op=) (net_tree_ips  $p$ )
        (oarrivemsg  $(\lambda\sigma \text{ } m. \text{msg\_fresh } \sigma \text{ } m \wedge \text{msg\_zhops } m)$ ),
        other quality_increases (net_tree_ips  $p$ )  $\rightarrow$ )
      global  $(\lambda\sigma. \forall i \in \text{net\_tree\_ips } p. \forall \text{dip.}$ 
         $\text{let } \text{nhop} = \text{the } (\text{nhop } (\text{rt } (\sigma \text{ } i)) \text{ dip})$ 
         $\text{in } \text{dip} \in \text{vD } (\text{rt } (\sigma \text{ } i)) \cap \text{vD } (\text{rt } (\sigma \text{ } \text{nhop})) \wedge \text{nhop} \neq \text{dip}$ 
         $\rightarrow (\text{rt } (\sigma \text{ } i)) \sqsubset_{\text{dip}} (\text{rt } (\sigma \text{ } \text{nhop}))$ )"
<proof>

```

## 2.11.4 Lift to closed networks

**lemma** `onet_nhop_quality_increases`:

```

shows "oclosed (opnet  $(\lambda i. \text{opaadv } i \langle\langle_i \text{ qmsg} \rangle_p$ )

```

```

 $\models (\lambda\_ \_ \_ . \text{True}, \text{other\_quality\_increases } (\text{net\_tree\_ips } p) \rightarrow)$ 
  global  $(\lambda\sigma . \forall i \in \text{net\_tree\_ips } p . \forall \text{dip} .$ 
    let  $\text{nhip} = \text{the } (\text{nhop } (\text{rt } (\sigma \ i)) \ \text{dip})$ 
    in  $\text{dip} \in \text{vD } (\text{rt } (\sigma \ i)) \cap \text{vD } (\text{rt } (\sigma \ \text{nhip})) \wedge \text{nhip} \neq \text{dip}$ 
       $\rightarrow (\text{rt } (\sigma \ i)) \sqsubset_{\text{dip}} (\text{rt } (\sigma \ \text{nhip}))$ )"
(is "_  $\models$  (_, ?U  $\rightarrow$ ) ?inv")
<proof>

```

### 2.11.5 Transfer into the standard model

```

interpretation aadv_openproc: openproc paadv opaadv id
rewrites "aadv_openproc.initmissing = initmissing"
<proof>

```

```

interpretation aadv_openproc_par_qmsg: openproc_parq paadv opaadv id qmsg
rewrites "aadv_openproc_par_qmsg.netglobal = netglobal"
  and "aadv_openproc_par_qmsg.initmissing = initmissing"
<proof>

```

```

lemma net_nhop_quality_increases:
  assumes "wf_net_tree n"
  shows "closed (pnet  $(\lambda i . \text{paadv } i \ \langle\langle \text{qmsg} \rangle\rangle \ n) \models \text{netglobal}$ 
     $(\lambda\sigma . \forall i \ \text{dip} . \text{let } \text{nhip} = \text{the } (\text{nhop } (\text{rt } (\sigma \ i)) \ \text{dip})$ 
      in  $\text{dip} \in \text{vD } (\text{rt } (\sigma \ i)) \cap \text{vD } (\text{rt } (\sigma \ \text{nhip})) \wedge \text{nhip} \neq \text{dip}$ 
         $\rightarrow (\text{rt } (\sigma \ i)) \sqsubset_{\text{dip}} (\text{rt } (\sigma \ \text{nhip}))$ )"
    (is "_  $\models \text{netglobal } (\lambda\sigma . \forall i . ?\text{inv } \sigma \ i)$ ")
<proof>

```

### 2.11.6 Loop freedom of AODV

```

theorem aadv_loop_freedom:
  assumes "wf_net_tree n"
  shows "closed (pnet  $(\lambda i . \text{paadv } i \ \langle\langle \text{qmsg} \rangle\rangle \ n) \models \text{netglobal } (\lambda\sigma . \forall \text{dip} . \text{irrefl } ((\text{rt\_graph } \sigma \ \text{dip})^+))$ "
<proof>

```

end



## Chapter 3

# Variant C: From Groupcast to Broadcast

Explanation [4, §10.4]: A node maintains a set of ‘precursor nodes’ for each of its valid routes. If the link to a route’s next hop is lost, an error message is groupcast to the associated precursor nodes. The idea is to reduce the number of messages received and handled. However, precursor lists are incomplete. They are updated only when a RREP message is sent. This can lead to packet loss. A possible solution is to abandon precursors and to replace every groupcast by a broadcast. At first glance this strategy seems to need more bandwidth, but this is not the case. Sending error messages to a set of precursors is implemented at the link layer by broadcasting the message anyway; a node receiving such a message then checks the header to determine whether it is one of the intended recipients. Instead of analysing the header only, a node can just as well read the message and decide whether the information contained in the message is of use. To be more precise: an error message is useful for a node if the node has established a route to one of the nodes listed in the message, and the next hop to a listed node is the sender of the error message. In case a node finds useful information inside the message, it should update its routing table and distribute another error message.

### 3.1 Predicates and functions used in the AODV model

```
theory C_Aodv_Data
imports C_Gtobcast
begin
```

#### 3.1.1 Sequence Numbers

Sequence numbers approximate the relative freshness of routing information.

```
definition inc :: "sqn  $\Rightarrow$  sqn"
  where "inc sn  $\equiv$  if sn = 0 then sn else sn + 1"
```

```
lemma less_than_inc [simp]: "x  $\leq$  inc x"
  <proof>
```

```
lemma inc_minus_suc_0 [simp]:
  "inc x - Suc 0 = x"
  <proof>
```

```
lemma inc_never_one' [simp, intro]: "inc x  $\neq$  Suc 0"
  <proof>
```

```
lemma inc_never_one [simp, intro]: "inc x  $\neq$  1"
  <proof>
```

#### 3.1.2 Modelling Routes

A route is a 5-tuple,  $(dsn, dsk, flag, hops, nhip)$  where  $dsn$  is the ‘destination sequence number’,  $dsk$  is the ‘destination-sequence-number status’,  $flag$  is the route status,  $hops$  is the number of hops to the destination, and  $nhip$  is the next hop toward the destination. In this variant, the set of ‘precursor nodes’ is not modelled.

```
type_synonym r = "sqn  $\times$  k  $\times$  f  $\times$  nat  $\times$  ip"
```

```

definition proj2 :: "r ⇒ sqn" ("π2")
  where "π2 ≡ λ(dsn, _, _, _). dsn"

```

```

definition proj3 :: "r ⇒ k" ("π3")
  where "π3 ≡ λ(_, dsk, _, _). dsk"

```

```

definition proj4 :: "r ⇒ f" ("π4")
  where "π4 ≡ λ(_, _, flag, _, _). flag"

```

```

definition proj5 :: "r ⇒ nat" ("π5")
  where "π5 ≡ λ(_, _, _, hops, _). hops"

```

```

definition proj6 :: "r ⇒ ip" ("π6")
  where "π6 ≡ λ(_, _, _, _, nhip). nhip"

```

```

lemma projs [simp]:
  "π2(dsn, dsk, flag, hops, nhip) = dsn"
  "π3(dsn, dsk, flag, hops, nhip) = dsk"
  "π4(dsn, dsk, flag, hops, nhip) = flag"
  "π5(dsn, dsk, flag, hops, nhip) = hops"
  "π6(dsn, dsk, flag, hops, nhip) = nhip"
  ⟨proof⟩

```

```

lemma proj3_pred [intro]: "[[ P kno; P unk ] ⇒ P (π3 x)"
  ⟨proof⟩

```

```

lemma proj4_pred [intro]: "[[ P val; P inv ] ⇒ P (π4 x)"
  ⟨proof⟩

```

```

lemma proj6_pair_snd [simp]:
  fixes dsn' r
  shows "π6 (dsn', snd (r)) = π6(r)"
  ⟨proof⟩

```

### 3.1.3 Routing Tables

Routing tables map ip addresses to route entries.

```

type_synonym rt = "ip → r"

```

```

syntax
  "_Sigma_route" :: "rt ⇒ ip → r" ("σroute'(_, _)'")

```

```

translations
  "σroute(rt, dip)" => "rt dip"

```

```

definition sqn :: "rt ⇒ ip ⇒ sqn"
  where "sqn rt dip ≡ case σroute(rt, dip) of Some r ⇒ π2(r) | None ⇒ 0"

```

```

definition sqnf :: "rt ⇒ ip ⇒ k"
  where "sqnf rt dip ≡ case σroute(rt, dip) of Some r ⇒ π3(r) | None ⇒ unk"

```

```

abbreviation flag :: "rt ⇒ ip → f"
  where "flag rt dip ≡ map_option π4 (σroute(rt, dip))"

```

```

abbreviation dhops :: "rt ⇒ ip → nat"
  where "dhops rt dip ≡ map_option π5 (σroute(rt, dip))"

```

```

abbreviation nhop :: "rt ⇒ ip → ip"
  where "nhop rt dip ≡ map_option π6 (σroute(rt, dip))"

```

```

definition vD :: "rt ⇒ ip set"
  where "vD rt ≡ {dip. flag rt dip = Some val}"

```

```

definition iD :: "rt ⇒ ip set"

```

```

where "iD rt  $\equiv$  {dip. flag rt dip = Some inv}"

definition kD :: "rt  $\Rightarrow$  ip set"
  where "kD rt  $\equiv$  {dip. rt dip  $\neq$  None}"

lemma kD_is_vD_and_iD: "kD rt = vD rt  $\cup$  iD rt"
  <proof>

lemma vD_iD_gives_kD [simp]:
  " $\bigwedge$  ip rt. ip  $\in$  vD rt  $\implies$  ip  $\in$  kD rt"
  " $\bigwedge$  ip rt. ip  $\in$  iD rt  $\implies$  ip  $\in$  kD rt"
  <proof>

lemma kD_Some [dest]:
  fixes dip rt
  assumes "dip  $\in$  kD rt"
  shows " $\exists$  dsn dsk flag hops nhip.
     $\sigma_{route}(rt, dip) = \text{Some } (dsn, dsk, flag, hops, nhip)$ "
  <proof>

lemma kD_None [dest]:
  fixes dip rt
  assumes "dip  $\notin$  kD rt"
  shows " $\sigma_{route}(rt, dip) = \text{None}$ "
  <proof>

lemma vD_Some [dest]:
  fixes dip rt
  assumes "dip  $\in$  vD rt"
  shows " $\exists$  dsn dsk hops nhip.
     $\sigma_{route}(rt, dip) = \text{Some } (dsn, dsk, val, hops, nhip)$ "
  <proof>

lemma vD_empty [simp]: "vD Map.empty = {}"
  <proof>

lemma iD_Some [dest]:
  fixes dip rt
  assumes "dip  $\in$  iD rt"
  shows " $\exists$  dsn dsk hops nhip.
     $\sigma_{route}(rt, dip) = \text{Some } (dsn, dsk, inv, hops, nhip)$ "
  <proof>

lemma val_is_vD [elim]:
  fixes ip rt
  assumes "ip  $\in$  kD(rt)"
  and "the (flag rt ip) = val"
  shows "ip  $\in$  vD(rt)"
  <proof>

lemma inv_is_iD [elim]:
  fixes ip rt
  assumes "ip  $\in$  kD(rt)"
  and "the (flag rt ip) = inv"
  shows "ip  $\in$  iD(rt)"
  <proof>

lemma iD_flag_is_inv [elim, simp]:
  fixes ip rt
  assumes "ip  $\in$  iD(rt)"
  shows "the (flag rt ip) = inv"
  <proof>

lemma kD_but_not_vD_is_iD [elim]:

```

```

fixes ip rt
assumes "ip ∈ kD(rt)"
and "ip ∉ vD(rt)"
shows "ip ∈ iD(rt)"
⟨proof⟩

```

```

lemma vD_or_iD [elim]:
fixes ip rt
assumes "ip ∈ kD(rt)"
and "ip ∈ vD(rt) ⇒ P rt ip"
and "ip ∈ iD(rt) ⇒ P rt ip"
shows "P rt ip"
⟨proof⟩

```

```

lemma proj5_eq_dhops: "∧ dip rt. dip ∈ kD(rt) ⇒ π5(the (rt dip)) = the (dhops rt dip)"
⟨proof⟩

```

```

lemma proj4_eq_flag: "∧ dip rt. dip ∈ kD(rt) ⇒ π4(the (rt dip)) = the (flag rt dip)"
⟨proof⟩

```

```

lemma proj2_eq_sqn: "∧ dip rt. dip ∈ kD(rt) ⇒ π2(the (rt dip)) = sqn rt dip"
⟨proof⟩

```

```

lemma kD_sqnf_is_proj3 [simp]:
"∧ ip rt. ip ∈ kD(rt) ⇒ sqnf rt ip = π3(the (rt ip))"
⟨proof⟩

```

```

lemma vD_flag_val [simp]:
"∧ dip rt. dip ∈ vD (rt) ⇒ the (flag rt dip) = val"
⟨proof⟩

```

```

lemma kD_update [simp]:
"∧ rt nip v. kD (rt(nip ↦ v)) = insert nip (kD rt)"
⟨proof⟩

```

```

lemma kD_empty [simp]: "kD Map.empty = {}"
⟨proof⟩

```

```

lemma ip_equal_or_known [elim]:
fixes rt ip ip'
assumes "ip = ip' ∨ ip ∈ kD(rt)"
and "ip = ip' ⇒ P rt ip ip'"
and "[ ip ≠ ip'; ip ∈ kD(rt) ] ⇒ P rt ip ip'"
shows "P rt ip ip'"
⟨proof⟩

```

### 3.1.4 Updating Routing Tables

Routing table entries are modified through explicit functions. The properties of these functions are important in invariant proofs.

#### Updating route entries

```

lemma in_kD_case [simp]:
fixes dip rt
assumes "dip ∈ kD(rt)"
shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = es (the (rt dip))"
⟨proof⟩

```

```

lemma not_in_kD_case [simp]:
fixes dip rt
assumes "dip ∉ kD(rt)"
shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = en"
⟨proof⟩

```

lemma rt\_Some\_sqn [dest]:

fixes rt and ip dsn dsk flag hops nhip  
 assumes "rt ip = Some (dsn, dsk, flag, hops, nhip)"  
 shows "sqn rt ip = dsn"  
 ⟨proof⟩

lemma not\_kD\_sqn [simp]:

fixes dip rt  
 assumes "dip  $\notin$  kD(rt)"  
 shows "sqn rt dip = 0"  
 ⟨proof⟩

definition update\_arg\_wf :: "r  $\Rightarrow$  bool"

where "update\_arg\_wf r  $\equiv$   $\pi_4(r) = \text{val} \wedge$   
 $(\pi_2(r) = 0) = (\pi_3(r) = \text{unk}) \wedge$   
 $(\pi_3(r) = \text{unk} \longrightarrow \pi_5(r) = 1)$ "

lemma update\_arg\_wf\_gives\_cases:

" $\bigwedge r. \text{update\_arg\_wf } r \implies (\pi_2(r) = 0) = (\pi_3(r) = \text{unk})$ "  
 ⟨proof⟩

lemma update\_arg\_wf\_tuples [simp]:

" $\bigwedge \text{nhip}. \text{update\_arg\_wf } (0, \text{unk}, \text{val}, \text{Suc } 0, \text{nhip})$ "  
 " $\bigwedge n \text{ hops nhip}. \text{update\_arg\_wf } (\text{Suc } n, \text{kno}, \text{val}, \text{hops}, \text{nhip})$ "  
 ⟨proof⟩

lemma update\_arg\_wf\_tuples' [elim]:

" $\bigwedge n \text{ hops nhip}. \text{Suc } 0 \leq n \implies \text{update\_arg\_wf } (n, \text{kno}, \text{val}, \text{hops}, \text{nhip})$ "  
 ⟨proof⟩

lemma wf\_r\_cases [intro]:

fixes P r  
 assumes "update\_arg\_wf r"  
 and c1: " $\bigwedge \text{nhip}. P (0, \text{unk}, \text{val}, \text{Suc } 0, \text{nhip})$ "  
 and c2: " $\bigwedge \text{dsn hops nhip}. \text{dsn} > 0 \implies P (\text{dsn}, \text{kno}, \text{val}, \text{hops}, \text{nhip})$ "  
 shows "P r"  
 ⟨proof⟩

definition update :: "rt  $\Rightarrow$  ip  $\Rightarrow$  r  $\Rightarrow$  rt"

where

"update rt ip r  $\equiv$   
 case  $\sigma_{\text{route}}(\text{rt}, \text{ip})$  of  
 None  $\Rightarrow$  rt (ip  $\mapsto$  r)  
 | Some s  $\Rightarrow$   
 if  $\pi_2(s) < \pi_2(r)$  then rt (ip  $\mapsto$  r)  
 else if  $\pi_2(s) = \pi_2(r) \wedge (\pi_5(s) > \pi_5(r) \vee \pi_4(s) = \text{inv})$   
 then rt (ip  $\mapsto$  r)  
 else if  $\pi_3(r) = \text{unk}$   
 then rt (ip  $\mapsto$  ( $\pi_2(s)$ , snd (r)))  
 else rt (ip  $\mapsto$  s)"

lemma update\_simps [simp]:

fixes r s nrt nr' ns rt ip  
 defines "s  $\equiv$  the  $\sigma_{\text{route}}(\text{rt}, \text{ip})$ "  
 and "nr'  $\equiv$  ( $\pi_2(s)$ ,  $\pi_3(r)$ ,  $\pi_4(r)$ ,  $\pi_5(r)$ ,  $\pi_6(r)$ )"  
 shows  
 " $\llbracket \text{ip} \notin \text{kD}(\text{rt}) \rrbracket \implies \text{update } \text{rt } \text{ip } r = \text{rt } (\text{ip} \mapsto r)$ "  
 " $\llbracket \text{ip} \in \text{kD}(\text{rt}); \text{sqn } \text{rt } \text{ip} < \pi_2(r) \rrbracket \implies \text{update } \text{rt } \text{ip } r = \text{rt } (\text{ip} \mapsto r)$ "  
 " $\llbracket \text{ip} \in \text{kD}(\text{rt}); \text{sqn } \text{rt } \text{ip} = \pi_2(r);$   
 the ( $\text{dhops } \text{rt } \text{ip}) > \pi_5(r) \rrbracket \implies \text{update } \text{rt } \text{ip } r = \text{rt } (\text{ip} \mapsto r)$ "  
 " $\llbracket \text{ip} \in \text{kD}(\text{rt}); \text{sqn } \text{rt } \text{ip} = \pi_2(r);$   
 flag rt ip = Some inv  $\rrbracket \implies \text{update } \text{rt } \text{ip } r = \text{rt } (\text{ip} \mapsto r)$ "  
 " $\llbracket \text{ip} \in \text{kD}(\text{rt}); \pi_3(r) = \text{unk}; (\pi_2(r) = 0) = (\pi_3(r) = \text{unk}) \rrbracket \implies \text{update } \text{rt } \text{ip } r = \text{rt } (\text{ip} \mapsto \text{nr}')$ "

"[[ip ∈ kD(rt); sqn rt ip ≥ π<sub>2</sub>(r); π<sub>3</sub>(r) = kno;  
sqn rt ip = π<sub>2</sub>(r) ⇒ the (dhops rt ip) ≤ π<sub>5</sub>(r) ∧ the (flag rt ip) = val ]]  
⇒ update rt ip r = rt (ip ↦ s)"

⟨proof⟩

lemma update\_cases [elim]:

assumes "(π<sub>2</sub>(r) = 0) = (π<sub>3</sub>(r) = unk)"

and c1: "[[ip ∉ kD(rt)] ⇒ P (rt (ip ↦ r))]"

and c2: "[[ip ∈ kD(rt); sqn rt ip < π<sub>2</sub>(r)]  
⇒ P (rt (ip ↦ r))]"

and c3: "[[ip ∈ kD(rt); sqn rt ip = π<sub>2</sub>(r); the (dhops rt ip) > π<sub>5</sub>(r)]  
⇒ P (rt (ip ↦ r))]"

and c4: "[[ip ∈ kD(rt); sqn rt ip = π<sub>2</sub>(r); the (flag rt ip) = inv]  
⇒ P (rt (ip ↦ r))]"

and c5: "[[ip ∈ kD(rt); π<sub>3</sub>(r) = unk]  
⇒ P (rt (ip ↦ (π<sub>2</sub>(the σ<sub>route</sub>(rt, ip)), π<sub>3</sub>(r),  
π<sub>4</sub>(r), π<sub>5</sub>(r), π<sub>6</sub>(r))))]"

and c6: "[[ip ∈ kD(rt); sqn rt ip ≥ π<sub>2</sub>(r); π<sub>3</sub>(r) = kno;  
sqn rt ip = π<sub>2</sub>(r) ⇒ the (dhops rt ip) ≤ π<sub>5</sub>(r) ∧ the (flag rt ip) = val ]]  
⇒ P (rt (ip ↦ the σ<sub>route</sub>(rt, ip)))]"

shows "(P (update rt ip r))"

⟨proof⟩

lemma update\_cases\_kD:

assumes "(π<sub>2</sub>(r) = 0) = (π<sub>3</sub>(r) = unk)"

and "ip ∈ kD(rt)"

and c2: "[sqn rt ip < π<sub>2</sub>(r) ⇒ P (rt (ip ↦ r))]"

and c3: "[[sqn rt ip = π<sub>2</sub>(r); the (dhops rt ip) > π<sub>5</sub>(r)]  
⇒ P (rt (ip ↦ r))]"

and c4: "[[sqn rt ip = π<sub>2</sub>(r); the (flag rt ip) = inv]  
⇒ P (rt (ip ↦ r))]"

and c5: "[π<sub>3</sub>(r) = unk ⇒ P (rt (ip ↦ (π<sub>2</sub>(the σ<sub>route</sub>(rt, ip)), π<sub>3</sub>(r),  
π<sub>4</sub>(r), π<sub>5</sub>(r), π<sub>6</sub>(r))))]"

and c6: "[[sqn rt ip ≥ π<sub>2</sub>(r); π<sub>3</sub>(r) = kno;  
sqn rt ip = π<sub>2</sub>(r) ⇒ the (dhops rt ip) ≤ π<sub>5</sub>(r) ∧ the (flag rt ip) = val ]]  
⇒ P (rt (ip ↦ the σ<sub>route</sub>(rt, ip)))]"

shows "(P (update rt ip r))"

⟨proof⟩

lemma in\_kD\_after\_update [simp]:

fixes rt nip dsn dsk flag hops nhip

shows "kD (update rt nip (dsn, dsk, flag, hops, nhip)) = insert nip (kD rt)"

⟨proof⟩

lemma nhop\_of\_update [simp]:

fixes rt dip dsn dsk flag hops nhip

assumes "rt ≠ update rt dip (dsn, dsk, flag, hops, nhip)"

shows "the (nhop (update rt dip (dsn, dsk, flag, hops, nhip)) dip) = nhip"

⟨proof⟩

lemma sqn\_if\_updated:

fixes rip v rt ip

shows "sqn (λx. if x = rip then Some v else rt x) ip  
= (if ip = rip then π<sub>2</sub>(v) else sqn rt ip)"

⟨proof⟩

lemma update\_sqn [simp]:

fixes rt dip rip dsn dsk hops nhip

assumes "(dsn = 0) = (dsk = unk)"

shows "sqn rt dip ≤ sqn (update rt rip (dsn, dsk, val, hops, nhip)) dip"

⟨proof⟩

lemma sqn\_update\_bigger [simp]:

```

    fixes rt ip ip' dsn dsk flag hops nhip
  assumes "1 ≤ hops"
  shows "sqn rt ip ≤ sqn (update rt ip' (dsn, dsk, flag, hops, nhip)) ip"
  ⟨proof⟩

lemma dhops_update [intro]:
  fixes rt dsn dsk flag hops ip rip nhip
  assumes ex: "∀ ip ∈ kD rt. the (dhops rt ip) ≥ 1"
  and ip: "(ip = rip ∧ Suc 0 ≤ hops) ∨ (ip ≠ rip ∧ ip ∈ kD rt)"
  shows "Suc 0 ≤ the (dhops (update rt rip (dsn, dsk, flag, hops, nhip)) ip)"
  ⟨proof⟩

lemma update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip
  assumes "ip ≠ dip"
  shows "(update rt dip (dsn, dsk, flag, hops, nhip)) ip = rt ip"
  ⟨proof⟩

lemma nhop_update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip
  assumes "ip ≠ dip"
  shows "nhop (update rt dip (dsn, dsk, flag, hops, nhip)) ip = nhop rt ip"
  ⟨proof⟩

lemma dhops_update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip
  assumes "ip ≠ dip"
  shows "dhops (update rt dip (dsn, dsk, flag, hops, nhip)) ip = dhops rt ip"
  ⟨proof⟩

lemma sqn_update_same [simp]:
  "∧ rt ip dsn dsk flag hops nhip. sqn (rt(ip ↦ v)) ip = π2(v)"
  ⟨proof⟩

lemma dhops_update_changed [simp]:
  fixes rt dip osn hops nhip
  assumes "rt ≠ update rt dip (osn, kno, val, hops, nhip)"
  shows "the (dhops (update rt dip (osn, kno, val, hops, nhip)) dip) = hops"
  ⟨proof⟩

lemma nhop_update_unk_val [simp]:
  "∧ rt dip ip dsn hops.
  the (nhop (update rt dip (dsn, unk, val, hops, ip)) dip) = ip"
  ⟨proof⟩

lemma nhop_update_changed [simp]:
  fixes rt dip dsn dsk flg hops sip
  assumes "update rt dip (dsn, dsk, flg, hops, sip) ≠ rt"
  shows "the (nhop (update rt dip (dsn, dsk, flg, hops, sip)) dip) = sip"
  ⟨proof⟩

lemma update_rt_split_asm:
  "∧ rt ip dsn dsk flag hops sip.
  P (update rt ip (dsn, dsk, flag, hops, sip))
  =
  (¬(rt = update rt ip (dsn, dsk, flag, hops, sip) ∧ ¬P rt
  ∨ rt ≠ update rt ip (dsn, dsk, flag, hops, sip)
  ∧ ¬P (update rt ip (dsn, dsk, flag, hops, sip))))"
  ⟨proof⟩

lemma sqn_update [simp]: "∧ rt dip dsn flg hops sip.
  rt ≠ update rt dip (dsn, kno, flg, hops, sip)
  ⇒ sqn (update rt dip (dsn, kno, flg, hops, sip)) dip = dsn"
  ⟨proof⟩

```

**lemma** *sqnf\_update [simp]*: " $\wedge$ rt dip dsn dsk flg hops sip.  
 rt  $\neq$  update rt dip (dsn, dsk, flg, hops, sip)  
 $\implies$  sqnf (update rt dip (dsn, dsk, flg, hops, sip)) dip = dsk"  
 <proof>

**lemma** *update\_kno\_dsn\_greater\_zero*:  
 " $\wedge$ rt dip ip dsn hops.  $1 \leq$  dsn  $\implies 1 \leq$  (sqn (update rt dip (dsn, kno, val, hops, ip)) dip)"  
 <proof>

**lemma** *proj3\_update [simp]*: " $\wedge$ rt dip dsn dsk flg hops sip.  
 rt  $\neq$  update rt dip (dsn, dsk, flg, hops, sip)  
 $\implies \pi_3$ (the (update rt dip (dsn, dsk, flg, hops, sip) dip)) = dsk"  
 <proof>

**lemma** *nhop\_update\_changed\_kno\_val [simp]*: " $\wedge$ rt ip dsn dsk hops nhip.  
 rt  $\neq$  update rt ip (dsn, kno, val, hops, nhip)  
 $\implies$  the (nhop (update rt ip (dsn, kno, val, hops, nhip)) ip) = nhip"  
 <proof>

**lemma** *flag\_update [simp]*: " $\wedge$ rt dip dsn flg hops sip.  
 rt  $\neq$  update rt dip (dsn, kno, flg, hops, sip)  
 $\implies$  the (flag (update rt dip (dsn, kno, flg, hops, sip)) dip) = flg"  
 <proof>

**lemma** *the\_flag\_Some [dest!]*:  
 fixes ip rt  
 assumes "the (flag rt ip) = x"  
 and "ip  $\in$  kD rt"  
 shows "flag rt ip = Some x"  
 <proof>

**lemma** *kD\_update\_unchanged [dest]*:  
 fixes rt dip dsn dsk flag hops nhip  
 assumes "rt = update rt dip (dsn, dsk, flag, hops, nhip)"  
 shows "dip  $\in$  kD(rt)"  
 <proof>

**lemma** *nhop\_update [simp]*: " $\wedge$ rt dip dsn dsk flg hops sip.  
 rt  $\neq$  update rt dip (dsn, dsk, flg, hops, sip)  
 $\implies$  the (nhop (update rt dip (dsn, dsk, flg, hops, sip)) dip) = sip"  
 <proof>

**lemma** *sqn\_update\_another [simp]*:  
 fixes dip ip rt dsn dsk flag hops nhip  
 assumes "ip  $\neq$  dip"  
 shows "sqn (update rt dip (dsn, dsk, flag, hops, nhip)) ip = sqn rt ip"  
 <proof>

**lemma** *sqnf\_update\_another [simp]*:  
 fixes dip ip rt dsn dsk flag hops nhip  
 assumes "ip  $\neq$  dip"  
 shows "sqnf (update rt dip (dsn, dsk, flag, hops, nhip)) ip = sqnf rt ip"  
 <proof>

**lemma** *vD\_update\_val [dest]*:  
 " $\wedge$ dip rt dip' dsn dsk hops nhip.  
 dip  $\in$  vD(update rt dip' (dsn, dsk, val, hops, nhip))  $\implies$  (dip  $\in$  vD(rt)  $\vee$  dip=dip)"  
 <proof>

## Invalidating route entries

**definition** *invalidate* :: "rt  $\Rightarrow$  (ip  $\rightarrow$  sqn)  $\Rightarrow$  rt"  
 where "invalidate rt dests  $\equiv$



```

λip. case (rt ip, dests ip) of
  (None, _) ⇒ None
| (Some s, None) ⇒ Some s
| (Some (_, dsk, _, hops, nhip), Some rsn) ⇒
  Some (rsn, dsk, inv, hops, nhip)"

lemma proj3_invalidate [simp]:
  "∧dip. π3(the ((invalidate rt dests) dip)) = π3(the (rt dip))"
  ⟨proof⟩

lemma proj5_invalidate [simp]:
  "∧dip. π5(the ((invalidate rt dests) dip)) = π5(the (rt dip))"
  ⟨proof⟩

lemma proj6_invalidate [simp]:
  "∧dip. π6(the ((invalidate rt dests) dip)) = π6(the (rt dip))"
  ⟨proof⟩

lemma invalidate_kD_inv [simp]:
  "∧rt dests. kD (invalidate rt dests) = kD rt"
  ⟨proof⟩

lemma invalidate_sqn:
  fixes rt dip dests
  assumes "∀rsn. dests dip = Some rsn → sqn rt dip ≤ rsn"
  shows "sqn rt dip ≤ sqn (invalidate rt dests) dip"
  ⟨proof⟩

lemma sqn_invalidate_in_dests [simp]:
  fixes dests ipa rsn rt
  assumes "dests ipa = Some rsn"
  and "ipa ∈ kD(rt)"
  shows "sqn (invalidate rt dests) ipa = rsn"
  ⟨proof⟩

lemma dhops_invalidate [simp]:
  "∧dip. the (dhops (invalidate rt dests) dip) = the (dhops rt dip)"
  ⟨proof⟩

lemma sqnf_invalidate [simp]:
  "∧dip. sqnf (invalidate (rt ξ) (dests ξ)) dip = sqnf (rt ξ) dip"
  ⟨proof⟩

lemma nhop_invalidate [simp]:
  "∧dip. the (nhop (invalidate (rt ξ) (dests ξ)) dip) = the (nhop (rt ξ) dip)"
  ⟨proof⟩

lemma invalidate_other [simp]:
  fixes rt dests dip
  assumes "dip ∉ dom(dests)"
  shows "invalidate rt dests dip = rt dip"
  ⟨proof⟩

lemma invalidate_none [simp]:
  fixes rt dests dip
  assumes "dip ∉ kD(rt)"
  shows "invalidate rt dests dip = None"
  ⟨proof⟩

lemma vD_invalidate_vD_not_dests:
  "∧dip rt dests. dip ∈ vD(invalidate rt dests) ⇒ dip ∈ vD(rt) ∧ dests dip = None"
  ⟨proof⟩

```

```

lemma sqn_invalidate_not_in_dests [simp]:
  fixes dests dip rt
  assumes "dip ∉ dom(dests)"
  shows "sqn (invalidate rt dests) dip = sqn rt dip"
  ⟨proof⟩

```

```

lemma invalidate_changes:
  fixes rt dests dip dsn dsk flag hops nhip
  assumes "invalidate rt dests dip = Some (dsn, dsk, flag, hops, nhip)"
  shows " dsn = (case dests dip of None ⇒ π2(the (rt dip)) | Some rsn ⇒ rsn)
    ∧ dsk = π3(the (rt dip))
    ∧ flag = (if dests dip = None then π4(the (rt dip)) else inv)
    ∧ hops = π5(the (rt dip))
    ∧ nhip = π6(the (rt dip))"
  ⟨proof⟩

```

```

lemma proj3_inv: "∧dip rt dests. dip ∈ kD (rt)
  ⇒ π3(the (invalidate rt dests dip)) = π3(the (rt dip))"
  ⟨proof⟩

```

```

lemma dests_iD_invalidate [simp]:
  assumes "dests ip = Some rsn"
  and "ip ∈ kD(rt)"
  shows "ip ∈ iD(invalidate rt dests)"
  ⟨proof⟩

```

### 3.1.5 Route Requests

Generate a fresh route request identifier.

```

definition nrreqid :: "(ip × rreqid) set ⇒ ip ⇒ rreqid"
  where "nrreqid rreqs ip ≡ Max ({n. (ip, n) ∈ rreqs} ∪ {0}) + 1"

```

### 3.1.6 Queued Packets

Functions for sending data packets.

```

type_synonym store = "ip → (p × data list)"

```

```

definition sigma_queue :: "store ⇒ ip ⇒ data list" ("σqueue'(_, _)'")
  where "σqueue(store, dip) ≡ case store dip of None ⇒ [] | Some (p, q) ⇒ q"

```

```

definition qD :: "store ⇒ ip set"
  where "qD ≡ dom"

```

```

definition add :: "data ⇒ ip ⇒ store ⇒ store"
  where "add d dip store ≡ case store dip of
    None ⇒ store (dip ↦ (req, [d]))
    | Some (p, q) ⇒ store (dip ↦ (p, q @ [d]))"

```

```

lemma qD_add [simp]:
  fixes d dip store
  shows "qD(add d dip store) = insert dip (qD store)"
  ⟨proof⟩

```

```

definition drop :: "ip ⇒ store → store"
  where "drop dip store ≡
  map_option (λ(p, q). if tl q = [] then store (dip := None)
    else store (dip ↦ (p, tl q))) (store dip)"

```

```

definition sigma_p_flag :: "store ⇒ ip → p" ("σp-flag'(_, _)'")
  where "σp-flag(store, dip) ≡ map_option fst (store dip)"

```

```

definition unsetRRF :: "store ⇒ ip ⇒ store"

```

```

where "unsetRRF store dip  $\equiv$  case store dip of
      None  $\Rightarrow$  store
      | Some (p, q)  $\Rightarrow$  store (dip  $\mapsto$  (noreq, q))"

```

```

definition setRRF :: "store  $\Rightarrow$  (ip  $\rightarrow$  sqn)  $\Rightarrow$  store"
  where "setRRF store dests  $\equiv$   $\lambda$ dip. if dests dip = None then store dip
        else map_option ( $\lambda$ (_, q). (req, q)) (store dip)"

```

### 3.1.7 Comparison with the original technical report

The major differences with the AODV technical report of Fehnker et al are:

1. *nhop* is partial, thus a ‘*the*’ is needed, similarly for *dhops* and *addpreRT*.
2. *precs* is partial.
3.  $\sigma_{p\text{-flag}}(\text{store}, \text{dip})$  is partial.
4. The routing table (*rt*) is modelled as a map ( $\text{ip} \Rightarrow r \text{ option}$ ) rather than a set of 7-tuples, likewise, the *r* is a 6-tuple rather than a 7-tuple, i.e., the destination ip-address (*dip*) is taken from the argument to the function, rather than a part of the result. Well-definedness then follows from the structure of the type and more related facts are available automatically, rather than having to be acquired through tedious proofs.
5. Similar remarks hold for the dests mapping passed to *invalidate*, and *store*.

end

## 3.2 AODV protocol messages

```

theory C_Aodv_Message
imports C_Gtobcast
begin

```

```

datatype msg =
  Rreq nat rreqid ip sqn k ip sqn ip
  | Rrep nat ip sqn ip ip
  | Rerr "ip  $\rightarrow$  sqn" ip
  | Newpkt data ip
  | Pkt data ip ip

```

```

instantiation msg :: msg

```

```

begin

```

```

  definition newpkt_def [simp]: "newpkt  $\equiv$   $\lambda$ (d, dip). Newpkt d dip"
  definition eq_newpkt_def: "eq_newpkt m  $\equiv$  case m of Newpkt d dip  $\Rightarrow$  True | _  $\Rightarrow$  False"

```

```

  instance <proof>

```

```

end

```

The *msg* type models the different messages used within AODV. The instantiation as a *msg* is a technicality due to the special treatment of *newpkt* messages in the AWN SOS rules. This use of classes allows a clean separation of the AWN-specific definitions and these AODV-specific definitions.

```

definition rreq :: "nat  $\times$  rreqid  $\times$  ip  $\times$  sqn  $\times$  k  $\times$  ip  $\times$  sqn  $\times$  ip  $\Rightarrow$  msg"
  where "rreq  $\equiv$   $\lambda$ (hops, rreqid, dip, dsn, dsk, oip, osn, sip).
        Rreq hops rreqid dip dsn dsk oip osn sip"

```

```

lemma rreq_simp [simp]:

```

```

  "rreq(hops, rreqid, dip, dsn, dsk, oip, osn, sip) = Rreq hops rreqid dip dsn dsk oip osn sip"
  <proof>

```

```

definition rrep :: "nat  $\times$  ip  $\times$  sqn  $\times$  ip  $\times$  ip  $\Rightarrow$  msg"

```

```

  where "rrep  $\equiv$   $\lambda$ (hops, dip, dsn, oip, sip). Rrep hops dip dsn oip sip"

```

```

lemma rrep_simp [simp]:

```

```

"rrep(hops, dip, dsn, oip, sip) = Rrep hops dip dsn oip sip"
⟨proof⟩

definition rerr :: "(ip  $\rightarrow$  sqn)  $\times$  ip  $\Rightarrow$  msg"
  where "rerr  $\equiv$   $\lambda$ (dests, sip). Rerr dests sip"

lemma rerr_simp [simp]:
  "rerr(dests, sip) = Rerr dests sip"
  ⟨proof⟩

lemma not_eq_newpkt_rreq [simp]: " $\neg$ eq_newpkt (Rreq hops rreqid dip dsn dsk oip osn sip)"
  ⟨proof⟩

lemma not_eq_newpkt_rrep [simp]: " $\neg$ eq_newpkt (Rrep hops dip dsn oip sip)"
  ⟨proof⟩

lemma not_eq_newpkt_rerr [simp]: " $\neg$ eq_newpkt (Rerr dests sip)"
  ⟨proof⟩

lemma not_eq_newpkt_pkt [simp]: " $\neg$ eq_newpkt (Pkt d dip sip)"
  ⟨proof⟩

definition pkt :: "data  $\times$  ip  $\times$  ip  $\Rightarrow$  msg"
  where "pkt  $\equiv$   $\lambda$ (d, dip, sip). Pkt d dip sip"

lemma pkt_simp [simp]:
  "pkt(d, dip, sip) = Pkt d dip sip"
  ⟨proof⟩

end

```

### 3.3 The AODV protocol

```

theory C_Aodv
imports C_Aodv_Data C_Aodv_Message
  "../.../AWN/AWN_SOS_Labels" "../.../AWN/AWN_Invariants"
begin

```

#### 3.3.1 Data state

```

record state =
  ip      :: "ip"
  sn      :: "sqn"
  rt      :: "rt"
  rreqs  :: "(ip  $\times$  rreqid) set"
  store  :: "store"

  msg     :: "msg"
  data    :: "data"
  dests  :: "ip  $\rightarrow$  sqn"
  rreqid :: "rreqid"
  dip     :: "ip"
  oip     :: "ip"
  hops    :: "nat"
  dsn     :: "sqn"
  dsk     :: "k"
  osn     :: "sqn"
  sip     :: "ip"

abbreviation aodv_init :: "ip  $\Rightarrow$  state"
where "aodv_init i  $\equiv$  (
  ip = i,
  sn = 1,
  rt = empty,

```

```

rreqs = {},
store = empty,

msg    = (SOME x. True),
data   = (SOME x. True),
destds = (SOME x. True),
rreqid = (SOME x. True),
dip    = (SOME x. True),
oip    = (SOME x. True),
hops   = (SOME x. True),
dsn    = (SOME x. True),
dsk    = (SOME x. True),
osn    = (SOME x. True),
sip    = (SOME x. x ≠ i)
)"

```

**lemma** `some_neq_not_eq [simp]: "¬((SOME x :: nat. x ≠ i) = i)"`  
`<proof>`

**definition** `clear_locals :: "state ⇒ state"`

```

where "clear_locals ξ = ξ (|
  msg      := (SOME x. True),
  data     := (SOME x. True),
  destds   := (SOME x. True),
  rreqid   := (SOME x. True),
  dip      := (SOME x. True),
  oip      := (SOME x. True),
  hops     := (SOME x. True),
  dsn      := (SOME x. True),
  dsk      := (SOME x. True),
  osn      := (SOME x. True),
  sip      := (SOME x. x ≠ ip ξ)
)"

```

**lemma** `clear_locals_sip_not_ip [simp]: "¬(sip (clear_locals ξ) = ip ξ)"`  
`<proof>`

**lemma** `clear_locals_but_not_globals [simp]:`

```

"ip (clear_locals ξ) = ip ξ"
"sn (clear_locals ξ) = sn ξ"
"rt (clear_locals ξ) = rt ξ"
"rreqs (clear_locals ξ) = rreqs ξ"
"store (clear_locals ξ) = store ξ"
<proof>

```

### 3.3.2 Auxilliary message handling definitions

**definition** `is_newpkt`

```

where "is_newpkt ξ ≡ case msg ξ of
  Newpkt data' dip' ⇒ { ξ(|data := data', dip := dip'|) }
  | _ ⇒ {}"

```

**definition** `is_pkt`

```

where "is_pkt ξ ≡ case msg ξ of
  Pkt data' dip' oip' ⇒ { ξ(|data := data', dip := dip', oip := oip'|) }
  | _ ⇒ {}"

```

**definition** `is_rreq`

```

where "is_rreq ξ ≡ case msg ξ of
  Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip' ⇒
  { ξ(|hops := hops', rreqid := rreqid', dip := dip', dsn := dsn',
    dsk := dsk', oip := oip', osn := osn', sip := sip'|) }
  | _ ⇒ {}"

```

```

lemma is_rreq_asm [dest!]:
  assumes "ξ' ∈ is_rreq ξ"
  shows "(∃ hops' rreqid' dip' dsn' dsk' oip' osn' sip'.
    msg ξ = Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip' ∧
    ξ' = ξ(| hops := hops', rreqid := rreqid', dip := dip', dsn := dsn',
    dsk := dsk', oip := oip', osn := osn', sip := sip' |))"
  <proof>

```

```

definition is_rrep
where "is_rrep ξ ≡ case msg ξ of
  Rrep hops' dip' dsn' oip' sip' ⇒
    { ξ(| hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' |) }
  | _ ⇒ {}"

```

```

lemma is_rrep_asm [dest!]:
  assumes "ξ' ∈ is_rrep ξ"
  shows "(∃ hops' dip' dsn' oip' sip'.
    msg ξ = Rrep hops' dip' dsn' oip' sip' ∧
    ξ' = ξ(| hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' |))"
  <proof>

```

```

definition is_rerr
where "is_rerr ξ ≡ case msg ξ of
  Rerr dests' sip' ⇒ { ξ(| dests := dests', sip := sip' |) }
  | _ ⇒ {}"

```

```

lemma is_rerr_asm [dest!]:
  assumes "ξ' ∈ is_rerr ξ"
  shows "(∃ dests' sip'.
    msg ξ = Rerr dests' sip' ∧
    ξ' = ξ(| dests := dests', sip := sip' |))"
  <proof>

```

```

lemmas is_msg_defs =
  is_rerr_def is_rrep_def is_rreq_def is_pkt_def is_newpkt_def

```

```

lemma is_msg_inv_ip [simp]:
  "ξ' ∈ is_rerr ξ ⇒ ip ξ' = ip ξ"
  "ξ' ∈ is_rrep ξ ⇒ ip ξ' = ip ξ"
  "ξ' ∈ is_rreq ξ ⇒ ip ξ' = ip ξ"
  "ξ' ∈ is_pkt ξ ⇒ ip ξ' = ip ξ"
  "ξ' ∈ is_newpkt ξ ⇒ ip ξ' = ip ξ"
  <proof>

```

```

lemma is_msg_inv_sn [simp]:
  "ξ' ∈ is_rerr ξ ⇒ sn ξ' = sn ξ"
  "ξ' ∈ is_rrep ξ ⇒ sn ξ' = sn ξ"
  "ξ' ∈ is_rreq ξ ⇒ sn ξ' = sn ξ"
  "ξ' ∈ is_pkt ξ ⇒ sn ξ' = sn ξ"
  "ξ' ∈ is_newpkt ξ ⇒ sn ξ' = sn ξ"
  <proof>

```

```

lemma is_msg_inv_rt [simp]:
  "ξ' ∈ is_rerr ξ ⇒ rt ξ' = rt ξ"
  "ξ' ∈ is_rrep ξ ⇒ rt ξ' = rt ξ"
  "ξ' ∈ is_rreq ξ ⇒ rt ξ' = rt ξ"
  "ξ' ∈ is_pkt ξ ⇒ rt ξ' = rt ξ"
  "ξ' ∈ is_newpkt ξ ⇒ rt ξ' = rt ξ"
  <proof>

```

```

lemma is_msg_inv_rreqs [simp]:
  "ξ' ∈ is_rerr ξ ⇒ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_rrep ξ ⇒ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_rreq ξ ⇒ rreqs ξ' = rreqs ξ"

```

```

"ξ' ∈ is_pkt ξ    ⇒ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_newpkt ξ ⇒ rreqs ξ' = rreqs ξ"
⟨proof⟩

```

**lemma** *is\_msg\_inv\_store* [simp]:

```

"ξ' ∈ is_rerr ξ    ⇒ store ξ' = store ξ"
"ξ' ∈ is_rrep ξ    ⇒ store ξ' = store ξ"
"ξ' ∈ is_rreq ξ    ⇒ store ξ' = store ξ"
"ξ' ∈ is_pkt ξ     ⇒ store ξ' = store ξ"
"ξ' ∈ is_newpkt ξ  ⇒ store ξ' = store ξ"
⟨proof⟩

```

**lemma** *is\_msg\_inv\_sip* [simp]:

```

"ξ' ∈ is_pkt ξ     ⇒ sip ξ' = sip ξ"
"ξ' ∈ is_newpkt ξ ⇒ sip ξ' = sip ξ"
⟨proof⟩

```

### 3.3.3 The protocol process

**datatype** *pseqp* =

```

  PAadv
| PNewPkt
| PPkt
| PRreq
| PRrep
| PRerr

```

**fun** *nat\_of\_seqp* :: "pseqp ⇒ nat"

**where**

```

  "nat_of_seqp PAadv = 1"
| "nat_of_seqp PPkt = 2"
| "nat_of_seqp PNewPkt = 3"
| "nat_of_seqp PRreq = 4"
| "nat_of_seqp PRrep = 5"
| "nat_of_seqp PRerr = 6"

```

**instantiation** "pseqp" :: ord

**begin**

**definition** *less\_eq\_seqp* [iff]: "l1 ≤ l2 = (nat\_of\_seqp l1 ≤ nat\_of\_seqp l2)"

**definition** *less\_seqp* [iff]: "l1 < l2 = (nat\_of\_seqp l1 < nat\_of\_seqp l2)"

**instance** ⟨proof⟩

**end**

**abbreviation** *AADV*

**where**

```
"AAV ≡ λ_. [[clear_locals]] call(PAadv)"
```

**abbreviation** *PKT*

**where**

```
"PKT args ≡
```

```

  [ξ. let (data, dip, oip) = args ξ in
    (clear_locals ξ) (| data := data, dip := dip, oip := oip |)]
  call(PPkt)"

```

**abbreviation** *NEWPKT*

**where**

```

"NEWPKT args ≡
  [ξ. let (data, dip) = args ξ in
    (clear_locals ξ) (| data := data, dip := dip |)]
  call(PNewPkt)"

```

**abbreviation** *RREQ*

**where**

```
"RREQ args ≡
```

```

[[ξ. let (hops, rreqid, dip, dsn, dsk, oip, osn, sip) = args ξ in
  (clear_locals ξ) (| hops := hops, rreqid := rreqid, dip := dip,
    dsn := dsn, dsk := dsk, oip := oip,
    osn := osn, sip := sip |)]
call(PRreq)"

abbreviation RREP
where
  "RREP args ≡
  [[ξ. let (hops, dip, dsn, oip, sip) = args ξ in
    (clear_locals ξ) (| hops := hops, dip := dip, dsn := dsn,
      oip := oip, sip := sip |)]
  call(PRrep)"

abbreviation RERR
where
  "RERR args ≡
  [[ξ. let (dests, sip) = args ξ in
    (clear_locals ξ) (| dests := dests, sip := sip |)]
  call(PRerr)"

fun ΓAODV :: "(state, msg, pseqp, pseqp label) seqp_env"
where
  "ΓAODV PAodv = labelled PAodv (
    receive(λmsg' ξ. ξ (| msg := msg' |)).
    (
      <is_newpkt> NEWPKT(λξ. (data ξ, ip ξ))
      ⊕ <is_pkt> PKT(λξ. (data ξ, dip ξ, oip ξ))
      ⊕ <is_rreq>
        [[ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ) |)]
        RREQ(λξ. (hops ξ, rreqid ξ, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, sip ξ))
      ⊕ <is_rrep>
        [[ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ) |)]
        RREP(λξ. (hops ξ, dip ξ, dsn ξ, oip ξ, sip ξ))
      ⊕ <is_rerr>
        [[ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ) |)]
        RERR(λξ. (dests ξ, sip ξ))
    )
    ⊕ <λξ. { ξ(| dip := dip |) | dip. dip ∈ qD(store ξ) ∩ vD(rt ξ) }>
      [[ξ. ξ (| data := hd(σqueue(store ξ, dip ξ)) |)]
      unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, ip ξ)).
      [[ξ. ξ (| store := the (drop (dip ξ) (store ξ)) |)]
      AODV()
      ▷ [[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
        then Some (inc (sqn (rt ξ) rip)) else None) |)]
        [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)]
        [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |)]
        broadcast(λξ. rerr(dests ξ, ip ξ)). AODV()
    ⊕ <λξ. { ξ(| dip := dip |)
      | dip. dip ∈ qD(store ξ) - vD(rt ξ) ∧ the (σp-flag(store ξ, dip)) = req }>
      [[ξ. ξ (| store := unsetRRF (store ξ) (dip ξ) |)]
      [[ξ. ξ (| sn := inc (sn ξ) |)]
      [[ξ. ξ (| rreqid := nrreqid (rreqs ξ) (ip ξ) |)]
      [[ξ. ξ (| rreqs := rreqs ξ ∪ {(ip ξ, rreqid ξ)} |)]
      broadcast(λξ. rreq(0, rreqid ξ, dip ξ, sqn (rt ξ) (dip ξ), sqnf (rt ξ) (dip ξ), ip ξ, sn ξ,
        ip ξ)). AODV())"

  | "ΓAODV PNewPkt = labelled PNewPkt (
    <ξ. dip ξ = ip ξ>
    deliver(λξ. data ξ).AODV()
    ⊕ <ξ. dip ξ ≠ ip ξ>
    [[ξ. ξ (| store := add (data ξ) (dip ξ) (store ξ) |)]
    AODV())"

  | "ΓAODV PPkt = labelled PPkt (

```



```

⟨ξ. dip ξ = ip ξ⟩
  deliver(λξ. data ξ).AODV()
⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
(
  ⟨ξ. dip ξ ∈ vD (rt ξ)⟩
    unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, oip ξ)).AODV()
  ▷
    [[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
      then Some (inc (sqn (rt ξ) rip)) else None) |]]
    [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |]]
    [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |]]
    broadcast(λξ. rerr(dests ξ, ip ξ)).AODV()
  ⊕ ⟨ξ. dip ξ ∉ vD (rt ξ)⟩
  (
    ⟨ξ. dip ξ ∈ iD (rt ξ)⟩
      broadcast(λξ. rerr([dip ξ ↦ sqn (rt ξ) (dip ξ)], ip ξ)). AODV()
    ⊕ ⟨ξ. dip ξ ∉ iD (rt ξ)⟩
      AODV()
  )
)
))"

```

```

| "ΓAODV PRreq = labelled PRreq (
  ⟨ξ. (oip ξ, rreqid ξ) ∈ rreqs ξ⟩
  AODV()
  ⊕ ⟨ξ. (oip ξ, rreqid ξ) ∉ rreqs ξ⟩
  [[ξ. ξ (| rt := update (rt ξ) (oip ξ) (osn ξ, kno, val, hops ξ + 1, sip ξ) |]]
  [[ξ. ξ (| rreqs := rreqs ξ ∪ {(oip ξ, rreqid ξ)} |]]
  (
    ⟨ξ. dip ξ = ip ξ⟩
    [[ξ. ξ (| sn := max (sn ξ) (dsn ξ) |]]
    unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(0, dip ξ, sn ξ, oip ξ, ip ξ)).AODV()
  ▷
    [[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
      then Some (inc (sqn (rt ξ) rip)) else None) |]]
    [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |]]
    [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |]]
    broadcast(λξ. rerr(dests ξ, ip ξ)).AODV()
  ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
  (
    ⟨ξ. dip ξ ∈ vD (rt ξ) ∧ dsn ξ ≤ sqn (rt ξ) (dip ξ) ∧ sqnf (rt ξ) (dip ξ) = kno⟩
    unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(the (dhops (rt ξ) (dip ξ)), dip ξ,
      sqn (rt ξ) (dip ξ), oip ξ, ip ξ)).
    AODV()
  ▷
    [[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
      then Some (inc (sqn (rt ξ) rip)) else None) |]]
    [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |]]
    [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |]]
    broadcast(λξ. rerr(dests ξ, ip ξ)).AODV()
  ⊕ ⟨ξ. dip ξ ∉ vD (rt ξ) ∨ sqn (rt ξ) (dip ξ) < dsn ξ ∨ sqnf (rt ξ) (dip ξ) = unk⟩
    broadcast(λξ. rreq(hops ξ + 1, rreqid ξ, dip ξ, max (sqn (rt ξ) (dip ξ)) (dsn ξ),
      dsk ξ, oip ξ, osn ξ, ip ξ)).
    AODV()
  )
)
))"

```

```

| "ΓAODV PRrep = labelled PRrep (
  ⟨ξ. rt ξ ≠ update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ) ⟩
  (
    [[ξ. ξ (| rt := update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ) | ] ]
  (
    ⟨ξ. oip ξ = ip ξ ⟩
    AODV()
  ⊕ ⟨ξ. oip ξ ≠ ip ξ ⟩

```

```

(
  ⟨ξ. oip ξ ∈ vD (rt ξ)⟩
  unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(hops ξ + 1, dip ξ,
    dsn ξ, oip ξ, ip ξ)).
  AODV()
▷
  [[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
    then Some (inc (sqn (rt ξ) rip)) else None) |)]]
  [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)]]
  [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |)]]
  broadcast(λξ. rerr(dests ξ, ip ξ)).AODV()
⊕ ⟨ξ. oip ξ ∉ vD (rt ξ)⟩
  AODV()
)
)
)
⊕ ⟨ξ. rt ξ = update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ) ⟩
  AODV()
)"

| "ΓAODV PRerr = labelled PRerr (
  [[ξ. ξ (| dests := (λrip. case (dests ξ) rip of None ⇒ None
    | Some rsn ⇒ if rip ∈ vD (rt ξ) ∧ the (nhop (rt ξ) rip) = sip ξ
      ∧ sqn (rt ξ) rip < rsn then Some rsn else None) |)]]
  [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)]]
  [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |)]]
  (
    ⟨ξ. dests ξ ≠ Map.empty⟩
    broadcast(λξ. rerr(dests ξ, ip ξ)). AODV()
    ⊕ ⟨ξ. dests ξ = Map.empty⟩
    AODV()
  ))"

```

```

declare ΓAODV.simps [simp del, code del]
lemmas ΓAODV.simps [simp, code] = ΓAODV.simps [simplified]

```

```

fun ΓAODV.skeleton
where
  "ΓAODV.skeleton PAadv = seqp_skeleton (ΓAODV PAadv)"
  | "ΓAODV.skeleton PNewPkt = seqp_skeleton (ΓAODV PNewPkt)"
  | "ΓAODV.skeleton PPkt = seqp_skeleton (ΓAODV PPkt)"
  | "ΓAODV.skeleton PRreq = seqp_skeleton (ΓAODV PRreq)"
  | "ΓAODV.skeleton PRrep = seqp_skeleton (ΓAODV PRrep)"
  | "ΓAODV.skeleton PRerr = seqp_skeleton (ΓAODV PRerr)"

```

```

lemma ΓAODV.skeleton_wf [simp]:
  "wellformed ΓAODV.skeleton"
  ⟨proof⟩

```

```

declare ΓAODV.skeleton.simps [simp del, code del]
lemmas ΓAODV.skeleton.simps [simp, code]
  = ΓAODV.skeleton.simps [simplified ΓAODV.simps seqp_skeleton.simps]

```

```

lemma aodv_proc_cases [dest]:
  fixes p pn
  shows "p ∈ cterms1 (ΓAODV pn) ⇒
    (p ∈ cterms1 (ΓAODV PAadv) ∨
     p ∈ cterms1 (ΓAODV PNewPkt) ∨
     p ∈ cterms1 (ΓAODV PPkt) ∨
     p ∈ cterms1 (ΓAODV PRreq) ∨
     p ∈ cterms1 (ΓAODV PRrep) ∨
     p ∈ cterms1 (ΓAODV PRerr))"

```

*<proof>*

**definition**  $\sigma_{AODV} :: "ip \Rightarrow (state \times (state, msg, pseqp, pseqp\ label) seqp) set"$   
**where** " $\sigma_{AODV} i \equiv \{(aodv\_init\ i, \Gamma_{AODV}\ PAodv)\}$ "

**abbreviation**  $paodv$

$:: "ip \Rightarrow (state \times (state, msg, pseqp, pseqp\ label) seqp, msg\ seq\_action) automaton"$

**where**

" $paodv\ i \equiv (\mid\ init = \sigma_{AODV}\ i, trans = seqp\_sos\ \Gamma_{AODV}\ \mid)$ "

**lemma**  $aodv\_trans: "trans\ (paodv\ i) = seqp\_sos\ \Gamma_{AODV}"$

*<proof>*

**lemma**  $aodv\_control\_within\ [simp]: "control\_within\ \Gamma_{AODV}\ (init\ (paodv\ i))"$

*<proof>*

**lemma**  $aodv\_wf\ [simp]:$

" $wellformed\ \Gamma_{AODV}$ "

*<proof>*

**lemmas**  $aodv\_labels\_not\_empty\ [simp] = labels\_not\_empty\ [OF\ aodv\_wf]$

**lemma**  $aodv\_ex\_label\ [intro]: "\exists l. l \in labels\ \Gamma_{AODV}\ p"$

*<proof>*

**lemma**  $aodv\_ex\_labelE\ [elim]:$

**assumes** " $\forall l \in labels\ \Gamma_{AODV}\ p. P\ l\ p$ "

**and** " $\exists p\ l. P\ l\ p \implies Q$ "

**shows** " $Q$ "

*<proof>*

**lemma**  $aodv\_simple\_labels\ [simp]: "simple\_labels\ \Gamma_{AODV}"$

*<proof>*

**lemma**  $\sigma_{AODV\_labels}\ [simp]: "(\xi, p) \in \sigma_{AODV}\ i \implies labels\ \Gamma_{AODV}\ p = \{PAodv-:0\}"$

*<proof>*

**lemma**  $aodv\_init\_kD\_empty\ [simp]:$

" $(\xi, p) \in \sigma_{AODV}\ i \implies kD\ (rt\ \xi) = \{\}$ "

*<proof>*

**lemma**  $aodv\_init\_sip\_not\_ip\ [simp]: "\neg (sip\ (aodv\_init\ i) = i)"$  *<proof>*

**lemma**  $aodv\_init\_sip\_not\_ip'\ [simp]:$

**assumes** " $(\xi, p) \in \sigma_{AODV}\ i$ "

**shows** " $sip\ \xi \neq ip\ \xi$ "

*<proof>*

**lemma**  $aodv\_init\_sip\_not\_i\ [simp]:$

**assumes** " $(\xi, p) \in \sigma_{AODV}\ i$ "

**shows** " $sip\ \xi \neq i$ "

*<proof>*

**lemma**  $clear\_locals\_sip\_not\_ip':$

**assumes** " $ip\ \xi = i$ "

**shows** " $\neg (sip\ (clear\_locals\ \xi) = i)"$

*<proof>*

Stop the simplifier from descending into process terms.

**declare**  $seqp\_congs\ [cong]$

Configure the main invariant tactic for AODV.

**declare**

$\Gamma_{AODV\_simps}\ [cterm\_env]$

```

aadv_proc_cases [ctermssl_cases]
seq_invariant_ctermssl [OF aadv_wf aadv_control_within aadv_simple_labels aadv_trans,
                          cterms_intros]
seq_step_invariant_ctermssl [OF aadv_wf aadv_control_within aadv_simple_labels aadv_trans,
                              cterms_intros]

```

end

### 3.4 Invariant assumptions and properties

```

theory C_Aadv_Predicates
imports C_Aadv
begin

```

Definitions for expression assumptions on incoming messages and properties of outgoing messages.

```

abbreviation not_Pkt :: "msg  $\Rightarrow$  bool"
where "not_Pkt m  $\equiv$  case m of Pkt _ _ _  $\Rightarrow$  False | _  $\Rightarrow$  True"

```

```

definition msg_sender :: "msg  $\Rightarrow$  ipc"
where "msg_sender m  $\equiv$  case m of Rreq _ _ _ _ _ ipc  $\Rightarrow$  ipc
      | Rrep _ _ _ _ ipc  $\Rightarrow$  ipc
      | Rerr _ ipc  $\Rightarrow$  ipc
      | Pkt _ _ ipc  $\Rightarrow$  ipc"

```

```

lemma msg_sender_simps [simp]:
  " $\bigwedge$  hops rreqid dip dsn dsk oip osn sip.
    msg_sender (Rreq hops rreqid dip dsn dsk oip osn sip) = sip"
  " $\bigwedge$  hops dip dsn oip sip. msg_sender (Rrep hops dip dsn oip sip) = sip"
  " $\bigwedge$  dests sip. msg_sender (Rerr dests sip) = sip"
  " $\bigwedge$  d dip sip. msg_sender (Pkt d dip sip) = sip"
  <proof>

```

```

definition msg_zhops :: "msg  $\Rightarrow$  bool"
where "msg_zhops m  $\equiv$  case m of
      Rreq hopsc _ dipc _ _ oipc _ sipc  $\Rightarrow$  hopsc = 0  $\longrightarrow$  oipc = sipc
      | Rrep hopsc dipc _ _ sipc  $\Rightarrow$  hopsc = 0  $\longrightarrow$  dipc = sipc
      | _  $\Rightarrow$  True"

```

```

lemma msg_zhops_simps [simp]:
  " $\bigwedge$  hops rreqid dip dsn dsk oip osn sip.
    msg_zhops (Rreq hops rreqid dip dsn dsk oip osn sip) = (hops = 0  $\longrightarrow$  oip = sip)"
  " $\bigwedge$  hops dip dsn oip sip. msg_zhops (Rrep hops dip dsn oip sip) = (hops = 0  $\longrightarrow$  dip = sip)"
  " $\bigwedge$  dests sip. msg_zhops (Rerr dests sip) = True"
  " $\bigwedge$  d dip. msg_zhops (Newpkt d dip) = True"
  " $\bigwedge$  d dip sip. msg_zhops (Pkt d dip sip) = True"
  <proof>

```

```

definition rreq_rrep_sn :: "msg  $\Rightarrow$  bool"
where "rreq_rrep_sn m  $\equiv$  case m of Rreq _ _ _ _ _ osnc _  $\Rightarrow$  osnc  $\geq$  1
      | Rrep _ _ dsnc _ _  $\Rightarrow$  dsnc  $\geq$  1
      | _  $\Rightarrow$  True"

```

```

lemma rreq_rrep_sn_simps [simp]:
  " $\bigwedge$  hops rreqid dip dsn dsk oip osn sip.
    rreq_rrep_sn (Rreq hops rreqid dip dsn dsk oip osn sip) = (osn  $\geq$  1)"
  " $\bigwedge$  hops dip dsn oip sip. rreq_rrep_sn (Rrep hops dip dsn oip sip) = (dsn  $\geq$  1)"
  " $\bigwedge$  dests sip. rreq_rrep_sn (Rerr dests sip) = True"
  " $\bigwedge$  d dip. rreq_rrep_sn (Newpkt d dip) = True"
  " $\bigwedge$  d dip sip. rreq_rrep_sn (Pkt d dip sip) = True"
  <proof>

```

```

definition rreq_rrep_fresh :: "rt  $\Rightarrow$  msg  $\Rightarrow$  bool"
where "rreq_rrep_fresh crt m  $\equiv$  case m of Rreq hopsc _ _ _ oipc osnc ipcc  $\Rightarrow$  (ipcc  $\neq$  oipc  $\longrightarrow$ 
      oipc  $\in$  kD(crt)  $\wedge$  (sqn crt oipc  $>$  osnc)

```

```

      ∨ (sqn crt oipc = osnc
        ∧ the (dhops crt oipc) ≤ hopscc
        ∧ the (flag crt oipc) = val)))
    | Rrep hopscc dipcc dsnc _ ipcc ⇒ (ipcc ≠ dipcc →
      dipcc ∈ kD(crt)
      ∧ sqn crt dipcc = dsnc
      ∧ the (dhops crt dipcc) = hopscc
      ∧ the (flag crt dipcc) = val)
  | _ ⇒ True"

```

**lemma rreq\_rrep\_fresh [simp]:**

```

"∧hops rreqid dip dsn dsk oip osn sip.
  rreq_rrep_fresh crt (Rreq hops rreqid dip dsn dsk oip osn sip) =
    (sip ≠ oip → oip ∈ kD(crt)
     ∧ (sqn crt oip > osn
       ∨ (sqn crt oip = osn
         ∧ the (dhops crt oip) ≤ hops
         ∧ the (flag crt oip) = val))))"
"∧hops dip dsn oip sip. rreq_rrep_fresh crt (Rrep hops dip dsn oip sip) =
  (sip ≠ dip → dip ∈ kD(crt)
   ∧ sqn crt dip = dsn
   ∧ the (dhops crt dip) = hops
   ∧ the (flag crt dip) = val)"
"∧dests sip.      rreq_rrep_fresh crt (Rerr dests sip) = True"
"∧d dip.          rreq_rrep_fresh crt (Newpkt d dip) = True"
"∧d dip sip.     rreq_rrep_fresh crt (Pkt d dip sip) = True"
⟨proof⟩

```

**definition rerr\_invalid :: "rt ⇒ msg ⇒ bool"**

```

where "rerr_invalid crt m ≡ case m of Rerr destsc _ ⇒ (∀rip ∈ dom(destsc).
  (rip ∈ iD(crt) ∧ the (destsc rip) = sqn crt rip))
  | _ ⇒ True"

```

**lemma rerr\_invalid [simp]:**

```

"∧hops rreqid dip dsn dsk oip osn sip.
  rerr_invalid crt (Rreq hops rreqid dip dsn dsk oip osn sip) = True"
"∧hops dip dsn oip sip. rerr_invalid crt (Rrep hops dip dsn oip sip) = True"
"∧dests sip.          rerr_invalid crt (Rerr dests sip) = (∀rip ∈ dom(dests).
  rip ∈ iD(crt) ∧ the (dests rip) = sqn crt rip)"
"∧d dip.              rerr_invalid crt (Newpkt d dip) = True"
"∧d dip sip.         rerr_invalid crt (Pkt d dip sip) = True"
⟨proof⟩

```

**definition**

```

initmissing :: "(nat ⇒ state option) × 'a ⇒ (nat ⇒ state) × 'a"

```

**where**

```

"initmissing σ = (λi. case (fst σ) i of None ⇒ aadv_init i | Some s ⇒ s, snd σ)"

```

**lemma not\_in\_net\_ips\_fst\_init\_missing [simp]:**

```

assumes "i ∉ net_ips σ"
  shows "fst (initmissing (netgmap fst σ)) i = aadv_init i"
⟨proof⟩

```

**lemma fst\_initmissing\_netgmap\_pair\_fst [simp]:**

```

"fst (initmissing (netgmap (λ(p, q). (fst (id p), snd (id p), q)) s))
  = fst (initmissing (netgmap fst s))"
⟨proof⟩

```

We introduce a streamlined alternative to `initmissing` with `netgmap` to simplify invariant statements and thus facilitate their comprehension and presentation.

**lemma fst\_initmissing\_netgmap\_default\_aadv\_init\_netlift:**

```

"fst (initmissing (netgmap fst s)) = default aadv_init (netlift fst s)"
⟨proof⟩

```

**definition**

```
netglobal :: "(nat  $\Rightarrow$  state)  $\Rightarrow$  bool  $\Rightarrow$  ((state  $\times$  'b)  $\times$  'c) net_state  $\Rightarrow$  bool"
```

**where**

```
"netglobal P  $\equiv$  ( $\lambda$ s. P (default aadv_init (netlift fst s)))"
```

**end**

## 3.5 Quality relations between routes

**theory** C\_Fresher

**imports** C\_Aadv\_Data

**begin**

### 3.5.1 Net sequence numbers

**On individual routes**

**definition**

```
nsqnr :: "r  $\Rightarrow$  sqn"
```

**where**

```
"nsqnr r  $\equiv$  if  $\pi_4(r) = \text{val} \vee \pi_2(r) = 0$  then  $\pi_2(r)$  else ( $\pi_2(r) - 1$ )"
```

**lemma** nsqnr\_def':

```
"nsqnr r = (if  $\pi_4(r) = \text{inv}$  then  $\pi_2(r) - 1$  else  $\pi_2(r)$ )"
```

*<proof>*

**lemma** nsqn<sub>r</sub>\_zero [simp]:

```
" $\bigwedge$ dsn dsk flag hops nhip. nsqnr (0, dsk, flag, hops, nhip) = 0"
```

*<proof>*

**lemma** nsqn<sub>r</sub>\_val [simp]:

```
" $\bigwedge$ dsn dsk hops nhip. nsqnr (dsn, dsk, val, hops, nhip) = dsn"
```

*<proof>*

**lemma** nsqn<sub>r</sub>\_inv [simp]:

```
" $\bigwedge$ dsn dsk hops nhip. nsqnr (dsn, dsk, inv, hops, nhip) = dsn - 1"
```

*<proof>*

**lemma** nsqn<sub>r</sub>\_lte\_dsn [simp]:

```
" $\bigwedge$ dsn dsk flag hops nhip. nsqnr (dsn, dsk, flag, hops, nhip)  $\leq$  dsn"
```

*<proof>*

**On routes in routing tables**

**definition**

```
nsqn :: "rt  $\Rightarrow$  ip  $\Rightarrow$  sqn"
```

**where**

```
"nsqn  $\equiv$   $\lambda$ rt dip. case  $\sigma_{\text{route}}(rt, dip)$  of None  $\Rightarrow$  0 | Some r  $\Rightarrow$  nsqnr(r)"
```

**lemma** nsqn\_sqn\_def:

```
" $\bigwedge$ rt dip. nsqn rt dip = (if flag rt dip = Some val  $\vee$  sqn rt dip = 0  
then sqn rt dip else sqn rt dip - 1)"
```

*<proof>*

**lemma** not\_in\_kD\_nsqn [simp]:

```
assumes "dip  $\notin$  kD(rt)"
```

```
shows "nsqn rt dip = 0"
```

*<proof>*

**lemma** kD\_nsqn:

```
assumes "dip  $\in$  kD(rt)"
```

```
shows "nsqn rt dip = nsqnr(the ( $\sigma_{\text{route}}(rt, dip)$ ))"
```

*<proof>*

```

lemma nsqnr_r_flag_pred [simp, intro]:
  fixes dsn dsk flag hops nhip
  assumes "P (nsqnr (dsn, dsk, val, hops, nhip))"
    and "P (nsqnr (dsn, dsk, inv, hops, nhip))"
  shows "P (nsqnr (dsn, dsk, flag, hops, nhip))"
  ⟨proof⟩

lemma sqn_nsqn:
  "∧rt dip. sqn rt dip - 1 ≤ nsqn rt dip"
  ⟨proof⟩

lemma nsqn_sqn: "nsqn rt dip ≤ sqn rt dip"
  ⟨proof⟩

lemma val_nsqn_sqn [elim, simp]:
  assumes "ip ∈ kD(rt)"
    and "the (flag rt ip) = val"
  shows "nsqn rt ip = sqn rt ip"
  ⟨proof⟩

lemma vD_nsqn_sqn [elim, simp]:
  assumes "ip ∈ vD(rt)"
  shows "nsqn rt ip = sqn rt ip"
  ⟨proof⟩

lemma inv_nsqn_sqn [elim, simp]:
  assumes "ip ∈ kD(rt)"
    and "the (flag rt ip) = inv"
  shows "nsqn rt ip = sqn rt ip - 1"
  ⟨proof⟩

lemma iD_nsqn_sqn [elim, simp]:
  assumes "ip ∈ iD(rt)"
  shows "nsqn rt ip = sqn rt ip - 1"
  ⟨proof⟩

lemma nsqn_update_changed_kno_val [simp]: "∧rt ip dsn dsk hops nhip.
  rt ≠ update rt ip (dsn, kno, val, hops, nhip)
  ⇒ nsqn (update rt ip (dsn, kno, val, hops, nhip)) ip = dsn"
  ⟨proof⟩

lemma nsqn_update_other [simp]:
  fixes dsn dsk flag hops dip nhip rt ip
  assumes "dip ≠ ip"
  shows "nsqn (update rt ip (dsn, dsk, flag, hops, nhip)) dip = nsqn rt dip"
  ⟨proof⟩

lemma nsqn_invalidate_eq:
  assumes "dip ∈ kD(rt)"
    and "dests dip = Some rsn"
  shows "nsqn (invalidate rt dests) dip = rsn - 1"
  ⟨proof⟩

lemma nsqn_invalidate_other [simp]:
  assumes "dip ∈ kD(rt)"
    and "dip ∉ dom dests"
  shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
  ⟨proof⟩

```

### 3.5.2 Comparing routes

**definition**

```
fresher :: "r ⇒ r ⇒ bool" ("(_/ ⊑ _)" [51, 51] 50)
```

where

"fresher r r'  $\equiv ((\text{nsqn}_r r < \text{nsqn}_r r') \vee (\text{nsqn}_r r = \text{nsqn}_r r' \wedge \pi_5(r) \geq \pi_5(r'))))$ "

**lemma fresherI1 [intro]:**

assumes "nsqn<sub>r</sub> r < nsqn<sub>r</sub> r'"  
 shows "r  $\sqsubseteq$  r'"  
 <proof>

**lemma fresherI2 [intro]:**

assumes "nsqn<sub>r</sub> r = nsqn<sub>r</sub> r'"  
 and " $\pi_5(r) \geq \pi_5(r')$ "  
 shows "r  $\sqsubseteq$  r'"  
 <proof>

**lemma fresherI [intro]:**

assumes " $(\text{nsqn}_r r < \text{nsqn}_r r') \vee (\text{nsqn}_r r = \text{nsqn}_r r' \wedge \pi_5(r) \geq \pi_5(r'))$ "  
 shows "r  $\sqsubseteq$  r'"  
 <proof>

**lemma fresherE [elim]:**

assumes "r  $\sqsubseteq$  r'"  
 and "nsqn<sub>r</sub> r < nsqn<sub>r</sub> r'  $\implies P r r'$ "  
 and "nsqn<sub>r</sub> r = nsqn<sub>r</sub> r'  $\wedge \pi_5(r) \geq \pi_5(r') \implies P r r'$ "  
 shows "P r r'"  
 <proof>

**lemma fresher\_refl [simp]:** "r  $\sqsubseteq$  r"

<proof>

**lemma fresher\_trans [elim, trans]:**

"[ x  $\sqsubseteq$  y; y  $\sqsubseteq$  z ]  $\implies x \sqsubseteq z$ "  
 <proof>

**lemma not\_fresher\_trans [elim, trans]:**

"[  $\neg(x \sqsubseteq y)$ ;  $\neg(z \sqsubseteq x)$  ]  $\implies \neg(z \sqsubseteq y)$ "  
 <proof>

**lemma fresher\_dsn\_flag\_hops\_const [simp]:**

fixes dsn dsk dsk' flag hops nhip nhip'  
 shows "(dsn, dsk, flag, hops, nhip)  $\sqsubseteq$  (dsn, dsk', flag, hops, nhip)'"  
 <proof>

### 3.5.3 Comparing routing tables

**definition**

rt\_fresher :: "ip  $\Rightarrow$  rt  $\Rightarrow$  rt  $\Rightarrow$  bool"

where

"rt\_fresher  $\equiv \lambda \text{dip } rt \text{ rt}'. (\text{the } (\sigma_{\text{route}}(rt, \text{dip}))) \sqsubseteq (\text{the } (\sigma_{\text{route}}(rt', \text{dip})))$ "

**abbreviation**

rt\_fresher\_syn :: "rt  $\Rightarrow$  ip  $\Rightarrow$  rt  $\Rightarrow$  bool" ("(\_/  $\sqsubseteq$  \_)" [51, 999, 51] 50)

where

"rt1  $\sqsubseteq_i$  rt2  $\equiv$  rt\_fresher i rt1 rt2"

**lemma rt\_fresher\_def':**

"(rt1  $\sqsubseteq_i$  rt2) = (nsqn<sub>r</sub> (the (rt1 i)) < nsqn<sub>r</sub> (the (rt2 i))  $\vee$   
 nsqn<sub>r</sub> (the (rt1 i)) = nsqn<sub>r</sub> (the (rt2 i))  $\wedge \pi_5$  (the (rt2 i))  $\leq \pi_5$  (the (rt1 i)))"

<proof>

**lemma single\_rt\_fresher [intro]:**

assumes "the (rt1 ip)  $\sqsubseteq$  the (rt2 ip)"  
 shows "rt1  $\sqsubseteq_{\text{ip}}$  rt2"  
 <proof>

**lemma rt\_fresher\_single [intro]:**



```

assumes "rt1  $\sqsubseteq_{ip}$  rt2"
shows "the (rt1 ip)  $\sqsubseteq$  the (rt2 ip)"
⟨proof⟩

lemma rt_fresher_def2:
assumes "dip  $\in$  kD(rt1)"
and "dip  $\in$  kD(rt2)"
shows "(rt1  $\sqsubseteq_{dip}$  rt2) = (nsqn rt1 dip < nsqn rt2 dip
 $\vee$  (nsqn rt1 dip = nsqn rt2 dip
 $\wedge$  the (dhops rt1 dip)  $\geq$  the (dhops rt2 dip)))"
⟨proof⟩

lemma rt_fresherI1 [intro]:
assumes "dip  $\in$  kD(rt1)"
and "dip  $\in$  kD(rt2)"
and "nsqn rt1 dip < nsqn rt2 dip"
shows "rt1  $\sqsubseteq_{dip}$  rt2"
⟨proof⟩

lemma rt_fresherI2 [intro]:
assumes "dip  $\in$  kD(rt1)"
and "dip  $\in$  kD(rt2)"
and "nsqn rt1 dip = nsqn rt2 dip"
and "the (dhops rt1 dip)  $\geq$  the (dhops rt2 dip)"
shows "rt1  $\sqsubseteq_{dip}$  rt2"
⟨proof⟩

lemma rt_fresherE [elim]:
assumes "rt1  $\sqsubseteq_{dip}$  rt2"
and "dip  $\in$  kD(rt1)"
and "dip  $\in$  kD(rt2)"
and "[[ nsqn rt1 dip < nsqn rt2 dip ]  $\implies$  P rt1 rt2 dip"
and "[[ nsqn rt1 dip = nsqn rt2 dip;
the (dhops rt1 dip)  $\geq$  the (dhops rt2 dip) ]  $\implies$  P rt1 rt2 dip"
shows "P rt1 rt2 dip"
⟨proof⟩

lemma rt_fresher_refl [simp]: "rt  $\sqsubseteq_{dip}$  rt"
⟨proof⟩

lemma rt_fresher_trans [elim, trans]:
assumes "rt1  $\sqsubseteq_{dip}$  rt2"
and "rt2  $\sqsubseteq_{dip}$  rt3"
shows "rt1  $\sqsubseteq_{dip}$  rt3"
⟨proof⟩

lemma rt_fresher_if_Some [intro!]:
assumes "the (rt dip)  $\sqsubseteq$  r"
shows "rt  $\sqsubseteq_{dip}$  ( $\lambda ip.$  if ip = dip then Some r else rt ip)"
⟨proof⟩

definition rt_fresh_as :: "ip  $\Rightarrow$  rt  $\Rightarrow$  rt  $\Rightarrow$  bool"
where
"rt_fresh_as  $\equiv$   $\lambda dip$  rt1 rt2. (rt1  $\sqsubseteq_{dip}$  rt2)  $\wedge$  (rt2  $\sqsubseteq_{dip}$  rt1)"

abbreviation
rt_fresh_as_syn :: "rt  $\Rightarrow$  ip  $\Rightarrow$  rt  $\Rightarrow$  bool" ("(_/  $\approx$  _)" [51, 999, 51] 50)
where
"rt1  $\approx_i$  rt2  $\equiv$  rt_fresh_as i rt1 rt2"

lemma rt_fresh_as_refl [simp]: " $\wedge$ rt dip. rt  $\approx_{dip}$  rt"
⟨proof⟩

lemma rt_fresh_as_trans [simp, intro, trans]:

```

" $\wedge rt1\ rt2\ rt3\ dip. \llbracket rt1 \approx_{dip} rt2; rt2 \approx_{dip} rt3 \rrbracket \implies rt1 \approx_{dip} rt3$ "  
 <proof>

lemma *rt\_fresh\_asI* [intro]:

assumes "rt1  $\sqsubseteq_{dip}$  rt2"  
 and "rt2  $\sqsubseteq_{dip}$  rt1"  
 shows "rt1  $\approx_{dip}$  rt2"  
 <proof>

lemma *rt\_fresh\_as\_fresherI* [intro]:

assumes "dip  $\in kD(rt1)$ "  
 and "dip  $\in kD(rt2)$ "  
 and "the (rt1 dip)  $\sqsubseteq$  the (rt2 dip)"  
 and "the (rt2 dip)  $\sqsubseteq$  the (rt1 dip)"  
 shows "rt1  $\approx_{dip}$  rt2"  
 <proof>

lemma *nsqn\_rt\_fresh\_asI*:

assumes "dip  $\in kD(rt)$ "  
 and "dip  $\in kD(rt')$ "  
 and "nsqn rt dip = nsqn rt' dip"  
 and " $\pi_5(\text{the } (rt\ dip)) = \pi_5(\text{the } (rt'\ dip))$ "  
 shows "rt  $\approx_{dip}$  rt'"  
 <proof>

lemma *rt\_fresh\_asE* [elim]:

assumes "rt1  $\approx_{dip}$  rt2"  
 and " $\llbracket rt1 \sqsubseteq_{dip} rt2; rt2 \sqsubseteq_{dip} rt1 \rrbracket \implies P\ rt1\ rt2\ dip$ "  
 shows "P rt1 rt2 dip"  
 <proof>

lemma *rt\_fresh\_asD1* [dest]:

assumes "rt1  $\approx_{dip}$  rt2"  
 shows "rt1  $\sqsubseteq_{dip}$  rt2"  
 <proof>

lemma *rt\_fresh\_asD2* [dest]:

assumes "rt1  $\approx_{dip}$  rt2"  
 shows "rt2  $\sqsubseteq_{dip}$  rt1"  
 <proof>

lemma *rt\_fresh\_as\_sym*:

assumes "rt1  $\approx_{dip}$  rt2"  
 shows "rt2  $\approx_{dip}$  rt1"  
 <proof>

lemma *not\_rt\_fresh\_asI1* [intro]:

assumes " $\neg (rt1 \sqsubseteq_{dip} rt2)$ "  
 shows " $\neg (rt1 \approx_{dip} rt2)$ "  
 <proof>

lemma *not\_rt\_fresh\_asI2* [intro]:

assumes " $\neg (rt2 \sqsubseteq_{dip} rt1)$ "  
 shows " $\neg (rt1 \approx_{dip} rt2)$ "  
 <proof>

lemma *not\_single\_rt\_fresher* [elim]:

assumes " $\neg (\text{the } (rt1\ ip) \sqsubseteq \text{the } (rt2\ ip))$ "  
 shows " $\neg (rt1 \sqsubseteq_{ip} rt2)$ "  
 <proof>

lemmas *not\_single\_rt\_fresh\_asI1* [intro] = *not\_rt\_fresh\_asI1* [OF *not\_single\_rt\_fresher*]

lemmas *not\_single\_rt\_fresh\_asI2* [intro] = *not\_rt\_fresh\_asI2* [OF *not\_single\_rt\_fresher*]

```

lemma not_rt_fresher_single [elim]:
  assumes "¬(rt1  $\sqsubseteq_{ip}$  rt2)"
  shows "¬(the (rt1 ip)  $\sqsubseteq$  the (rt2 ip))"
  <proof>

lemma rt_fresh_as_nsqnr:
  assumes "dip  $\in$  kD(rt1)"
  and "dip  $\in$  kD(rt2)"
  and "rt1  $\approx_{dip}$  rt2"
  shows "nsqnr (the (rt2 dip)) = nsqnr (the (rt1 dip))"
  <proof>

lemma rt_fresher_mapupd [intro!]:
  assumes "dip  $\in$  kD(rt)"
  and "the (rt dip)  $\sqsubseteq$  r"
  shows "rt  $\sqsubseteq_{dip}$  rt(dip  $\mapsto$  r)"
  <proof>

lemma rt_fresher_map_update_other [intro!]:
  assumes "dip  $\in$  kD(rt)"
  and "dip  $\neq$  ip"
  shows "rt  $\sqsubseteq_{dip}$  rt(ip  $\mapsto$  r)"
  <proof>

lemma rt_fresher_update_other [simp]:
  assumes inkD: "dip  $\in$  kD(rt)"
  and "dip  $\neq$  ip"
  shows "rt  $\sqsubseteq_{dip}$  update rt ip r"
  <proof>

theorem rt_fresher_update [simp]:
  assumes "dip  $\in$  kD(rt)"
  and "the (dhops rt dip)  $\geq$  1"
  and "update_arg_wf r"
  shows "rt  $\sqsubseteq_{dip}$  update rt ip r"
  <proof>

theorem rt_fresher_invalidate [simp]:
  assumes "dip  $\in$  kD(rt)"
  and indests: " $\forall rip \in \text{dom}(\text{dests}). rip \in vD(\text{rt}) \wedge \text{sqn rt rip} < \text{the}(\text{dests rip})$ "
  shows "rt  $\sqsubseteq_{dip}$  invalidate rt dests"
  <proof>

lemma nsqnr_invalidate [simp]:
  assumes "dip  $\in$  kD(rt)"
  and "dip  $\in$  dom(dests)"
  shows "nsqnr (the (invalidate rt dests dip)) = the (dests dip) - 1"
  <proof>

lemma rt_fresh_as_inc_invalidate [simp]:
  assumes "dip  $\in$  kD(rt)"
  and " $\forall rip \in \text{dom}(\text{dests}). rip \in vD(\text{rt}) \wedge \text{the}(\text{dests rip}) = \text{inc}(\text{sqn rt rip})$ "
  shows "rt  $\approx_{dip}$  invalidate rt dests"
  <proof>

lemmas rt_fresher_inc_invalidate [simp] = rt_fresh_as_inc_invalidate [THEN rt_fresh_asD1]

```

### 3.5.4 Strictly comparing routing tables

definition rt\_strictly\_fresher :: "ip  $\Rightarrow$  rt  $\Rightarrow$  rt  $\Rightarrow$  bool"

where

"rt\_strictly\_fresher  $\equiv$   $\lambda$ dip rt1 rt2. (rt1  $\sqsubseteq_{dip}$  rt2)  $\wedge$   $\neg$ (rt1  $\approx_{dip}$  rt2)"

abbreviation

```

    rt_strictly_fresher_syn :: "rt  $\Rightarrow$  ip  $\Rightarrow$  rt  $\Rightarrow$  bool" ("(_/  $\sqsubseteq$  _)" [51, 999, 51] 50)
where
  "rt1  $\sqsubseteq_i$  rt2  $\equiv$  rt_strictly_fresher i rt1 rt2"

lemma rt_strictly_fresher_def'':
  "rt1  $\sqsubseteq_i$  rt2 = ((rt1  $\sqsubseteq_i$  rt2)  $\wedge$   $\neg$ (rt2  $\sqsubseteq_i$  rt1))"
  <proof>

lemma rt_strictly_fresherI' [intro]:
  assumes "rt1  $\sqsubseteq_i$  rt2"
    and " $\neg$ (rt2  $\sqsubseteq_i$  rt1)"
  shows "rt1  $\sqsubseteq_i$  rt2"
  <proof>

lemma rt_strictly_fresherE' [elim]:
  assumes "rt1  $\sqsubseteq_i$  rt2"
    and "[| rt1  $\sqsubseteq_i$  rt2;  $\neg$ (rt2  $\sqsubseteq_i$  rt1) |]  $\Longrightarrow$  P rt1 rt2 i"
  shows "P rt1 rt2 i"
  <proof>

lemma rt_strictly_fresherI [intro]:
  assumes "rt1  $\sqsubseteq_i$  rt2"
    and " $\neg$ (rt1  $\approx_i$  rt2)"
  shows "rt1  $\sqsubseteq_i$  rt2"
  <proof>

lemmas rt_strictly_fresher_singleI [elim] = rt_strictly_fresherI [OF single_rt_fresher]

lemma rt_strictly_fresherE [elim]:
  assumes "rt1  $\sqsubseteq_i$  rt2"
    and "[| rt1  $\sqsubseteq_i$  rt2;  $\neg$ (rt1  $\approx_i$  rt2) |]  $\Longrightarrow$  P rt1 rt2 i"
  shows "P rt1 rt2 i"
  <proof>

lemma rt_strictly_fresher_def':
  "rt1  $\sqsubseteq_i$  rt2 =
  (nsqnr (the (rt1 i)) < nsqnr (the (rt2 i))
   $\vee$  (nsqnr (the (rt1 i)) = nsqnr (the (rt2 i))  $\wedge$   $\pi_5$ (the (rt1 i)) >  $\pi_5$ (the (rt2 i))))"
  <proof>

lemma rt_strictly_fresher_fresherD [dest]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
  shows "the (rt1 dip)  $\sqsubseteq$  the (rt2 dip)"
  <proof>

lemma rt_strictly_fresher_not_fresh_asD [dest]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
  shows " $\neg$  rt1  $\approx_{dip}$  rt2"
  <proof>

lemma rt_strictly_fresher_trans [elim, trans]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
    and "rt2  $\sqsubseteq_{dip}$  rt3"
  shows "rt1  $\sqsubseteq_{dip}$  rt3"
  <proof>

lemma rt_strictly_fresher_irefl [simp]: " $\neg$  (rt  $\sqsubseteq_{dip}$  rt)"
  <proof>

lemma rt_fresher_trans_rt_strictly_fresher [elim, trans]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
    and "rt2  $\sqsubseteq_{dip}$  rt3"
  shows "rt1  $\sqsubseteq_{dip}$  rt3"
  <proof>

```

```

lemma rt_fresher_trans_rt_strictly_fresher' [elim, trans]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
    and "rt2  $\sqsubseteq_{dip}$  rt3"
  shows "rt1  $\sqsubseteq_{dip}$  rt3"
  <proof>

lemma rt_fresher_imp_nsqn_le:
  assumes "rt1  $\sqsubseteq_{ip}$  rt2"
    and "ip  $\in kD$  rt1"
    and "ip  $\in kD$  rt2"
  shows "nsqn rt1 ip  $\leq$  nsqn rt2 ip"
  <proof>

lemma rt_strictly_fresher_ltI [intro]:
  assumes "dip  $\in kD$ (rt1)"
    and "dip  $\in kD$ (rt2)"
    and "nsqn rt1 dip < nsqn rt2 dip"
  shows "rt1  $\sqsubseteq_{dip}$  rt2"
  <proof>

lemma rt_strictly_fresher_eqI [intro]:
  assumes "i  $\in kD$ (rt1)"
    and "i  $\in kD$ (rt2)"
    and "nsqn rt1 i = nsqn rt2 i"
    and " $\pi_5$ (the (rt2 i)) <  $\pi_5$ (the (rt1 i))"
  shows "rt1  $\sqsubseteq_i$  rt2"
  <proof>

lemma invalidate_rtsf_left [simp]:
  " $\bigwedge$ dests dip rt rt'. dests dip = None  $\implies$  (invalidate rt dests  $\sqsubseteq_{dip}$  rt') = (rt  $\sqsubseteq_{dip}$  rt')"
  <proof>

lemma vD_invalidate_rt_strictly_fresher [simp]:
  assumes "dip  $\in vD$ (invalidate rt1 dests)"
  shows "(invalidate rt1 dests  $\sqsubseteq_{dip}$  rt2) = (rt1  $\sqsubseteq_{dip}$  rt2)"
  <proof>

lemma rt_strictly_fresher_update_other [elim!]:
  " $\bigwedge$ dip ip rt r rt'. [ $dip \neq ip$ ; rt  $\sqsubseteq_{dip}$  rt']  $\implies$  update rt ip r  $\sqsubseteq_{dip}$  rt'"
  <proof>

lemma lt_sqn_imp_update_strictly_fresher:
  assumes "dip  $\in vD$  (rt2 nhip)"
    and *: "osn < sqn (rt2 nhip) dip"
    and **: "rt  $\neq$  update rt dip (osn, kno, val, hops, nhip)"
  shows "update rt dip (osn, kno, val, hops, nhip)  $\sqsubseteq_{dip}$  rt2 nhip"
  <proof>

lemma dhops_le_hops_imp_update_strictly_fresher:
  assumes "dip  $\in vD$ (rt2 nhip)"
    and sqn: "sqn (rt2 nhip) dip = osn"
    and hop: "the (dhops (rt2 nhip) dip)  $\leq$  hops"
    and **: "rt  $\neq$  update rt dip (osn, kno, val, Suc hops, nhip)"
  shows "update rt dip (osn, kno, val, Suc hops, nhip)  $\sqsubseteq_{dip}$  rt2 nhip"
  <proof>

lemma nsqn_invalidate:
  assumes "dip  $\in kD$ (rt)"
    and " $\forall ip \in \text{dom}(\text{dests}). ip \in vD(\text{rt}) \wedge \text{the}(\text{dests } ip) = \text{inc}(\text{sqn } \text{rt } ip)"$ "
  shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
  <proof>

```

end

## 3.6 Invariant proofs on individual processes

```
theory C_Seq_Invariants
imports "../.../AWN/Invariants" C_Aodv C_Aodv_Data C_Aodv_Predicates C_Fresher
begin
```

The proposition numbers are taken from the December 2013 version of the Fehnker et al technical report.

Proposition 7.2

```
lemma sequence_number_increases:
  "paodv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), \_, (\xi', \_)). sn \xi \leq sn \xi'$ )"
  <proof>
```

```
lemma sequence_number_one_or_bigger:
  "paodv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_). 1 \leq sn \xi$ )"
  <proof>
```

We can get rid of the onl/onll if desired...

```
lemma sequence_number_increases':
  "paodv i  $\models_A$  ( $\lambda((\xi, \_), \_, (\xi', \_)). sn \xi \leq sn \xi'$ )"
  <proof>
```

```
lemma sequence_number_one_or_bigger':
  "paodv i  $\models$  ( $\lambda(\xi, \_). 1 \leq sn \xi$ )"
  <proof>
```

```
lemma sip_in_kD:
  "paodv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). l \in (\{PAodv-:7\} \cup \{PAodv-:5\} \cup \{PRrep-:0..PRrep-:1\}$ 
     $\cup \{PRreq-:0..PRreq-:3\}) \longrightarrow sip \xi \in kD (rt \xi)$ )"
  <proof>
```

```
lemma rrep_1_update_changes:
  "paodv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). (l = PRrep-:1 \longrightarrow$ 
     $rt \xi \neq update (rt \xi) (dip \xi) (dsn \xi, kno, val, hops \xi + 1, sip \xi))$ )"
  <proof>
```

Proposition 7.38

```
lemma includes_nhip:
  "paodv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). \forall dip \in kD(rt \xi). the (nhop (rt \xi) dip) \in kD(rt \xi)$ )"
  <proof>
```

Proposition 7.4

```
lemma known_destinations_increase:
  "paodv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), \_, (\xi', \_)). kD (rt \xi) \subseteq kD (rt \xi')$ )"
  <proof>
```

Proposition 7.5

```
lemma rreqs_increase:
  "paodv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), \_, (\xi', \_)). rreqs \xi \subseteq rreqs \xi'$ )"
  <proof>
```

```
lemma dests_bigger_than_sqn:
  "paodv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). l \in \{PAodv-:15..PAodv-:17\}$ 
     $\cup \{PPkt-:7..PPkt-:9\}$ 
     $\cup \{PRreq-:9..PRreq-:11\}$ 
     $\cup \{PRreq-:17..PRreq-:19\}$ 
     $\cup \{PRrep-:8..PRrep-:10\}$ 
     $\cup \{PRerr-:1..PRerr-:4\} \cup \{PRerr-:6\}$ 
     $\longrightarrow (\forall ip \in dom(dests \xi). ip \in kD(rt \xi) \wedge sqn (rt \xi) ip \leq the (dests \xi ip))$ )"
  <proof>
```

Proposition 7.6

```
lemma sqns_increase:
```

"paadv i  $\models_A$  onll  $\Gamma_{AODV} (\lambda((\xi, \_), \_, (\xi', \_)). \forall ip. sqn (rt \xi) ip \leq sqn (rt \xi') ip)"$   
 <proof>

Proposition 7.7

**lemma ip\_constant:**

"paadv i  $\models$  onl  $\Gamma_{AODV} (\lambda(\xi, \_). ip \xi = i)"$   
 <proof>

Proposition 7.8

**lemma sender\_ip\_valid':**

"paadv i  $\models_A$  onll  $\Gamma_{AODV} (\lambda((\xi, \_), a, \_). anycast (\lambda m. not\_Pkt m \longrightarrow msg\_sender m = ip \xi) a)"$   
 <proof>

**lemma sender\_ip\_valid:**

"paadv i  $\models_A$  onll  $\Gamma_{AODV} (\lambda((\xi, \_), a, \_). anycast (\lambda m. not\_Pkt m \longrightarrow msg\_sender m = i) a)"$   
 <proof>

**lemma received\_msg\_inv:**

"paadv i  $\models$  (recvmmsg P  $\rightarrow$ ) onl  $\Gamma_{AODV} (\lambda(\xi, l). l \in \{PAodv-:1\} \longrightarrow P (msg \xi))"$   
 <proof>

Proposition 7.9

**lemma sip\_not\_ip':**

"paadv i  $\models$  (recvmmsg ( $\lambda m. not\_Pkt m \longrightarrow msg\_sender m \neq i$ )  $\rightarrow$ ) onl  $\Gamma_{AODV} (\lambda(\xi, \_). sip \xi \neq ip \xi)"$   
 <proof>

**lemma sip\_not\_ip:**

"paadv i  $\models$  (recvmmsg ( $\lambda m. not\_Pkt m \longrightarrow msg\_sender m \neq i$ )  $\rightarrow$ ) onl  $\Gamma_{AODV} (\lambda(\xi, \_). sip \xi \neq i)"$   
 <proof>

Neither *sip\_not\_ip'* nor *sip\_not\_ip* is needed to show loop freedom.

Proposition 7.10

**lemma hop\_count\_positive:**

"paadv i  $\models$  onl  $\Gamma_{AODV} (\lambda(\xi, \_). \forall ip \in kD (rt \xi). the (dhops (rt \xi) ip) \geq 1)"$   
 <proof>

**lemma rreq\_dip\_in\_vD\_dip\_eq\_ip:**

"paadv i  $\models$  onl  $\Gamma_{AODV} (\lambda(\xi, l). (l \in \{PRreq-:14\} \longrightarrow dip \xi \in vD(rt \xi))$   
 $\wedge (l \in \{PRreq-:5, PRreq-:6\} \longrightarrow dip \xi = ip \xi)$   
 $\wedge (l \in \{PRreq-:13..PRreq-:14\} \longrightarrow dip \xi \neq ip \xi))"$   
 <proof>

Proposition 7.11

**lemma anycast\_msg\_zhops:**

" $\wedge rreqid dip dsn dsk oip osn sip.$   
 paadv i  $\models_A$  onll  $\Gamma_{AODV} (\lambda(\_, a, \_). anycast msg\_zhops a)"$   
 <proof>

**lemma hop\_count\_zero\_oip\_dip\_sip:**

"paadv i  $\models$  (recvmmsg msg\_zhops  $\rightarrow$ ) onl  $\Gamma_{AODV} (\lambda(\xi, l).$   
 $(l \in \{PAodv-:4..PAodv-:5\} \cup \{PRreq-:n/n. True\} \longrightarrow$   
 $(hops \xi = 0 \longrightarrow oip \xi = sip \xi))$   
 $\wedge$   
 $((l \in \{PAodv-:6..PAodv-:7\} \cup \{PRrep-:n/n. True\} \longrightarrow$   
 $(hops \xi = 0 \longrightarrow dip \xi = sip \xi))))"$   
 <proof>

**lemma osn\_rreq:**

"paadv i  $\models$  (recvmmsg rreq\_rrep\_sn  $\rightarrow$ ) onl  $\Gamma_{AODV} (\lambda(\xi, l).$   
 $l \in \{PAodv-:4, PAodv-:5\} \cup \{PRreq-:n/n. True\} \longrightarrow 1 \leq osn \xi)"$   
 <proof>

**lemma osn\_rreq':**

"paadv i  $\models$  (recvmsg ( $\lambda m. rreq\_rrep\_sn\ m \wedge msg\_zhops\ m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l)$ .  
 $l \in \{PAadv-:4, PAadv-:5\} \cup \{PRreq-:n/n. True\} \rightarrow 1 \leq osn\ \xi$ )"  
 <proof>

**lemma dsn\_rrep:**

"paadv i  $\models$  (recvmsg rreq\_rrep\_sn  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l)$ .  
 $l \in \{PAadv-:6, PAadv-:7\} \cup \{PRrep-:n/n. True\} \rightarrow 1 \leq dsn\ \xi$ )"  
 <proof>

**lemma dsn\_rrep':**

"paadv i  $\models$  (recvmsg ( $\lambda m. rreq\_rrep\_sn\ m \wedge msg\_zhops\ m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l)$ .  
 $l \in \{PAadv-:6, PAadv-:7\} \cup \{PRrep-:n/n. True\} \rightarrow 1 \leq dsn\ \xi$ )"  
 <proof>

**lemma hop\_count\_zero\_oip\_dip\_sip':**

"paadv i  $\models$  (recvmsg ( $\lambda m. rreq\_rrep\_sn\ m \wedge msg\_zhops\ m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l)$ .  
 $(l \in \{PAadv-:4..PAadv-:5\} \cup \{PRreq-:n/n. True\} \rightarrow$   
 $(hops\ \xi = 0 \rightarrow oip\ \xi = sip\ \xi))$   
 $\wedge$   
 $((l \in \{PAadv-:6..PAadv-:7\} \cup \{PRrep-:n/n. True\} \rightarrow$   
 $(hops\ \xi = 0 \rightarrow dip\ \xi = sip\ \xi))))$ "  
 <proof>

Proposition 7.12

**lemma zero\_seq\_unk\_hops\_one':**

"paadv i  $\models$  (recvmsg ( $\lambda m. rreq\_rrep\_sn\ m \wedge msg\_zhops\ m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_)$ .  
 $\forall dip \in kD(rt\ \xi). (sqn\ (rt\ \xi)\ dip = 0 \rightarrow sqnf\ (rt\ \xi)\ dip = unk)$   
 $\wedge (sqnf\ (rt\ \xi)\ dip = unk \rightarrow the\ (dhops\ (rt\ \xi)\ dip) = 1)$   
 $\wedge (the\ (dhops\ (rt\ \xi)\ dip) = 1 \rightarrow the\ (nhop\ (rt\ \xi)\ dip) = dip))$ "  
 <proof>

**lemma zero\_seq\_unk\_hops\_one:**

"paadv i  $\models$  (recvmsg ( $\lambda m. rreq\_rrep\_sn\ m \wedge msg\_zhops\ m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_)$ .  
 $\forall dip \in kD(rt\ \xi). (sqn\ (rt\ \xi)\ dip = 0 \rightarrow (sqnf\ (rt\ \xi)\ dip = unk$   
 $\wedge the\ (dhops\ (rt\ \xi)\ dip) = 1$   
 $\wedge the\ (nhop\ (rt\ \xi)\ dip) = dip))$ "  
 <proof>

**lemma kD\_unk\_or\_atleast\_one:**

"paadv i  $\models$  (recvmsg rreq\_rrep\_sn  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l)$ .  
 $\forall dip \in kD(rt\ \xi). \pi_3(the\ (rt\ \xi\ dip)) = unk \vee 1 \leq \pi_2(the\ (rt\ \xi\ dip))$ )"  
 <proof>

Proposition 7.13

**lemma rreq\_rrep\_sn\_any\_step\_invariant:**

"paadv i  $\models_A$  (recvmsg rreq\_rrep\_sn  $\rightarrow$ ) onll  $\Gamma_{AODV}$  ( $\lambda(\_, a, \_)$ . anycast rreq\_rrep\_sn a)"  
 <proof>

Proposition 7.14

**lemma rreq\_rrep\_fresh\_any\_step\_invariant:**

"paadv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), a, \_)$ . anycast (rreq\_rrep\_fresh (rt  $\xi$ )) a)"  
 <proof>

Proposition 7.15

**lemma rerr\_invalid\_any\_step\_invariant:**

"paadv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), a, \_)$ . anycast (rerr\_invalid (rt  $\xi$ )) a)"  
 <proof>

Proposition 7.16

Some well-definedness obligations are irrelevant for the Isabelle development:

1. In each routing table there is at most one entry for each destination: guaranteed by type.



2. In each store of queued data packets there is at most one data queue for each destination: guaranteed by structure.
3. Whenever a set of pairs  $(rip, rsn)$  is assigned to the variable  $dests$  of type  $ip \rightarrow sqn$ , or to the first argument of the function  $rerr$ , this set is a partial function, i.e., there is at most one entry  $(rip, rsn)$  for each destination  $rip$ : guaranteed by type.

**lemma** *dests\_vD\_inc\_sqn*:

```
"paadv i  $\models$ 
  on1  $\Gamma_{AODV} (\lambda(\xi, l). (l \in \{PAadv-:15, PPkt-:7, PRreq-:9, PRreq-:17, PRrep-:8\}$ 
     $\rightarrow (\forall ip \in \text{dom}(dests \ \xi). ip \in vD(rt \ \xi) \wedge \text{the}(dests \ \xi \ ip) = \text{inc}(sqn(rt \ \xi) \ ip)))$ 
   $\wedge (l = PRerr-:1$ 
     $\rightarrow (\forall ip \in \text{dom}(dests \ \xi). ip \in vD(rt \ \xi) \wedge \text{the}(dests \ \xi \ ip) > sqn(rt \ \xi) \ ip)))"$ 
<proof>
```

Proposition 7.27

**lemma** *route\_tables\_fresher*:

```
"paadv i  $\models_A (\text{recvm}sg \ rreq\_rrep\_sn \rightarrow) \text{onll} \ \Gamma_{AODV} (\lambda((\xi, \_), \_, (\xi', \_)).$ 
 $\forall dip \in kD(rt \ \xi). rt \ \xi \sqsubseteq_{dip} rt \ \xi')$ "
<proof>
```

end

### 3.7 The quality increases predicate

**theory** *C\_Quality\_Increases*

**imports** *C\_Aadv\_Predicates C\_Fresher*

**begin**

**definition** *quality\_increases* :: "state  $\Rightarrow$  state  $\Rightarrow$  bool"

where "quality\_increases  $\xi \ \xi' \equiv (\forall dip \in kD(rt \ \xi). dip \in kD(rt \ \xi') \wedge rt \ \xi \sqsubseteq_{dip} rt \ \xi')$   
 $\wedge (\forall dip. sqn(rt \ \xi) \ dip \leq sqn(rt \ \xi') \ dip)"$

**lemma** *quality\_increasesI [intro!]*:

```
assumes " $\wedge dip. dip \in kD(rt \ \xi) \implies dip \in kD(rt \ \xi')$ "
  and " $\wedge dip. \llbracket dip \in kD(rt \ \xi); dip \in kD(rt \ \xi') \rrbracket \implies rt \ \xi \sqsubseteq_{dip} rt \ \xi'$ "
  and " $\wedge dip. sqn(rt \ \xi) \ dip \leq sqn(rt \ \xi') \ dip"$ 
shows "quality_increases  $\xi \ \xi'$ "
<proof>
```

**lemma** *quality\_increasesE [elim]*:

```
fixes dip
assumes "quality_increases  $\xi \ \xi'$ "
  and "dip  $\in kD(rt \ \xi)$ "
  and " $\llbracket dip \in kD(rt \ \xi'); rt \ \xi \sqsubseteq_{dip} rt \ \xi'; sqn(rt \ \xi) \ dip \leq sqn(rt \ \xi') \ dip \rrbracket \implies R \ dip \ \xi \ \xi'$ "
shows "R dip  $\xi \ \xi'$ "
<proof>
```

**lemma** *quality\_increases\_rt\_fresherD [dest]*:

```
fixes ip
assumes "quality_increases  $\xi \ \xi'$ "
  and "ip  $\in kD(rt \ \xi)$ "
shows "rt  $\xi \sqsubseteq_{ip} rt \ \xi'$ "
<proof>
```

**lemma** *quality\_increases\_sqnE [elim]*:

```
fixes dip
assumes "quality_increases  $\xi \ \xi'$ "
  and "sqn(rt  $\xi$ ) dip  $\leq sqn(rt \ \xi') \ dip \implies R \ dip \ \xi \ \xi'$ "
shows "R dip  $\xi \ \xi'$ "
<proof>
```

**lemma** *quality\_increases\_refl [intro, simp]*: "quality\_increases  $\xi \ \xi$ "

*<proof>*

```
lemma strictly_fresher_quality_increases_right [elim]:  
  fixes  $\sigma$   $\sigma'$  dip  
  assumes "rt ( $\sigma$  i)  $\sqsubset_{\text{dip}}$  rt ( $\sigma$  nhip)"  
    and qinc: "quality_increases ( $\sigma$  nhip) ( $\sigma'$  nhip)"  
    and "dip  $\in$  kD(rt ( $\sigma$  nhip))"  
  shows "rt ( $\sigma$  i)  $\sqsubset_{\text{dip}}$  rt ( $\sigma'$  nhip)"  
<proof>
```

```
lemma kD_quality_increases [elim]:  
  assumes "i  $\in$  kD(rt  $\xi$ )"  
    and "quality_increases  $\xi$   $\xi'$ "  
  shows "i  $\in$  kD(rt  $\xi')$ "  
<proof>
```

```
lemma kD_nsqn_quality_increases [elim]:  
  assumes "i  $\in$  kD(rt  $\xi$ )"  
    and "quality_increases  $\xi$   $\xi'$ "  
  shows "i  $\in$  kD(rt  $\xi')$   $\wedge$  nsqn (rt  $\xi$ ) i  $\leq$  nsqn (rt  $\xi')$  i"  
<proof>
```

```
lemma nsqn_quality_increases [elim]:  
  assumes "i  $\in$  kD(rt  $\xi$ )"  
    and "quality_increases  $\xi$   $\xi'$ "  
  shows "nsqn (rt  $\xi$ ) i  $\leq$  nsqn (rt  $\xi')$  i"  
<proof>
```

```
lemma kD_nsqn_quality_increases_trans [elim]:  
  assumes "i  $\in$  kD(rt  $\xi$ )"  
    and "s  $\leq$  nsqn (rt  $\xi$ ) i"  
    and "quality_increases  $\xi$   $\xi'$ "  
  shows "i  $\in$  kD(rt  $\xi')$   $\wedge$  s  $\leq$  nsqn (rt  $\xi')$  i"  
<proof>
```

```
lemma nsqn_quality_increases_nsqn_lt_lt [elim]:  
  assumes "i  $\in$  kD(rt  $\xi$ )"  
    and "quality_increases  $\xi$   $\xi'$ "  
    and "s < nsqn (rt  $\xi$ ) i"  
  shows "s < nsqn (rt  $\xi')$  i"  
<proof>
```

```
lemma nsqn_quality_increases_dhops [elim]:  
  assumes "i  $\in$  kD(rt  $\xi$ )"  
    and "quality_increases  $\xi$   $\xi'$ "  
    and "nsqn (rt  $\xi$ ) i = nsqn (rt  $\xi')$  i"  
  shows "the (dhops (rt  $\xi$ ) i)  $\geq$  the (dhops (rt  $\xi')$  i)"  
<proof>
```

```
lemma nsqn_quality_increases_nsqn_eq_le [elim]:  
  assumes "i  $\in$  kD(rt  $\xi$ )"  
    and "quality_increases  $\xi$   $\xi'$ "  
    and "s = nsqn (rt  $\xi$ ) i"  
  shows "s < nsqn (rt  $\xi')$  i  $\vee$  (s = nsqn (rt  $\xi')$  i  $\wedge$  the (dhops (rt  $\xi$ ) i)  $\geq$  the (dhops (rt  $\xi')$  i))"  
<proof>
```

```
lemma quality_increases_rreq_rrep_props [elim]:  
  fixes sn ip hops sip  
  assumes qinc: "quality_increases ( $\sigma$  sip) ( $\sigma'$  sip)"  
    and "1  $\leq$  sn"  
    and *: "ip  $\in$  kD(rt ( $\sigma$  sip))  $\wedge$  sn  $\leq$  nsqn (rt ( $\sigma$  sip)) ip  
       $\wedge$  (nsqn (rt ( $\sigma$  sip)) ip = sn  
         $\rightarrow$  (the (dhops (rt ( $\sigma$  sip)) ip)  $\leq$  hops  
           $\vee$  the (flag (rt ( $\sigma$  sip)) ip) = inv))"
```

```

shows "ip∈kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
      ∧ (nsqn (rt (σ' sip)) ip = sn
        → (the (dhops (rt (σ' sip)) ip) ≤ hops
            ∨ the (flag (rt (σ' sip)) ip) = inv))"
  (is "_ ∧ ?nsqnafter")
⟨proof⟩

```

lemma quality\_increases\_rreq\_rrep\_props':

```

fixes sn ip hops sip
assumes "∀j. quality_increases (σ j) (σ' j)"
and "1 ≤ sn"
and *: "ip∈kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
      ∧ (nsqn (rt (σ sip)) ip = sn
        → (the (dhops (rt (σ sip)) ip) ≤ hops
            ∨ the (flag (rt (σ sip)) ip) = inv))"
shows "ip∈kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
      ∧ (nsqn (rt (σ' sip)) ip = sn
        → (the (dhops (rt (σ' sip)) ip) ≤ hops
            ∨ the (flag (rt (σ' sip)) ip) = inv))"
⟨proof⟩

```

lemma rteq\_quality\_increases:

```

assumes "∀j. j ≠ i → quality_increases (σ j) (σ' j)"
and "rt (σ' i) = rt (σ i)"
shows "∀j. quality_increases (σ j) (σ' j)"
⟨proof⟩

```

definition msg\_fresh :: "(ip ⇒ state) ⇒ msg ⇒ bool"

```

where "msg_fresh σ m ≡
  case m of Rreq hopsc _ _ _ oipc osnc sipc ⇒ osnc ≥ 1 ∧ (sipc ≠ oipc →
    oipc∈kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) oipc ≥ osnc
    ∧ (nsqn (rt (σ sipc)) oipc = osnc
      → (hopsc ≥ the (dhops (rt (σ sipc)) oipc)
        ∨ the (flag (rt (σ sipc)) oipc) = inv)))
  | Rrep hopsc dipc dsnc _ sipc ⇒ dsnc ≥ 1 ∧ (sipc ≠ dipc →
    dipc∈kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) dipc ≥ dsnc
    ∧ (nsqn (rt (σ sipc)) dipc = dsnc
      → (hopsc ≥ the (dhops (rt (σ sipc)) dipc)
        ∨ the (flag (rt (σ sipc)) dipc) = inv)))
  | Rerr destsc sipc ⇒ (∀ripc∈dom(destsc). (ripc∈kD(rt (σ sipc))
    ∧ the (destsc ripc) - 1 ≤ nsqn (rt (σ sipc)) ripc))
  | _ ⇒ True"

```

lemma msg\_fresh [simp]:

```

"∧hops rreqid dip dsn dsk oip osn sip.
  msg_fresh σ (Rreq hops rreqid dip dsn dsk oip osn sip) =
  (osn ≥ 1 ∧ (sip ≠ oip → oip∈kD(rt (σ sip))
    ∧ nsqn (rt (σ sip)) oip ≥ osn
    ∧ (nsqn (rt (σ sip)) oip = osn
      → (hops ≥ the (dhops (rt (σ sip)) oip)
        ∨ the (flag (rt (σ sip)) oip) = inv))))"
"∧hops dip dsn oip sip. msg_fresh σ (Rrep hops dip dsn oip sip) =
  (dsn ≥ 1 ∧ (sip ≠ dip → dip∈kD(rt (σ sip))
    ∧ nsqn (rt (σ sip)) dip ≥ dsn
    ∧ (nsqn (rt (σ sip)) dip = dsn
      → (hops ≥ the (dhops (rt (σ sip)) dip)
        ∨ the (flag (rt (σ sip)) dip) = inv))))"
"∧dests sip. msg_fresh σ (Rerr dests sip) =
  (∀ripc∈dom(dests). (ripc∈kD(rt (σ sip))
    ∧ the (dests ripc) - 1 ≤ nsqn (rt (σ sip)) ripc))"
"∧d dip. msg_fresh σ (Newpkt d dip) = True"
"∧d dip sip. msg_fresh σ (Pkt d dip sip) = True"
⟨proof⟩

```

**lemma** *msg\_fresh\_inc\_sn* [*simp*, *elim*]:

"*msg\_fresh*  $\sigma$   $m \implies$  *rreq\_rrep\_sn*  $m$ "  
<*proof*>

**lemma** *recv\_msg\_fresh\_inc\_sn* [*simp*, *elim*]:

"*orecvmsg* (*msg\_fresh*)  $\sigma$   $m \implies$  *recvmsg* *rreq\_rrep\_sn*  $m$ "  
<*proof*>

**lemma** *rreq\_nsqn\_is\_fresh* [*simp*]:

*fixes*  $\sigma$  *msg* *hops* *rreqid* *dip* *dsn* *dsk* *oip* *osn* *sip*  
*assumes* "*rreq\_rrep\_fresh* (*rt* ( $\sigma$  *sip*)) (*Rreq* *hops* *rreqid* *dip* *dsn* *dsk* *oip* *osn* *sip*)"  
  *and* "*rreq\_rrep\_sn* (*Rreq* *hops* *rreqid* *dip* *dsn* *dsk* *oip* *osn* *sip*)"  
*shows* "*msg\_fresh*  $\sigma$  (*Rreq* *hops* *rreqid* *dip* *dsn* *dsk* *oip* *osn* *sip*)"  
  (*is* "*msg\_fresh*  $\sigma$  ?*msg*")  
<*proof*>

**lemma** *rrep\_nsqn\_is\_fresh* [*simp*]:

*fixes*  $\sigma$  *msg* *hops* *dip* *dsn* *oip* *sip*  
*assumes* "*rreq\_rrep\_fresh* (*rt* ( $\sigma$  *sip*)) (*Rrep* *hops* *dip* *dsn* *oip* *sip*)"  
  *and* "*rreq\_rrep\_sn* (*Rrep* *hops* *dip* *dsn* *oip* *sip*)"  
*shows* "*msg\_fresh*  $\sigma$  (*Rrep* *hops* *dip* *dsn* *oip* *sip*)"  
  (*is* "*msg\_fresh*  $\sigma$  ?*msg*")  
<*proof*>

**lemma** *rerr\_nsqn\_is\_fresh* [*simp*]:

*fixes*  $\sigma$  *msg* *dests* *sip*  
*assumes* "*rerr\_invalid* (*rt* ( $\sigma$  *sip*)) (*Rerr* *dests* *sip*)"  
*shows* "*msg\_fresh*  $\sigma$  (*Rerr* *dests* *sip*)"  
  (*is* "*msg\_fresh*  $\sigma$  ?*msg*")  
<*proof*>

**lemma** *quality\_increases\_msg\_fresh* [*elim*]:

*assumes* *qinc*: " $\forall j.$  *quality\_increases* ( $\sigma$   $j$ ) ( $\sigma'$   $j$ )"  
  *and* "*msg\_fresh*  $\sigma$   $m$ "  
*shows* "*msg\_fresh*  $\sigma'$   $m$ "  
<*proof*>

**end**

### 3.8 The 'open' AODV model

**theory** *C\_OAodv*

**imports** *C\_Aodv* "../.../AWN/OAWN\_SOS\_Labels" "../.../AWN/OAWN\_Convert"

**begin**

Definitions for stating and proving global network properties over individual processes.

**definition**  $\sigma_{AODV}'$  :: " $((ip \Rightarrow state) \times ((state, msg, pseqp, pseqp label) seqp)) set$ "  
**where** " $\sigma_{AODV}' \equiv \{(\lambda i. aodv\_init\ i, \Gamma_{AODV}\ PAodv)\}$ "

**abbreviation** *opaodv*

:: " $ip \Rightarrow ((ip \Rightarrow state) \times (state, msg, pseqp, pseqp label) seqp, msg\ seq\_action)$  automaton"

**where**

"*opaodv*  $i \equiv \{\mid init = \sigma_{AODV}', trans = oseqp\_sos\ \Gamma_{AODV}\ i\ \}$ "

**lemma** *initiali\_aodv* [*intro!*, *simp*]: "*initiali*  $i$  (*init* (*opaodv*  $i$ )) (*init* (*paodv*  $i$ ))"

<*proof*>

**lemma** *oaodv\_control\_within* [*simp*]: "*control\_within*  $\Gamma_{AODV}$  (*init* (*opaodv*  $i$ ))"

<*proof*>

**lemma**  $\sigma_{AODV}'\_labels$  [*simp*]: " $(\sigma, p) \in \sigma_{AODV}' \implies labels\ \Gamma_{AODV}\ p = \{PAodv-:0\}$ "

<*proof*>

**lemma** *oaodv\_init\_kD\_empty* [*simp*]:

```
"( $\sigma$ , p)  $\in$   $\sigma_{AODV}' \implies kD$  (rt ( $\sigma$  i)) = {}"
⟨proof⟩
```

```
lemma oaodv_init_vD_empty [simp]:
```

```
"( $\sigma$ , p)  $\in$   $\sigma_{AODV}' \implies vD$  (rt ( $\sigma$  i)) = {}"
⟨proof⟩
```

```
lemma oaodv_trans: "trans (opaodv i) = oseqp_sos  $\Gamma_{AODV}$  i"
```

```
⟨proof⟩
```

```
declare
```

```
oseq_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros]
oseq_step_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros]
```

```
end
```

### 3.9 Global invariant proofs over sequential processes

```
theory C_Global_Invariants
```

```
imports C_Seq_Invariants
```

```
  C_Aodv_Predicates
```

```
  C_Fresher
```

```
  C_Quality_Increases
```

```
  "../.../AWN/OAWN_Convert"
```

```
  C_OAodv
```

```
begin
```

```
lemma other_quality_increases [elim]:
```

```
  assumes "other quality_increases I  $\sigma$   $\sigma'$ "
  shows " $\forall j$ . quality_increases ( $\sigma$  j) ( $\sigma'$  j)"
⟨proof⟩
```

```
lemma weaken_otherwith [elim]:
```

```
  fixes m
  assumes *: "otherwith P I (orecvmsg Q)  $\sigma$   $\sigma'$  a"
  and weakenP: " $\bigwedge \sigma m$ . P  $\sigma$  m  $\implies$  P'  $\sigma$  m"
  and weakenQ: " $\bigwedge \sigma m$ . Q  $\sigma$  m  $\implies$  Q'  $\sigma$  m"
  shows "otherwith P' I (orecvmsg Q')  $\sigma$   $\sigma'$  a"
⟨proof⟩
```

```
lemma oreceived_msg_inv:
```

```
  assumes other: " $\bigwedge \sigma \sigma' m$ . [ $P$   $\sigma$  m; other Q {i}  $\sigma$   $\sigma'$ ]  $\implies$  P  $\sigma'$  m"
  and local: " $\bigwedge \sigma m$ . P  $\sigma$  m  $\implies$  P ( $\sigma$ (i :=  $\sigma$  i(msg := m))) m"
  shows "opaodv i  $\models$  (otherwith Q {i} (orecvmsg P), other Q {i}  $\rightarrow$ )
  onl  $\Gamma_{AODV}$  ( $\lambda(\sigma, l)$ .  $l \in \{PAodv-:1\} \rightarrow P \sigma$  (msg ( $\sigma$  i)))"
⟨proof⟩
```

(Equivalent to) Proposition 7.27

```
lemma local_quality_increases:
```

```
"paodv i  $\models_A$  (recvmsg rreq_rrep_sn  $\rightarrow$ ) onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), \_, (\xi', \_))$ . quality_increases  $\xi$   $\xi'$ )"
⟨proof⟩
```

```
lemmas olocal_quality_increases =
```

```
  open_seq_step_invariant [OF local_quality_increases initiali_aodv oaodv_trans aodv_trans,
  simplified seql_onll_swap]
```

```
lemma oquality_increases:
```

```
"opaodv i  $\models_A$  (otherwith quality_increases {i} (orecvmsg ( $\lambda$ _. rreq_rrep_sn)),
  other quality_increases {i}  $\rightarrow$ )
  onll  $\Gamma_{AODV}$  ( $\lambda((\sigma, \_), \_, (\sigma', \_))$ .  $\forall j$ . quality_increases ( $\sigma$  j) ( $\sigma'$  j))"
(is " $\_ \models_A$  (?S,  $\_ \rightarrow$ )  $\_$ ")
⟨proof⟩
```

```
lemma rreq_rrep_nsqn_fresh_any_step_invariant:
```

"opaadv i  $\models_A$  (act (recvmsg rreq\_rrep\_sn), other A {i}  $\rightarrow$ )  
 onll  $\Gamma_{AODV}$  ( $\lambda((\sigma, \_), a, \_)$ . anycast (msg\_fresh  $\sigma$ ) a)"  
 <proof>

**lemma oreceived\_rreq\_rrep\_nsqn\_fresh\_inv:**

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onll  $\Gamma_{AODV}$  ( $\lambda(\sigma, l)$ .  $l \in \{PAodv-:1\} \rightarrow$  msg\_fresh  $\sigma$  (msg ( $\sigma$  i)))"  
 <proof>

**lemma oquality\_increases\_nsqn\_fresh:**

"opaadv i  $\models_A$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onll  $\Gamma_{AODV}$  ( $\lambda((\sigma, \_), \_, (\sigma', \_))$ .  $\forall j$ . quality\_increases ( $\sigma$  j) ( $\sigma'$  j))"  
 <proof>

**lemma oosn\_rreq:**

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onll  $\Gamma_{AODV}$  (seq1 i ( $\lambda(\xi, l)$ .  $l \in \{PAodv-:4, PAodv-:5\} \cup \{PRreq-:n \mid n. \text{True}\} \rightarrow 1 \leq$  osn  $\xi$ ))"  
 <proof>

**lemma rreq\_sip:**

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onll  $\Gamma_{AODV}$  ( $\lambda(\sigma, l)$ .  
 ( $l \in \{PAodv-:4, PAodv-:5, PRreq-:0, PRreq-:2\} \wedge$  sip ( $\sigma$  i)  $\neq$  oip ( $\sigma$  i))  
 $\rightarrow$  oip ( $\sigma$  i)  $\in$  kD(rt ( $\sigma$  (sip ( $\sigma$  i))))  
 $\wedge$  nsqn (rt ( $\sigma$  (sip ( $\sigma$  i)))) (oip ( $\sigma$  i))  $\geq$  osn ( $\sigma$  i)  
 $\wedge$  (nsqn (rt ( $\sigma$  (sip ( $\sigma$  i)))) (oip ( $\sigma$  i)) = osn ( $\sigma$  i))  
 $\rightarrow$  (hops ( $\sigma$  i)  $\geq$  the (dhops (rt ( $\sigma$  (sip ( $\sigma$  i)))) (oip ( $\sigma$  i))))  
 $\vee$  the (flag (rt ( $\sigma$  (sip ( $\sigma$  i)))) (oip ( $\sigma$  i))) = inv)))"  
 (is "\_  $\models$  (?S, ?U  $\rightarrow$ ) \_")  
 <proof>

**lemma odsn\_rrep:**

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onll  $\Gamma_{AODV}$  (seq1 i ( $\lambda(\xi, l)$ .  $l \in \{PAodv-:6, PAodv-:7\} \cup \{PRrep-:n \mid n. \text{True}\} \rightarrow 1 \leq$  dsn  $\xi$ ))"  
 <proof>

**lemma rrep\_sip:**

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onll  $\Gamma_{AODV}$  ( $\lambda(\sigma, l)$ .  
 ( $l \in \{PAodv-:6, PAodv-:7, PRrep-:0, PRrep-:1\} \wedge$  sip ( $\sigma$  i)  $\neq$  dip ( $\sigma$  i))  
 $\rightarrow$  dip ( $\sigma$  i)  $\in$  kD(rt ( $\sigma$  (sip ( $\sigma$  i))))  
 $\wedge$  nsqn (rt ( $\sigma$  (sip ( $\sigma$  i)))) (dip ( $\sigma$  i))  $\geq$  dsn ( $\sigma$  i)  
 $\wedge$  (nsqn (rt ( $\sigma$  (sip ( $\sigma$  i)))) (dip ( $\sigma$  i)) = dsn ( $\sigma$  i))  
 $\rightarrow$  (hops ( $\sigma$  i)  $\geq$  the (dhops (rt ( $\sigma$  (sip ( $\sigma$  i)))) (dip ( $\sigma$  i))))  
 $\vee$  the (flag (rt ( $\sigma$  (sip ( $\sigma$  i)))) (dip ( $\sigma$  i))) = inv)))"  
 (is "\_  $\models$  (?S, ?U  $\rightarrow$ ) \_")  
 <proof>

**lemma rerr\_sip:**

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onll  $\Gamma_{AODV}$  ( $\lambda(\sigma, l)$ .  
 $l \in \{PAodv-:8, PAodv-:9, PRerr-:0, PRerr-:1\}$   
 $\rightarrow$  ( $\forall \text{ripc} \in \text{dom}(\text{dests } (\sigma \text{ i})). \text{ripc} \in \text{kD}(\text{rt } (\sigma \text{ (sip } (\sigma \text{ i})))) \wedge$   
 the (dests ( $\sigma$  i) ripc) - 1  $\leq$  nsqn (rt ( $\sigma$  (sip ( $\sigma$  i)))) ripc))"  
 (is "\_  $\models$  (?S, ?U  $\rightarrow$ ) \_")  
 <proof>

lemma prerr\_guard: "paadv i  $\models$   
 onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l)$ . ( $l = PRerr-:1$   
 $\rightarrow (\forall ip \in \text{dom}(\text{dests } \xi). ip \in vD(\text{rt } \xi)$   
 $\wedge \text{the } (\text{nhop } (\text{rt } \xi) ip) = sip \xi$   
 $\wedge \text{sqn } (\text{rt } \xi) ip < \text{the } (\text{dests } \xi ip))$ )")  
 <proof>

lemmas odests\_vD\_inc\_sqn =  
 open\_seq\_invariant [OF dests\_vD\_inc\_sqn initiali\_aadv oaadv\_trans aadv\_trans,  
 simplified seq1\_onl\_swap,  
 THEN oinvariant\_anyact]

lemmas oprerr\_guard =  
 open\_seq\_invariant [OF prerr\_guard initiali\_aadv oaadv\_trans aadv\_trans,  
 simplified seq1\_onl\_swap,  
 THEN oinvariant\_anyact]

Proposition 7.28

lemma seq\_compare\_next\_hop':  
 "opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\sigma, \_)$ .  
 $\forall dip$ . let nhip = the (nhop (rt ( $\sigma$  i)) dip)  
 in dip  $\in$  kD(rt ( $\sigma$  i))  $\wedge$  nhip  $\neq$  dip  $\rightarrow$   
 dip  $\in$  kD(rt ( $\sigma$  nhip))  $\wedge$  nsqn (rt ( $\sigma$  i)) dip  $\leq$  nsqn (rt ( $\sigma$  nhip)) dip)"  
 (is " $\_ \models$  (?S, ?U  $\rightarrow$ )  $\_$ ")  
 <proof>

Proposition 7.30

lemmas okD\_unk\_or\_atleast\_one =  
 open\_seq\_invariant [OF kD\_unk\_or\_atleast\_one initiali\_aadv,  
 simplified seq1\_onl\_swap]

lemmas ozero\_seq\_unk\_hops\_one =  
 open\_seq\_invariant [OF zero\_seq\_unk\_hops\_one initiali\_aadv,  
 simplified seq1\_onl\_swap]

lemma oreachable\_fresh\_okD\_unk\_or\_atleast\_one:  
 fixes dip  
 assumes " $(\sigma, p) \in$  oreachable (opaadv i)  
 (otherwith (op=) {i} (orecvmsg ( $\lambda\sigma m$ . msg\_fresh  $\sigma m$   
 $\wedge$  msg\_zhops m)))  
 (other quality\_increases {i}))"  
 and "dip  $\in$  kD(rt ( $\sigma$  i))"  
 shows " $\pi_3(\text{the } (\text{rt } (\sigma i) dip)) = \text{unk} \vee 1 \leq \pi_2(\text{the } (\text{rt } (\sigma i) dip))$ "  
 (is "?P dip")  
 <proof>

lemma oreachable\_fresh\_ozero\_seq\_unk\_hops\_one:  
 fixes dip  
 assumes " $(\sigma, p) \in$  oreachable (opaadv i)  
 (otherwith (op=) {i} (orecvmsg ( $\lambda\sigma m$ . msg\_fresh  $\sigma m$   
 $\wedge$  msg\_zhops m)))  
 (other quality\_increases {i}))"  
 and "dip  $\in$  kD(rt ( $\sigma$  i))"  
 shows "sqn (rt ( $\sigma$  i)) dip = 0  $\rightarrow$   
 sqnf (rt ( $\sigma$  i)) dip = unk  
 $\wedge$  the (dhops (rt ( $\sigma$  i)) dip) = 1  
 $\wedge$  the (nhop (rt ( $\sigma$  i)) dip) = dip"  
 (is "?P dip")  
 <proof>

lemma seq\_nhop\_quality\_increases':  
 shows "opaadv i  $\models$  (otherwith (op=) {i}  
 (orecvmsg ( $\lambda\sigma m$ . msg\_fresh  $\sigma m \wedge$  msg\_zhops m))),

```

    other quality_increases {i} →)
  onl  $\Gamma_{AODV} (\lambda(\sigma, \_). \forall dip. \text{let } nhip = \text{the } (\text{nhop } (\text{rt } (\sigma \ i)) \ dip)
    \text{in } dip \in vD(\text{rt } (\sigma \ i)) \cap vD(\text{rt } (\sigma \ nhip))
    \wedge nhip \neq dip
    \rightarrow (\text{rt } (\sigma \ i)) \sqsubset_{dip} (\text{rt } (\sigma \ nhip)))"$ 

```

```

(is "_  $\models$  (?S i, _  $\rightarrow$  _)")
<proof>

```

**lemma seq\_compare\_next\_hop:**

```

fixes w
shows "opaadv i  $\models$  (otherwith (op=) {i} (orecvmsg msg_fresh),
  other quality_increases {i} →)
  global ( $\lambda\sigma. \forall dip. \text{let } nhip = \text{the } (\text{nhop } (\text{rt } (\sigma \ i)) \ dip)
    \text{in } dip \in kD(\text{rt } (\sigma \ i)) \wedge nhip \neq dip \rightarrow
    dip \in kD(\text{rt } (\sigma \ nhip))
    \wedge nsqn(\text{rt } (\sigma \ i)) \ dip \leq nsqn(\text{rt } (\sigma \ nhip)) \ dip)"$ 

```

```

<proof>

```

**lemma seq\_nhop\_quality\_increases:**

```

shows "opaadv i  $\models$  (otherwith (op=) {i}
  (orecvmsg ( $\lambda\sigma \ m. \text{msg\_fresh } \sigma \ m \wedge \text{msg\_zhops } m$ )),
  other quality_increases {i} →)
  global ( $\lambda\sigma. \forall dip. \text{let } nhip = \text{the } (\text{nhop } (\text{rt } (\sigma \ i)) \ dip)
    \text{in } dip \in vD(\text{rt } (\sigma \ i)) \cap vD(\text{rt } (\sigma \ nhip)) \wedge nhip \neq dip
    \rightarrow (\text{rt } (\sigma \ i)) \sqsubset_{dip} (\text{rt } (\sigma \ nhip)))"$ 

```

```

<proof>

```

end

### 3.10 Routing graphs and loop freedom

```

theory C_Loop_Freedom
imports C_Aodv_Predicates C_Fresher
begin

```

Define the central theorem that relates an invariant over network states to the absence of loops in the associate routing graph.

**definition**

```

rt_graph :: "(ip  $\Rightarrow$  state)  $\Rightarrow$  ip  $\Rightarrow$  ip rel"
where
  "rt_graph  $\sigma = (\lambda dip.
    \{(ip, ip') \mid ip \ ip' \ \text{dsn} \ \text{dsk} \ \text{hops}.
    ip \neq dip \wedge \text{rt } (\sigma \ ip) \ dip = \text{Some } (\text{dsn}, \text{dsk}, \text{val}, \text{hops}, ip')\})"$ 

```

Given the state of a network  $\sigma$ , a routing graph for a given destination  $dip$  abstracts the details of routing tables into nodes (ip addresses) and vertices (valid routes between ip addresses).

**lemma rt\_graphE [elim]:**

```

fixes n dip ip ip'
assumes "(ip, ip')  $\in$  rt_graph  $\sigma$  dip"
shows "ip  $\neq$  dip  $\wedge$  ( $\exists r. \text{rt } (\sigma \ ip) = r
  \wedge (\exists \text{dsn} \ \text{dsk} \ \text{hops}. r \ dip = \text{Some } (\text{dsn}, \text{dsk}, \text{val}, \text{hops}, ip')))"$ 

```

```

<proof>

```

**lemma rt\_graph\_vD [dest]:**

```

" $\bigwedge ip \ ip' \ \sigma \ dip. (ip, ip') \in \text{rt\_graph } \sigma \ dip \implies dip \in vD(\text{rt } (\sigma \ ip))"$ 
<proof>

```

**lemma rt\_graph\_vD\_trans [dest]:**

```

" $\bigwedge ip \ ip' \ \sigma \ dip. (ip, ip') \in (\text{rt\_graph } \sigma \ dip)^+ \implies dip \in vD(\text{rt } (\sigma \ ip))"$ 
<proof>

```

**lemma rt\_graph\_not\_dip [dest]:**

```

" $\bigwedge ip \ ip' \ \sigma \ dip. (ip, ip') \in \text{rt\_graph } \sigma \ dip \implies ip \neq dip"$ 

```



*<proof>*

**lemma** *rt\_graph\_not\_dip\_trans* [*dest*]:

" $\bigwedge ip\ ip'\ \sigma\ dip.\ (ip,\ ip') \in (rt\_graph\ \sigma\ dip)^+ \implies ip \neq dip$ "

*<proof>*

NB: the property below cannot be lifted to the transitive closure

**lemma** *rt\_graph\_nhip\_is\_nhop* [*dest*]:

" $\bigwedge ip\ ip'\ \sigma\ dip.\ (ip,\ ip') \in rt\_graph\ \sigma\ dip \implies ip' = the\ (nhop\ (rt\ (\sigma\ ip))\ dip)$ "

*<proof>*

**theorem** *inv\_to\_loop\_freedom*:

assumes " $\forall i\ dip.\ let\ nhip = the\ (nhop\ (rt\ (\sigma\ i))\ dip)$   
in  $dip \in vD\ (rt\ (\sigma\ i)) \cap vD\ (rt\ (\sigma\ nhip)) \wedge nhip \neq dip$   
 $\longrightarrow (rt\ (\sigma\ i)) \sqsubset_{dip}\ (rt\ (\sigma\ nhip))$ "

shows " $\forall dip.\ irrefl\ ((rt\_graph\ \sigma\ dip)^+)$ "

*<proof>*

end

## 3.11 Lift and transfer invariants to show loop freedom

**theory** *C\_Aodv\_Loop\_Freedom*

imports "*../../AWN/OClosed\_Transfer*" "*../../AWN/Qmsg\_Lifting*" *C\_Global\_Invariants* *C\_Loop\_Freedom*

begin

### 3.11.1 Lift to parallel processes with queues

**lemma** *par\_step\_no\_change\_on\_send\_or\_receive*:

fixes  $\sigma\ s\ a\ \sigma'\ s'$   
assumes " $((\sigma,\ s),\ a,\ (\sigma',\ s')) \in oparp\_sos\ i\ (oseqp\_sos\ \Gamma_{AODV}\ i)\ (seqp\_sos\ \Gamma_{QMSG})$ "  
and " $a \neq \tau$ "  
shows " $\sigma' i = \sigma i$ "

*<proof>*

**lemma** *par\_nhop\_quality\_increases*:

shows " $opaodv\ i\ \langle\langle_i\ qmsg \models (otherwith\ (op=)\ \{i\}\ (orecvmsg\ (\lambda\sigma\ m.\ msg\_fresh\ \sigma\ m \wedge msg\_zhops\ m)),$   
 $other\ quality\_increases\ \{i\} \rightarrow)$   
global  $(\lambda\sigma.\ \forall dip.\ let\ nhip = the\ (nhop\ (rt\ (\sigma\ i))\ dip)$   
in  $dip \in vD\ (rt\ (\sigma\ i)) \cap vD\ (rt\ (\sigma\ nhip)) \wedge nhip \neq dip$   
 $\longrightarrow (rt\ (\sigma\ i)) \sqsubset_{dip}\ (rt\ (\sigma\ nhip))$ )"

*<proof>*

**lemma** *par\_rreq\_rrep\_sn\_quality\_increases*:

" $opaodv\ i\ \langle\langle_i\ qmsg \models_A\ (\lambda\sigma\ \_.\ orecvmsg\ (\lambda\ \_.\ rreq\_rrep\_sn)\ \sigma,\ other\ (\lambda\ \_.\ True)\ \{i\} \rightarrow)$   
globala  $(\lambda(\sigma,\ \_,\ \sigma').\ quality\_increases\ (\sigma\ i)\ (\sigma'\ i))$ "

*<proof>*

**lemma** *par\_rreq\_rrep\_nsqn\_fresh\_any\_step*:

shows " $opaodv\ i\ \langle\langle_i\ qmsg \models_A\ (\lambda\sigma\ \_.\ orecvmsg\ (\lambda\ \_.\ rreq\_rrep\_sn)\ \sigma,$   
 $other\ (\lambda\ \_.\ True)\ \{i\} \rightarrow)$   
globala  $(\lambda(\sigma,\ a,\ \sigma').\ anycast\ (msg\_fresh\ \sigma)\ a)$ "

*<proof>*

**lemma** *par\_anycast\_msg\_zhops*:

shows " $opaodv\ i\ \langle\langle_i\ qmsg \models_A\ (\lambda\sigma\ \_.\ orecvmsg\ (\lambda\ \_.\ rreq\_rrep\_sn)\ \sigma,\ other\ (\lambda\ \_.\ True)\ \{i\} \rightarrow)$   
globala  $(\lambda(\_,\ a,\ \_).\ anycast\ msg\_zhops\ a)$ "

*<proof>*

### 3.11.2 Lift to nodes

**lemma** *node\_step\_no\_change\_on\_send\_or\_receive*:

assumes " $((\sigma,\ NodeS\ i\ P\ R),\ a,\ (\sigma',\ NodeS\ i'\ P'\ R')) \in onode\_sos$ "

(oparp\_sos i (oseqp\_sos  $\Gamma_{AODV}$  i) (seqp\_sos  $\Gamma_{QMSG}$ ))"

and "a  $\neq \tau$ "  
 shows " $\sigma' i = \sigma i$ "  
 <proof>

lemma node\_nhop\_quality\_increases:

shows " $\langle i : opaadv i \langle \langle_i qmsg : R \rangle_o \models$   
 (otherwith (op=) {i}  
 (oarrivemsg ( $\lambda \sigma m. msg\_fresh \sigma m \wedge msg\_zhops m$ )),  
 other quality\_increases {i}  
 $\rightarrow$ ) global ( $\lambda \sigma. \forall dip. let nhip = the (nhop (rt (\sigma i)) dip)$   
 in  $dip \in vD (rt (\sigma i)) \cap vD (rt (\sigma nhip)) \wedge nhip \neq dip$   
 $\rightarrow (rt (\sigma i)) \sqsubset_{dip} (rt (\sigma nhip))$ )"

<proof>

lemma node\_quality\_increases:

" $\langle i : opaadv i \langle \langle_i qmsg : R \rangle_o \models_A (\lambda \sigma \_ . oarrivemsg (\lambda \_ . rreq\_rrep\_sn) \sigma,$   
 other ( $\lambda \_ \_ . True$ ) {i}  $\rightarrow$ )  
 globala ( $\lambda (\sigma, \_ , \sigma'). quality\_increases (\sigma i) (\sigma' i)$ )"

<proof>

lemma node\_rreq\_rrep\_nsqn\_fresh\_any\_step:

shows " $\langle i : opaadv i \langle \langle_i qmsg : R \rangle_o \models_A$   
 ( $\lambda \sigma \_ . oarrivemsg (\lambda \_ . rreq\_rrep\_sn) \sigma, other (\lambda \_ \_ . True) \{i\} \rightarrow$ )  
 globala ( $\lambda (\sigma, a, \sigma'). castmsg (msg\_fresh \sigma) a$ )"

<proof>

lemma node\_anycast\_msg\_zhops:

shows " $\langle i : opaadv i \langle \langle_i qmsg : R \rangle_o \models_A$   
 ( $\lambda \sigma \_ . oarrivemsg (\lambda \_ . rreq\_rrep\_sn) \sigma, other (\lambda \_ \_ . True) \{i\} \rightarrow$ )  
 globala ( $\lambda (\_, a, \_). castmsg (msg\_zhops a)$ )"

<proof>

lemma node\_silent\_change\_only:

shows " $\langle i : opaadv i \langle \langle_i qmsg : R_i \rangle_o \models_A (\lambda \sigma \_ . oarrivemsg (\lambda \_ \_ . True) \sigma,$   
 other ( $\lambda \_ \_ . True$ ) {i}  $\rightarrow$ )  
 globala ( $\lambda (\sigma, a, \sigma'). a \neq \tau \rightarrow \sigma' i = \sigma i$ )"

<proof>

### 3.11.3 Lift to partial networks

lemma arrive\_rreq\_rrep\_nsqn\_fresh\_inc\_sn [simp]:

assumes "oarrivemsg ( $\lambda \sigma m. msg\_fresh \sigma m \wedge P \sigma m$ )  $\sigma m$ "  
 shows "oarrivemsg ( $\lambda \_ . rreq\_rrep\_sn$ )  $\sigma m$ "

<proof>

lemma opnet\_nhop\_quality\_increases:

shows "opnet ( $\lambda i. opaadv i \langle \langle_i qmsg \rangle p \models$   
 (otherwith (op=) (net\_tree\_ips p)  
 (oarrivemsg ( $\lambda \sigma m. msg\_fresh \sigma m \wedge msg\_zhops m$ )),  
 other quality\_increases (net\_tree\_ips p)  $\rightarrow$ )  
 global ( $\lambda \sigma. \forall i \in net\_tree\_ips p. \forall dip.$   
 let  $nhip = the (nhop (rt (\sigma i)) dip)$   
 in  $dip \in vD (rt (\sigma i)) \cap vD (rt (\sigma nhip)) \wedge nhip \neq dip$   
 $\rightarrow (rt (\sigma i)) \sqsubset_{dip} (rt (\sigma nhip))$ )"

<proof>

### 3.11.4 Lift to closed networks

lemma onet\_nhop\_quality\_increases:

shows "oclosed (opnet ( $\lambda i. opaadv i \langle \langle_i qmsg \rangle p$ )  
 $\models (\lambda \_ \_ \_ . True, other quality\_increases (net\_tree\_ips p) \rightarrow)$   
 global ( $\lambda \sigma. \forall i \in net\_tree\_ips p. \forall dip.$   
 let  $nhip = the (nhop (rt (\sigma i)) dip)$

```

      in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
      → (rt (σ i)) ⊆dip (rt (σ nhip)))"
(is "_ ⊨ (_, ?U →) ?inv")
⟨proof⟩

```

### 3.11.5 Transfer into the standard model

```

interpretation aadv_openproc: openproc paadv opaadv id
rewrites "aadv_openproc.initmissing = initmissing"
⟨proof⟩

```

```

interpretation aadv_openproc_par_qmsg: openproc_parq paadv opaadv id qmsg
rewrites "aadv_openproc_par_qmsg.netglobal = netglobal"
  and "aadv_openproc_par_qmsg.initmissing = initmissing"
⟨proof⟩

```

```

lemma net_nhop_quality_increases:
  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paadv i ⟨⟨ qmsg ⟩⟩ n) ⊨ netglobal
    (λσ. ∀i dip. let nhip = the (nhop (rt (σ i)) dip)
      in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
      → (rt (σ i)) ⊆dip (rt (σ nhip)))"
    (is "_ ⊨ netglobal (λσ. ∀i. ?inv σ i)")
⟨proof⟩

```

### 3.11.6 Loop freedom of AODV

```

theorem aadv_loop_freedom:
  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paadv i ⟨⟨ qmsg ⟩⟩ n) ⊨ netglobal (λσ. ∀dip. irrefl ((rt_graph σ dip)+))"
⟨proof⟩

```

end

# Chapter 4

## Variant D: Forwarding the Route Request

Explanation [4, §10.5]: In AODV's route discovery process, a destination node (or an intermediate node with an active route to the destination) will generate a RREP message in response to a received RREQ message. The RREQ message is then dropped and not forwarded. This termination of the route discovery process at the destination can lead to other nodes inadvertently creating non-optimal routes to the source node [5]. A possible modification to solve this problem is to allow the destination node to continue to forward the RREQ message. A route request is only stopped if it has been handled before. The forwarded RREQ message from the destination node needs to be modified to include a Boolean flag `handled` that indicates a RREP message has already been generated and sent in response to the former message. In case the flag is set to true, it prevents other nodes (with valid route to the destination) from sending a RREP message in response to their reception of the forwarded RREQ message.

### 4.1 Predicates and functions used in the AODV model

```
theory D_Aodv_Data
imports D_Fwdrreqs
begin
```

#### 4.1.1 Sequence Numbers

Sequence numbers approximate the relative freshness of routing information.

```
definition inc :: "sqn  $\Rightarrow$  sqn"
  where "inc sn  $\equiv$  if sn = 0 then sn else sn + 1"
```

```
lemma less_than_inc [simp]: "x  $\leq$  inc x"
  <proof>
```

```
lemma inc_minus_suc_0 [simp]:
  "inc x - Suc 0 = x"
  <proof>
```

```
lemma inc_never_one' [simp, intro]: "inc x  $\neq$  Suc 0"
  <proof>
```

```
lemma inc_never_one [simp, intro]: "inc x  $\neq$  1"
  <proof>
```

#### 4.1.2 Modelling Routes

A route is a 6-tuple,  $(dsn, dsk, flag, hops, nhop, pre)$  where  $dsn$  is the 'destination sequence number',  $dsk$  is the 'destination-sequence-number status',  $flag$  is the route status,  $hops$  is the number of hops to the destination,  $nhop$  is the next hop toward the destination, and  $pre$  is the set of 'precursor nodes'—those interested in hearing about changes to the route.

```
type_synonym r = "sqn  $\times$  k  $\times$  f  $\times$  nat  $\times$  ip  $\times$  ip set"
```

```
definition proj2 :: "r  $\Rightarrow$  sqn" ("π2")
```

```

where "π2 ≡ λ(dsn, _, _, _, _, _). dsn"

definition proj3 :: "r ⇒ k" ("π3")
  where "π3 ≡ λ(_, dsk, _, _, _, _). dsk"

definition proj4 :: "r ⇒ f" ("π4")
  where "π4 ≡ λ(_, _, flag, _, _, _). flag"

definition proj5 :: "r ⇒ nat" ("π5")
  where "π5 ≡ λ(_, _, _, hops, _, _). hops"

definition proj6 :: "r ⇒ ip" ("π6")
  where "π6 ≡ λ(_, _, _, _, nhip, _). nhip"

definition proj7 :: "r ⇒ ip set" ("π7")
  where "π7 ≡ λ(_, _, _, _, _, pre). pre"

lemma projs [simp]:
  "π2(dsn, dsk, flag, hops, nhip, pre) = dsn"
  "π3(dsn, dsk, flag, hops, nhip, pre) = dsk"
  "π4(dsn, dsk, flag, hops, nhip, pre) = flag"
  "π5(dsn, dsk, flag, hops, nhip, pre) = hops"
  "π6(dsn, dsk, flag, hops, nhip, pre) = nhip"
  "π7(dsn, dsk, flag, hops, nhip, pre) = pre"
  <proof>

lemma proj3_pred [intro]: "[[ P kno; P unk ] ⇒ P (π3 x)]"
  <proof>

lemma proj4_pred [intro]: "[[ P val; P inv ] ⇒ P (π4 x)]"
  <proof>

lemma proj6_pair_snd [simp]:
  fixes dsn' r
  shows "π6(dsn', snd (r)) = π6(r)"
  <proof>

```

### 4.1.3 Routing Tables

Routing tables map ip addresses to route entries.

```

type_synonym rt = "ip → r"

syntax
  "_Sigma_route" :: "rt ⇒ ip → r" ("σroute'(_, _)'")

translations
  "σroute(rt, dip)" => "rt dip"

definition sqn :: "rt ⇒ ip ⇒ sqn"
  where "sqn rt dip ≡ case σroute(rt, dip) of Some r ⇒ π2(r) | None ⇒ 0"

definition sqnf :: "rt ⇒ ip ⇒ k"
  where "sqnf rt dip ≡ case σroute(rt, dip) of Some r ⇒ π3(r) | None ⇒ unk"

abbreviation flag :: "rt ⇒ ip → f"
  where "flag rt dip ≡ map_option π4 (σroute(rt, dip))"

abbreviation dhops :: "rt ⇒ ip → nat"
  where "dhops rt dip ≡ map_option π5 (σroute(rt, dip))"

abbreviation nhop :: "rt ⇒ ip → ip"
  where "nhop rt dip ≡ map_option π6 (σroute(rt, dip))"

abbreviation precs :: "rt ⇒ ip → ip set"

```

```

where "precs rt dip  $\equiv$  map_option  $\pi_7$  ( $\sigma_{route}(rt, dip)$ )"

definition vD :: "rt  $\Rightarrow$  ip set"
  where "vD rt  $\equiv$  {dip. flag rt dip = Some val}"

definition iD :: "rt  $\Rightarrow$  ip set"
  where "iD rt  $\equiv$  {dip. flag rt dip = Some inv}"

definition kD :: "rt  $\Rightarrow$  ip set"
  where "kD rt  $\equiv$  {dip. rt dip  $\neq$  None}"

lemma kD_is_vD_and_iD: "kD rt = vD rt  $\cup$  iD rt"
  <proof>

lemma vD_iD_gives_kD [simp]:
  " $\bigwedge ip$  rt. ip  $\in$  vD rt  $\implies$  ip  $\in$  kD rt"
  " $\bigwedge ip$  rt. ip  $\in$  iD rt  $\implies$  ip  $\in$  kD rt"
  <proof>

lemma kD_Some [dest]:
  fixes dip rt
  assumes "dip  $\in$  kD rt"
  shows " $\exists$  dsn dsk flag hops nhip pre.
     $\sigma_{route}(rt, dip) = \text{Some } (dsn, dsk, flag, hops, nhip, pre)$ "
  <proof>

lemma kD_None [dest]:
  fixes dip rt
  assumes "dip  $\notin$  kD rt"
  shows " $\sigma_{route}(rt, dip) = \text{None}$ "
  <proof>

lemma vD_Some [dest]:
  fixes dip rt
  assumes "dip  $\in$  vD rt"
  shows " $\exists$  dsn dsk hops nhip pre.
     $\sigma_{route}(rt, dip) = \text{Some } (dsn, dsk, val, hops, nhip, pre)$ "
  <proof>

lemma vD_empty [simp]: "vD Map.empty = {}"
  <proof>

lemma iD_Some [dest]:
  fixes dip rt
  assumes "dip  $\in$  iD rt"
  shows " $\exists$  dsn dsk hops nhip pre.
     $\sigma_{route}(rt, dip) = \text{Some } (dsn, dsk, inv, hops, nhip, pre)$ "
  <proof>

lemma val_is_vD [elim]:
  fixes ip rt
  assumes "ip  $\in$  kD(rt)"
  and "the (flag rt ip) = val"
  shows "ip  $\in$  vD(rt)"
  <proof>

lemma inv_is_iD [elim]:
  fixes ip rt
  assumes "ip  $\in$  kD(rt)"
  and "the (flag rt ip) = inv"
  shows "ip  $\in$  iD(rt)"
  <proof>

lemma iD_flag_is_inv [elim, simp]:

```

```

    fixes ip rt
    assumes "ip∈iD(rt)"
    shows "the (flag rt ip) = inv"
    ⟨proof⟩

lemma kD_but_not_vD_is_iD [elim]:
    fixes ip rt
    assumes "ip∈kD(rt)"
    and "ip∉vD(rt)"
    shows "ip∈iD(rt)"
    ⟨proof⟩

lemma vD_or_iD [elim]:
    fixes ip rt
    assumes "ip∈kD(rt)"
    and "ip∈vD(rt) ⇒ P rt ip"
    and "ip∈iD(rt) ⇒ P rt ip"
    shows "P rt ip"
    ⟨proof⟩

lemma proj5_eq_dhops: "∧dip rt. dip∈kD(rt) ⇒ π5(the (rt dip)) = the (dhops rt dip)"
    ⟨proof⟩

lemma proj4_eq_flag: "∧dip rt. dip∈kD(rt) ⇒ π4(the (rt dip)) = the (flag rt dip)"
    ⟨proof⟩

lemma proj2_eq_sqn: "∧dip rt. dip∈kD(rt) ⇒ π2(the (rt dip)) = sqn rt dip"
    ⟨proof⟩

lemma kD_sqnf_is_proj3 [simp]:
    "∧ip rt. ip∈kD(rt) ⇒ sqnf rt ip = π3(the (rt ip))"
    ⟨proof⟩

lemma vD_flag_val [simp]:
    "∧dip rt. dip ∈ vD (rt) ⇒ the (flag rt dip) = val"
    ⟨proof⟩

lemma kD_update [simp]:
    "∧rt nip v. kD (rt(nip ↦ v)) = insert nip (kD rt)"
    ⟨proof⟩

lemma kD_empty [simp]: "kD Map.empty = {}"
    ⟨proof⟩

lemma ip_equal_or_known [elim]:
    fixes rt ip ip'
    assumes "ip = ip' ∨ ip∈kD(rt)"
    and "ip = ip' ⇒ P rt ip ip'"
    and "[[ ip ≠ ip'; ip∈kD(rt) ] ⇒ P rt ip ip'"
    shows "P rt ip ip'"
    ⟨proof⟩

```

#### 4.1.4 Updating Routing Tables

Routing table entries are modified through explicit functions. The properties of these functions are important in invariant proofs.

##### Updating Precursor Lists

```

definition addpre :: "r ⇒ ip set ⇒ r"
    where "addpre r npre ≡ let (dsn, dsk, flag, hops, nhip, pre) = r in
        (dsn, dsk, flag, hops, nhip, pre ∪ npre)"

```

```

lemma proj2_addpre:

```

```

fixes v pre
shows " $\pi_2(\text{addpre } v \text{ pre}) = \pi_2(v)$ "
<proof>

lemma proj3_addpre:
fixes v pre
shows " $\pi_3(\text{addpre } v \text{ pre}) = \pi_3(v)$ "
<proof>

lemma proj4_addpre:
fixes v pre
shows " $\pi_4(\text{addpre } v \text{ pre}) = \pi_4(v)$ "
<proof>

lemma proj5_addpre:
fixes v pre
shows " $\pi_5(\text{addpre } v \text{ pre}) = \pi_5(v)$ "
<proof>

lemma proj6_addpre:
fixes dsn dsk flag hops nhip pre npre
shows " $\pi_6(\text{addpre } v \text{ npre}) = \pi_6(v)$ "
<proof>

lemma proj7_addpre:
fixes dsn dsk flag hops nhip pre npre
shows " $\pi_7(\text{addpre } v \text{ npre}) = \pi_7(v) \cup \text{npre}$ "
<proof>

lemma addpre_empty: "addpre r {} = r"
<proof>

lemma addpre_r:
"addpre (dsn, dsk, fl, hops, nhip, pre) npre = (dsn, dsk, fl, hops, nhip, pre  $\cup$  npre)"
<proof>

lemmas addpre_simps [simp] = proj2_addpre proj3_addpre proj4_addpre proj5_addpre
proj6_addpre proj7_addpre addpre_empty addpre_r

definition addpreRT :: "rt  $\Rightarrow$  ip  $\Rightarrow$  ip set  $\rightarrow$  rt"
where "addpreRT rt dip npre  $\equiv$ 
map_option ( $\lambda s. \text{rt } (\text{dip } \mapsto \text{addpre } s \text{ npre})$ ) ( $\sigma_{\text{route}}(\text{rt}, \text{dip})$ )"

lemma snd_addpre [simp]:
" $\bigwedge \text{dsn dsn}' v \text{pre}. (\text{dsn}, \text{snd}(\text{addpre } (\text{dsn}', v) \text{pre})) = \text{addpre } (\text{dsn}, v) \text{pre}$ "
<proof>

lemma proj2_addpreRT [simp]:
fixes ip rt ip' npre
assumes "ip  $\in$  kD rt"
and "ip'  $\in$  kD rt"
shows " $\pi_2(\text{the } (\text{the } (\text{addpreRT } \text{rt } \text{ip}' \text{ npre}) \text{ ip})) = \pi_2(\text{the } (\text{rt } \text{ip}))$ "
<proof>

lemma proj3_addpreRT [simp]:
fixes ip rt ip' npre
assumes "ip  $\in$  kD rt"
and "ip'  $\in$  kD rt"
shows " $\pi_3(\text{the } (\text{the } (\text{addpreRT } \text{rt } \text{ip}' \text{ npre}) \text{ ip})) = \pi_3(\text{the } (\text{rt } \text{ip}))$ "
<proof>

lemma proj5_addpreRT [simp]:
" $\bigwedge \text{rt dip ip npre}. \text{dip} \in \text{kD}(\text{rt}) \implies \pi_5(\text{the } (\text{the } (\text{addpreRT } \text{rt } \text{dip } \text{npre}) \text{ ip})) = \pi_5(\text{the } (\text{rt } \text{ip}))$ "
<proof>

```



```

lemma flag_addpreRT [simp]:
  fixes rt pre ip dip
  assumes "dip ∈ kD rt"
  shows "flag (the (addpreRT rt dip pre)) ip = flag rt ip"
  ⟨proof⟩

lemma kD_addpreRT [simp]:
  fixes rt dip npre
  assumes "dip ∈ kD rt"
  shows "kD (the (addpreRT rt dip npre)) = kD rt"
  ⟨proof⟩

lemma vD_addpreRT [simp]:
  fixes rt dip npre
  assumes "dip ∈ kD rt"
  shows "vD (the (addpreRT rt dip npre)) = vD rt"
  ⟨proof⟩

lemma iD_addpreRT [simp]:
  fixes rt dip npre
  assumes "dip ∈ kD rt"
  shows "iD (the (addpreRT rt dip npre)) = iD rt"
  ⟨proof⟩

lemma nhop_addpreRT [simp]:
  fixes rt pre ip dip
  assumes "dip ∈ kD rt"
  shows "nhop (the (addpreRT rt dip pre)) ip = nhop rt ip"
  ⟨proof⟩

lemma sqn_addpreRT [simp]:
  fixes rt pre ip dip
  assumes "dip ∈ kD rt"
  shows "sqn (the (addpreRT rt dip pre)) ip = sqn rt ip"
  ⟨proof⟩

lemma dhops_addpreRT [simp]:
  fixes rt pre ip dip
  assumes "dip ∈ kD rt"
  shows "dhops (the (addpreRT rt dip pre)) ip = dhops rt ip"
  ⟨proof⟩

lemma sqnf_addpreRT [simp]:
  "∧ip dip. ip ∈ kD(rt ξ) ⇒ sqnf (the (addpreRT (rt ξ) ip npre)) dip = sqnf (rt ξ) dip"
  ⟨proof⟩

Updating route entries

lemma in_kD_case [simp]:
  fixes dip rt
  assumes "dip ∈ kD(rt)"
  shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = es (the (rt dip))"
  ⟨proof⟩

lemma not_in_kD_case [simp]:
  fixes dip rt
  assumes "dip ∉ kD(rt)"
  shows "(case rt dip of None ⇒ en | Some r ⇒ es r) = en"
  ⟨proof⟩

lemma rt_Some_sqn [dest]:
  fixes rt and ip dsn dsk flag hops nhip pre
  assumes "rt ip = Some (dsn, dsk, flag, hops, nhip, pre)"

```

```

  shows "sqn rt ip = dsn"
  ⟨proof⟩

lemma not_kD_sqn [simp]:
  fixes dip rt
  assumes "dip ∉ kD(rt)"
  shows "sqn rt dip = 0"
  ⟨proof⟩

definition update_arg_wf :: "r ⇒ bool"
where "update_arg_wf r ≡ π4(r) = val ∧
      (π2(r) = 0) = (π3(r) = unk) ∧
      (π3(r) = unk ⟶ π5(r) = 1)"

lemma update_arg_wf_gives_cases:
  "∧r. update_arg_wf r ⟹ (π2(r) = 0) = (π3(r) = unk)"
  ⟨proof⟩

lemma update_arg_wf_tuples [simp]:
  "∧nhip pre. update_arg_wf (0, unk, val, Suc 0, nhip, pre)"
  "∧n hops nhip pre. update_arg_wf (Suc n, kno, val, hops, nhip, pre)"
  ⟨proof⟩

lemma update_arg_wf_tuples' [elim]:
  "∧n hops nhip pre. Suc 0 ≤ n ⟹ update_arg_wf (n, kno, val, hops, nhip, pre)"
  ⟨proof⟩

lemma wf_r_cases [intro]:
  fixes P r
  assumes "update_arg_wf r"
  and c1: "∧nhip pre. P (0, unk, val, Suc 0, nhip, pre)"
  and c2: "∧dsn hops nhip pre. dsn > 0 ⟹ P (dsn, kno, val, hops, nhip, pre)"
  shows "P r"
  ⟨proof⟩

definition update :: "rt ⇒ ip ⇒ r ⇒ rt"
where
  "update rt ip r ≡
  case σroute(rt, ip) of
  None ⇒ rt (ip ↦ r)
  | Some s ⇒
    if π2(s) < π2(r) then rt (ip ↦ addpre r (π7(s)))
    else if π2(s) = π2(r) ∧ (π5(s) > π5(r) ∨ π4(s) = inv)
    then rt (ip ↦ addpre r (π7(s)))
    else if π3(r) = unk
    then rt (ip ↦ (π2(s), snd (addpre r (π7(s))))
    else rt (ip ↦ addpre s (π7(r)))"

lemma update_simps [simp]:
  fixes r s nrt nr nr' ns rt ip
  defines "s ≡ the σroute(rt, ip)"
  and "nr ≡ addpre r (π7(s))"
  and "nr' ≡ (π2(s), π3(nr), π4(nr), π5(nr), π6(nr), π7(nr))"
  and "ns ≡ addpre s (π7(r))"
  shows
  "[ip ∉ kD(rt)] ⟹ update rt ip r = rt (ip ↦ r)"
  "[ip ∈ kD(rt); sqn rt ip < π2(r)] ⟹ update rt ip r = rt (ip ↦ nr)"
  "[ip ∈ kD(rt); sqn rt ip = π2(r);
  the (dhops rt ip) > π5(r)] ⟹ update rt ip r = rt (ip ↦ nr)"
  "[ip ∈ kD(rt); sqn rt ip = π2(r);
  flag rt ip = Some inv] ⟹ update rt ip r = rt (ip ↦ nr)"
  "[ip ∈ kD(rt); π3(r) = unk; (π2(r) = 0) = (π3(r) = unk)] ⟹ update rt ip r = rt (ip ↦ nr)"
  "[ip ∈ kD(rt); sqn rt ip ≥ π2(r); π3(r) = kno;
  sqn rt ip = π2(r) ⟹ the (dhops rt ip) ≤ π5(r) ∧ the (flag rt ip) = val ]"

```

$\implies \text{update rt ip r} = \text{rt (ip } \mapsto \text{ns)}$ "

*<proof>*

**lemma update\_cases [elim]:**

assumes " $(\pi_2(r) = 0) = (\pi_3(r) = \text{unk})$ "

and c1: " $\llbracket \text{ip} \notin \text{kD}(\text{rt}) \rrbracket \implies P(\text{rt (ip } \mapsto \text{r)})$ "

and c2: " $\llbracket \text{ip} \in \text{kD}(\text{rt}); \text{sqn rt ip} < \pi_2(r) \rrbracket$   
 $\implies P(\text{rt (ip } \mapsto \text{addpre r } (\pi_7(\text{the } \sigma_{\text{route}}(\text{rt}, \text{ip}))))$ "

and c3: " $\llbracket \text{ip} \in \text{kD}(\text{rt}); \text{sqn rt ip} = \pi_2(r); \text{the (dhops rt ip)} > \pi_5(r) \rrbracket$   
 $\implies P(\text{rt (ip } \mapsto \text{addpre r } (\pi_7(\text{the } \sigma_{\text{route}}(\text{rt}, \text{ip}))))$ "

and c4: " $\llbracket \text{ip} \in \text{kD}(\text{rt}); \text{sqn rt ip} = \pi_2(r); \text{the (flag rt ip)} = \text{inv} \rrbracket$   
 $\implies P(\text{rt (ip } \mapsto \text{addpre r } (\pi_7(\text{the } \sigma_{\text{route}}(\text{rt}, \text{ip}))))$ "

and c5: " $\llbracket \text{ip} \in \text{kD}(\text{rt}); \pi_3(r) = \text{unk} \rrbracket$   
 $\implies P(\text{rt (ip } \mapsto (\pi_2(\text{the } \sigma_{\text{route}}(\text{rt}, \text{ip})), \pi_3(r),$   
 $\pi_4(r), \pi_5(r), \pi_6(r), \pi_7(\text{addpre r } (\pi_7(\text{the } \sigma_{\text{route}}(\text{rt}, \text{ip}))))))$ "

and c6: " $\llbracket \text{ip} \in \text{kD}(\text{rt}); \text{sqn rt ip} \geq \pi_2(r); \pi_3(r) = \text{kno};$   
 $\text{sqn rt ip} = \pi_2(r) \implies \text{the (dhops rt ip)} \leq \pi_5(r) \wedge \text{the (flag rt ip)} = \text{val} \rrbracket$   
 $\implies P(\text{rt (ip } \mapsto \text{addpre (the } \sigma_{\text{route}}(\text{rt}, \text{ip})) (\pi_7(r))))$ "

shows " $P(\text{update rt ip r})$ "

*<proof>*

**lemma update\_cases\_kD:**

assumes " $(\pi_2(r) = 0) = (\pi_3(r) = \text{unk})$ "

and " $\text{ip} \in \text{kD}(\text{rt})$ "

and c2: " $\llbracket \text{sqn rt ip} < \pi_2(r) \rrbracket \implies P(\text{rt (ip } \mapsto \text{addpre r } (\pi_7(\text{the } \sigma_{\text{route}}(\text{rt}, \text{ip}))))$ "

and c3: " $\llbracket \llbracket \text{sqn rt ip} = \pi_2(r); \text{the (dhops rt ip)} > \pi_5(r) \rrbracket$   
 $\implies P(\text{rt (ip } \mapsto \text{addpre r } (\pi_7(\text{the } \sigma_{\text{route}}(\text{rt}, \text{ip}))))$ "

and c4: " $\llbracket \llbracket \text{sqn rt ip} = \pi_2(r); \text{the (flag rt ip)} = \text{inv} \rrbracket$   
 $\implies P(\text{rt (ip } \mapsto \text{addpre r } (\pi_7(\text{the } \sigma_{\text{route}}(\text{rt}, \text{ip}))))$ "

and c5: " $\llbracket \pi_3(r) = \text{unk} \rrbracket \implies P(\text{rt (ip } \mapsto (\pi_2(\text{the } \sigma_{\text{route}}(\text{rt}, \text{ip})), \pi_3(r),$   
 $\pi_4(r), \pi_5(r), \pi_6(r),$   
 $\pi_7(\text{addpre r } (\pi_7(\text{the } \sigma_{\text{route}}(\text{rt}, \text{ip}))))))$ "

and c6: " $\llbracket \llbracket \text{sqn rt ip} \geq \pi_2(r); \pi_3(r) = \text{kno};$   
 $\text{sqn rt ip} = \pi_2(r) \implies \text{the (dhops rt ip)} \leq \pi_5(r) \wedge \text{the (flag rt ip)} = \text{val} \rrbracket$   
 $\implies P(\text{rt (ip } \mapsto \text{addpre (the } \sigma_{\text{route}}(\text{rt}, \text{ip})) (\pi_7(r))))$ "

shows " $P(\text{update rt ip r})$ "

*<proof>*

**lemma in\_kD\_after\_update [simp]:**

fixes rt nip dsn dsk flag hops nhip pre

shows " $\text{kD}(\text{update rt nip (dsn, dsk, flag, hops, nhip, pre)}) = \text{insert nip (kD rt)}$ "

*<proof>*

**lemma nhop\_of\_update [simp]:**

fixes rt dip dsn dsk flag hops nhip

assumes " $\text{rt} \neq \text{update rt dip (dsn, dsk, flag, hops, nhip, \{ })}$ "

shows " $\text{the (nhop (update rt dip (dsn, dsk, flag, hops, nhip, \{ }))) dip} = \text{nhip}$ "

*<proof>*

**lemma sqn\_if\_updated:**

fixes rip v rt ip

shows " $\text{sqn } (\lambda x. \text{if } x = \text{rip} \text{ then Some } v \text{ else rt } x) \text{ ip}$   
 $= (\text{if } \text{ip} = \text{rip} \text{ then } \pi_2(v) \text{ else sqn rt ip})$ "

*<proof>*

**lemma update\_sqn [simp]:**

fixes rt dip rip dsn dsk hops nhip pre

assumes " $(\text{dsn} = 0) = (\text{dsk} = \text{unk})$ "

shows " $\text{sqn rt dip} \leq \text{sqn (update rt rip (dsn, dsk, val, hops, nhip, pre)) dip}$ "

*<proof>*

**lemma sqn\_update\_bigger [simp]:**

fixes rt ip ip' dsn dsk flag hops nhip pre

```

assumes "1 ≤ hops"
  shows "sqn rt ip ≤ sqn (update rt ip' (dsn, dsk, flag, hops, nhip, pre)) ip"
⟨proof⟩

lemma dhops_update [intro]:
  fixes rt dsn dsk flag hops ip rip nhip pre
  assumes ex: "∀ ip ∈ kD rt. the (dhops rt ip) ≥ 1"
  and ip: "(ip = rip ∧ Suc 0 ≤ hops) ∨ (ip ≠ rip ∧ ip ∈ kD rt)"
  shows "Suc 0 ≤ the (dhops (update rt rip (dsn, dsk, flag, hops, nhip, pre)) ip)"
⟨proof⟩

lemma update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
  shows "(update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = rt ip"
⟨proof⟩

lemma nhop_update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
  shows "nhop (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = nhop rt ip"
⟨proof⟩

lemma dhops_update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip pre
  assumes "ip ≠ dip"
  shows "dhops (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = dhops rt ip"
⟨proof⟩

lemma sqn_update_same [simp]:
  "∧ rt ip dsn dsk flag hops nhip pre. sqn (rt(ip ↦ v)) ip = π2(v)"
⟨proof⟩

lemma dhops_update_changed [simp]:
  fixes rt dip osn hops nhip
  assumes "rt ≠ update rt dip (osn, kno, val, hops, nhip, {})"
  shows "the (dhops (update rt dip (osn, kno, val, hops, nhip, {})) dip) = hops"
⟨proof⟩

lemma nhop_update_unk_val [simp]:
  "∧ rt dip ip dsn hops npre.
  the (nhop (update rt dip (dsn, unk, val, hops, ip, npre)) dip) = ip"
⟨proof⟩

lemma nhop_update_changed [simp]:
  fixes rt dip dsn dsk flg hops sip
  assumes "update rt dip (dsn, dsk, flg, hops, sip, {}) ≠ rt"
  shows "the (nhop (update rt dip (dsn, dsk, flg, hops, sip, {})) dip) = sip"
⟨proof⟩

lemma update_rt_split_asm:
  "∧ rt ip dsn dsk flag hops sip.
  P (update rt ip (dsn, dsk, flag, hops, sip, {}))
  =
  (¬(rt = update rt ip (dsn, dsk, flag, hops, sip, {})) ∧ ¬P rt
  ∨ rt ≠ update rt ip (dsn, dsk, flag, hops, sip, {})
  ∧ ¬P (update rt ip (dsn, dsk, flag, hops, sip, {})))"
⟨proof⟩

lemma sqn_update [simp]: "∧ rt dip dsn flg hops sip.
rt ≠ update rt dip (dsn, kno, flg, hops, sip, {})
⇒ sqn (update rt dip (dsn, kno, flg, hops, sip, {})) dip = dsn"
⟨proof⟩

```

lemma sqnf\_update [simp]: " $\bigwedge$ rt dip dsn dsk flg hops sip.  
 rt  $\neq$  update rt dip (dsn, dsk, flg, hops, sip, { })  
 $\implies$  sqnf (update rt dip (dsn, dsk, flg, hops, sip, { })) dip = dsk"  
 <proof>

lemma update\_kno\_dsn\_greater\_zero:  
 " $\bigwedge$ rt dip ip dsn hops npre.  $1 \leq$  dsn  $\implies$   $1 \leq$  (sqn (update rt dip (dsn, kno, val, hops, ip, npre)) dip)"  
 <proof>

lemma proj3\_update [simp]: " $\bigwedge$ rt dip dsn dsk flg hops sip.  
 rt  $\neq$  update rt dip (dsn, dsk, flg, hops, sip, { })  
 $\implies$   $\pi_3$ (the (update rt dip (dsn, dsk, flg, hops, sip, { }) dip)) = dsk"  
 <proof>

lemma nhop\_update\_changed\_kno\_val [simp]: " $\bigwedge$ rt ip dsn dsk hops nhip.  
 rt  $\neq$  update rt ip (dsn, kno, val, hops, nhip, { })  
 $\implies$  the (nhop (update rt ip (dsn, kno, val, hops, nhip, { })) ip) = nhip"  
 <proof>

lemma flag\_update [simp]: " $\bigwedge$ rt dip dsn flg hops sip.  
 rt  $\neq$  update rt dip (dsn, kno, flg, hops, sip, { })  
 $\implies$  the (flag (update rt dip (dsn, kno, flg, hops, sip, { })) dip) = flg"  
 <proof>

lemma the\_flag\_Some [dest!]:  
 fixes ip rt  
 assumes "the (flag rt ip) = x"  
 and "ip  $\in$  kD rt"  
 shows "flag rt ip = Some x"  
 <proof>

lemma kD\_update\_unchanged [dest]:  
 fixes rt dip dsn dsk flag hops nhip pre  
 assumes "rt = update rt dip (dsn, dsk, flag, hops, nhip, pre)"  
 shows "dip  $\in$  kD(rt)"  
 <proof>

lemma nhop\_update [simp]: " $\bigwedge$ rt dip dsn dsk flg hops sip.  
 rt  $\neq$  update rt dip (dsn, dsk, flg, hops, sip, { })  
 $\implies$  the (nhop (update rt dip (dsn, dsk, flg, hops, sip, { })) dip) = sip"  
 <proof>

lemma sqn\_update\_another [simp]:  
 fixes dip ip rt dsn dsk flag hops nhip pre  
 assumes "ip  $\neq$  dip"  
 shows "sqn (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = sqn rt ip"  
 <proof>

lemma sqnf\_update\_another [simp]:  
 fixes dip ip rt dsn dsk flag hops nhip pre  
 assumes "ip  $\neq$  dip"  
 shows "sqnf (update rt dip (dsn, dsk, flag, hops, nhip, pre)) ip = sqnf rt ip"  
 <proof>

lemma vD\_update\_val [dest]:  
 " $\bigwedge$ dip rt dip' dsn dsk hops nhip pre.  
 dip  $\in$  vD(update rt dip' (dsn, dsk, val, hops, nhip, pre))  $\implies$  (dip  $\in$  vD(rt)  $\vee$  dip=dip)"  
 <proof>

## Invalidating route entries

definition invalidate :: "rt  $\Rightarrow$  (ip  $\rightarrow$  sqn)  $\Rightarrow$  rt"  
 where "invalidate rt dests  $\equiv$   
 $\lambda$ ip. case (rt ip, dests ip) of

```

  (None, _) ⇒ None
| (Some s, None) ⇒ Some s
| (Some (_, dsk, _, hops, nhop, pre), Some rsn) ⇒
  Some (rsn, dsk, inv, hops, nhop, pre)"

```

```

lemma proj3_invalidate [simp]:
  "∧dip. π3(the ((invalidate rt dests) dip)) = π3(the (rt dip))"
⟨proof⟩

```

```

lemma proj5_invalidate [simp]:
  "∧dip. π5(the ((invalidate rt dests) dip)) = π5(the (rt dip))"
⟨proof⟩

```

```

lemma proj6_invalidate [simp]:
  "∧dip. π6(the ((invalidate rt dests) dip)) = π6(the (rt dip))"
⟨proof⟩

```

```

lemma proj7_invalidate [simp]:
  "∧dip. π7(the ((invalidate rt dests) dip)) = π7(the (rt dip))"
⟨proof⟩

```

```

lemma invalidate_kD_inv [simp]:
  "∧rt dests. kD (invalidate rt dests) = kD rt"
⟨proof⟩

```

```

lemma invalidate_sqn:
  fixes rt dip dests
  assumes "∀rsn. dests dip = Some rsn → sqn rt dip ≤ rsn"
  shows "sqn rt dip ≤ sqn (invalidate rt dests) dip"
⟨proof⟩

```

```

lemma sqn_invalidate_in_dests [simp]:
  fixes dests ipa rsn rt
  assumes "dests ipa = Some rsn"
  and "ipa ∈ kD(rt)"
  shows "sqn (invalidate rt dests) ipa = rsn"
⟨proof⟩

```

```

lemma dhops_invalidate [simp]:
  "∧dip. the (dhops (invalidate rt dests) dip) = the (dhops rt dip)"
⟨proof⟩

```

```

lemma sqnf_invalidate [simp]:
  "∧dip. sqnf (invalidate (rt ξ) (dests ξ)) dip = sqnf (rt ξ) dip"
⟨proof⟩

```

```

lemma nhop_invalidate [simp]:
  "∧dip. the (nhop (invalidate (rt ξ) (dests ξ)) dip) = the (nhop (rt ξ) dip)"
⟨proof⟩

```

```

lemma invalidate_other [simp]:
  fixes rt dests dip
  assumes "dip ∉ dom(dests)"
  shows "invalidate rt dests dip = rt dip"
⟨proof⟩

```

```

lemma invalidate_none [simp]:
  fixes rt dests dip
  assumes "dip ∉ kD(rt)"
  shows "invalidate rt dests dip = None"
⟨proof⟩

```

```

lemma vD_invalidate_vD_not_dests:
  "∧dip rt dests. dip ∈ vD (invalidate rt dests) ⇒ dip ∈ vD(rt) ∧ dests dip = None"

```

*<proof>*

```
lemma sqn_invalidate_not_in_dests [simp]:  
  fixes dests dip rt  
  assumes "dip ∉ dom(dests)"  
  shows "sqn (invalidate rt dests) dip = sqn rt dip"  
  <proof>
```

```
lemma invalidate_changes:  
  fixes rt dests dip dsn dsk flag hops nhip pre  
  assumes "invalidate rt dests dip = Some (dsn, dsk, flag, hops, nhip, pre)"  
  shows " dsn = (case dests dip of None ⇒  $\pi_2$ (the (rt dip)) | Some rsn ⇒ rsn)  
    ∧ dsk =  $\pi_3$ (the (rt dip))  
    ∧ flag = (if dests dip = None then  $\pi_4$ (the (rt dip)) else inv)  
    ∧ hops =  $\pi_5$ (the (rt dip))  
    ∧ nhip =  $\pi_6$ (the (rt dip))  
    ∧ pre =  $\pi_7$ (the (rt dip))"  
  <proof>
```

```
lemma proj3_inv: " $\bigwedge$ dip rt dests. dip ∈ kD (rt)  
  ⇒  $\pi_3$ (the (invalidate rt dests dip)) =  $\pi_3$ (the (rt dip))"  
  <proof>
```

```
lemma dests_iD_invalidate [simp]:  
  assumes "dests ip = Some rsn"  
    and "ip ∈ kD(rt)"  
  shows "ip ∈ iD(invalidate rt dests)"  
  <proof>
```

#### 4.1.5 Route Requests

Generate a fresh route request identifier.

```
definition nrreqid :: "(ip × rreqid) set ⇒ ip ⇒ rreqid"  
  where "nrreqid rreqs ip ≡ Max ({n. (ip, n) ∈ rreqs} ∪ {0}) + 1"
```

#### 4.1.6 Queued Packets

Functions for sending data packets.

```
type_synonym store = "ip → (p × data list)"
```

```
definition sigma_queue :: "store ⇒ ip ⇒ data list" ("σqueue'(_, _)'")  
  where "σqueue(store, dip) ≡ case store dip of None ⇒ [] | Some (p, q) ⇒ q"
```

```
definition qD :: "store ⇒ ip set"  
  where "qD ≡ dom"
```

```
definition add :: "data ⇒ ip ⇒ store ⇒ store"  
  where "add d dip store ≡ case store dip of  
    None ⇒ store (dip ↦ (req, [d]))  
    | Some (p, q) ⇒ store (dip ↦ (p, q @ [d]))"
```

```
lemma qD_add [simp]:  
  fixes d dip store  
  shows "qD(add d dip store) = insert dip (qD store)"  
  <proof>
```

```
definition drop :: "ip ⇒ store → store"  
  where "drop dip store ≡  
    map_option (λ(p, q). if tl q = [] then store (dip := None)  
      else store (dip ↦ (p, tl q))) (store dip)"
```

```
definition sigma_p_flag :: "store ⇒ ip → p" ("σp-flag'(_, _)'")
```

where " $\sigma_{p\text{-flag}}(\text{store}, \text{dip}) \equiv \text{map\_option fst (store dip)}$ "

```

definition unsetRRF :: "store  $\Rightarrow$  ip  $\Rightarrow$  store"
  where "unsetRRF store dip  $\equiv$  case store dip of
        None  $\Rightarrow$  store
        | Some (p, q)  $\Rightarrow$  store (dip  $\mapsto$  (noreq, q))"

```

```

definition setRRF :: "store  $\Rightarrow$  (ip  $\rightarrow$  sqn)  $\Rightarrow$  store"
  where "setRRF store dests  $\equiv$   $\lambda$ dip. if dests dip = None then store dip
        else map_option ( $\lambda$ (_, q). (req, q)) (store dip)"

```

#### 4.1.7 Comparison with the original technical report

The major differences with the AODV technical report of Fehnker et al are:

1. *nhop* is partial, thus a ‘the’ is needed, similarly for *dhops* and *addpreRT*.
2. *precs* is partial.
3.  $\sigma_{p\text{-flag}}(\text{store}, \text{dip})$  is partial.
4. The routing table (*rt*) is modelled as a map ( $\text{ip} \Rightarrow r \text{ option}$ ) rather than a set of 7-tuples, likewise, the *r* is a 6-tuple rather than a 7-tuple, i.e., the destination ip-address (*dip*) is taken from the argument to the function, rather than a part of the result. Well-definedness then follows from the structure of the type and more related facts are available automatically, rather than having to be acquired through tedious proofs.
5. Similar remarks hold for the *dests* mapping passed to *invalidate*, and *store*.

end

## 4.2 AODV protocol messages

```

theory D_Aodv_Message
imports D_Fwdrrreqs
begin

```

```

datatype msg =
  Rreq nat rreqid ip sqn k ip sqn ip bool
  | Rrep nat ip sqn ip ip
  | Rerr "ip  $\rightarrow$  sqn" ip
  | Newpkt data ip
  | Pkt data ip ip

```

```

instantiation msg :: msg

```

```

begin

```

```

  definition newpkt_def [simp]: "newpkt  $\equiv$   $\lambda$ (d, dip). Newpkt d dip"

```

```

  definition eq_newpkt_def: "eq_newpkt m  $\equiv$  case m of Newpkt d dip  $\Rightarrow$  True | _  $\Rightarrow$  False"

```

```

  instance <proof>

```

```

end

```

The *msg* type models the different messages used within AODV. The instantiation as a *msg* is a technicality due to the special treatment of *newpkt* messages in the AWN SOS rules. This use of classes allows a clean separation of the AWN-specific definitions and these AODV-specific definitions.

```

definition rreq :: "nat  $\times$  rreqid  $\times$  ip  $\times$  sqn  $\times$  k  $\times$  ip  $\times$  sqn  $\times$  ip  $\times$  bool  $\Rightarrow$  msg"
  where "rreq  $\equiv$   $\lambda$ (hops, rreqid, dip, dsn, dsk, oip, osn, sip, handled).
        Rreq hops rreqid dip dsn dsk oip osn sip handled"

```

```

lemma rreq_simp [simp]:

```

```

  "rreq(hops, rreqid, dip, dsn, dsk, oip, osn, sip, handled) = Rreq hops rreqid dip dsn dsk oip osn sip
  handled"

```

```

  <proof>

```



```

definition rrep :: "nat × ip × sqn × ip × ip ⇒ msg"
  where "rrep ≡ λ(hops, dip, dsn, oip, sip). Rrep hops dip dsn oip sip"

```

```

lemma rrep_simp [simp]:
  "rrep(hops, dip, dsn, oip, sip) = Rrep hops dip dsn oip sip"
  ⟨proof⟩

```

```

definition rerr :: "(ip → sqn) × ip ⇒ msg"
  where "rerr ≡ λ(dests, sip). Rerr dests sip"

```

```

lemma rerr_simp [simp]:
  "rerr(dests, sip) = Rerr dests sip"
  ⟨proof⟩

```

```

lemma not_eq_newpkt_rreq [simp]: "¬eq_newpkt (Rreq hops rreqid dip dsn dsk oip osn sip handled)"
  ⟨proof⟩

```

```

lemma not_eq_newpkt_rrep [simp]: "¬eq_newpkt (Rrep hops dip dsn oip sip)"
  ⟨proof⟩

```

```

lemma not_eq_newpkt_rerr [simp]: "¬eq_newpkt (Rerr dests sip)"
  ⟨proof⟩

```

```

lemma not_eq_newpkt_pkt [simp]: "¬eq_newpkt (Pkt d dip sip)"
  ⟨proof⟩

```

```

definition pkt :: "data × ip × ip ⇒ msg"
  where "pkt ≡ λ(d, dip, sip). Pkt d dip sip"

```

```

lemma pkt_simp [simp]:
  "pkt(d, dip, sip) = Pkt d dip sip"
  ⟨proof⟩

```

end

## 4.3 The AODV protocol

```

theory D_Aodv
imports D_Aodv_Data D_Aodv_Message
  "../.../AWN/AWN_SOS_Labels" "../.../AWN/AWN_Invariants"
begin

```

### 4.3.1 Data state

```

record state =
  ip      :: "ip"
  sn      :: "sqn"
  rt      :: "rt"
  rreqs   :: "(ip × rreqid) set"
  store   :: "store"

  msg     :: "msg"
  data    :: "data"
  dests   :: "ip → sqn"
  pre     :: "ip set"
  rreqid  :: "rreqid"
  dip     :: "ip"
  oip     :: "ip"
  hops    :: "nat"
  dsn     :: "sqn"
  dsk     :: "k"
  osn     :: "sqn"
  sip     :: "ip"
  handled :: "bool"

```

abbreviation aadv\_init :: "ip  $\Rightarrow$  state"

```
where "aadv_init i  $\equiv$  (  
  ip = i,  
  sn = 1,  
  rt = empty,  
  rreqs = {},  
  store = empty,  
  
  msg    = (SOME x. True),  
  data   = (SOME x. True),  
  dests  = (SOME x. True),  
  pre    = (SOME x. True),  
  rreqid = (SOME x. True),  
  dip    = (SOME x. True),  
  oip    = (SOME x. True),  
  hops   = (SOME x. True),  
  dsn    = (SOME x. True),  
  dsk    = (SOME x. True),  
  osn    = (SOME x. True),  
  sip    = (SOME x. x  $\neq$  i),  
  handled= (SOME x. True)  
)"
```

lemma some\_neq\_not\_eq [simp]: " $\neg((\text{SOME } x :: \text{nat. } x \neq i) = i)$ "  
<proof>

definition clear\_locals :: "state  $\Rightarrow$  state"

```
where "clear_locals  $\xi$  =  $\xi$  (  
  msg    := (SOME x. True),  
  data   := (SOME x. True),  
  dests  := (SOME x. True),  
  pre    := (SOME x. True),  
  rreqid := (SOME x. True),  
  dip    := (SOME x. True),  
  oip    := (SOME x. True),  
  hops   := (SOME x. True),  
  dsn    := (SOME x. True),  
  dsk    := (SOME x. True),  
  osn    := (SOME x. True),  
  sip    := (SOME x. x  $\neq$  ip  $\xi$ ),  
  handled:= (SOME x. True)  
)"
```

lemma clear\_locals\_sip\_not\_ip [simp]: " $\neg(\text{sip } (\text{clear\_locals } \xi) = \text{ip } \xi)$ "  
<proof>

lemma clear\_locals\_but\_not\_globals [simp]:

```
"ip (clear_locals  $\xi$ ) = ip  $\xi$ "  
"sn (clear_locals  $\xi$ ) = sn  $\xi$ "  
"rt (clear_locals  $\xi$ ) = rt  $\xi$ "  
"rreqs (clear_locals  $\xi$ ) = rreqs  $\xi$ "  
"store (clear_locals  $\xi$ ) = store  $\xi$ "  
<proof>
```

### 4.3.2 Auxilliary message handling definitions

definition is\_newpkt

```
where "is_newpkt  $\xi$   $\equiv$  case msg  $\xi$  of  
  Newpkt data' dip'  $\Rightarrow$  {  $\xi$ (data := data', dip := dip') }  
  | _  $\Rightarrow$  {}"
```

definition is\_pkt

```
where "is_pkt  $\xi$   $\equiv$  case msg  $\xi$  of
```

Pkt data' dip' oip'  $\Rightarrow$  {  $\xi$ (| data := data', dip := dip', oip := oip' |) }  
 | \_  $\Rightarrow$  {}"

**definition is\_rreq**

where "is\_rreq  $\xi \equiv$  case msg  $\xi$  of  
     Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip' handled'  $\Rightarrow$   
         {  $\xi$ (| hops := hops', rreqid := rreqid', dip := dip', dsn := dsn',  
           dsk := dsk', oip := oip', osn := osn', sip := sip',  
           handled := handled' |) }  
 | \_  $\Rightarrow$  {}"

**lemma is\_rreq\_asm [dest!]:**

assumes " $\xi' \in$  is\_rreq  $\xi$ "  
 shows "( $\exists$ hops' rreqid' dip' dsn' dsk' oip' osn' sip' handled'.  
     msg  $\xi =$  Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip' handled'  $\wedge$   
      $\xi' = \xi$ (| hops := hops', rreqid := rreqid', dip := dip', dsn := dsn',  
           dsk := dsk', oip := oip', osn := osn', sip := sip',  
           handled := handled' |))"

*<proof>*

**definition is\_rrep**

where "is\_rrep  $\xi \equiv$  case msg  $\xi$  of  
     Rrep hops' dip' dsn' oip' sip'  $\Rightarrow$   
         {  $\xi$ (| hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' |) }  
 | \_  $\Rightarrow$  {}"

**lemma is\_rrep\_asm [dest!]:**

assumes " $\xi' \in$  is\_rrep  $\xi$ "  
 shows "( $\exists$ hops' dip' dsn' oip' sip'.  
     msg  $\xi =$  Rrep hops' dip' dsn' oip' sip'  $\wedge$   
      $\xi' = \xi$ (| hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' |))"

*<proof>*

**definition is\_rerr**

where "is\_rerr  $\xi \equiv$  case msg  $\xi$  of  
     Rerr dests' sip'  $\Rightarrow$  {  $\xi$ (| dests := dests', sip := sip' |) }  
 | \_  $\Rightarrow$  {}"

**lemma is\_rerr\_asm [dest!]:**

assumes " $\xi' \in$  is\_rerr  $\xi$ "  
 shows "( $\exists$ dests' sip'.  
     msg  $\xi =$  Rerr dests' sip'  $\wedge$   
      $\xi' = \xi$ (| dests := dests', sip := sip' |))"

*<proof>*

**lemmas is\_msg\_defs =**

is\_rerr\_def is\_rrep\_def is\_rreq\_def is\_pkt\_def is\_newpkt\_def

**lemma is\_msg\_inv\_ip [simp]:**

" $\xi' \in$  is\_rerr  $\xi \implies$  ip  $\xi' =$  ip  $\xi$ "  
 " $\xi' \in$  is\_rrep  $\xi \implies$  ip  $\xi' =$  ip  $\xi$ "  
 " $\xi' \in$  is\_rreq  $\xi \implies$  ip  $\xi' =$  ip  $\xi$ "  
 " $\xi' \in$  is\_pkt  $\xi \implies$  ip  $\xi' =$  ip  $\xi$ "  
 " $\xi' \in$  is\_newpkt  $\xi \implies$  ip  $\xi' =$  ip  $\xi$ "

*<proof>*

**lemma is\_msg\_inv\_sn [simp]:**

" $\xi' \in$  is\_rerr  $\xi \implies$  sn  $\xi' =$  sn  $\xi$ "  
 " $\xi' \in$  is\_rrep  $\xi \implies$  sn  $\xi' =$  sn  $\xi$ "  
 " $\xi' \in$  is\_rreq  $\xi \implies$  sn  $\xi' =$  sn  $\xi$ "  
 " $\xi' \in$  is\_pkt  $\xi \implies$  sn  $\xi' =$  sn  $\xi$ "  
 " $\xi' \in$  is\_newpkt  $\xi \implies$  sn  $\xi' =$  sn  $\xi$ "

*<proof>*

```

lemma is_msg_inv_rt [simp]:
  "ξ' ∈ is_rerr ξ  ⇒ rt ξ' = rt ξ"
  "ξ' ∈ is_rrep ξ  ⇒ rt ξ' = rt ξ"
  "ξ' ∈ is_rreq ξ  ⇒ rt ξ' = rt ξ"
  "ξ' ∈ is_pkt ξ   ⇒ rt ξ' = rt ξ"
  "ξ' ∈ is_newpkt ξ ⇒ rt ξ' = rt ξ"
  ⟨proof⟩

lemma is_msg_inv_rreqs [simp]:
  "ξ' ∈ is_rerr ξ  ⇒ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_rrep ξ  ⇒ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_rreq ξ  ⇒ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_pkt ξ   ⇒ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_newpkt ξ ⇒ rreqs ξ' = rreqs ξ"
  ⟨proof⟩

lemma is_msg_inv_store [simp]:
  "ξ' ∈ is_rerr ξ  ⇒ store ξ' = store ξ"
  "ξ' ∈ is_rrep ξ  ⇒ store ξ' = store ξ"
  "ξ' ∈ is_rreq ξ  ⇒ store ξ' = store ξ"
  "ξ' ∈ is_pkt ξ   ⇒ store ξ' = store ξ"
  "ξ' ∈ is_newpkt ξ ⇒ store ξ' = store ξ"
  ⟨proof⟩

lemma is_msg_inv_sip [simp]:
  "ξ' ∈ is_pkt ξ     ⇒ sip ξ' = sip ξ"
  "ξ' ∈ is_newpkt ξ ⇒ sip ξ' = sip ξ"
  ⟨proof⟩

```

### 4.3.3 The protocol process

```

datatype pseqp =
  PAadv
  | PNewPkt
  | PPkt
  | PRreq
  | PRrep
  | PRerr

fun nat_of_seqp :: "pseqp ⇒ nat"
where
  "nat_of_seqp PAadv = 1"
  | "nat_of_seqp PPkt = 2"
  | "nat_of_seqp PNewPkt = 3"
  | "nat_of_seqp PRreq = 4"
  | "nat_of_seqp PRrep = 5"
  | "nat_of_seqp PRerr = 6"

instantiation "pseqp" :: ord
begin
definition less_eq_seqp [iff]: "l1 ≤ l2 = (nat_of_seqp l1 ≤ nat_of_seqp l2)"
definition less_seqp [iff]: "l1 < l2 = (nat_of_seqp l1 < nat_of_seqp l2)"
instance ⟨proof⟩
end

abbreviation AODV
where
  "AODV ≡ λ_. [[clear_locals]] call(PAadv)"

abbreviation PKT
where
  "PKT args ≡

  [[ξ. let (data, dip, oip) = args ξ in

```

```

      (clear_locals ξ) (| data := data, dip := dip, oip := oip |)]
    call(PPkt)"
abbreviation NEWPKT
where
  "NEWPKT args ≡
    [[ξ. let (data, dip) = args ξ in
      (clear_locals ξ) (| data := data, dip := dip |)]
    call(PNewPkt)"

abbreviation RREQ
where
  "RREQ args ≡
    [[ξ. let (hops, rreqid, dip, dsn, dsk, oip, osn, sip, handled) = args ξ in
      (clear_locals ξ) (| hops := hops, rreqid := rreqid, dip := dip,
        dsn := dsn, dsk := dsk, oip := oip,
        osn := osn, sip := sip, handled := handled |)]
    call(PRreq)"

abbreviation RREP
where
  "RREP args ≡
    [[ξ. let (hops, dip, dsn, oip, sip) = args ξ in
      (clear_locals ξ) (| hops := hops, dip := dip, dsn := dsn,
        oip := oip, sip := sip |)]
    call(PRrep)"

abbreviation RERR
where
  "RERR args ≡
    [[ξ. let (dests, sip) = args ξ in
      (clear_locals ξ) (| dests := dests, sip := sip |)]
    call(PRerr)"

fun ΓAODV :: "(state, msg, pseqp, pseqp label) seqp_env"
where
  "ΓAODV PAodv = labelled PAodv (
    receive(λmsg' ξ. ξ (| msg := msg' |)).
    (
      ⟨is_newpkt⟩ NEWPKT(λξ. (data ξ, ip ξ))
    ⊕ ⟨is_pkt⟩ PKT(λξ. (data ξ, dip ξ, oip ξ))
    ⊕ ⟨is_rreq⟩
      [[ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, { }) |)]
      RREQ(λξ. (hops ξ, rreqid ξ, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, sip ξ, handled ξ))
    ⊕ ⟨is_rrep⟩
      [[ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, { }) |)]
      RREP(λξ. (hops ξ, dip ξ, dsn ξ, oip ξ, sip ξ))
    ⊕ ⟨is_rerr⟩
      [[ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, { }) |)]
      RERR(λξ. (dests ξ, sip ξ))
    )
  ⊕ ⟨λξ. { ξ(| dip := dip |) | dip. dip ∈ qD(store ξ) ∩ vD(rt ξ) }⟩
    [[ξ. ξ (| data := hd(σqueue(store ξ, dip ξ)) |)]
    unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, ip ξ)).
    [[ξ. ξ (| store := the (drop (dip ξ) (store ξ)) |)]
    AODV()
  ▷ [[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
    then Some (inc (sqn (rt ξ) rip)) else None) |)]
    [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)]
    [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |)]
    [[ξ. ξ (| pre := ⋃ { the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } |)]
    [[ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ { })
    then (dests ξ) rip else None) |)]
    groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)). AODV()
  ⊕ ⟨λξ. { ξ(| dip := dip |)
    | dip. dip ∈ qD(store ξ) - vD(rt ξ) ∧ the (σp-flag(store ξ, dip)) = req }⟩

```

```

[[ξ. ξ (| store := unsetRRF (store ξ) (dip ξ) )]]
[[ξ. ξ (| sn := inc (sn ξ) )]]
[[ξ. ξ (| rreqid := nrreqid (rreqs ξ) (ip ξ) )]]
[[ξ. ξ (| rreqs := rreqs ξ ∪ {(ip ξ, rreqid ξ)} )]]
broadcast(λξ. rreq(0, rreqid ξ, dip ξ, sqn (rt ξ) (dip ξ), sqnf (rt ξ) (dip ξ), ip ξ, sn ξ,
ip ξ, False)). AODV()"

```

```

| "ΓAODV PNewPkt = labelled PNewPkt (
  ⟨ξ. dip ξ = ip ξ⟩
  deliver(λξ. data ξ).AODV()
  ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
  [[ξ. ξ (| store := add (data ξ) (dip ξ) (store ξ) )]]
  AODV())"

```

```

| "ΓAODV PPkt = labelled PPkt (
  ⟨ξ. dip ξ = ip ξ⟩
  deliver(λξ. data ξ).AODV()
  ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
  (
    ⟨ξ. dip ξ ∈ vD (rt ξ)⟩
    unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, oip ξ)).AODV()
    ▷
    [[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
      then Some (inc (sqn (rt ξ) rip)) else None) )]]
    [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) )]]
    [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) )]]
    [[ξ. ξ (| pre := ∪ { the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } )]]
    [[ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ { })
      then (dests ξ) rip else None) )]]
    groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
  ⊕ ⟨ξ. dip ξ ∉ vD (rt ξ)⟩
  (
    ⟨ξ. dip ξ ∈ iD (rt ξ)⟩
    groupcast(λξ. the (precs (rt ξ) (dip ξ)), λξ. rerr([dip ξ ↦ sqn (rt ξ) (dip ξ)],
      ip ξ)).AODV()
    ⊕ ⟨ξ. dip ξ ∉ iD (rt ξ)⟩
    AODV()
  )
  )
  )"

```

```

| "ΓAODV PRreq = labelled PRreq (
  ⟨ξ. (oip ξ, rreqid ξ) ∈ rreqs ξ⟩
  AODV()
  ⊕ ⟨ξ. (oip ξ, rreqid ξ) ∉ rreqs ξ⟩
  [[ξ. ξ (| rt := update (rt ξ) (oip ξ) (osn ξ, kno, val, hops ξ + 1, sip ξ, { }) )]]
  [[ξ. ξ (| rreqs := rreqs ξ ∪ {(oip ξ, rreqid ξ)} )]]
  (
    ⟨ξ. handled ξ = False⟩
    (
      ⟨ξ. dip ξ = ip ξ⟩
      [[ξ. ξ (| sn := max (sn ξ) (dsn ξ) )]]
      unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(0, dip ξ, sn ξ, oip ξ, ip ξ)).
      broadcast(λξ. rreq(hops ξ + 1, rreqid ξ, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, ip ξ, True)).
      AODV()
    ▷
    [[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
      then Some (inc (sqn (rt ξ) rip)) else None) )]]
    [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) )]]
    [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) )]]
    [[ξ. ξ (| pre := ∪ { the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } )]]
    [[ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ { })
      then (dests ξ) rip else None) )]]
    groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
  ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
  )
  )"

```

```

(
  ⟨ξ. dip ξ ∈ vD (rt ξ) ∧ dsn ξ ≤ sqn (rt ξ) (dip ξ) ∧ sqnf (rt ξ) (dip ξ) = kno⟩
  [[ξ. ξ (| rt := the (addpreRT (rt ξ) (dip ξ) {sip ξ}) )]]
  [[ξ. ξ (| rt := the (addpreRT (rt ξ) (oip ξ) {the (nhop (rt ξ) (dip ξ))}) )]]
  unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(the (dhops (rt ξ) (dip ξ)), dip ξ,
    sqn (rt ξ) (dip ξ), oip ξ, ip ξ)).
    broadcast(λξ. rreq(hops ξ + 1, rreqid ξ, dip ξ, dsn ξ,
      dsk ξ, oip ξ, osn ξ, ip ξ, True)).
    AODV()
  ▷
  [[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
    then Some (inc (sqn (rt ξ) rip)) else None) )]]
  [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) )]]
  [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) )]]
  [[ξ. ξ (| pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } )]]
  [[ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ { })
    then (dests ξ) rip else None) )]]
  groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
  ⊕ ⟨ξ. dip ξ ∉ vD (rt ξ) ∨ sqn (rt ξ) (dip ξ) < dsn ξ ∨ sqnf (rt ξ) (dip ξ) = unk⟩
  broadcast(λξ. rreq(hops ξ + 1, rreqid ξ, dip ξ, max (sqn (rt ξ) (dip ξ)) (dsn ξ),
    dsk ξ, oip ξ, osn ξ, ip ξ, False)).
    AODV()
)
)
)
⊕ ⟨ξ. handled ξ = True⟩
broadcast(λξ. rreq(hops ξ + 1, rreqid ξ, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, ip ξ, True)).
AODV()
)"

```

```

| "ΓAODV PRrep = labelled PRrep (
  ⟨ξ. rt ξ ≠ update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, { })⟩
  (
    [[ξ. ξ (| rt := update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, { }) )]]
    (
      ⟨ξ. oip ξ = ip ξ⟩
      AODV()
      ⊕ ⟨ξ. oip ξ ≠ ip ξ⟩
      (
        ⟨ξ. oip ξ ∈ vD (rt ξ)⟩
        [[ξ. ξ (| rt := the (addpreRT (rt ξ) (dip ξ) {the (nhop (rt ξ) (oip ξ))}) )]]
        [[ξ. ξ (| rt := the (addpreRT (rt ξ) (the (nhop (rt ξ) (dip ξ)))
          {the (nhop (rt ξ) (oip ξ))}) )]]
        unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(hops ξ + 1, dip ξ, dsn ξ, oip ξ, ip ξ)).
          AODV()
        ▷
        [[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
          then Some (inc (sqn (rt ξ) rip)) else None) )]]
        [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) )]]
        [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) )]]
        [[ξ. ξ (| pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } )]]
        [[ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ { })
          then (dests ξ) rip else None) )]]
        groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
        ⊕ ⟨ξ. oip ξ ∉ vD (rt ξ)⟩
        AODV()
      )
    )
  )
  ⊕ ⟨ξ. rt ξ = update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, { })⟩
  AODV()
)"

```

```

| "ΓAODV PRerr = labelled PRerr (
  [[ξ. ξ (| dests := (λrip. case (dests ξ) rip of None ⇒ None

```

```

    | Some rsn  $\Rightarrow$  if rip  $\in$  vD (rt  $\xi$ )  $\wedge$  the (nhop (rt  $\xi$ ) rip) = sip  $\xi$ 
       $\wedge$  sqn (rt  $\xi$ ) rip < rsn then Some rsn else None  $\rangle\rangle$ 
  [|  $\xi$ .  $\xi$  (| rt := invalidate (rt  $\xi$ ) (dests  $\xi$ )  $\rangle\rangle$ 
  [|  $\xi$ .  $\xi$  (| store := setRRF (store  $\xi$ ) (dests  $\xi$ )  $\rangle\rangle$ 
  [|  $\xi$ .  $\xi$  (| pre :=  $\bigcup$ { the (precs (rt  $\xi$ ) rip) | rip. rip  $\in$  dom (dests  $\xi$ ) }  $\rangle\rangle$ 
  [|  $\xi$ .  $\xi$  (| dests := ( $\lambda$ rip. if ((dests  $\xi$ ) rip  $\neq$  None  $\wedge$  the (precs (rt  $\xi$ ) rip)  $\neq$  {})
      then (dests  $\xi$ ) rip else None)  $\rangle\rangle$ 
  groupcast( $\lambda$  $\xi$ . pre  $\xi$ ,  $\lambda$  $\xi$ . rerr(dests  $\xi$ , ip  $\xi$ )). AODV())"

declare  $\Gamma_{AODV}$ .simps [simp del, code del]
lemmas  $\Gamma_{AODV}$ .simps [simp, code] =  $\Gamma_{AODV}$ .simps [simplified]

fun  $\Gamma_{AODV}$ _skeleton
where
  " $\Gamma_{AODV}$ _skeleton PAodv = seqp_skeleton ( $\Gamma_{AODV}$  PAodv)"
  | " $\Gamma_{AODV}$ _skeleton PNewPkt = seqp_skeleton ( $\Gamma_{AODV}$  PNewPkt)"
  | " $\Gamma_{AODV}$ _skeleton PPkt = seqp_skeleton ( $\Gamma_{AODV}$  PPkt)"
  | " $\Gamma_{AODV}$ _skeleton PRreq = seqp_skeleton ( $\Gamma_{AODV}$  PRreq)"
  | " $\Gamma_{AODV}$ _skeleton PRrep = seqp_skeleton ( $\Gamma_{AODV}$  PRrep)"
  | " $\Gamma_{AODV}$ _skeleton PRerr = seqp_skeleton ( $\Gamma_{AODV}$  PRerr)"

lemma  $\Gamma_{AODV}$ _skeleton_wf [simp]:
  "wellformed  $\Gamma_{AODV}$ _skeleton"
  <proof>

declare  $\Gamma_{AODV}$ _skeleton.simps [simp del, code del]
lemmas  $\Gamma_{AODV}$ _skeleton.simps [simp, code]
  =  $\Gamma_{AODV}$ _skeleton.simps [simplified  $\Gamma_{AODV}$ .simps seqp_skeleton.simps]

lemma aodv_proc_cases [dest]:
  fixes p pn
  shows "p  $\in$  cterms1 ( $\Gamma_{AODV}$  pn)  $\implies$ 
    (p  $\in$  cterms1 ( $\Gamma_{AODV}$  PAodv)  $\vee$ 
    p  $\in$  cterms1 ( $\Gamma_{AODV}$  PNewPkt)  $\vee$ 
    p  $\in$  cterms1 ( $\Gamma_{AODV}$  PPkt)  $\vee$ 
    p  $\in$  cterms1 ( $\Gamma_{AODV}$  PRreq)  $\vee$ 
    p  $\in$  cterms1 ( $\Gamma_{AODV}$  PRrep)  $\vee$ 
    p  $\in$  cterms1 ( $\Gamma_{AODV}$  PRerr))"
  <proof>

definition  $\sigma_{AODV}$  :: "ip  $\Rightarrow$  (state  $\times$  (state, msg, pseqp, pseqp label) seqp) set"
where " $\sigma_{AODV}$  i  $\equiv$  {(aodv_init i,  $\Gamma_{AODV}$  PAodv)}"

abbreviation paodv
  :: "ip  $\Rightarrow$  (state  $\times$  (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
  " $\text{paodv } i \equiv (| \text{init} = \sigma_{AODV} i, \text{trans} = \text{seqp\_sos } \Gamma_{AODV} |)$ "

lemma aodv_trans: "trans (paodv i) = seqp_sos  $\Gamma_{AODV}$ "
  <proof>

lemma aodv_control_within [simp]: "control_within  $\Gamma_{AODV}$  (init (paodv i))"
  <proof>

lemma aodv_wf [simp]:
  "wellformed  $\Gamma_{AODV}$ "
  <proof>

lemmas aodv_labels_not_empty [simp] = labels_not_empty [OF aodv_wf]

lemma aodv_ex_label [intro]: " $\exists$ l. l $\in$ labels  $\Gamma_{AODV}$  p"
  <proof>

lemma aodv_ex_labelE [elim]:

```



```

assumes " $\forall l \in \text{labels } \Gamma_{AODV} p. P \ l \ p$ "
and " $\exists p \ l. P \ l \ p \implies Q$ "
shows " $Q$ "
<proof>

```

```

lemma aodv_simple_labels [simp]: "simple_labels  $\Gamma_{AODV}$ "
<proof>

```

```

lemma  $\sigma_{AODV\_labels}$  [simp]: " $(\xi, p) \in \sigma_{AODV} i \implies \text{labels } \Gamma_{AODV} p = \{PAodv-:0\}$ "
<proof>

```

```

lemma aodv_init_kD_empty [simp]:
" $(\xi, p) \in \sigma_{AODV} i \implies kD \ (rt \ \xi) = \{\}$ "
<proof>

```

```

lemma aodv_init_sip_not_ip [simp]: " $\neg(\text{sip} \ (\text{aodv\_init} \ i) = i)$ " <proof>

```

```

lemma aodv_init_sip_not_ip' [simp]:
assumes " $(\xi, p) \in \sigma_{AODV} i$ "
shows " $\text{sip} \ \xi \neq ip \ \xi$ "
<proof>

```

```

lemma aodv_init_sip_not_i [simp]:
assumes " $(\xi, p) \in \sigma_{AODV} i$ "
shows " $\text{sip} \ \xi \neq i$ "
<proof>

```

```

lemma clear_locals_sip_not_ip':
assumes " $ip \ \xi = i$ "
shows " $\neg(\text{sip} \ (\text{clear\_locals} \ \xi) = i)$ "
<proof>

```

Stop the simplifier from descending into process terms.

```

declare seqp_congs [cong]

```

Configure the main invariant tactic for AODV.

```

declare
 $\Gamma_{AODV\_simps}$  [cterms_env]
aodv_proc_cases [ctermst1_cases]
seq_invariant_ctermstI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
ctermst_intros]
seq_step_invariant_ctermstI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
ctermst_intros]

```

end

## 4.4 Invariant assumptions and properties

```

theory D_Aodv_Predicates
imports D_Aodv
begin

```

Definitions for expression assumptions on incoming messages and properties of outgoing messages.

```

abbreviation not_Pkt :: "msg  $\Rightarrow$  bool"
where "not_Pkt m  $\equiv$  case m of Pkt _ _ _  $\Rightarrow$  False | _  $\Rightarrow$  True"

```

```

definition msg_sender :: "msg  $\Rightarrow$  ip"
where "msg_sender m  $\equiv$  case m of Rreq _ _ _ _ _ _ ipc _  $\Rightarrow$  ipc
| Rrep _ _ _ _ ipc  $\Rightarrow$  ipc
| Rerr _ ipc  $\Rightarrow$  ipc
| Pkt _ _ ipc  $\Rightarrow$  ipc"

```

```

lemma msg_sender_simps [simp]:

```

```

"∧hops rreqid dip dsn dsk oip osn sip handled.
    msg_sender (Rreq hops rreqid dip dsn dsk oip osn sip handled) = sip"
"∧hops dip dsn oip sip. msg_sender (Rrep hops dip dsn oip sip) = sip"
"∧dests sip.          msg_sender (Rerr dests sip) = sip"
"∧d dip sip.         msg_sender (Pkt d dip sip) = sip"
⟨proof⟩

```

**definition** `msg_zhops` :: "msg ⇒ bool"

where "msg\_zhops m ≡ case m of

```

    Rreq hopsc _ dipc _ _ oipc _ sipc _ ⇒ hopsc = 0 → oipc = sipc
  | Rrep hopsc dipc _ _ sipc ⇒ hopsc = 0 → dipc = sipc
  | _ ⇒ True"

```

**lemma** `msg_zhops_simps` [simp]:

```

"∧hops rreqid dip dsn dsk oip osn sip handled.
    msg_zhops (Rreq hops rreqid dip dsn dsk oip osn sip handled) = (hops = 0 → oip = sip)"
"∧hops dip dsn oip sip. msg_zhops (Rrep hops dip dsn oip sip) = (hops = 0 → dip = sip)"
"∧dests sip.          msg_zhops (Rerr dests sip) = True"
"∧d dip.             msg_zhops (Newpkt d dip) = True"
"∧d dip sip.        msg_zhops (Pkt d dip sip) = True"
⟨proof⟩

```

**definition** `rreq_rrep_sn` :: "msg ⇒ bool"

where "rreq\_rrep\_sn m ≡ case m of Rreq \_ \_ \_ \_ \_ osnc \_ \_ ⇒ osnc ≥ 1

```

  | Rrep _ _ dsnc _ _ ⇒ dsnc ≥ 1
  | _ ⇒ True"

```

**lemma** `rreq_rrep_sn_simps` [simp]:

```

"∧hops rreqid dip dsn dsk oip osn sip handled.
    rreq_rrep_sn (Rreq hops rreqid dip dsn dsk oip osn sip handled) = (osn ≥ 1)"
"∧hops dip dsn oip sip. rreq_rrep_sn (Rrep hops dip dsn oip sip) = (dsn ≥ 1)"
"∧dests sip.          rreq_rrep_sn (Rerr dests sip) = True"
"∧d dip.             rreq_rrep_sn (Newpkt d dip) = True"
"∧d dip sip.        rreq_rrep_sn (Pkt d dip sip) = True"
⟨proof⟩

```

**definition** `rreq_rrep_fresh` :: "rt ⇒ msg ⇒ bool"

where "rreq\_rrep\_fresh crt m ≡ case m of Rreq hopsc \_ \_ \_ oipc osnc ipcc \_ ⇒ (ipcc ≠ oipc →

```

    oipc ∈ kD(crt) ∧ (sqn crt oipc > osnc
      ∨ (sqn crt oipc = osnc
        ∧ the (dhops crt oipc) ≤ hopsc
        ∧ the (flag crt oipc) = val)))
  | Rrep hopsc dipc dsnc _ ipcc ⇒ (ipcc ≠ dipc →
    dipc ∈ kD(crt)
    ∧ sqn crt dipc = dsnc
    ∧ the (dhops crt dipc) = hopsc
    ∧ the (flag crt dipc) = val)
  | _ ⇒ True"

```

**lemma** `rreq_rrep_fresh_simps` [simp]:

```

"∧hops rreqid dip dsn dsk oip osn sip handled.
    rreq_rrep_fresh crt (Rreq hops rreqid dip dsn dsk oip osn sip handled) =
      (sip ≠ oip → oip ∈ kD(crt)
        ∧ (sqn crt oip > osn
          ∨ (sqn crt oip = osn
            ∧ the (dhops crt oip) ≤ hops
            ∧ the (flag crt oip) = val))))"
"∧hops dip dsn oip sip. rreq_rrep_fresh crt (Rrep hops dip dsn oip sip) =
      (sip ≠ dip → dip ∈ kD(crt)
        ∧ sqn crt dip = dsnc
        ∧ the (dhops crt dip) = hops
        ∧ the (flag crt dip) = val)"
"∧dests sip.          rreq_rrep_fresh crt (Rerr dests sip) = True"
"∧d dip.             rreq_rrep_fresh crt (Newpkt d dip) = True"

```

```
" $\wedge$ d dip sip.          rreq_rrep_fresh crt (Pkt d dip sip) = True"
<proof>
```

```
definition rerr_invalid :: "rt  $\Rightarrow$  msg  $\Rightarrow$  bool"
where "rerr_invalid crt m  $\equiv$  case m of Rerr destsc _  $\Rightarrow$  ( $\forall$  ripc $\in$ dom(destsc).
                                     (ripc $\in$ iD(crt)  $\wedge$  the (destsc ripc) = sqn crt ripc))
      | _  $\Rightarrow$  True"
```

```
lemma rerr_invalid [simp]:
  " $\wedge$ hops rreqid dip dsn dsk oip osn sip handled.
      rerr_invalid crt (Rreq hops rreqid dip dsn dsk oip osn sip handled) = True"
  " $\wedge$ hops dip dsn oip sip. rerr_invalid crt (Rrep hops dip dsn oip sip) = True"
  " $\wedge$ dests sip.          rerr_invalid crt (Rerr dests sip) = ( $\forall$  rip $\in$ dom(dests).
                                     rip $\in$ iD(crt)  $\wedge$  the (dests rip) = sqn crt rip)"
  " $\wedge$ d dip.            rerr_invalid crt (Newpkt d dip) = True"
  " $\wedge$ dip sip.          rerr_invalid crt (Pkt d dip sip) = True"
<proof>
```

```
definition
  initmissing :: "(nat  $\Rightarrow$  state option)  $\times$  'a  $\Rightarrow$  (nat  $\Rightarrow$  state)  $\times$  'a"
where
```

```
"initmissing  $\sigma$  = ( $\lambda$ i. case (fst  $\sigma$ ) i of None  $\Rightarrow$  aadv_init i | Some s  $\Rightarrow$  s, snd  $\sigma$ )"
```

```
lemma not_in_net_ips_fst_init_missing [simp]:
  assumes "i  $\notin$  net_ips  $\sigma$ "
  shows "fst (initmissing (netgmap fst  $\sigma$ )) i = aadv_init i"
<proof>
```

```
lemma fst_initmissing_netgmap_pair_fst [simp]:
  "fst (initmissing (netgmap ( $\lambda$ (p, q). (fst (id p), snd (id p), q)) s))
    = fst (initmissing (netgmap fst s))"
<proof>
```

We introduce a streamlined alternative to `initmissing` with `netgmap` to simplify invariant statements and thus facilitate their comprehension and presentation.

```
lemma fst_initmissing_netgmap_default_aadv_init_netlift:
  "fst (initmissing (netgmap fst s)) = default aadv_init (netlift fst s)"
<proof>
```

```
definition
  netglobal :: "((nat  $\Rightarrow$  state)  $\Rightarrow$  bool)  $\Rightarrow$  ((state  $\times$  'b)  $\times$  'c) net_state  $\Rightarrow$  bool"
where
  "netglobal P  $\equiv$  ( $\lambda$ s. P (default aadv_init (netlift fst s)))"
```

**end**

## 4.5 Quality relations between routes

```
theory D_Fresher
imports D_Aadv_Data
begin
```

### 4.5.1 Net sequence numbers

#### On individual routes

```
definition
  nsqnr :: "r  $\Rightarrow$  sqn"
where
  "nsqnr r  $\equiv$  if  $\pi_4$ (r) = val  $\vee$   $\pi_2$ (r) = 0 then  $\pi_2$ (r) else ( $\pi_2$ (r) - 1)"
```

```
lemma nsqn_def':
  "nsqnr r = (if  $\pi_4$ (r) = inv then  $\pi_2$ (r) - 1 else  $\pi_2$ (r))"
<proof>
```

```

lemma nsqnr_zero [simp]:
  "∧ dsn dsk flag hops nhip pre. nsqnr (0, dsk, flag, hops, nhip, pre) = 0"
  ⟨proof⟩

lemma nsqnr_val [simp]:
  "∧ dsn dsk hops nhip pre. nsqnr (dsn, dsk, val, hops, nhip, pre) = dsn"
  ⟨proof⟩

lemma nsqnr_inv [simp]:
  "∧ dsn dsk hops nhip pre. nsqnr (dsn, dsk, inv, hops, nhip, pre) = dsn - 1"
  ⟨proof⟩

lemma nsqnr_lte_dsn [simp]:
  "∧ dsn dsk flag hops nhip pre. nsqnr (dsn, dsk, flag, hops, nhip, pre) ≤ dsn"
  ⟨proof⟩

```

## On routes in routing tables

### definition

```
nsqn :: "rt ⇒ ip ⇒ sqn"
```

### where

```
"nsqn ≡ λrt dip. case σroute(rt, dip) of None ⇒ 0 | Some r ⇒ nsqnr(r)"
```

### lemma nsqn\_sqn\_def:

```
"∧rt dip. nsqn rt dip = (if flag rt dip = Some val ∨ sqn rt dip = 0
  then sqn rt dip else sqn rt dip - 1)"
```

⟨proof⟩

### lemma not\_in\_kD\_nsqn [simp]:

```
assumes "dip ∉ kD(rt)"
```

```
shows "nsqn rt dip = 0"
```

⟨proof⟩

### lemma kD\_nsqn:

```
assumes "dip ∈ kD(rt)"
```

```
shows "nsqn rt dip = nsqnr(the (σroute(rt, dip)))"
```

⟨proof⟩

### lemma nsqn<sub>r</sub>\_r\_flag\_pred [simp, intro]:

```
fixes dsn dsk flag hops nhip pre
```

```
assumes "P (nsqnr (dsn, dsk, val, hops, nhip, pre))"
```

```
and "P (nsqnr (dsn, dsk, inv, hops, nhip, pre))"
```

```
shows "P (nsqnr (dsn, dsk, flag, hops, nhip, pre))"
```

⟨proof⟩

### lemma nsqn<sub>r</sub>\_addpreRT\_inv [simp]:

```
"∧rt dip npre dip'. dip ∈ kD(rt) ⇒
```

```
nsqnr (the (the (addpreRT rt dip npre) dip')) = nsqnr (the (rt dip'))"
```

⟨proof⟩

### lemma sqn\_nsqn:

```
"∧rt dip. sqn rt dip - 1 ≤ nsqn rt dip"
```

⟨proof⟩

### lemma nsqn\_sqn: "nsqn rt dip ≤ sqn rt dip"

⟨proof⟩

### lemma val\_nsqn\_sqn [elim, simp]:

```
assumes "ip ∈ kD(rt)"
```

```
and "the (flag rt ip) = val"
```

```
shows "nsqn rt ip = sqn rt ip"
```

⟨proof⟩

```

lemma vD_nsqn_sqn [elim, simp]:
  assumes "ip ∈ vD(rt)"
    shows "nsqn rt ip = sqn rt ip"
  ⟨proof⟩

```

```

lemma inv_nsqn_sqn [elim, simp]:
  assumes "ip ∈ kD(rt)"
    and "the (flag rt ip) = inv"
    shows "nsqn rt ip = sqn rt ip - 1"
  ⟨proof⟩

```

```

lemma iD_nsqn_sqn [elim, simp]:
  assumes "ip ∈ iD(rt)"
    shows "nsqn rt ip = sqn rt ip - 1"
  ⟨proof⟩

```

```

lemma nsqn_update_changed_kno_val [simp]: "∧rt ip dsn dsk hops nhip.
  rt ≠ update rt ip (dsn, kno, val, hops, nhip, { })
  ⇒ nsqn (update rt ip (dsn, kno, val, hops, nhip, { })) ip = dsn"
  ⟨proof⟩

```

```

lemma nsqn_addpreRT_inv [simp]:
  "∧rt dip npre dip'. dip ∈ kD(rt) ⇒
  nsqn (the (addpreRT rt dip npre)) dip' = nsqn rt dip'"
  ⟨proof⟩

```

```

lemma nsqn_update_other [simp]:
  fixes dsn dsk flag hops dip nhip pre rt ip
  assumes "dip ≠ ip"
    shows "nsqn (update rt ip (dsn, dsk, flag, hops, nhip, pre)) dip = nsqn rt dip"
  ⟨proof⟩

```

```

lemma nsqn_invalidate_eq:
  assumes "dip ∈ kD(rt)"
    and "dests dip = Some rsn"
    shows "nsqn (invalidate rt dests) dip = rsn - 1"
  ⟨proof⟩

```

```

lemma nsqn_invalidate_other [simp]:
  assumes "dip ∈ kD(rt)"
    and "dip ∉ dom dests"
    shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
  ⟨proof⟩

```

## 4.5.2 Comparing routes

**definition**

```

fresher :: "r ⇒ r ⇒ bool" ("(_/ ⊆ _)" [51, 51] 50)

```

where

```

"fresher r r' ≡ ((nsqnr r < nsqnr r') ∨ (nsqnr r = nsqnr r' ∧ π5(r) ≥ π5(r')))"

```

```

lemma fresherI1 [intro]:
  assumes "nsqnr r < nsqnr r'"
    shows "r ⊆ r'"
  ⟨proof⟩

```

```

lemma fresherI2 [intro]:
  assumes "nsqnr r = nsqnr r'"
    and "π5(r) ≥ π5(r')"
    shows "r ⊆ r'"
  ⟨proof⟩

```

```

lemma fresherI [intro]:
  assumes "(nsqnr r < nsqnr r') ∨ (nsqnr r = nsqnr r' ∧ π5(r) ≥ π5(r'))"

```

shows "r  $\sqsubseteq$  r'"  
 ⟨proof⟩

lemma fresherE [elim]:  
 assumes "r  $\sqsubseteq$  r'"  
 and "nsqn<sub>r</sub> r < nsqn<sub>r</sub> r'  $\implies$  P r r'"  
 and "nsqn<sub>r</sub> r = nsqn<sub>r</sub> r'  $\wedge$   $\pi_5$ (r)  $\geq$   $\pi_5$ (r')  $\implies$  P r r'"  
 shows "P r r'"  
 ⟨proof⟩

lemma fresher\_refl [simp]: "r  $\sqsubseteq$  r"  
 ⟨proof⟩

lemma fresher\_trans [elim, trans]:  
 "[ x  $\sqsubseteq$  y; y  $\sqsubseteq$  z ]  $\implies$  x  $\sqsubseteq$  z"  
 ⟨proof⟩

lemma not\_fresher\_trans [elim, trans]:  
 "[  $\neg$ (x  $\sqsubseteq$  y);  $\neg$ (z  $\sqsubseteq$  x) ]  $\implies$   $\neg$ (z  $\sqsubseteq$  y)"  
 ⟨proof⟩

lemma fresher\_dsn\_flag\_hops\_const [simp]:  
 fixes dsn dsk' flag hops nhip' pre pre'  
 shows "(dsn, dsk, flag, hops, nhip, pre)  $\sqsubseteq$  (dsn, dsk', flag, hops, nhip', pre)"  
 ⟨proof⟩

lemma addpre\_fresher [simp]: " $\wedge$ r npre. r  $\sqsubseteq$  (addpre r npre)"  
 ⟨proof⟩

### 4.5.3 Comparing routing tables

definition

rt\_fresher :: "ip  $\Rightarrow$  rt  $\Rightarrow$  rt  $\Rightarrow$  bool"

where

"rt\_fresher  $\equiv$   $\lambda$ dip rt rt'. (the ( $\sigma_{route}$ (rt, dip)))  $\sqsubseteq$  (the ( $\sigma_{route}$ (rt', dip)))"

abbreviation

rt\_fresher\_syn :: "rt  $\Rightarrow$  ip  $\Rightarrow$  rt  $\Rightarrow$  bool" ("(\_/  $\sqsubseteq$  \_)" [51, 999, 51] 50)

where

"rt1  $\sqsubseteq_i$  rt2  $\equiv$  rt\_fresher i rt1 rt2"

lemma rt\_fresher\_def':

"(rt1  $\sqsubseteq_i$  rt2) = (nsqn<sub>r</sub> (the (rt1 i)) < nsqn<sub>r</sub> (the (rt2 i))  $\vee$   
 nsqn<sub>r</sub> (the (rt1 i)) = nsqn<sub>r</sub> (the (rt2 i))  $\wedge$   $\pi_5$  (the (rt2 i))  $\leq$   $\pi_5$  (the (rt1 i)))"

⟨proof⟩

lemma single\_rt\_fresher [intro]:

assumes "the (rt1 ip)  $\sqsubseteq$  the (rt2 ip)"

shows "rt1  $\sqsubseteq_{ip}$  rt2"

⟨proof⟩

lemma rt\_fresher\_single [intro]:

assumes "rt1  $\sqsubseteq_{ip}$  rt2"

shows "the (rt1 ip)  $\sqsubseteq$  the (rt2 ip)"

⟨proof⟩

lemma rt\_fresher\_def2:

assumes "dip  $\in$  kD(rt1)"

and "dip  $\in$  kD(rt2)"

shows "(rt1  $\sqsubseteq_{dip}$  rt2) = (nsqn rt1 dip < nsqn rt2 dip  
 $\vee$  (nsqn rt1 dip = nsqn rt2 dip  
 $\wedge$  the (dhops rt1 dip)  $\geq$  the (dhops rt2 dip)))"

⟨proof⟩

```

lemma rt_fresherI1 [intro]:
  assumes "dip ∈ kD(rt1)"
    and "dip ∈ kD(rt2)"
    and "nsqn rt1 dip < nsqn rt2 dip"
  shows "rt1 ⊆dip rt2"
  ⟨proof⟩

lemma rt_fresherI2 [intro]:
  assumes "dip ∈ kD(rt1)"
    and "dip ∈ kD(rt2)"
    and "nsqn rt1 dip = nsqn rt2 dip"
    and "the (dhops rt1 dip) ≥ the (dhops rt2 dip)"
  shows "rt1 ⊆dip rt2"
  ⟨proof⟩

lemma rt_fresherE [elim]:
  assumes "rt1 ⊆dip rt2"
    and "dip ∈ kD(rt1)"
    and "dip ∈ kD(rt2)"
    and "[| nsqn rt1 dip < nsqn rt2 dip |] ⇒ P rt1 rt2 dip"
    and "[| nsqn rt1 dip = nsqn rt2 dip;
      the (dhops rt1 dip) ≥ the (dhops rt2 dip) |] ⇒ P rt1 rt2 dip"
  shows "P rt1 rt2 dip"
  ⟨proof⟩

lemma rt_fresher_refl [simp]: "rt ⊆dip rt"
  ⟨proof⟩

lemma rt_fresher_trans [elim, trans]:
  assumes "rt1 ⊆dip rt2"
    and "rt2 ⊆dip rt3"
  shows "rt1 ⊆dip rt3"
  ⟨proof⟩

lemma rt_fresher_if_Some [intro!]:
  assumes "the (rt dip) ⊆ r"
  shows "rt ⊆dip (λip. if ip = dip then Some r else rt ip)"
  ⟨proof⟩

definition rt_fresh_as :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_fresh_as ≡ λdip rt1 rt2. (rt1 ⊆dip rt2) ∧ (rt2 ⊆dip rt1)"

abbreviation
  rt_fresh_as_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" ("(_/ ≈ip _)" [51, 999, 51] 50)
where
  "rt1 ≈ip rt2 ≡ rt_fresh_as ip rt1 rt2"

lemma rt_fresh_as_refl [simp]: "∧rt dip. rt ≈dip rt"
  ⟨proof⟩

lemma rt_fresh_as_trans [simp, intro, trans]:
  "∧rt1 rt2 rt3 dip. [| rt1 ≈dip rt2; rt2 ≈dip rt3 |] ⇒ rt1 ≈dip rt3"
  ⟨proof⟩

lemma rt_fresh_asI [intro!]:
  assumes "rt1 ⊆dip rt2"
    and "rt2 ⊆dip rt1"
  shows "rt1 ≈dip rt2"
  ⟨proof⟩

lemma rt_fresh_as_fresherI [intro]:
  assumes "dip ∈ kD(rt1)"
    and "dip ∈ kD(rt2)"

```

```

    and "the (rt1 dip)  $\sqsubseteq$  the (rt2 dip)"
    and "the (rt2 dip)  $\sqsubseteq$  the (rt1 dip)"
  shows "rt1  $\approx_{\text{dip}}$  rt2"
<proof>

lemma nsqn_rt_fresh_asI:
  assumes "dip  $\in$  kD(rt)"
    and "dip  $\in$  kD(rt'"
    and "nsqn rt dip = nsqn rt' dip"
    and " $\pi_5(\text{the } (rt \text{ dip})) = \pi_5(\text{the } (rt' \text{ dip}))$ "
  shows "rt  $\approx_{\text{dip}}$  rt'"
<proof>

lemma rt_fresh_asE [elim]:
  assumes "rt1  $\approx_{\text{dip}}$  rt2"
    and "[[ rt1  $\sqsubseteq_{\text{dip}}$  rt2; rt2  $\sqsubseteq_{\text{dip}}$  rt1 ]  $\implies$  P rt1 rt2 dip]"
  shows "P rt1 rt2 dip"
<proof>

lemma rt_fresh_asD1 [dest]:
  assumes "rt1  $\approx_{\text{dip}}$  rt2"
  shows "rt1  $\sqsubseteq_{\text{dip}}$  rt2"
<proof>

lemma rt_fresh_asD2 [dest]:
  assumes "rt1  $\approx_{\text{dip}}$  rt2"
  shows "rt2  $\sqsubseteq_{\text{dip}}$  rt1"
<proof>

lemma rt_fresh_as_sym:
  assumes "rt1  $\approx_{\text{dip}}$  rt2"
  shows "rt2  $\approx_{\text{dip}}$  rt1"
<proof>

lemma not_rt_fresh_asI1 [intro]:
  assumes " $\neg$  (rt1  $\sqsubseteq_{\text{dip}}$  rt2)"
  shows " $\neg$  (rt1  $\approx_{\text{dip}}$  rt2)"
<proof>

lemma not_rt_fresh_asI2 [intro]:
  assumes " $\neg$  (rt2  $\sqsubseteq_{\text{dip}}$  rt1)"
  shows " $\neg$  (rt1  $\approx_{\text{dip}}$  rt2)"
<proof>

lemma not_single_rt_fresher [elim]:
  assumes " $\neg$ (the (rt1 ip)  $\sqsubseteq$  the (rt2 ip))"
  shows " $\neg$ (rt1  $\sqsubseteq_{\text{ip}}$  rt2)"
<proof>

lemmas not_single_rt_fresh_asI1 [intro] = not_rt_fresh_asI1 [OF not_single_rt_fresher]
lemmas not_single_rt_fresh_asI2 [intro] = not_rt_fresh_asI2 [OF not_single_rt_fresher]

lemma not_rt_fresher_single [elim]:
  assumes " $\neg$ (rt1  $\sqsubseteq_{\text{ip}}$  rt2)"
  shows " $\neg$ (the (rt1 ip)  $\sqsubseteq$  the (rt2 ip))"
<proof>

lemma rt_fresh_as_nsqnr:
  assumes "dip  $\in$  kD(rt1)"
    and "dip  $\in$  kD(rt2)"
    and "rt1  $\approx_{\text{dip}}$  rt2"
  shows "nsqnr (the (rt2 dip)) = nsqnr (the (rt1 dip))"
<proof>

```



```

lemma rt_fresher_mapupd [intro!]:
  assumes "dip ∈ kD(rt)"
    and "the (rt dip) ⊆ r"
  shows "rt ⊆dip rt(dip ↦ r)"
  ⟨proof⟩

```

```

lemma rt_fresher_map_update_other [intro!]:
  assumes "dip ∈ kD(rt)"
    and "dip ≠ ip"
  shows "rt ⊆dip rt(ip ↦ r)"
  ⟨proof⟩

```

```

lemma rt_fresher_update_other [simp]:
  assumes inkD: "dip ∈ kD(rt)"
    and "dip ≠ ip"
  shows "rt ⊆dip update rt ip r"
  ⟨proof⟩

```

```

theorem rt_fresher_update [simp]:
  assumes "dip ∈ kD(rt)"
    and "the (dhops rt dip) ≥ 1"
    and "update_arg_wf r"
  shows "rt ⊆dip update rt ip r"
  ⟨proof⟩

```

```

theorem rt_fresher_invalidate [simp]:
  assumes "dip ∈ kD(rt)"
    and indests: "∀ rip ∈ dom(dests). rip ∈ vD(rt) ∧ sqn rt rip < the (dests rip)"
  shows "rt ⊆dip invalidate rt dests"
  ⟨proof⟩

```

```

lemma nsqnr_invalidate [simp]:
  assumes "dip ∈ kD(rt)"
    and "dip ∈ dom(dests)"
  shows "nsqnr (the (invalidate rt dests dip)) = the (dests dip) - 1"
  ⟨proof⟩

```

```

lemma rt_fresh_as_inc_invalidate [simp]:
  assumes "dip ∈ kD(rt)"
    and "∀ rip ∈ dom(dests). rip ∈ vD(rt) ∧ the (dests rip) = inc (sqn rt rip)"
  shows "rt ≈dip invalidate rt dests"
  ⟨proof⟩

```

```

lemmas rt_fresher_inc_invalidate [simp] = rt_fresh_as_inc_invalidate [THEN rt_fresh_asD1]

```

```

lemma rt_fresh_as_addpreRT [simp]:
  assumes "ip ∈ kD(rt)"
  shows "rt ≈dip the (addpreRT rt ip npre)"
  ⟨proof⟩

```

```

lemmas rt_fresher_addpreRT [simp] = rt_fresh_as_addpreRT [THEN rt_fresh_asD1]

```

#### 4.5.4 Strictly comparing routing tables

```

definition rt_strictly_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"

```

where

```

"rt_strictly_fresher ≡ λdip rt1 rt2. (rt1 ⊆dip rt2) ∧ ¬(rt1 ≈dip rt2)"

```

abbreviation

```

rt_strictly_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" ("(_/ ⊆ _)" [51, 999, 51] 50)

```

where

```

"rt1 ⊆i rt2 ≡ rt_strictly_fresher i rt1 rt2"

```

```

lemma rt_strictly_fresher_def'':

```

```

"rt1  $\sqsubseteq_i$  rt2 = ((rt1  $\sqsubseteq_i$  rt2)  $\wedge$   $\neg$ (rt2  $\sqsubseteq_i$  rt1))"
⟨proof⟩

lemma rt_strictly_fresherI' [intro]:
  assumes "rt1  $\sqsubseteq_i$  rt2"
    and " $\neg$ (rt2  $\sqsubseteq_i$  rt1)"
  shows "rt1  $\sqsubseteq_i$  rt2"
⟨proof⟩

lemma rt_strictly_fresherE' [elim]:
  assumes "rt1  $\sqsubseteq_i$  rt2"
    and "[[ rt1  $\sqsubseteq_i$  rt2;  $\neg$ (rt2  $\sqsubseteq_i$  rt1) ]  $\implies$  P rt1 rt2 i]"
  shows "P rt1 rt2 i"
⟨proof⟩

lemma rt_strictly_fresherI [intro]:
  assumes "rt1  $\sqsubseteq_i$  rt2"
    and " $\neg$ (rt1  $\approx_i$  rt2)"
  shows "rt1  $\sqsubseteq_i$  rt2"
⟨proof⟩

lemmas rt_strictly_fresher_singleI [elim] = rt_strictly_fresherI [OF single_rt_fresher]

lemma rt_strictly_fresherE [elim]:
  assumes "rt1  $\sqsubseteq_i$  rt2"
    and "[[ rt1  $\sqsubseteq_i$  rt2;  $\neg$ (rt1  $\approx_i$  rt2) ]  $\implies$  P rt1 rt2 i]"
  shows "P rt1 rt2 i"
⟨proof⟩

lemma rt_strictly_fresher_def':
  "rt1  $\sqsubseteq_i$  rt2 =
  (nsqnr (the (rt1 i)) < nsqnr (the (rt2 i))
   $\vee$  (nsqnr (the (rt1 i)) = nsqnr (the (rt2 i))  $\wedge$   $\pi_5$ (the (rt1 i)) >  $\pi_5$ (the (rt2 i))))"
⟨proof⟩

lemma rt_strictly_fresher_fresherD [dest]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
  shows "the (rt1 dip)  $\sqsubseteq$  the (rt2 dip)"
⟨proof⟩

lemma rt_strictly_fresher_not_fresh_asD [dest]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
  shows " $\neg$  rt1  $\approx_{dip}$  rt2"
⟨proof⟩

lemma rt_strictly_fresher_trans [elim, trans]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
    and "rt2  $\sqsubseteq_{dip}$  rt3"
  shows "rt1  $\sqsubseteq_{dip}$  rt3"
⟨proof⟩

lemma rt_strictly_fresher_irefl [simp]: " $\neg$  (rt  $\sqsubseteq_{dip}$  rt)"
⟨proof⟩

lemma rt_fresher_trans_rt_strictly_fresher [elim, trans]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
    and "rt2  $\sqsubseteq_{dip}$  rt3"
  shows "rt1  $\sqsubseteq_{dip}$  rt3"
⟨proof⟩

lemma rt_fresher_trans_rt_strictly_fresher' [elim, trans]:
  assumes "rt1  $\sqsubseteq_{dip}$  rt2"
    and "rt2  $\sqsubseteq_{dip}$  rt3"
  shows "rt1  $\sqsubseteq_{dip}$  rt3"

```

```

⟨proof⟩

lemma rt_fresher_imp_nsqn_le:
  assumes "rt1  $\sqsubseteq_{ip}$  rt2"
    and "ip  $\in kD$  rt1"
    and "ip  $\in kD$  rt2"
  shows "nsqn rt1 ip  $\leq$  nsqn rt2 ip"
⟨proof⟩

lemma rt_strictly_fresher_ltI [intro]:
  assumes "dip  $\in kD$ (rt1)"
    and "dip  $\in kD$ (rt2)"
    and "nsqn rt1 dip < nsqn rt2 dip"
  shows "rt1  $\sqsubset_{dip}$  rt2"
⟨proof⟩

lemma rt_strictly_fresher_eqI [intro]:
  assumes "i  $\in kD$ (rt1)"
    and "i  $\in kD$ (rt2)"
    and "nsqn rt1 i = nsqn rt2 i"
    and " $\pi_5$ (the (rt2 i)) <  $\pi_5$ (the (rt1 i))"
  shows "rt1  $\sqsubset_i$  rt2"
⟨proof⟩

lemma invalidate_rtsf_left [simp]:
  " $\bigwedge$ dests dip rt rt'. dests dip = None  $\implies$  (invalidate rt dests  $\sqsubset_{dip}$  rt') = (rt  $\sqsubset_{dip}$  rt')"
⟨proof⟩

lemma vD_invalidate_rt_strictly_fresher [simp]:
  assumes "dip  $\in vD$ (invalidate rt1 dests)"
  shows "(invalidate rt1 dests  $\sqsubset_{dip}$  rt2) = (rt1  $\sqsubset_{dip}$  rt2)"
⟨proof⟩

lemma rt_strictly_fresher_update_other [elim!]:
  " $\bigwedge$ dip ip rt r rt'. [ $dip \neq ip$ ; rt  $\sqsubset_{dip}$  rt']  $\implies$  update rt ip r  $\sqsubset_{dip}$  rt'"
⟨proof⟩

lemma addpreRT_strictly_fresher [simp]:
  assumes "dip  $\in kD$ (rt)"
  shows "(the (addpreRT rt dip npre)  $\sqsubset_{ip}$  rt2) = (rt  $\sqsubset_{ip}$  rt2)"
⟨proof⟩

lemma lt_sqn_imp_update_strictly_fresher:
  assumes "dip  $\in vD$  (rt2 nhip)"
    and *: "osn < sqn (rt2 nhip) dip"
    and **: "rt  $\neq$  update rt dip (osn, kno, val, hops, nhip, {})"
  shows "update rt dip (osn, kno, val, hops, nhip, {})  $\sqsubset_{dip}$  rt2 nhip"
⟨proof⟩

lemma dhops_le_hops_imp_update_strictly_fresher:
  assumes "dip  $\in vD$ (rt2 nhip)"
    and sqn: "sqn (rt2 nhip) dip = osn"
    and hop: "the (dhops (rt2 nhip) dip)  $\leq$  hops"
    and **: "rt  $\neq$  update rt dip (osn, kno, val, Suc hops, nhip, {})"
  shows "update rt dip (osn, kno, val, Suc hops, nhip, {})  $\sqsubset_{dip}$  rt2 nhip"
⟨proof⟩

lemma nsqn_invalidate:
  assumes "dip  $\in kD$ (rt)"
    and " $\forall ip \in \text{dom}(\text{dests}). ip \in vD(\text{rt}) \wedge \text{the}(\text{dests } ip) = \text{inc}(\text{sqn } \text{rt } ip)"$ "
  shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
⟨proof⟩

end

```

## 4.6 Invariant proofs on individual processes

```
theory D_Seq_Invariants
imports "../../AWN/Invariants" D_Aodv D_Aodv_Data D_Aodv_Predicates D_Fresher
begin
```

The proposition numbers are taken from the December 2013 version of the Fehnker et al technical report.

Proposition 7.2

```
lemma sequence_number_increases:
  "paodv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), \_, (\xi', \_)). sn \xi \leq sn \xi'$ )"
  <proof>
```

```
lemma sequence_number_one_or_bigger:
  "paodv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_). 1 \leq sn \xi$ )"
  <proof>
```

We can get rid of the onl/onll if desired...

```
lemma sequence_number_increases':
  "paodv i  $\models_A$  ( $\lambda((\xi, \_), \_, (\xi', \_)). sn \xi \leq sn \xi'$ )"
  <proof>
```

```
lemma sequence_number_one_or_bigger':
  "paodv i  $\models$  ( $\lambda(\xi, \_). 1 \leq sn \xi$ )"
  <proof>
```

```
lemma sip_in_kD:
  "paodv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). l \in (\{PAodv-:7\} \cup \{PAodv-:5\} \cup \{PRrep-:0..PRrep-:1\}$ 
     $\cup \{PRreq-:0..PRreq-:3\}) \longrightarrow sip \xi \in kD (rt \xi)$ )"
  <proof>
```

```
lemma rrep_1_update_changes:
  "paodv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). (l = PRrep-:1 \longrightarrow$ 
     $rt \xi \neq update (rt \xi) (dip \xi) (dsn \xi, kno, val, hops \xi + 1, sip \xi, \{ })))$ "
  <proof>
```

```
lemma addpreRT_partly_welldefined:
  "paodv i  $\models$ 
    onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). (l \in \{PRreq-:18..PRreq-:20\} \cup \{PRrep-:2..PRrep-:6\} \longrightarrow dip \xi \in kD (rt \xi))$ 
     $\wedge (l \in \{PRreq-:3..PRreq-:19\} \longrightarrow oip \xi \in kD (rt \xi))$ )"
  <proof>
```

Proposition 7.38

```
lemma includes_nhip:
  "paodv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). \forall dip \in kD(rt \xi). the (nhop (rt \xi) dip) \in kD(rt \xi)$ )"
  <proof>
```

Proposition 7.22: needed in Proposition 7.4

```
lemma addpreRT_welldefined:
  "paodv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). (l \in \{PRreq-:18..PRreq-:20\} \longrightarrow dip \xi \in kD (rt \xi)) \wedge$ 
     $(l = PRreq-:19 \longrightarrow oip \xi \in kD (rt \xi)) \wedge$ 
     $(l = PRrep-:5 \longrightarrow dip \xi \in kD (rt \xi)) \wedge$ 
     $(l = PRrep-:6 \longrightarrow (the (nhop (rt \xi) (dip \xi))) \in kD (rt \xi)))$ )"
  (is " $\_ \models onl \Gamma_{AODV} ?P$ ")
  <proof>
```

Proposition 7.4

```
lemma known_destinations_increase:
  "paodv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), \_, (\xi', \_)). kD (rt \xi) \subseteq kD (rt \xi')$ )"
  <proof>
```

Proposition 7.5

```
lemma rreqs_increase:
```

"paodv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), \_, (\xi', \_)). rreqs \xi \subseteq rreqs \xi'$ )"  
 ⟨proof⟩

**lemma dests\_bigger\_than\_sqn:**

"paodv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). l \in \{PAodv-:15..PAodv-:19\}$   
 $\cup \{PPkt-:7..PPkt-:11\}$   
 $\cup \{PRreq-:11..PRreq-:15\}$   
 $\cup \{PRreq-:24..PRreq-:28\}$   
 $\cup \{PRrep-:10..PRrep-:14\}$   
 $\cup \{PRerr-:1..PRerr-:5\}$   
 $\rightarrow (\forall ip \in \text{dom}(\text{dests } \xi). ip \in kD(\text{rt } \xi) \wedge \text{sqn}(\text{rt } \xi) ip \leq \text{the}(\text{dests } \xi ip))$ )"  
 ⟨proof⟩

Proposition 7.6

**lemma sqns\_increase:**

"paodv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), \_, (\xi', \_)). \forall ip. \text{sqn}(\text{rt } \xi) ip \leq \text{sqn}(\text{rt } \xi') ip$ )"  
 ⟨proof⟩

Proposition 7.7

**lemma ip\_constant:**

"paodv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_). ip \xi = i$ )"  
 ⟨proof⟩

Proposition 7.8

**lemma sender\_ip\_valid':**

"paodv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), a, \_). \text{anycast}(\lambda m. \text{not\_Pkt } m \rightarrow \text{msg\_sender } m = ip \xi) a$ )"  
 ⟨proof⟩

**lemma sender\_ip\_valid:**

"paodv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), a, \_). \text{anycast}(\lambda m. \text{not\_Pkt } m \rightarrow \text{msg\_sender } m = i) a$ )"  
 ⟨proof⟩

**lemma received\_msg\_inv:**

"paodv i  $\models$  ( $\text{recvm}sg P \rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). l \in \{PAodv-:1\} \rightarrow P(\text{msg } \xi)$ )"  
 ⟨proof⟩

Proposition 7.9

**lemma sip\_not\_ip':**

"paodv i  $\models$  ( $\text{recvm}sg(\lambda m. \text{not\_Pkt } m \rightarrow \text{msg\_sender } m \neq i) \rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_). \text{sip } \xi \neq ip \xi$ )"  
 ⟨proof⟩

**lemma sip\_not\_ip:**

"paodv i  $\models$  ( $\text{recvm}sg(\lambda m. \text{not\_Pkt } m \rightarrow \text{msg\_sender } m \neq i) \rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_). \text{sip } \xi \neq i$ )"  
 ⟨proof⟩

Neither *sip\_not\_ip'* nor *sip\_not\_ip* is needed to show loop freedom.

Proposition 7.10

**lemma hop\_count\_positive:**

"paodv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_). \forall ip \in kD(\text{rt } \xi). \text{the}(\text{dhops}(\text{rt } \xi) ip) \geq 1$ )"  
 ⟨proof⟩

**lemma rreq\_dip\_in\_vD\_dip\_eq\_ip:**

"paodv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). (l \in \{PRreq-:18..PRreq-:21\} \rightarrow \text{dip } \xi \in vD(\text{rt } \xi))$   
 $\wedge (l \in \{PRreq-:6, PRreq-:7\} \rightarrow \text{dip } \xi = ip \xi)$   
 $\wedge (l \in \{PRreq-:17..PRreq-:21\} \rightarrow \text{dip } \xi \neq ip \xi)$ )"  
 ⟨proof⟩

Proposition 7.11

**lemma anycast\_msg\_zhops:**

" $\wedge rreqid \text{ dip dsn dsk oip osn sip.}$   
 paodv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda(\_, a, \_). \text{anycast } \text{msg\_zhops } a$ )"  
 ⟨proof⟩

**lemma hop\_count\_zero\_oip\_dip\_sip:**

"paadv i  $\models$  (recvmsg msg\_zhops  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l)$ .  
 $(l \in \{PAadv-:4..PAadv-:5\} \cup \{PRreq-:n/n. True\} \rightarrow$   
 $(hops \xi = 0 \rightarrow oip \xi = sip \xi))$   
 $\wedge$   
 $((l \in \{PAadv-:6..PAadv-:7\} \cup \{PRrep-:n/n. True\} \rightarrow$   
 $(hops \xi = 0 \rightarrow dip \xi = sip \xi))))$ "

*<proof>*

**lemma osn\_rreq:**

"paadv i  $\models$  (recvmsg rreq\_rrep\_sn  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l)$ .  
 $l \in \{PAadv-:4, PAadv-:5\} \cup \{PRreq-:n/n. True\} \rightarrow 1 \leq osn \xi$ )"

*<proof>*

**lemma osn\_rreq':**

"paadv i  $\models$  (recvmsg ( $\lambda m. rreq_rrep\_sn m \wedge msg\_zhops m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l)$ .  
 $l \in \{PAadv-:4, PAadv-:5\} \cup \{PRreq-:n/n. True\} \rightarrow 1 \leq osn \xi$ )"

*<proof>*

**lemma dsn\_rrep:**

"paadv i  $\models$  (recvmsg rreq\_rrep\_sn  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l)$ .  
 $l \in \{PAadv-:6, PAadv-:7\} \cup \{PRrep-:n/n. True\} \rightarrow 1 \leq dsn \xi$ )"

*<proof>*

**lemma dsn\_rrep':**

"paadv i  $\models$  (recvmsg ( $\lambda m. rreq_rrep\_sn m \wedge msg\_zhops m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l)$ .  
 $l \in \{PAadv-:6, PAadv-:7\} \cup \{PRrep-:n/n. True\} \rightarrow 1 \leq dsn \xi$ )"

*<proof>*

**lemma hop\_count\_zero\_oip\_dip\_sip':**

"paadv i  $\models$  (recvmsg ( $\lambda m. rreq_rrep\_sn m \wedge msg\_zhops m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l)$ .  
 $(l \in \{PAadv-:4..PAadv-:5\} \cup \{PRreq-:n/n. True\} \rightarrow$   
 $(hops \xi = 0 \rightarrow oip \xi = sip \xi))$   
 $\wedge$   
 $((l \in \{PAadv-:6..PAadv-:7\} \cup \{PRrep-:n/n. True\} \rightarrow$   
 $(hops \xi = 0 \rightarrow dip \xi = sip \xi))))$ "

*<proof>*

Proposition 7.12

**lemma zero\_seq\_unk\_hops\_one':**

"paadv i  $\models$  (recvmsg ( $\lambda m. rreq_rrep\_sn m \wedge msg\_zhops m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_)$ .  
 $\forall dip \in kD(rt \xi). (sqn (rt \xi) dip = 0 \rightarrow sqnf (rt \xi) dip = unk)$   
 $\wedge (sqnf (rt \xi) dip = unk \rightarrow the (dhops (rt \xi) dip) = 1)$   
 $\wedge (the (dhops (rt \xi) dip) = 1 \rightarrow the (nhop (rt \xi) dip) = dip))$ "

*<proof>*

**lemma zero\_seq\_unk\_hops\_one:**

"paadv i  $\models$  (recvmsg ( $\lambda m. rreq_rrep\_sn m \wedge msg\_zhops m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_)$ .  
 $\forall dip \in kD(rt \xi). (sqn (rt \xi) dip = 0 \rightarrow (sqnf (rt \xi) dip = unk$   
 $\wedge the (dhops (rt \xi) dip) = 1$   
 $\wedge the (nhop (rt \xi) dip) = dip))$ "

*<proof>*

**lemma kD\_unk\_or\_atleast\_one:**

"paadv i  $\models$  (recvmsg rreq\_rrep\_sn  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l)$ .  
 $\forall dip \in kD(rt \xi). \pi_3(the (rt \xi) dip) = unk \vee 1 \leq \pi_2(the (rt \xi) dip))$ "

*<proof>*

Proposition 7.13

**lemma rreq\_rrep\_sn\_any\_step\_invariant:**

"paadv i  $\models_A$  (recvmsg rreq\_rrep\_sn  $\rightarrow$ ) onll  $\Gamma_{AODV}$  ( $\lambda(\_, a, \_)$ . anycast rreq\_rrep\_sn a)"

*<proof>*

Proposition 7.14

**lemma rreq\_rrep\_fresh\_any\_step\_invariant:**

```
"paadv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), a, \_)$ . anycast (rreq_rrep_fresh (rt  $\xi$ )) a)"
<proof>
```

Proposition 7.15

**lemma rerr\_invalid\_any\_step\_invariant:**

```
"paadv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), a, \_)$ . anycast (rerr_invalid (rt  $\xi$ )) a)"
<proof>
```

Proposition 7.16

Some well-definedness obligations are irrelevant for the Isabelle development:

1. In each routing table there is at most one entry for each destination: guaranteed by type.
2. In each store of queued data packets there is at most one data queue for each destination: guaranteed by structure.
3. Whenever a set of pairs  $(rip, rsn)$  is assigned to the variable  $dests$  of type  $ip \rightarrow sqn$ , or to the first argument of the function  $rerr$ , this set is a partial function, i.e., there is at most one entry  $(rip, rsn)$  for each destination  $rip$ : guaranteed by type.

**lemma dests\_vD\_inc\_sqn:**

```
"paadv i  $\models$ 
  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l)$ . ( $l \in \{PAadv-:15, PPkt-:7, PRreq-:11, PRreq-:24, PRrep-:10\}$ 
     $\rightarrow (\forall ip \in \text{dom}(dests \ \xi)$ .  $ip \in vD(\text{rt } \xi) \wedge \text{the}(dests \ \xi \ ip) = \text{inc}(sqn(\text{rt } \xi) \ ip))$ )
     $\wedge (l = PRerr-:1$ 
     $\rightarrow (\forall ip \in \text{dom}(dests \ \xi)$ .  $ip \in vD(\text{rt } \xi) \wedge \text{the}(dests \ \xi \ ip) > sqn(\text{rt } \xi) \ ip))$ )"
<proof>
```

Proposition 7.27

**lemma route\_tables\_fresher:**

```
"paadv i  $\models_A$  (recvmmsg rreq_rrep_sn  $\rightarrow$ ) onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), \_, (\xi', \_))$ .
   $\forall dip \in kD(\text{rt } \xi)$ .  $\text{rt } \xi \sqsubseteq_{dip} \text{rt } \xi'$ )"
<proof>
```

end

## 4.7 The quality increases predicate

**theory D\_Quality\_Increases**

**imports D\_Aadv\_Predicates D\_Fresher**

**begin**

**definition quality\_increases** :: "state  $\Rightarrow$  state  $\Rightarrow$  bool"

```
where "quality_increases  $\xi \ \xi' \equiv (\forall dip \in kD(\text{rt } \xi)$ .  $dip \in kD(\text{rt } \xi')$   $\wedge \text{rt } \xi \sqsubseteq_{dip} \text{rt } \xi'$ )
   $\wedge (\forall dip$ .  $sqn(\text{rt } \xi) \ dip \leq sqn(\text{rt } \xi') \ dip)$ "
```

**lemma quality\_increasesI [intro!]:**

```
assumes " $\wedge dip$ .  $dip \in kD(\text{rt } \xi) \implies dip \in kD(\text{rt } \xi')$ "
  and " $\wedge dip$ .  $\llbracket dip \in kD(\text{rt } \xi); dip \in kD(\text{rt } \xi') \rrbracket \implies \text{rt } \xi \sqsubseteq_{dip} \text{rt } \xi'$ "
  and " $\wedge dip$ .  $sqn(\text{rt } \xi) \ dip \leq sqn(\text{rt } \xi') \ dip$ "
shows "quality_increases  $\xi \ \xi'$ "
<proof>
```

**lemma quality\_increasesE [elim]:**

```
fixes dip
assumes "quality_increases  $\xi \ \xi'$ "
  and " $dip \in kD(\text{rt } \xi)$ "
  and " $\llbracket dip \in kD(\text{rt } \xi'); \text{rt } \xi \sqsubseteq_{dip} \text{rt } \xi'; sqn(\text{rt } \xi) \ dip \leq sqn(\text{rt } \xi') \ dip \rrbracket \implies R \ dip \ \xi \ \xi'$ "
shows " $R \ dip \ \xi \ \xi'$ "
```

*<proof>*

lemma quality\_increases\_rt\_fresherD [dest]:

fixes ip  
assumes "quality\_increases  $\xi \xi'$ "  
and "ip  $\in kD(rt \xi)$ "  
shows "rt  $\xi \sqsubseteq_{ip} rt \xi'$ "

*<proof>*

lemma quality\_increases\_sqnE [elim]:

fixes dip  
assumes "quality\_increases  $\xi \xi'$ "  
and "sqn (rt  $\xi$ ) dip  $\leq$  sqn (rt  $\xi'$ ) dip  $\implies R \text{ dip } \xi \xi'$ "  
shows "R dip  $\xi \xi'$ "

*<proof>*

lemma quality\_increases\_refl [intro, simp]: "quality\_increases  $\xi \xi$ "

*<proof>*

lemma strictly\_fresher\_quality\_increases\_right [elim]:

fixes  $\sigma \sigma'$  dip  
assumes "rt ( $\sigma i$ )  $\sqsubseteq_{dip} rt (\sigma \text{ nhip})$ "  
and qinc: "quality\_increases ( $\sigma \text{ nhip}$ ) ( $\sigma' \text{ nhip}$ )"  
and "dip  $\in kD(rt (\sigma \text{ nhip}))$ "  
shows "rt ( $\sigma i$ )  $\sqsubseteq_{dip} rt (\sigma' \text{ nhip})$ "

*<proof>*

lemma kD\_quality\_increases [elim]:

assumes "i  $\in kD(rt \xi)$ "  
and "quality\_increases  $\xi \xi'$ "  
shows "i  $\in kD(rt \xi')$ "

*<proof>*

lemma kD\_nsqn\_quality\_increases [elim]:

assumes "i  $\in kD(rt \xi)$ "  
and "quality\_increases  $\xi \xi'$ "  
shows "i  $\in kD(rt \xi') \wedge \text{nsqn} (rt \xi) i \leq \text{nsqn} (rt \xi') i$ "

*<proof>*

lemma nsqn\_quality\_increases [elim]:

assumes "i  $\in kD(rt \xi)$ "  
and "quality\_increases  $\xi \xi'$ "  
shows "nsqn (rt  $\xi$ ) i  $\leq$  nsqn (rt  $\xi'$ ) i"

*<proof>*

lemma kD\_nsqn\_quality\_increases\_trans [elim]:

assumes "i  $\in kD(rt \xi)$ "  
and "s  $\leq$  nsqn (rt  $\xi$ ) i"  
and "quality\_increases  $\xi \xi'$ "  
shows "i  $\in kD(rt \xi') \wedge s \leq$  nsqn (rt  $\xi')$  i"

*<proof>*

lemma nsqn\_quality\_increases\_nsqn\_lt\_lt [elim]:

assumes "i  $\in kD(rt \xi)$ "  
and "quality\_increases  $\xi \xi'$ "  
and "s < nsqn (rt  $\xi$ ) i"  
shows "s < nsqn (rt  $\xi')$  i"

*<proof>*

lemma nsqn\_quality\_increases\_dhops [elim]:

assumes "i  $\in kD(rt \xi)$ "  
and "quality\_increases  $\xi \xi'$ "  
and "nsqn (rt  $\xi$ ) i = nsqn (rt  $\xi')$  i"  
shows "the (dhops (rt  $\xi$ ) i)  $\geq$  the (dhops (rt  $\xi')$  i)"



*<proof>*

lemma nsqn\_quality\_increases\_nsqn\_eq\_le [elim]:

assumes "i ∈ kD(rt ξ)"  
and "quality\_increases ξ ξ'"  
and "s = nsqn (rt ξ) i"  
shows "s < nsqn (rt ξ') i ∨ (s = nsqn (rt ξ') i ∧ the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i))"  
*<proof>*

lemma quality\_increases\_rreq\_rrep\_props [elim]:

fixes sn ip hops sip  
assumes qinc: "quality\_increases (σ sip) (σ' sip)"  
and "1 ≤ sn"  
and \*: "ip ∈ kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip  
∧ (nsqn (rt (σ sip)) ip = sn  
→ (the (dhops (rt (σ sip)) ip) ≤ hops  
∨ the (flag (rt (σ sip)) ip) = inv))"  
shows "ip ∈ kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip  
∧ (nsqn (rt (σ' sip)) ip = sn  
→ (the (dhops (rt (σ' sip)) ip) ≤ hops  
∨ the (flag (rt (σ' sip)) ip) = inv))"  
(is "\_ ∧ ?nsqnafter")  
*<proof>*

lemma quality\_increases\_rreq\_rrep\_props':

fixes sn ip hops sip  
assumes "∀ j. quality\_increases (σ j) (σ' j)"  
and "1 ≤ sn"  
and \*: "ip ∈ kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip  
∧ (nsqn (rt (σ sip)) ip = sn  
→ (the (dhops (rt (σ sip)) ip) ≤ hops  
∨ the (flag (rt (σ sip)) ip) = inv))"  
shows "ip ∈ kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip  
∧ (nsqn (rt (σ' sip)) ip = sn  
→ (the (dhops (rt (σ' sip)) ip) ≤ hops  
∨ the (flag (rt (σ' sip)) ip) = inv))"  
*<proof>*

lemma rteq\_quality\_increases:

assumes "∀ j. j ≠ i → quality\_increases (σ j) (σ' j)"  
and "rt (σ' i) = rt (σ i)"  
shows "∀ j. quality\_increases (σ j) (σ' j)"  
*<proof>*

definition msg\_fresh :: "(ip ⇒ state) ⇒ msg ⇒ bool"

where "msg\_fresh σ m ≡

case m of Rreq hops<sub>c</sub> \_ \_ \_ oipc osnc sip<sub>c</sub> \_ ⇒ osnc ≥ 1 ∧ (sip<sub>c</sub> ≠ oipc →  
oipc ∈ kD(rt (σ sip<sub>c</sub>)) ∧ nsqn (rt (σ sip<sub>c</sub>)) oipc ≥ osnc  
∧ (nsqn (rt (σ sip<sub>c</sub>)) oipc = osnc  
→ (hops<sub>c</sub> ≥ the (dhops (rt (σ sip<sub>c</sub>)) oipc)  
∨ the (flag (rt (σ sip<sub>c</sub>)) oipc) = inv)))  
| Rrep hops<sub>c</sub> dip<sub>c</sub> dsnc \_ sip<sub>c</sub> ⇒ dsnc ≥ 1 ∧ (sip<sub>c</sub> ≠ dip<sub>c</sub> →  
dip<sub>c</sub> ∈ kD(rt (σ sip<sub>c</sub>)) ∧ nsqn (rt (σ sip<sub>c</sub>)) dip<sub>c</sub> ≥ dsnc  
∧ (nsqn (rt (σ sip<sub>c</sub>)) dip<sub>c</sub> = dsnc  
→ (hops<sub>c</sub> ≥ the (dhops (rt (σ sip<sub>c</sub>)) dip<sub>c</sub>)  
∨ the (flag (rt (σ sip<sub>c</sub>)) dip<sub>c</sub>) = inv)))  
| Rerr destsc sip<sub>c</sub> ⇒ (∀ rip<sub>c</sub> ∈ dom(destsc). (rip<sub>c</sub> ∈ kD(rt (σ sip<sub>c</sub>))  
∧ the (destsc rip<sub>c</sub>) - 1 ≤ nsqn (rt (σ sip<sub>c</sub>)) rip<sub>c</sub>)  
| \_ ⇒ True"

lemma msg\_fresh [simp]:

"∧ hops rreqid dip dsnc dsk oip osn sip handled.  
msg\_fresh σ (Rreq hops rreqid dip dsnc dsk oip osn sip handled) =  
(osn ≥ 1 ∧ (sip ≠ oip → oip ∈ kD(rt (σ sip)))"

```

      ∧ nsqn (rt (σ sip)) oip ≥ osn
      ∧ (nsqn (rt (σ sip)) oip = osn
        → (hops ≥ the (dhops (rt (σ sip)) oip)
            ∨ the (flag (rt (σ sip)) oip) = inv)))"
"∧hops dip dsn oip sip. msg_fresh σ (Rrep hops dip dsn oip sip) =
  (dsn ≥ 1 ∧ (sip ≠ dip → dip ∈ kD(rt (σ sip))
    ∧ nsqn (rt (σ sip)) dip ≥ dsn
    ∧ (nsqn (rt (σ sip)) dip = dsn
      → (hops ≥ the (dhops (rt (σ sip)) dip)
        ∨ the (flag (rt (σ sip)) dip) = inv))))"
"∧dests sip.      msg_fresh σ (Rerr dests sip) =
  (∀ ripc ∈ dom(dests). (ripc ∈ kD(rt (σ sip))
    ∧ the (dests ripc) - 1 ≤ nsqn (rt (σ sip)) ripc))"
"∧d dip.          msg_fresh σ (Newpkt d dip) = True"
"∧d dip sip.     msg_fresh σ (Pkt d dip sip) = True"
⟨proof⟩

```

lemma msg\_fresh\_inc\_sn [simp, elim]:

```

"msg_fresh σ m ⇒ rreq_rrep_sn m"
⟨proof⟩

```

lemma recv\_msg\_fresh\_inc\_sn [simp, elim]:

```

"orecvmsg (msg_fresh) σ m ⇒ recvmmsg rreq_rrep_sn m"
⟨proof⟩

```

lemma rreq\_nsqn\_is\_fresh [simp]:

```

fixes σ msg hops rreqid dip dsn dsk oip osn sip handled
assumes "rreq_rrep_fresh (rt (σ sip)) (Rreq hops rreqid dip dsn dsk oip osn sip handled)"
and "rreq_rrep_sn (Rreq hops rreqid dip dsn dsk oip osn sip handled)"
shows "msg_fresh σ (Rreq hops rreqid dip dsn dsk oip osn sip handled)"
(is "msg_fresh σ ?msg")
⟨proof⟩

```

lemma rrep\_nsqn\_is\_fresh [simp]:

```

fixes σ msg hops dip dsn oip sip
assumes "rreq_rrep_fresh (rt (σ sip)) (Rrep hops dip dsn oip sip)"
and "rreq_rrep_sn (Rrep hops dip dsn oip sip)"
shows "msg_fresh σ (Rrep hops dip dsn oip sip)"
(is "msg_fresh σ ?msg")
⟨proof⟩

```

lemma rerr\_nsqn\_is\_fresh [simp]:

```

fixes σ msg dests sip
assumes "rerr_invalid (rt (σ sip)) (Rerr dests sip)"
shows "msg_fresh σ (Rerr dests sip)"
(is "msg_fresh σ ?msg")
⟨proof⟩

```

lemma quality\_increases\_msg\_fresh [elim]:

```

assumes qinc: "∀ j. quality_increases (σ j) (σ' j)"
and "msg_fresh σ m"
shows "msg_fresh σ' m"
⟨proof⟩

```

end

## 4.8 The ‘open’ AODV model

```

theory D_OAodv
imports D_Aodv "../..../AWN/OAWN_SOS_Labels" "../..../AWN/OAWN_Convert"
begin

```

Definitions for stating and proving global network properties over individual processes.

```

definition σAODV' :: "(ip ⇒ state) × ((state, msg, pseqp, pseqp label) seqp) set"

```

where " $\sigma_{AODV}' \equiv \{(\lambda i. aadv\_init\ i, \Gamma_{AODV}\ PAadv)\}$ "

abbreviation opaadv

:: "ip  $\Rightarrow$  ((ip  $\Rightarrow$  state)  $\times$  (state, msg, pseq, pseq label) seq, msg seq\_action) automaton"

where

"opaadv i  $\equiv$  ( $\lfloor$  init =  $\sigma_{AODV}'$ , trans = oseqp\_sos  $\Gamma_{AODV}$  i  $\rfloor$ )"

lemma initiali\_aadv [intro!, simp]: "initiali i (init (opaadv i)) (init (paadv i))"

$\langle$ proof $\rangle$

lemma oaadv\_control\_within [simp]: "control\_within  $\Gamma_{AODV}$  (init (opaadv i))"

$\langle$ proof $\rangle$

lemma  $\sigma_{AODV}'$ \_labels [simp]: " $(\sigma, p) \in \sigma_{AODV}' \implies$  labels  $\Gamma_{AODV}$  p = {PAadv-:0}"

$\langle$ proof $\rangle$

lemma oaadv\_init\_kD\_empty [simp]:

" $(\sigma, p) \in \sigma_{AODV}' \implies$  kD (rt ( $\sigma$  i)) = {}"

$\langle$ proof $\rangle$

lemma oaadv\_init\_vD\_empty [simp]:

" $(\sigma, p) \in \sigma_{AODV}' \implies$  vD (rt ( $\sigma$  i)) = {}"

$\langle$ proof $\rangle$

lemma oaadv\_trans: "trans (opaadv i) = oseqp\_sos  $\Gamma_{AODV}$  i"

$\langle$ proof $\rangle$

declare

oseq\_invariant\_ctermsI [OF aadv\_wf oaadv\_control\_within aadv\_simple\_labels oaadv\_trans, cterms\_intros]  
oseq\_step\_invariant\_ctermsI [OF aadv\_wf oaadv\_control\_within aadv\_simple\_labels oaadv\_trans, cterms\_intros]

end

## 4.9 Global invariant proofs over sequential processes

theory D\_Global\_Invariants

imports D\_Seq\_Invariants

D\_Aadv\_Predicates

D\_Fresher

D\_Quality\_Increases

"../../../../AWN/OAWN\_Convert"

D\_OAadv

begin

lemma other\_quality\_increases [elim]:

assumes "other quality\_increases I  $\sigma$   $\sigma'$ "

shows " $\forall j. quality\_increases$  ( $\sigma$  j) ( $\sigma'$  j)"

$\langle$ proof $\rangle$

lemma weaken\_otherwith [elim]:

fixes m

assumes \*: "otherwith P I (orecvmsg Q)  $\sigma$   $\sigma'$  a"

and weakenP: " $\bigwedge \sigma m. P \sigma m \implies P' \sigma m$ "

and weakenQ: " $\bigwedge \sigma m. Q \sigma m \implies Q' \sigma m$ "

shows "otherwith P' I (orecvmsg Q')  $\sigma$   $\sigma'$  a"

$\langle$ proof $\rangle$

lemma oreceived\_msg\_inv:

assumes other: " $\bigwedge \sigma \sigma' m. \llbracket P \sigma m; other\ Q\ \{i\}\ \sigma \sigma' \rrbracket \implies P \sigma' m$ "

and local: " $\bigwedge \sigma m. P \sigma m \implies P (\sigma(i := \sigma i(\text{msg} := m))) m$ "

shows "opaadv i  $\models$  (otherwith Q {i} (orecvmsg P), other Q {i}  $\rightarrow$ )

onl  $\Gamma_{AODV} (\lambda(\sigma, l). l \in \{PAadv-:1\} \rightarrow P \sigma (\text{msg} (\sigma i)))$ "

$\langle$ proof $\rangle$

(Equivalent to) Proposition 7.27

**lemma local\_quality\_increases:**

"paadv i  $\models_A$  (recvmmsg rreq\_rrep\_sn  $\rightarrow$ ) onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), \_, (\xi', \_)).$  quality\_increases  $\xi \xi'$ )"  
 <proof>

**lemmas olocal\_quality\_increases =**

open\_seq\_step\_invariant [OF local\_quality\_increases initiali\_aadv oaadv\_trans aadv\_trans,  
 simplified seqll\_onll\_swap]

**lemma oquality\_increases:**

"opaadv i  $\models_A$  (otherwith quality\_increases {i} (orecvmsg ( $\lambda\_.$  rreq\_rrep\_sn)),  
 other quality\_increases {i}  $\rightarrow$ )  
 onll  $\Gamma_{AODV}$  ( $\lambda((\sigma, \_), \_, (\sigma', \_)). \forall j.$  quality\_increases ( $\sigma j$ ) ( $\sigma' j$ ))"  
 (is " $\_ \models_A$  (?S,  $\_ \rightarrow$ )  $\_$ ")  
 <proof>

**lemma rreq\_rrep\_nsqn\_fresh\_any\_step\_invariant:**

"opaadv i  $\models_A$  (act (recvmmsg rreq\_rrep\_sn), other A {i}  $\rightarrow$ )  
 onll  $\Gamma_{AODV}$  ( $\lambda((\sigma, \_), a, \_).$  anycast (msg\_fresh  $\sigma$ ) a)"  
 <proof>

**lemma oreceived\_rreq\_rrep\_nsqn\_fresh\_inv:**

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onl  $\Gamma_{AODV}$  ( $\lambda(\sigma, l).$   $l \in \{PAadv:-1\} \rightarrow$  msg\_fresh  $\sigma$  (msg ( $\sigma i$ )))"  
 <proof>

**lemma oquality\_increases\_nsqn\_fresh:**

"opaadv i  $\models_A$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onll  $\Gamma_{AODV}$  ( $\lambda((\sigma, \_), \_, (\sigma', \_)). \forall j.$  quality\_increases ( $\sigma j$ ) ( $\sigma' j$ ))"  
 <proof>

**lemma oosn\_rreq:**

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onl  $\Gamma_{AODV}$  (seq1 i ( $\lambda(\xi, l).$   $l \in \{PAadv:-4, PAadv:-5\} \cup \{PRreq:-n \mid n. \text{True}\} \rightarrow 1 \leq$  osn  $\xi$ ))"  
 <proof>

**lemma rreq\_sip:**

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onl  $\Gamma_{AODV}$  ( $\lambda(\sigma, l).$   
 ( $l \in \{PAadv:-4, PAadv:-5, PRreq:-0, PRreq:-2\} \wedge$  sip ( $\sigma i$ )  $\neq$  oip ( $\sigma i$ ))  
 $\rightarrow$  oip ( $\sigma i$ )  $\in$  kD(rt ( $\sigma$  (sip ( $\sigma i$ ))))  
 $\wedge$  nsqn (rt ( $\sigma$  (sip ( $\sigma i$ )))) (oip ( $\sigma i$ ))  $\geq$  osn ( $\sigma i$ )  
 $\wedge$  (nsqn (rt ( $\sigma$  (sip ( $\sigma i$ )))) (oip ( $\sigma i$ )) = osn ( $\sigma i$ )  
 $\rightarrow$  (hops ( $\sigma i$ )  $\geq$  the (dhops (rt ( $\sigma$  (sip ( $\sigma i$ )))) (oip ( $\sigma i$ )))  
 $\vee$  the (flag (rt ( $\sigma$  (sip ( $\sigma i$ )))) (oip ( $\sigma i$ ))) = inv)))"  
 (is " $\_ \models$  (?S, ?U  $\rightarrow$ )  $\_$ ")  
 <proof>

**lemma odsn\_rrep:**

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onl  $\Gamma_{AODV}$  (seq1 i ( $\lambda(\xi, l).$   $l \in \{PAadv:-6, PAadv:-7\} \cup \{PRrep:-n \mid n. \text{True}\} \rightarrow 1 \leq$  dsn  $\xi$ ))"  
 <proof>

**lemma rrep\_sip:**

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onl  $\Gamma_{AODV}$  ( $\lambda(\sigma, l).$   
 ( $l \in \{PAadv:-6, PAadv:-7, PRrep:-0, PRrep:-1\} \wedge$  sip ( $\sigma i$ )  $\neq$  dip ( $\sigma i$ ))  
 $\rightarrow$  dip ( $\sigma i$ )  $\in$  kD(rt ( $\sigma$  (sip ( $\sigma i$ ))))

```

      ∧ nsqn (rt (σ (sip (σ i)))) (dip (σ i)) ≥ dsn (σ i)
      ∧ (nsqn (rt (σ (sip (σ i)))) (dip (σ i)) = dsn (σ i)
        → (hops (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (dip (σ i)))
          ∨ the (flag (rt (σ (sip (σ i)))) (dip (σ i))) = inv)))"
  (is "_ ⊨ (?S, ?U →) _")
  ⟨proof⟩

```

**lemma rerr\_sip:**

```

"opaadv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
  other quality_increases {i} →)
  onl ΓAODV (λ(σ, l).
    l ∈ {PAadv-:8, PAadv-:9, PRerr-:0, PRerr-:1}
    → (∀ ripc ∈ dom(dests (σ i)). ripc ∈ kD(rt (σ (sip (σ i)))) ∧
      the (dests (σ i) ripc) - 1 ≤ nsqn (rt (σ (sip (σ i)))) ripc))"
  (is "_ ⊨ (?S, ?U →) _")
  ⟨proof⟩

```

**lemma prerr\_guard:** "paadv i ⊨

```

  onl ΓAODV (λ(ξ, l). (l = PRerr-:1
    → (∀ ip ∈ dom(dests ξ). ip ∈ vD(rt ξ)
      ∧ the (nhop (rt ξ) ip) = sip ξ
      ∧ sqn (rt ξ) ip < the (dests ξ ip))))"

```

⟨proof⟩

**lemmas oadpreRT\_welldefined =**

```

  open_seq_invariant [OF addpreRT_welldefined initiali_aadv oaadv_trans aadv_trans,
    simplified seq_l_onl_swap,
    THEN oinvariant_anyact]

```

**lemmas odests\_vD\_inc\_sqn =**

```

  open_seq_invariant [OF dests_vD_inc_sqn initiali_aadv oaadv_trans aadv_trans,
    simplified seq_l_onl_swap,
    THEN oinvariant_anyact]

```

**lemmas oprerr\_guard =**

```

  open_seq_invariant [OF prerr_guard initiali_aadv oaadv_trans aadv_trans,
    simplified seq_l_onl_swap,
    THEN oinvariant_anyact]

```

Proposition 7.28

**lemma seq\_compare\_next\_hop':**

```

"opaadv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
  other quality_increases {i} →) onl ΓAODV (λ(σ, _).
  ∀ dip. let nhop = the (nhop (rt (σ i)) dip)
    in dip ∈ kD(rt (σ i)) ∧ nhop ≠ dip →
      dip ∈ kD(rt (σ nhop)) ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhop)) dip)"
  (is "_ ⊨ (?S, ?U →) _")
  ⟨proof⟩

```

Proposition 7.30

**lemmas okD\_unk\_or\_atleast\_one =**

```

  open_seq_invariant [OF kD_unk_or_atleast_one initiali_aadv,
    simplified seq_l_onl_swap]

```

**lemmas ozero\_seq\_unk\_hops\_one =**

```

  open_seq_invariant [OF zero_seq_unk_hops_one initiali_aadv,
    simplified seq_l_onl_swap]

```

**lemma oreachable\_fresh\_okD\_unk\_or\_atleast\_one:**

```

  fixes dip
  assumes "(σ, p) ∈ oreachable (opaadv i)
    (otherwith (op=) {i} (orecvmsg (λσ m. msg_fresh σ m
      ∧ msg_zhops m)))
    (other quality_increases {i})"

```

```

    and "dip ∈ kD(rt (σ i))"
    shows "π3(the (rt (σ i) dip)) = unk ∨ 1 ≤ π2(the (rt (σ i) dip))"
    (is "?P dip")
  ⟨proof⟩

```

**lemma** *oreachable\_fresh\_ozero\_seq\_unk\_hops\_one*:

```

  fixes dip
  assumes "(σ, p) ∈ oreachable (opaadv i)
    (otherwith (op=) {i} (orecvmsg (λσ m. msg_fresh σ m
    ∧ msg_zhops m)))
    (other quality_increases {i})"
  and "dip ∈ kD(rt (σ i))"
  shows "sqn (rt (σ i)) dip = 0 →
    sqnf (rt (σ i)) dip = unk
    ∧ the (dhops (rt (σ i)) dip) = 1
    ∧ the (nhop (rt (σ i)) dip) = dip"
  (is "?P dip")
  ⟨proof⟩

```

**lemma** *seq\_nhop\_quality\_increases'*:

```

  shows "opaadv i ⊨ (otherwith (op=) {i}
    (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
    other quality_increases {i} →)
    onl ΓAODV (λ(σ, _). ∀dip. let nhip = the (nhop (rt (σ i)) dip)
    in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip))
    ∧ nhip ≠ dip
    → (rt (σ i)) ⊑dip (rt (σ nhip)))"
  (is "_ ⊨ (?S i, _ →) _")
  ⟨proof⟩

```

**lemma** *seq\_compare\_next\_hop*:

```

  fixes w
  shows "opaadv i ⊨ (otherwith (op=) {i} (orecvmsg msg_fresh),
    other quality_increases {i} →)
    global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
    in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip →
    dip ∈ kD(rt (σ nhip))
    ∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip)"
  ⟨proof⟩

```

**lemma** *seq\_nhop\_quality\_increases*:

```

  shows "opaadv i ⊨ (otherwith (op=) {i}
    (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
    other quality_increases {i} →)
    global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
    in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
    → (rt (σ i)) ⊑dip (rt (σ nhip)))"
  ⟨proof⟩

```

**end**

## 4.10 Routing graphs and loop freedom

```

theory D_Loop_Freedom
imports D_Aodv_Predicates D_Fresher
begin

```

Define the central theorem that relates an invariant over network states to the absence of loops in the associate routing graph.

**definition**

```

  rt_graph :: "(ip ⇒ state) ⇒ ip ⇒ ip rel"
where
  "rt_graph σ = (λdip.

```

```
{(ip, ip') | ip ip' dsn dsk hops pre.
  ip ≠ dip ∧ rt (σ ip) dip = Some (dsn, dsk, val, hops, ip', pre)}
```

Given the state of a network  $\sigma$ , a routing graph for a given destination ip address  $dip$  abstracts the details of routing tables into nodes (ip addresses) and vertices (valid routes between ip addresses).

```
lemma rt_graphE [elim]:
  fixes n dip ip ip'
  assumes "(ip, ip') ∈ rt_graph σ dip"
  shows "ip ≠ dip ∧ (∃r. rt (σ ip) = r
    ∧ (∃dsn dsk hops pre. r dip = Some (dsn, dsk, val, hops, ip', pre)))"
  <proof>
```

```
lemma rt_graph_vD [dest]:
  "∧ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⇒ dip ∈ vD(rt (σ ip))"
  <proof>
```

```
lemma rt_graph_vD_trans [dest]:
  "∧ip ip' σ dip. (ip, ip') ∈ (rt_graph σ dip)+ ⇒ dip ∈ vD(rt (σ ip))"
  <proof>
```

```
lemma rt_graph_not_dip [dest]:
  "∧ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⇒ ip ≠ dip"
  <proof>
```

```
lemma rt_graph_not_dip_trans [dest]:
  "∧ip ip' σ dip. (ip, ip') ∈ (rt_graph σ dip)+ ⇒ ip ≠ dip"
  <proof>
```

NB: the property below cannot be lifted to the transitive closure

```
lemma rt_graph_nhip_is_nhop [dest]:
  "∧ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⇒ ip' = the (nhop (rt (σ ip)) dip)"
  <proof>
```

```
theorem inv_to_loop_freedom:
  assumes "∀i dip. let nhip = the (nhop (rt (σ i)) dip)
    in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
    → (rt (σ i)) □dip (rt (σ nhip))"
  shows "∀dip. irrefl ((rt_graph σ dip)+)"
  <proof>
```

end

## 4.11 Lift and transfer invariants to show loop freedom

```
theory D_Aodv_Loop_Freedom
imports "../.../AWN/OClosed_Transfer" "../.../AWN/Qmsg_Lifting" D_Global_Invariants D_Loop_Freedom
begin
```

### 4.11.1 Lift to parallel processes with queues

```
lemma par_step_no_change_on_send_or_receive:
  fixes σ s a σ' s'
  assumes "((σ, s), a, (σ', s')) ∈ oparp_sos i (oseqp_sos ΓAODV i) (seqp_sos ΓQMSG)"
  and "a ≠ τ"
  shows "σ' i = σ i"
  <proof>
```

```
lemma par_nhop_quality_increases:
  shows "opaodv i <<i qmsg ⊢ (otherwith (op=) {i} (orecvmsg (λσ m.
    msg_fresh σ m ∧ msg_zhops m)),
    other quality_increases {i} →)
  global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
    in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
```

$\rightarrow (\text{rt } (\sigma \ i)) \sqsubseteq_{\text{dip}} (\text{rt } (\sigma \ \text{nhip}))$ )"

$\langle \text{proof} \rangle$

**lemma par\_rreq\_rrep\_sn\_quality\_increases:**

"opaadv  $i \langle \langle i \ \text{qmsg} \models_A (\lambda \sigma \_. \text{orecvmsg } (\lambda \_. \text{rreq\_rrep\_sn}) \sigma, \text{other } (\lambda \_ \_. \text{True}) \{i\} \rightarrow) \text{globala } (\lambda(\sigma, \_, \sigma')). \text{quality\_increases } (\sigma \ i) (\sigma' \ i)) \rangle \rangle$ "

$\langle \text{proof} \rangle$

**lemma par\_rreq\_rrep\_nsqn\_fresh\_any\_step:**

shows "opaadv  $i \langle \langle i \ \text{qmsg} \models_A (\lambda \sigma \_. \text{orecvmsg } (\lambda \_. \text{rreq\_rrep\_sn}) \sigma, \text{other } (\lambda \_ \_. \text{True}) \{i\} \rightarrow) \text{globala } (\lambda(\sigma, a, \sigma'). \text{anycast } (\text{msg\_fresh } \sigma) a) \rangle \rangle$ "

$\langle \text{proof} \rangle$

**lemma par\_anycast\_msg\_zhops:**

shows "opaadv  $i \langle \langle i \ \text{qmsg} \models_A (\lambda \sigma \_. \text{orecvmsg } (\lambda \_. \text{rreq\_rrep\_sn}) \sigma, \text{other } (\lambda \_ \_. \text{True}) \{i\} \rightarrow) \text{globala } (\lambda(\_, a, \_). \text{anycast } \text{msg\_zhops } a) \rangle \rangle$ "

$\langle \text{proof} \rangle$

## 4.11.2 Lift to nodes

**lemma node\_step\_no\_change\_on\_send\_or\_receive:**

assumes " $((\sigma, \text{NodeS } i \ P \ R), a, (\sigma', \text{NodeS } i' \ P' \ R')) \in \text{onode\_sos}$   
 $(\text{oparp\_sos } i \ (\text{oseqp\_sos } \Gamma_{AODV} \ i) \ (\text{seqp\_sos } \Gamma_{QMSG}))$ "

and " $a \neq \tau$ "

shows " $\sigma' \ i = \sigma \ i$ "

$\langle \text{proof} \rangle$

**lemma node\_nhop\_quality\_increases:**

shows " $\langle i : \text{opaadv } i \langle \langle i \ \text{qmsg} : R \rangle_o \models (\text{otherwith } (\text{op}=) \{i\} \ (\text{oarrivemsg } (\lambda \sigma \ m. \text{msg\_fresh } \sigma \ m \wedge \text{msg\_zhops } m)), \text{other } \text{quality\_increases } \{i\} \rightarrow) \text{global } (\lambda \sigma. \forall \text{dip}. \text{let } \text{nhip} = \text{the } (\text{nhop } (\text{rt } (\sigma \ i)) \ \text{dip}) \text{in } \text{dip} \in \text{vD } (\text{rt } (\sigma \ i)) \cap \text{vD } (\text{rt } (\sigma \ \text{nhip})) \wedge \text{nhip} \neq \text{dip} \rightarrow (\text{rt } (\sigma \ i)) \sqsubseteq_{\text{dip}} (\text{rt } (\sigma \ \text{nhip})) \rangle \rangle$ "

$\langle \text{proof} \rangle$

**lemma node\_quality\_increases:**

" $\langle i : \text{opaadv } i \langle \langle i \ \text{qmsg} : R \rangle_o \models_A (\lambda \sigma \_. \text{oarrivemsg } (\lambda \_. \text{rreq\_rrep\_sn}) \sigma, \text{other } (\lambda \_ \_. \text{True}) \{i\} \rightarrow) \text{globala } (\lambda(\sigma, \_, \sigma'). \text{quality\_increases } (\sigma \ i) (\sigma' \ i)) \rangle \rangle$ "

$\langle \text{proof} \rangle$

**lemma node\_rreq\_rrep\_nsqn\_fresh\_any\_step:**

shows " $\langle i : \text{opaadv } i \langle \langle i \ \text{qmsg} : R \rangle_o \models_A (\lambda \sigma \_. \text{oarrivemsg } (\lambda \_. \text{rreq\_rrep\_sn}) \sigma, \text{other } (\lambda \_ \_. \text{True}) \{i\} \rightarrow) \text{globala } (\lambda(\sigma, a, \sigma'). \text{castmsg } (\text{msg\_fresh } \sigma) a) \rangle \rangle$ "

$\langle \text{proof} \rangle$

**lemma node\_anycast\_msg\_zhops:**

shows " $\langle i : \text{opaadv } i \langle \langle i \ \text{qmsg} : R \rangle_o \models_A (\lambda \sigma \_. \text{oarrivemsg } (\lambda \_. \text{rreq\_rrep\_sn}) \sigma, \text{other } (\lambda \_ \_. \text{True}) \{i\} \rightarrow) \text{globala } (\lambda(\_, a, \_). \text{castmsg } \text{msg\_zhops } a) \rangle \rangle$ "

$\langle \text{proof} \rangle$

**lemma node\_silent\_change\_only:**

shows " $\langle i : \text{opaadv } i \langle \langle i \ \text{qmsg} : R_i \rangle_o \models_A (\lambda \sigma \_. \text{oarrivemsg } (\lambda \_ \_. \text{True}) \sigma, \text{other } (\lambda \_ \_. \text{True}) \{i\} \rightarrow) \text{globala } (\lambda(\sigma, a, \sigma'). a \neq \tau \rightarrow \sigma' \ i = \sigma \ i) \rangle \rangle$ "

$\langle \text{proof} \rangle$



### 4.11.3 Lift to partial networks

```
lemma arrive_rreq_rrep_nsqn_fresh_inc_sn [simp]:
  assumes "oarrivemsg ( $\lambda\sigma m. \text{msg\_fresh } \sigma m \wedge P \sigma m$ )  $\sigma m$ "
  shows "oarrivemsg ( $\lambda_. \text{rreq\_rrep\_sn}$ )  $\sigma m$ "
  <proof>
```

```
lemma opnet_nhop_quality_increases:
  shows "opnet ( $\lambda i. \text{opaodv } i \langle\langle_i \text{qmsg}\rangle\rangle p \models$ 
    (otherwith (op=) (net_tree_ips p)
      (oarrivemsg ( $\lambda\sigma m. \text{msg\_fresh } \sigma m \wedge \text{msg\_zhops } m$ )),
      other quality_increases (net_tree_ips p)  $\rightarrow$ )
    global ( $\lambda\sigma. \forall i \in \text{net\_tree\_ips } p. \forall \text{dip.}$ 
      let nhip = the (nhop (rt ( $\sigma$  i)) dip)
      in dip  $\in vD$  (rt ( $\sigma$  i))  $\cap vD$  (rt ( $\sigma$  nhip))  $\wedge$  nhip  $\neq$  dip
       $\rightarrow$  (rt ( $\sigma$  i))  $\sqsubset_{\text{dip}}$  (rt ( $\sigma$  nhip)))")
  <proof>
```

### 4.11.4 Lift to closed networks

```
lemma onet_nhop_quality_increases:
  shows "oclosed (opnet ( $\lambda i. \text{opaodv } i \langle\langle_i \text{qmsg}\rangle\rangle p$ )
     $\models (\lambda\_ \_ \_. \text{True}, \text{other quality\_increases (net\_tree\_ips } p) \rightarrow)$ 
    global ( $\lambda\sigma. \forall i \in \text{net\_tree\_ips } p. \forall \text{dip.}$ 
      let nhip = the (nhop (rt ( $\sigma$  i)) dip)
      in dip  $\in vD$  (rt ( $\sigma$  i))  $\cap vD$  (rt ( $\sigma$  nhip))  $\wedge$  nhip  $\neq$  dip
       $\rightarrow$  (rt ( $\sigma$  i))  $\sqsubset_{\text{dip}}$  (rt ( $\sigma$  nhip)))")
  (is " $\_ \models (\_, ?U \rightarrow) ?inv$ ")
  <proof>
```

### 4.11.5 Transfer into the standard model

```
interpretation aadv_openproc: openproc paodv opaodv id
  rewrites "aadv_openproc.initmissing = initmissing"
  <proof>
```

```
interpretation aadv_openproc_par_qmsg: openproc_parq paodv opaodv id qmsg
  rewrites "aadv_openproc_par_qmsg.netglobal = netglobal"
  and "aadv_openproc_par_qmsg.initmissing = initmissing"
  <proof>
```

```
lemma net_nhop_quality_increases:
  assumes "wf_net_tree n"
  shows "closed (pnet ( $\lambda i. \text{paodv } i \langle\langle \text{qmsg}\rangle\rangle n \models \text{netglobal}$ 
    ( $\lambda\sigma. \forall i \text{dip. let nhip = the (nhop (rt ( $\sigma$  i)) dip)$ 
      in dip  $\in vD$  (rt ( $\sigma$  i))  $\cap vD$  (rt ( $\sigma$  nhip))  $\wedge$  nhip  $\neq$  dip
       $\rightarrow$  (rt ( $\sigma$  i))  $\sqsubset_{\text{dip}}$  (rt ( $\sigma$  nhip)))")
    (is " $\_ \models \text{netglobal } (\lambda\sigma. \forall i. ?inv \sigma i)$ ")
  <proof>
```

### 4.11.6 Loop freedom of AODV

```
theorem aadv_loop_freedom:
  assumes "wf_net_tree n"
  shows "closed (pnet ( $\lambda i. \text{paodv } i \langle\langle \text{qmsg}\rangle\rangle n \models \text{netglobal } (\lambda\sigma. \forall \text{dip. irrefl } ((\text{rt\_graph } \sigma \text{ dip})^+))$ )")
  <proof>
```

end

# Chapter 5

## Variants A–D: All proposed modifications

This model combines the changes proposed in each of the individual variant models.

### 5.1 Predicates and functions used in the AODV model

```
theory E_Aodv_Data
imports E_All_ABCD
begin
```

#### 5.1.1 Sequence Numbers

Sequence numbers approximate the relative freshness of routing information.

```
definition inc :: "sqn  $\Rightarrow$  sqn"
  where "inc sn  $\equiv$  if sn = 0 then sn else sn + 1"
```

```
lemma less_than_inc [simp]: "x  $\leq$  inc x"
  <proof>
```

```
lemma inc_minus_suc_0 [simp]:
  "inc x - Suc 0 = x"
  <proof>
```

```
lemma inc_never_one' [simp, intro]: "inc x  $\neq$  Suc 0"
  <proof>
```

```
lemma inc_never_one [simp, intro]: "inc x  $\neq$  1"
  <proof>
```

#### 5.1.2 Modelling Routes

A route is a t-tuple,  $(dsn, dsk, flag, hops, nhop)$  where  $dsn$  is the ‘destination sequence number’,  $dsk$  is the ‘destination-sequence-number status’,  $flag$  is the route status,  $hops$  is the number of hops to the destination, and  $nhop$  is the next hop toward the destination.

```
type_synonym r = "sqn  $\times$  k  $\times$  f  $\times$  nat  $\times$  ip"
```

```
definition proj2 :: "r  $\Rightarrow$  sqn" ("π2")
  where "π2  $\equiv$   $\lambda$ (dsn, _, _, _, _). dsn"
```

```
definition proj3 :: "r  $\Rightarrow$  k" ("π3")
  where "π3  $\equiv$   $\lambda$ (_, dsk, _, _, _). dsk"
```

```
definition proj4 :: "r  $\Rightarrow$  f" ("π4")
  where "π4  $\equiv$   $\lambda$ (_, _, flag, _, _). flag"
```

```
definition proj5 :: "r  $\Rightarrow$  nat" ("π5")
  where "π5  $\equiv$   $\lambda$ (_, _, _, hops, _). hops"
```

```
definition proj6 :: "r  $\Rightarrow$  ip" ("π6")
```

where " $\pi_6 \equiv \lambda(\_, \_, \_, \_, \_). \text{nhip}$ ". nhip"

lemma projs [simp]:

" $\pi_2(\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}) = \text{dsn}$ "  
" $\pi_3(\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}) = \text{dsk}$ "  
" $\pi_4(\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}) = \text{flag}$ "  
" $\pi_5(\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}) = \text{hops}$ "  
" $\pi_6(\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}) = \text{nhip}$ "  
<proof>

lemma proj3\_pred [intro]: "[[ P kno; P unk ]  $\implies$  P ( $\pi_3$  x)]"  
<proof>

lemma proj4\_pred [intro]: "[[ P val; P inv ]  $\implies$  P ( $\pi_4$  x)]"  
<proof>

lemma proj6\_pair\_snd [simp]:

fixes dsn' r  
shows " $\pi_6(\text{dsn}', \text{snd}(r)) = \pi_6(r)$ "  
<proof>

### 5.1.3 Routing Tables

Routing tables map ip addresses to route entries.

type\_synonym rt = "ip  $\rightarrow$  r"

syntax

"\_Sigma\_route" :: "rt  $\Rightarrow$  ip  $\rightarrow$  r" ("sigma\_route'(\_, \_)'")

translations

"sigma\_route(rt, dip)"  $\Rightarrow$  "rt dip"

definition sqn :: "rt  $\Rightarrow$  ip  $\Rightarrow$  sqn"

where "sqn rt dip  $\equiv$  case sigma\_route(rt, dip) of Some r  $\Rightarrow$   $\pi_2(r)$  | None  $\Rightarrow$  0"

definition sqnf :: "rt  $\Rightarrow$  ip  $\Rightarrow$  k"

where "sqnf rt dip  $\equiv$  case sigma\_route(rt, dip) of Some r  $\Rightarrow$   $\pi_3(r)$  | None  $\Rightarrow$  unk"

abbreviation flag :: "rt  $\Rightarrow$  ip  $\rightarrow$  f"

where "flag rt dip  $\equiv$  map\_option  $\pi_4$  (sigma\_route(rt, dip))"

abbreviation dhops :: "rt  $\Rightarrow$  ip  $\rightarrow$  nat"

where "dhops rt dip  $\equiv$  map\_option  $\pi_5$  (sigma\_route(rt, dip))"

abbreviation nhop :: "rt  $\Rightarrow$  ip  $\rightarrow$  ip"

where "nhop rt dip  $\equiv$  map\_option  $\pi_6$  (sigma\_route(rt, dip))"

definition vD :: "rt  $\Rightarrow$  ip set"

where "vD rt  $\equiv$  {dip. flag rt dip = Some val}"

definition iD :: "rt  $\Rightarrow$  ip set"

where "iD rt  $\equiv$  {dip. flag rt dip = Some inv}"

definition kD :: "rt  $\Rightarrow$  ip set"

where "kD rt  $\equiv$  {dip. rt dip  $\neq$  None}"

lemma kD\_is\_vD\_and\_iD: "kD rt = vD rt  $\cup$  iD rt"

<proof>

lemma vD\_iD\_gives\_kD [simp]:

" $\bigwedge$  ip rt. ip  $\in$  vD rt  $\implies$  ip  $\in$  kD rt"

" $\bigwedge$  ip rt. ip  $\in$  iD rt  $\implies$  ip  $\in$  kD rt"

<proof>

```

lemma kD_Some [dest]:
  fixes dip rt
  assumes "dip ∈ kD rt"
  shows "∃ dsn dsk flag hops nhip.
    σroute(rt, dip) = Some (dsn, dsk, flag, hops, nhip)"
  ⟨proof⟩

lemma kD_None [dest]:
  fixes dip rt
  assumes "dip ∉ kD rt"
  shows "σroute(rt, dip) = None"
  ⟨proof⟩

lemma vD_Some [dest]:
  fixes dip rt
  assumes "dip ∈ vD rt"
  shows "∃ dsn dsk hops nhip.
    σroute(rt, dip) = Some (dsn, dsk, val, hops, nhip)"
  ⟨proof⟩

lemma vD_empty [simp]: "vD Map.empty = {}"
  ⟨proof⟩

lemma iD_Some [dest]:
  fixes dip rt
  assumes "dip ∈ iD rt"
  shows "∃ dsn dsk hops nhip.
    σroute(rt, dip) = Some (dsn, dsk, inv, hops, nhip)"
  ⟨proof⟩

lemma val_is_vD [elim]:
  fixes ip rt
  assumes "ip ∈ kD(rt)"
  and "the (flag rt ip) = val"
  shows "ip ∈ vD(rt)"
  ⟨proof⟩

lemma inv_is_iD [elim]:
  fixes ip rt
  assumes "ip ∈ kD(rt)"
  and "the (flag rt ip) = inv"
  shows "ip ∈ iD(rt)"
  ⟨proof⟩

lemma iD_flag_is_inv [elim, simp]:
  fixes ip rt
  assumes "ip ∈ iD(rt)"
  shows "the (flag rt ip) = inv"
  ⟨proof⟩

lemma kD_but_not_vD_is_iD [elim]:
  fixes ip rt
  assumes "ip ∈ kD(rt)"
  and "ip ∉ vD(rt)"
  shows "ip ∈ iD(rt)"
  ⟨proof⟩

lemma vD_or_iD [elim]:
  fixes ip rt
  assumes "ip ∈ kD(rt)"
  and "ip ∈ vD(rt) ⇒ P rt ip"
  and "ip ∈ iD(rt) ⇒ P rt ip"
  shows "P rt ip"
  ⟨proof⟩

```

lemma proj5\_eq\_dhops: " $\bigwedge \text{dip rt. dip} \in \text{kD}(\text{rt}) \implies \pi_5(\text{the } (\text{rt dip})) = \text{the } (\text{dhops rt dip})$ "  
 <proof>

lemma proj4\_eq\_flag: " $\bigwedge \text{dip rt. dip} \in \text{kD}(\text{rt}) \implies \pi_4(\text{the } (\text{rt dip})) = \text{the } (\text{flag rt dip})$ "  
 <proof>

lemma proj2\_eq\_sqn: " $\bigwedge \text{dip rt. dip} \in \text{kD}(\text{rt}) \implies \pi_2(\text{the } (\text{rt dip})) = \text{sqn rt dip}$ "  
 <proof>

lemma kD\_sqnf\_is\_proj3 [simp]:  
 " $\bigwedge \text{ip rt. ip} \in \text{kD}(\text{rt}) \implies \text{sqnf rt ip} = \pi_3(\text{the } (\text{rt ip}))$ "  
 <proof>

lemma vD\_flag\_val [simp]:  
 " $\bigwedge \text{dip rt. dip} \in \text{vD } (\text{rt}) \implies \text{the } (\text{flag rt dip}) = \text{val}$ "  
 <proof>

lemma kD\_update [simp]:  
 " $\bigwedge \text{rt nip v. kD } (\text{rt}(\text{nip} \mapsto \text{v})) = \text{insert nip } (\text{kD rt})$ "  
 <proof>

lemma kD\_empty [simp]: " $\text{kD Map.empty} = \{\}$ "  
 <proof>

lemma ip\_equal\_or\_known [elim]:  
 fixes rt ip ip'  
 assumes " $\text{ip} = \text{ip}' \vee \text{ip} \in \text{kD}(\text{rt})$ "  
 and " $\text{ip} = \text{ip}' \implies P \text{ rt ip ip}'$ "  
 and " $\llbracket \text{ip} \neq \text{ip}'; \text{ip} \in \text{kD}(\text{rt}) \rrbracket \implies P \text{ rt ip ip}'$ "  
 shows " $P \text{ rt ip ip}'$ "  
 <proof>

### 5.1.4 Updating Routing Tables

Routing table entries are modified through explicit functions. The properties of these functions are important in invariant proofs.

#### Updating route entries

lemma in\_kD\_case [simp]:  
 fixes dip rt  
 assumes " $\text{dip} \in \text{kD}(\text{rt})$ "  
 shows " $(\text{case rt dip of None} \Rightarrow \text{en} \mid \text{Some } r \Rightarrow \text{es } r) = \text{es } (\text{the } (\text{rt dip}))$ "  
 <proof>

lemma not\_in\_kD\_case [simp]:  
 fixes dip rt  
 assumes " $\text{dip} \notin \text{kD}(\text{rt})$ "  
 shows " $(\text{case rt dip of None} \Rightarrow \text{en} \mid \text{Some } r \Rightarrow \text{es } r) = \text{en}$ "  
 <proof>

lemma rt\_Some\_sqn [dest]:  
 fixes rt and ip dsn dsk flag hops nhip  
 assumes " $\text{rt ip} = \text{Some } (\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip})$ "  
 shows " $\text{sqn rt ip} = \text{dsn}$ "  
 <proof>

lemma not\_kD\_sqn [simp]:  
 fixes dip rt  
 assumes " $\text{dip} \notin \text{kD}(\text{rt})$ "  
 shows " $\text{sqn rt dip} = 0$ "  
 <proof>

```

definition update_arg_wf :: "r ⇒ bool"
where "update_arg_wf r ≡ π4(r) = val ∧
      (π2(r) = 0) = (π3(r) = unk) ∧
      (π3(r) = unk → π5(r) = 1)"

```

```

lemma update_arg_wf_gives_cases:
"∧r. update_arg_wf r ⇒ (π2(r) = 0) = (π3(r) = unk)"
⟨proof⟩

```

```

lemma update_arg_wf_tuples [simp]:
"∧nhip. update_arg_wf (0, unk, val, Suc 0, nhip)"
"∧n hops nhip. update_arg_wf (Suc n, kno, val, hops, nhip)"
⟨proof⟩

```

```

lemma update_arg_wf_tuples' [elim]:
"∧n hops nhip. Suc 0 ≤ n ⇒ update_arg_wf (n, kno, val, hops, nhip)"
⟨proof⟩

```

```

lemma wf_r_cases [intro]:
  fixes P r
  assumes "update_arg_wf r"
    and c1: "∧nhip. P (0, unk, val, Suc 0, nhip)"
    and c2: "∧dsn hops nhip. dsn > 0 ⇒ P (dsn, kno, val, hops, nhip)"
  shows "P r"
⟨proof⟩

```

```

definition update :: "rt ⇒ ip ⇒ r ⇒ rt"
where
"update rt ip r ≡
  case σroute(rt, ip) of
    None ⇒ rt (ip ↦ r)
  | Some s ⇒
    if π2(s) < π2(r) then rt (ip ↦ r)
    else if π2(s) = π2(r) ∧ (π5(s) > π5(r) ∨ π4(s) = inv)
      then rt (ip ↦ r)
    else if π3(r) = unk
      then rt (ip ↦ (π2(s), snd (r)))
    else rt (ip ↦ s)"

```

```

lemma update_simps [simp]:
  fixes r s nrt nr' ns rt ip
  defines "s ≡ the σroute(rt, ip)"
    and "nr' ≡ (π2(s), π3(r), π4(r), π5(r), π6(r))"
  shows
"[[ip ∉ kD(rt)]] ⇒ update rt ip r = rt (ip ↦ r)"
"[[ip ∈ kD(rt); sqn rt ip < π2(r)]] ⇒ update rt ip r = rt (ip ↦ r)"
"[[ip ∈ kD(rt); sqn rt ip = π2(r);
  the (dhops rt ip) > π5(r)]] ⇒ update rt ip r = rt (ip ↦ r)"
"[[ip ∈ kD(rt); sqn rt ip = π2(r);
  flag rt ip = Some inv]] ⇒ update rt ip r = rt (ip ↦ r)"
"[[ip ∈ kD(rt); π3(r) = unk; (π2(r) = 0) = (π3(r) = unk)]] ⇒ update rt ip r = rt (ip ↦ nr' )"
"[[ip ∈ kD(rt); sqn rt ip ≥ π2(r); π3(r) = kno;
  sqn rt ip = π2(r) ⇒ the (dhops rt ip) ≤ π5(r) ∧ the (flag rt ip) = val ]]]
  ⇒ update rt ip r = rt (ip ↦ s)"
⟨proof⟩

```

```

lemma update_cases [elim]:
  assumes "(π2(r) = 0) = (π3(r) = unk)"
    and c1: "[[ip ∉ kD(rt)]] ⇒ P (rt (ip ↦ r))"

    and c2: "[[ip ∈ kD(rt); sqn rt ip < π2(r)]]
  ⇒ P (rt (ip ↦ r))"
    and c3: "[[ip ∈ kD(rt); sqn rt ip = π2(r); the (dhops rt ip) > π5(r)]]
  ⇒ P (rt (ip ↦ r))"

```

```

and c4: "[[ip ∈ kD(rt); sqn rt ip = π2(r); the (flag rt ip) = inv]]
  ⇒ P (rt (ip ↦ r ))"
and c5: "[[ip ∈ kD(rt); π3(r) = unk]]
  ⇒ P (rt (ip ↦ (π2(the σroute(rt, ip)), π3(r),
  π4(r), π5(r), π6(r))))"
and c6: "[[ip ∈ kD(rt); sqn rt ip ≥ π2(r); π3(r) = kno;
  sqn rt ip = π2(r) ⇒ the (dhops rt ip) ≤ π5(r) ∧ the (flag rt ip) = val]]
  ⇒ P (rt (ip ↦ the σroute(rt, ip)))"
shows "(P (update rt ip r))"
⟨proof⟩

```

lemma update\_cases\_kD:

```

assumes "(π2(r) = 0) = (π3(r) = unk)"
and "ip ∈ kD(rt)"
and c2: "sqn rt ip < π2(r) ⇒ P (rt (ip ↦ r ))"
and c3: "[[sqn rt ip = π2(r); the (dhops rt ip) > π5(r)]]
  ⇒ P (rt (ip ↦ r ))"
and c4: "[[sqn rt ip = π2(r); the (flag rt ip) = inv]]
  ⇒ P (rt (ip ↦ r ))"
and c5: "π3(r) = unk ⇒ P (rt (ip ↦ (π2(the σroute(rt, ip)), π3(r),
  π4(r), π5(r), π6(r))))"
and c6: "[[sqn rt ip ≥ π2(r); π3(r) = kno;
  sqn rt ip = π2(r) ⇒ the (dhops rt ip) ≤ π5(r) ∧ the (flag rt ip) = val]]
  ⇒ P (rt (ip ↦ the σroute(rt, ip)))"
shows "(P (update rt ip r))"
⟨proof⟩

```

lemma in\_kD\_after\_update [simp]:

```

fixes rt nip dsn dsk flag hops nhip pre
shows "kD (update rt nip (dsn, dsk, flag, hops, nhip)) = insert nip (kD rt)"
⟨proof⟩

```

lemma nhop\_of\_update [simp]:

```

fixes rt dip dsn dsk flag hops nhip
assumes "rt ≠ update rt dip (dsn, dsk, flag, hops, nhip)"
shows "the (nhop (update rt dip (dsn, dsk, flag, hops, nhip)) dip) = nhip"
⟨proof⟩

```

lemma sqn\_if\_updated:

```

fixes rip v rt ip
shows "sqn (λx. if x = rip then Some v else rt x) ip
  = (if ip = rip then π2(v) else sqn rt ip)"
⟨proof⟩

```

lemma update\_sqn [simp]:

```

fixes rt dip rip dsn dsk hops nhip
assumes "(dsn = 0) = (dsk = unk)"
shows "sqn rt dip ≤ sqn (update rt rip (dsn, dsk, val, hops, nhip)) dip"
⟨proof⟩

```

lemma sqn\_update\_bigger [simp]:

```

fixes rt ip ip' dsn dsk flag hops nhip
assumes "1 ≤ hops"
shows "sqn rt ip ≤ sqn (update rt ip' (dsn, dsk, flag, hops, nhip)) ip"
⟨proof⟩

```

lemma dhops\_update [intro]:

```

fixes rt dsn dsk flag hops ip rip nhip
assumes ex: "∀ip∈kD rt. the (dhops rt ip) ≥ 1"
and ip: "(ip = rip ∧ Suc 0 ≤ hops) ∨ (ip ≠ rip ∧ ip∈kD rt)"
shows "Suc 0 ≤ the (dhops (update rt rip (dsn, dsk, flag, hops, nhip)) ip)"
⟨proof⟩

```

lemma update\_another [simp]:

fixes dip ip rt dsn dsk flag hops nhip  
 assumes "ip  $\neq$  dip"  
 shows "(update rt dip (dsn, dsk, flag, hops, nhip)) ip = rt ip"  
 <proof>

**lemma** nhop\_update\_another [simp]:  
 fixes dip ip rt dsn dsk flag hops nhip  
 assumes "ip  $\neq$  dip"  
 shows "nhop (update rt dip (dsn, dsk, flag, hops, nhip)) ip = nhop rt ip"  
 <proof>

**lemma** dhops\_update\_another [simp]:  
 fixes dip ip rt dsn dsk flag hops nhip  
 assumes "ip  $\neq$  dip"  
 shows "dhops (update rt dip (dsn, dsk, flag, hops, nhip)) ip = dhops rt ip"  
 <proof>

**lemma** sqn\_update\_same [simp]:  
 " $\bigwedge$ rt ip dsn dsk flag hops nhip. sqn (rt(ip  $\mapsto$  v)) ip =  $\pi_2$ (v)"  
 <proof>

**lemma** dhops\_update\_changed [simp]:  
 fixes rt dip osn hops nhip  
 assumes "rt  $\neq$  update rt dip (osn, kno, val, hops, nhip)"  
 shows "the (dhops (update rt dip (osn, kno, val, hops, nhip)) dip) = hops"  
 <proof>

**lemma** nhop\_update\_unk\_val [simp]:  
 " $\bigwedge$ rt dip ip dsn hops.  
 the (nhop (update rt dip (dsn, unk, val, hops, ip)) dip) = ip"  
 <proof>

**lemma** nhop\_update\_changed [simp]:  
 fixes rt dip dsn dsk flg hops sip  
 assumes "update rt dip (dsn, dsk, flg, hops, sip)  $\neq$  rt"  
 shows "the (nhop (update rt dip (dsn, dsk, flg, hops, sip)) dip) = sip"  
 <proof>

**lemma** update\_rt\_split\_asm:  
 " $\bigwedge$ rt ip dsn dsk flag hops sip.  
 P (update rt ip (dsn, dsk, flag, hops, sip))  
 =  
 ( $\neg$ (rt = update rt ip (dsn, dsk, flag, hops, sip)  $\wedge$   $\neg$ P rt  
 $\vee$  rt  $\neq$  update rt ip (dsn, dsk, flag, hops, sip)  
 $\wedge$   $\neg$ P (update rt ip (dsn, dsk, flag, hops, sip))))"  
 <proof>

**lemma** sqn\_update [simp]: " $\bigwedge$ rt dip dsn flg hops sip.  
 rt  $\neq$  update rt dip (dsn, kno, flg, hops, sip)  
 $\implies$  sqn (update rt dip (dsn, kno, flg, hops, sip)) dip = dsn"  
 <proof>

**lemma** sqnf\_update [simp]: " $\bigwedge$ rt dip dsn dsk flg hops sip.  
 rt  $\neq$  update rt dip (dsn, dsk, flg, hops, sip)  
 $\implies$  sqnf (update rt dip (dsn, dsk, flg, hops, sip)) dip = dsk"  
 <proof>

**lemma** update\_kno\_dsn\_greater\_zero:  
 " $\bigwedge$ rt dip ip dsn hops.  $1 \leq$  dsn  $\implies$   $1 \leq$  (sqn (update rt dip (dsn, kno, val, hops, ip)) dip)"  
 <proof>

**lemma** proj3\_update [simp]: " $\bigwedge$ rt dip dsn dsk flg hops sip.  
 rt  $\neq$  update rt dip (dsn, dsk, flg, hops, sip)  
 $\implies$   $\pi_3$ (the (update rt dip (dsn, dsk, flg, hops, sip) dip)) = dsk"



*<proof>*

**lemma** *nhop\_update\_changed\_kno\_val* [simp]: " $\wedge$ rt ip dsn dsk hops nhip.  
rt  $\neq$  update rt ip (dsn, kno, val, hops, nhip)  
 $\implies$  the (nhop (update rt ip (dsn, kno, val, hops, nhip)) ip) = nhip"  
*<proof>*

**lemma** *flag\_update* [simp]: " $\wedge$ rt dip dsn flg hops sip.  
rt  $\neq$  update rt dip (dsn, kno, flg, hops, sip)  
 $\implies$  the (flag (update rt dip (dsn, kno, flg, hops, sip)) dip) = flg"  
*<proof>*

**lemma** *the\_flag\_Some* [dest!]:  
fixes ip rt  
assumes "the (flag rt ip) = x"  
and "ip  $\in$  kD rt"  
shows "flag rt ip = Some x"  
*<proof>*

**lemma** *kD\_update\_unchanged* [dest]:  
fixes rt dip dsn dsk flag hops nhip  
assumes "rt = update rt dip (dsn, dsk, flag, hops, nhip)"  
shows "dip  $\in$  kD(rt)"  
*<proof>*

**lemma** *nhop\_update* [simp]: " $\wedge$ rt dip dsn dsk flg hops sip.  
rt  $\neq$  update rt dip (dsn, dsk, flg, hops, sip)  
 $\implies$  the (nhop (update rt dip (dsn, dsk, flg, hops, sip)) dip) = sip"  
*<proof>*

**lemma** *sqn\_update\_another* [simp]:  
fixes dip ip rt dsn dsk flag hops nhip  
assumes "ip  $\neq$  dip"  
shows "sqn (update rt dip (dsn, dsk, flag, hops, nhip)) ip = sqn rt ip"  
*<proof>*

**lemma** *sqnf\_update\_another* [simp]:  
fixes dip ip rt dsn dsk flag hops nhip  
assumes "ip  $\neq$  dip"  
shows "sqnf (update rt dip (dsn, dsk, flag, hops, nhip)) ip = sqnf rt ip"  
*<proof>*

**lemma** *vD\_update\_val* [dest]:  
" $\wedge$ dip rt dip' dsn dsk hops nhip.  
dip  $\in$  vD(update rt dip' (dsn, dsk, val, hops, nhip))  $\implies$  (dip  $\in$  vD(rt)  $\vee$  dip=dip)"  
*<proof>*

## Invalidating route entries

**definition** *invalidate* :: "rt  $\Rightarrow$  (ip  $\rightarrow$  sqn)  $\Rightarrow$  rt"  
**where** "invalidate rt dests  $\equiv$   
 $\lambda$ ip. case (rt ip, dests ip) of  
  (None, \_)  $\Rightarrow$  None  
  | (Some s, None)  $\Rightarrow$  Some s  
  | (Some (\_, dsk, \_, hops, nhip), Some rsn)  $\Rightarrow$   
    Some (rsn, dsk, inv, hops, nhip)"

**lemma** *proj3\_invalidate* [simp]:  
" $\wedge$ dip.  $\pi_3$ (the ((invalidate rt dests) dip)) =  $\pi_3$ (the (rt dip))"  
*<proof>*

**lemma** *proj5\_invalidate* [simp]:  
" $\wedge$ dip.  $\pi_5$ (the ((invalidate rt dests) dip)) =  $\pi_5$ (the (rt dip))"  
*<proof>*

```

lemma proj6_invalidate [simp]:
  " $\bigwedge$ dip.  $\pi_6(\text{the } ((\text{invalidate } rt \text{ dests}) \text{ dip})) = \pi_6(\text{the } (rt \text{ dip}))"$ 
  <proof>

```

### 5.1.5 Route Requests

```

lemma invalidate_kD_inv [simp]:
  " $\bigwedge$ rt dests.  $kD (\text{invalidate } rt \text{ dests}) = kD \text{ rt}"$ 
  <proof>

```

```

lemma invalidate_sqn:
  fixes rt dip dests
  assumes " $\forall$ rsn.  $\text{dests } dip = \text{Some } rsn \longrightarrow \text{sqn } rt \text{ dip} \leq rsn"$ 
  shows " $\text{sqn } rt \text{ dip} \leq \text{sqn } (\text{invalidate } rt \text{ dests}) \text{ dip}"$ 
  <proof>

```

```

lemma sqn_invalidate_in_dests [simp]:
  fixes dests ipa rsn rt
  assumes " $\text{dests } ipa = \text{Some } rsn"$ 
    and " $ipa \in kD(rt)"$ 
  shows " $\text{sqn } (\text{invalidate } rt \text{ dests}) \text{ ipa} = rsn"$ 
  <proof>

```

```

lemma dhops_invalidate [simp]:
  " $\bigwedge$ dip.  $\text{the } (\text{dhops } (\text{invalidate } rt \text{ dests}) \text{ dip}) = \text{the } (\text{dhops } rt \text{ dip})"$ 
  <proof>

```

```

lemma sqnf_invalidate [simp]:
  " $\bigwedge$ dip.  $\text{sqnf } (\text{invalidate } (rt \ \xi) \ (\text{dests } \xi)) \text{ dip} = \text{sqnf } (rt \ \xi) \text{ dip}"$ 
  <proof>

```

```

lemma nhop_invalidate [simp]:
  " $\bigwedge$ dip.  $\text{the } (\text{nhop } (\text{invalidate } (rt \ \xi) \ (\text{dests } \xi)) \text{ dip}) = \text{the } (\text{nhop } (rt \ \xi) \text{ dip})"$ 
  <proof>

```

```

lemma invalidate_other [simp]:
  fixes rt dests dip
  assumes " $dip \notin \text{dom}(\text{dests})"$ 
  shows " $\text{invalidate } rt \text{ dests } dip = rt \text{ dip}"$ 
  <proof>

```

```

lemma invalidate_none [simp]:
  fixes rt dests dip
  assumes " $dip \notin kD(rt)"$ 
  shows " $\text{invalidate } rt \text{ dests } dip = \text{None}"$ 
  <proof>

```

```

lemma vD_invalidate_vD_not_dests:
  " $\bigwedge$ dip rt dests.  $dip \in vD(\text{invalidate } rt \text{ dests}) \implies dip \in vD(rt) \wedge \text{dests } dip = \text{None}"$ 
  <proof>

```

```

lemma sqn_invalidate_not_in_dests [simp]:
  fixes dests dip rt
  assumes " $dip \notin \text{dom}(\text{dests})"$ 
  shows " $\text{sqn } (\text{invalidate } rt \text{ dests}) \text{ dip} = \text{sqn } rt \text{ dip}"$ 
  <proof>

```

```

lemma invalidate_changes:
  fixes rt dests dip dsn dsk flag hops nhop pre
  assumes " $\text{invalidate } rt \text{ dests } dip = \text{Some } (dsn, dsk, flag, hops, nhop)"$ 
  shows "  $dsn = (\text{case } \text{dests } dip \text{ of } \text{None} \Rightarrow \pi_2(\text{the } (rt \text{ dip})) \mid \text{Some } rsn \Rightarrow rsn)$ 
     $\wedge dsk = \pi_3(\text{the } (rt \text{ dip}))$ 
     $\wedge flag = (\text{if } \text{dests } dip = \text{None} \text{ then } \pi_4(\text{the } (rt \text{ dip})) \text{ else } inv)$ "

```

```

    ∧ hops = π5(the (rt dip))
    ∧ nhip = π6(the (rt dip))"
⟨proof⟩

```

```

lemma proj3_inv: "∧dip rt dests. dip∈kD (rt)
    ⇒ π3(the (invalidate rt dests dip)) = π3(the (rt dip))"
⟨proof⟩

```

```

lemma dests_iD_invalidate [simp]:
  assumes "dests ip = Some rsn"
    and "ip∈kD(rt)"
  shows "ip∈iD(invalidate rt dests)"
⟨proof⟩

```

### 5.1.6 Queued Packets

Functions for sending data packets.

```

type_synonym store = "ip → (p × data list)"

```

```

definition sigma_queue :: "store ⇒ ip ⇒ data list" ("σqueue'(_, _)'")
  where "σqueue(store, dip) ≡ case store dip of None ⇒ [] | Some (p, q) ⇒ q"

```

```

definition qD :: "store ⇒ ip set"
  where "qD ≡ dom"

```

```

definition add :: "data ⇒ ip ⇒ store ⇒ store"
  where "add d dip store ≡ case store dip of
    None ⇒ store (dip ↦ (req, [d]))
    | Some (p, q) ⇒ store (dip ↦ (p, q @ [d]))"

```

```

lemma qD_add [simp]:
  fixes d dip store
  shows "qD(add d dip store) = insert dip (qD store)"
⟨proof⟩

```

```

definition drop :: "ip ⇒ store → store"
  where "drop dip store ≡
    map_option (λ(p, q). if tl q = [] then store (dip := None)
      else store (dip ↦ (p, tl q))) (store dip)"

```

```

definition sigma_p_flag :: "store ⇒ ip → p" ("σp-flag'(_, _)'")
  where "σp-flag(store, dip) ≡ map_option fst (store dip)"

```

```

definition unsetRRF :: "store ⇒ ip ⇒ store"
  where "unsetRRF store dip ≡ case store dip of
    None ⇒ store
    | Some (p, q) ⇒ store (dip ↦ (noreq, q))"

```

```

definition setRRF :: "store ⇒ (ip → sqn) ⇒ store"
  where "setRRF store dests ≡ λdip. if dests dip = None then store dip
    else map_option (λ(_, q). (req, q)) (store dip)"

```

### 5.1.7 Comparison with the original technical report

The major differences with the AODV technical report of Fehnker et al are:

1. *nhop* is partial, thus a ‘*the*’ is needed, similarly for *dhops* and *addpreRT*.
2. *precs* is partial.
3.  $\sigma_{p\text{-flag}}(\text{store}, \text{dip})$  is partial.
4. The routing table (*rt*) is modelled as a map ( $\text{ip} \Rightarrow r \text{ option}$ ) rather than a set of 7-tuples, likewise, the *r* is a 6-tuple rather than a 7-tuple, i.e., the destination ip-address (*dip*) is taken from the argument to the

function, rather than a part of the result. Well-definedness then follows from the structure of the type and more related facts are available automatically, rather than having to be acquired through tedious proofs.

5. Similar remarks hold for the *dests* mapping passed to *invalidate*, and *store*.

end

## 5.2 AODV protocol messages

```
theory E_Aodv_Message
imports E_All_ABCD
begin
```

```
datatype msg =
  Rreq nat ip sqn k ip sqn ip bool
  | Rrep nat ip sqn ip ip
  | Rerr "ip  $\rightarrow$  sqn" ip
  | Newpkt data ip
  | Pkt data ip ip
```

```
instantiation msg :: msg
```

```
begin
```

```
  definition newpkt_def [simp]: "newpkt  $\equiv$   $\lambda$ (d, dip). Newpkt d dip"
```

```
  definition eq_newpkt_def: "eq_newpkt m  $\equiv$  case m of Newpkt d dip  $\Rightarrow$  True | _  $\Rightarrow$  False"
```

```
  instance <proof>
```

```
end
```

The *msg* type models the different messages used within AODV. The instantiation as a *msg* is a technicality due to the special treatment of *newpkt* messages in the AWN SOS rules. This use of classes allows a clean separation of the AWN-specific definitions and these AODV-specific definitions.

```
definition rreq :: "nat  $\times$  ip  $\times$  sqn  $\times$  k  $\times$  ip  $\times$  sqn  $\times$  ip  $\times$  bool  $\Rightarrow$  msg"
```

```
  where "rreq  $\equiv$   $\lambda$ (hops, dip, dsn, dsk, oip, osn, sip, handled).
```

```
    Rreq hops dip dsn dsk oip osn sip handled"
```

```
lemma rreq_simp [simp]:
```

```
  "rreq(hops, dip, dsn, dsk, oip, osn, sip, handled) = Rreq hops dip dsn dsk oip osn sip handled"
```

```
<proof>
```

```
definition rrep :: "nat  $\times$  ip  $\times$  sqn  $\times$  ip  $\times$  ip  $\Rightarrow$  msg"
```

```
  where "rrep  $\equiv$   $\lambda$ (hops, dip, dsn, oip, sip). Rrep hops dip dsn oip sip"
```

```
lemma rrep_simp [simp]:
```

```
  "rrep(hops, dip, dsn, oip, sip) = Rrep hops dip dsn oip sip"
```

```
<proof>
```

```
definition rerr :: "(ip  $\rightarrow$  sqn)  $\times$  ip  $\Rightarrow$  msg"
```

```
  where "rerr  $\equiv$   $\lambda$ (dests, sip). Rerr dests sip"
```

```
lemma rerr_simp [simp]:
```

```
  "rerr(dests, sip) = Rerr dests sip"
```

```
<proof>
```

```
lemma not_eq_newpkt_rreq [simp]: " $\neg$ eq_newpkt (Rreq hops dip dsn dsk oip osn sip handled)"
```

```
<proof>
```

```
lemma not_eq_newpkt_rrep [simp]: " $\neg$ eq_newpkt (Rrep hops dip dsn oip sip)"
```

```
<proof>
```

```
lemma not_eq_newpkt_rerr [simp]: " $\neg$ eq_newpkt (Rerr dests sip)"
```

```
<proof>
```

```
lemma not_eq_newpkt_pkt [simp]: " $\neg$ eq_newpkt (Pkt d dip sip)"
```

⟨proof⟩

```
definition pkt :: "data × ip × ip ⇒ msg"
  where "pkt ≡ λ(d, dip, sip). Pkt d dip sip"
```

```
lemma pkt_simp [simp]:
  "pkt(d, dip, sip) = Pkt d dip sip"
  ⟨proof⟩
```

end

## 5.3 The AODV protocol

```
theory E_Aodv
imports E_Aodv_Data E_Aodv_Message
  "../.../AWN/AWN_SOS_Labels" "../.../AWN/AWN_Invariants"
begin
```

### 5.3.1 Data state

```
record state =
  ip      :: "ip"
  sn      :: "sqn"
  rt      :: "rt"
  rreqs   :: "(ip × sqn) set"
  store   :: "store"

  msg     :: "msg"
  data    :: "data"
  dests   :: "ip → sqn"

  dip     :: "ip"
  oip     :: "ip"
  hops    :: "nat"
  dsn     :: "sqn"
  dsk     :: "k"
  osn     :: "sqn"
  sip     :: "ip"
  handled :: "bool"

abbreviation aodv_init :: "ip ⇒ state"
where "aodv_init i ≡ (|
  ip = i,
  sn = 1,
  rt = empty,
  rreqs = {},
  store = empty,

  msg = (SOME x. True),
  data = (SOME x. True),
  dests = (SOME x. True),

  dip = (SOME x. True),
  oip = (SOME x. True),
  hops = (SOME x. True),
  dsn = (SOME x. True),
  dsk = (SOME x. True),
  osn = (SOME x. True),
  sip = (SOME x. x ≠ i),
  handled = (SOME x. True)
|)"
```

```
lemma some_neq_not_eq [simp]: "¬((SOME x :: nat. x ≠ i) = i)"
  ⟨proof⟩
```

**definition** `clear_locals` :: "state  $\Rightarrow$  state"

```
where "clear_locals  $\xi$  =  $\xi$  ( $\mid$ 
  msg      := (SOME x. True),
  data     := (SOME x. True),
  dests    := (SOME x. True),

  dip      := (SOME x. True),
  oip      := (SOME x. True),
  hops     := (SOME x. True),
  dsn      := (SOME x. True),
  dsk      := (SOME x. True),
  osn      := (SOME x. True),
  sip      := (SOME x. x  $\neq$  ip  $\xi$ ),
  handled := (SOME x. True)
 $\mid$ )"
```

**lemma** `clear_locals_sip_not_ip` [simp]: " $\neg$ (sip (clear\_locals  $\xi$ ) = ip  $\xi$ )"  
 <proof>

**lemma** `clear_locals_but_not_globals` [simp]:  
 "ip (clear\_locals  $\xi$ ) = ip  $\xi$ "  
 "sn (clear\_locals  $\xi$ ) = sn  $\xi$ "  
 "rt (clear\_locals  $\xi$ ) = rt  $\xi$ "  
 "rreqs (clear\_locals  $\xi$ ) = rreqs  $\xi$ "  
 "store (clear\_locals  $\xi$ ) = store  $\xi$ "  
 <proof>

### 5.3.2 Auxilliary message handling definitions

**definition** `is_newpkt`  
 where "is\_newpkt  $\xi \equiv$  case msg  $\xi$  of  
     Newpkt data' dip'  $\Rightarrow$  {  $\xi$ (data := data', dip := dip') }  
     | \_  $\Rightarrow$  {}"

**definition** `is_pkt`  
 where "is\_pkt  $\xi \equiv$  case msg  $\xi$  of  
     Pkt data' dip' oip'  $\Rightarrow$  {  $\xi$ (data := data', dip := dip', oip := oip' ) }  
     | \_  $\Rightarrow$  {}"

**definition** `is_rreq`  
 where "is\_rreq  $\xi \equiv$  case msg  $\xi$  of  
     Rreq hops' dip' dsn' dsk' oip' osn' sip' handled'  $\Rightarrow$   
     {  $\xi$ ( hops := hops', dip := dip', dsn := dsn',  
        dsk := dsk', oip := oip', osn := osn', sip := sip',  
        handled := handled' ) }  
     | \_  $\Rightarrow$  {}"

**lemma** `is_rreq_asm` [dest!]:  
 assumes " $\xi' \in$  is\_rreq  $\xi$ "  
 shows " $(\exists$  hops' dip' dsn' dsk' oip' osn' sip' handled'.  
     msg  $\xi =$  Rreq hops' dip' dsn' dsk' oip' osn' sip' handled'  $\wedge$   
      $\xi' = \xi$ ( hops := hops', dip := dip', dsn := dsn',  
        dsk := dsk', oip := oip', osn := osn', sip := sip',  
        handled := handled' ))"  
 <proof>

**definition** `is_rrep`  
 where "is\_rrep  $\xi \equiv$  case msg  $\xi$  of  
     Rrep hops' dip' dsn' oip' sip'  $\Rightarrow$   
     {  $\xi$ ( hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' ) }  
     | \_  $\Rightarrow$  {}"

**lemma** `is_rrep_asm` [dest!]:

```

assumes "ξ' ∈ is_rrep ξ"
shows "(∃ hops' dip' dsn' oip' sip'.
  msg ξ = Rrep hops' dip' dsn' oip' sip' ∧
  ξ' = ξ(| hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' |))"
⟨proof⟩

```

**definition is\_rerr**

```

where "is_rerr ξ ≡ case msg ξ of
  Rerr dests' sip' ⇒ { ξ(| dests := dests', sip := sip' |) }
  | _ ⇒ {}"

```

**lemma is\_rerr\_asm [dest!]:**

```

assumes "ξ' ∈ is_rerr ξ"
shows "(∃ dests' sip'.
  msg ξ = Rerr dests' sip' ∧
  ξ' = ξ(| dests := dests', sip := sip' |))"
⟨proof⟩

```

**lemmas is\_msg\_defs =**

```

is_rerr_def is_rrep_def is_rreq_def is_pkt_def is_newpkt_def

```

**lemma is\_msg\_inv\_ip [simp]:**

```

"ξ' ∈ is_rerr ξ ⇒ ip ξ' = ip ξ"
"ξ' ∈ is_rrep ξ ⇒ ip ξ' = ip ξ"
"ξ' ∈ is_rreq ξ ⇒ ip ξ' = ip ξ"
"ξ' ∈ is_pkt ξ ⇒ ip ξ' = ip ξ"
"ξ' ∈ is_newpkt ξ ⇒ ip ξ' = ip ξ"
⟨proof⟩

```

**lemma is\_msg\_inv\_sn [simp]:**

```

"ξ' ∈ is_rerr ξ ⇒ sn ξ' = sn ξ"
"ξ' ∈ is_rrep ξ ⇒ sn ξ' = sn ξ"
"ξ' ∈ is_rreq ξ ⇒ sn ξ' = sn ξ"
"ξ' ∈ is_pkt ξ ⇒ sn ξ' = sn ξ"
"ξ' ∈ is_newpkt ξ ⇒ sn ξ' = sn ξ"
⟨proof⟩

```

**lemma is\_msg\_inv\_rt [simp]:**

```

"ξ' ∈ is_rerr ξ ⇒ rt ξ' = rt ξ"
"ξ' ∈ is_rrep ξ ⇒ rt ξ' = rt ξ"
"ξ' ∈ is_rreq ξ ⇒ rt ξ' = rt ξ"
"ξ' ∈ is_pkt ξ ⇒ rt ξ' = rt ξ"
"ξ' ∈ is_newpkt ξ ⇒ rt ξ' = rt ξ"
⟨proof⟩

```

**lemma is\_msg\_inv\_rreqs [simp]:**

```

"ξ' ∈ is_rerr ξ ⇒ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_rrep ξ ⇒ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_rreq ξ ⇒ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_pkt ξ ⇒ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_newpkt ξ ⇒ rreqs ξ' = rreqs ξ"
⟨proof⟩

```

**lemma is\_msg\_inv\_store [simp]:**

```

"ξ' ∈ is_rerr ξ ⇒ store ξ' = store ξ"
"ξ' ∈ is_rrep ξ ⇒ store ξ' = store ξ"
"ξ' ∈ is_rreq ξ ⇒ store ξ' = store ξ"
"ξ' ∈ is_pkt ξ ⇒ store ξ' = store ξ"
"ξ' ∈ is_newpkt ξ ⇒ store ξ' = store ξ"
⟨proof⟩

```

**lemma is\_msg\_inv\_sip [simp]:**

```

"ξ' ∈ is_pkt ξ ⇒ sip ξ' = sip ξ"
"ξ' ∈ is_newpkt ξ ⇒ sip ξ' = sip ξ"

```

⟨proof⟩

### 5.3.3 The protocol process

```
datatype pseqp =
  PAadv
  | PNewPkt
  | PPkt
  | PRreq
  | PRrep
  | PRerr

fun nat_of_seqp :: "pseqp ⇒ nat"
where
  "nat_of_seqp PAadv = 1"
  | "nat_of_seqp PPkt = 2"
  | "nat_of_seqp PNewPkt = 3"
  | "nat_of_seqp PRreq = 4"
  | "nat_of_seqp PRrep = 5"
  | "nat_of_seqp PRerr = 6"

instantiation "pseqp" :: ord
begin
definition less_eq_seqp [iff]: "l1 ≤ l2 = (nat_of_seqp l1 ≤ nat_of_seqp l2)"
definition less_seqp [iff]: "l1 < l2 = (nat_of_seqp l1 < nat_of_seqp l2)"
instance ⟨proof⟩
end

abbreviation AODV
where
  "AODV ≡ λ_. [[clear_locals]] call(PAadv)"

abbreviation PKT
where
  "PKT args ≡

  [[ξ. let (data, dip, oip) = args ξ in
    (clear_locals ξ) (| data := data, dip := dip, oip := oip |)]
  call(PPkt)"]

abbreviation NEWPKT
where
  "NEWPKT args ≡
  [[ξ. let (data, dip) = args ξ in
    (clear_locals ξ) (| data := data, dip := dip |)]
  call(PNewPkt)"]

abbreviation RREQ
where
  "RREQ args ≡
  [[ξ. let (hops, dip, dsn, dsk, oip, osn, sip, handled) = args ξ in
    (clear_locals ξ) (| hops := hops, dip := dip,
      dsn := dsn, dsk := dsk, oip := oip,
      osn := osn, sip := sip, handled := handled |)]
  call(PRreq)"]

abbreviation RREP
where
  "RREP args ≡
  [[ξ. let (hops, dip, dsn, oip, sip) = args ξ in
    (clear_locals ξ) (| hops := hops, dip := dip, dsn := dsn,
      oip := oip, sip := sip |)]
  call(PRrep)"]

abbreviation RERR
```



where

```
"RERR args ≡
[[ξ. let (dests, sip) = args ξ in
  (clear_locals ξ) (| dests := dests, sip := sip |)]
call(PRerr)"]
```

fun  $\Gamma_{AODV} :: "(state, msg, pseqp, pseqp label) seqp\_env"$

where

```
" $\Gamma_{AODV}$  PAadv = labelled PAadv (
  receive( $\lambda$ msg' ξ. ξ (| msg := msg' |)).
  (
    <is_newpkt> NEWPKT( $\lambda$ ξ. (data ξ, ip ξ))
    ⊕ <is_pkt> PKT( $\lambda$ ξ. (data ξ, dip ξ, oip ξ))
    ⊕ <is_rreq>
      [[ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ) |)]
      RREQ( $\lambda$ ξ. (hops ξ, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, sip ξ, handled ξ))]
    ⊕ <is_rrep>
      [[ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ) |)]
      RREP( $\lambda$ ξ. (hops ξ, dip ξ, dsn ξ, oip ξ, sip ξ))]
    ⊕ <is_rerr>
      [[ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ) |)]
      RERR( $\lambda$ ξ. (dests ξ, sip ξ))]
  )
  ⊕ < $\lambda$ ξ. { ξ(| dip := dip |) | dip. dip ∈ qD(store ξ) ∩ vD(rt ξ) }>
      [[ξ. ξ (| data := hd( $\sigma_{queue}$ (store ξ, dip ξ)) |)]
      unicast( $\lambda$ ξ. the (nhop (rt ξ) (dip ξ)),  $\lambda$ ξ. pkt(data ξ, dip ξ, ip ξ)).
      [[ξ. ξ (| store := the (drop (dip ξ) (store ξ)) |)]
      AODV()
      ▷ [[ξ. ξ (| dests := ( $\lambda$ rip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
        then Some (inc (sqn (rt ξ) rip)) else None) |)]
        [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)]
        [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |)]
        broadcast( $\lambda$ ξ. rerr(dests ξ, ip ξ)). AODV()
      ⊕ < $\lambda$ ξ. { ξ(| dip := dip |)
        | dip. dip ∈ qD(store ξ) - vD(rt ξ) ∧ the ( $\sigma_{p-flag}$ (store ξ, dip)) = req }>
        [[ξ. ξ (| store := unsetRRF (store ξ) (dip ξ) |)]
        [[ξ. ξ (| sn := inc (sn ξ) |)]
        [[ξ. ξ (| rreqs := rreqs ξ ∪ {(ip ξ, sn ξ)} |)]
        broadcast( $\lambda$ ξ. rreq(0, dip ξ, sqn (rt ξ) (dip ξ), sqnf (rt ξ) (dip ξ), ip ξ, sn ξ,
          ip ξ, False)). AODV())"
```

```
| " $\Gamma_{AODV}$  PNewPkt = labelled PNewPkt (
  <ξ. dip ξ = ip ξ>
  deliver( $\lambda$ ξ. data ξ).AODV()
  ⊕ <ξ. dip ξ ≠ ip ξ>
  [[ξ. ξ (| store := add (data ξ) (dip ξ) (store ξ) |)]
  AODV())"
```

```
| " $\Gamma_{AODV}$  PPkt = labelled PPkt (
  <ξ. dip ξ = ip ξ>
  deliver( $\lambda$ ξ. data ξ).AODV()
  ⊕ <ξ. dip ξ ≠ ip ξ>
  (
    <ξ. dip ξ ∈ vD (rt ξ)>
    unicast( $\lambda$ ξ. the (nhop (rt ξ) (dip ξ)),  $\lambda$ ξ. pkt(data ξ, dip ξ, oip ξ)).AODV()
    ▷
    [[ξ. ξ (| dests := ( $\lambda$ rip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
      then Some (inc (sqn (rt ξ) rip)) else None) |)]
    [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)]
    [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |)]
    broadcast( $\lambda$ ξ. rerr(dests ξ, ip ξ)).AODV()
  ⊕ <ξ. dip ξ ∉ vD (rt ξ)>
  (
    <ξ. dip ξ ∈ iD (rt ξ)>
    broadcast( $\lambda$ ξ. rerr([dip ξ ↦ sqn (rt ξ) (dip ξ)], ip ξ)). AODV()
```

```

    ⊕ ⟨ξ. dip ξ ∉ iD (rt ξ)⟩
      AODV()
  )
))"

| "ΓAODV PRreq = labelled PRreq (
  ⟨ξ. (oip ξ, osn ξ) ∈ rreqs ξ⟩
  AODV()
  ⊕ ⟨ξ. (oip ξ, osn ξ) ∉ rreqs ξ⟩
  [[ξ. ξ (| rt := update (rt ξ) (oip ξ) (osn ξ, kno, val, hops ξ + 1, sip ξ) |)]
  [[ξ. ξ (| rreqs := rreqs ξ ∪ {(oip ξ, osn ξ)} |)]
  (
    ⟨ξ. handled ξ = False⟩
    (
      ⟨ξ. dip ξ = ip ξ⟩
      [[ξ. ξ (| sn := max (sn ξ) (dsn ξ) |)]
      unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(0, dip ξ, sn ξ, oip ξ, ip ξ)).
      broadcast(λξ. rreq(hops ξ + 1, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, ip ξ, True)).
      AODV()
    ▷
      [[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
        then Some (inc (sqn (rt ξ) rip)) else None) |)]
      [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)]
      [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |)]
      broadcast(λξ. rerr(dests ξ, ip ξ)).AODV()
    ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
    (
      ⟨ξ. dip ξ ∈ vD (rt ξ) ∧ dsn ξ ≤ sqn (rt ξ) (dip ξ) ∧ sqnf (rt ξ) (dip ξ) = kno⟩
      unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(the (dhops (rt ξ) (dip ξ)), dip ξ,
        sqn (rt ξ) (dip ξ), oip ξ, ip ξ)).
      broadcast(λξ. rreq(hops ξ + 1, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, ip ξ, True)).
      AODV()
    ▷
      [[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
        then Some (inc (sqn (rt ξ) rip)) else None) |)]
      [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)]
      [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |)]
      broadcast(λξ. rerr(dests ξ, ip ξ)).AODV()
    ⊕ ⟨ξ. dip ξ ∉ vD (rt ξ) ∨ sqn (rt ξ) (dip ξ) < dsn ξ ∨ sqnf (rt ξ) (dip ξ) = unk⟩
      broadcast(λξ. rreq(hops ξ + 1, dip ξ, max (sqn (rt ξ) (dip ξ)) (dsn ξ),
        dsk ξ, oip ξ, osn ξ, ip ξ, False)).
      AODV()
    )
  )
  ⊕ ⟨ξ. handled ξ = True⟩
  broadcast(λξ. rreq(hops ξ + 1, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, ip ξ, True)).
  AODV()
))"

```

```

| "ΓAODV PRrep = labelled PRrep (
  [[ξ. ξ (| rt := update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ) |)]
  (
    ⟨ξ. oip ξ = ip ξ⟩
    AODV()
  ⊕ ⟨ξ. oip ξ ≠ ip ξ⟩
  (
    ⟨ξ. oip ξ ∈ vD (rt ξ) ∧ dip ξ ∈ vD (rt ξ)⟩
    unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(the (dhops (rt ξ) (dip ξ)), dip ξ,
      sqn (rt ξ) (dip ξ), oip ξ, ip ξ)).
    AODV()
  ▷
    [[ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
      then Some (inc (sqn (rt ξ) rip)) else None) |)]
    [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)]
  )
  )

```

```

    [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |)]
    broadcast(λξ. rerr(dests ξ, ip ξ)).AODV()
  ⊕ ⟨ξ. oip ξ ∉ vD (rt ξ) ∨ dip ξ ∉ vD (rt ξ)⟩
    AODV()
  )
)
)"

| "ΓAODV PRerr = labelled PRerr (
  [[ξ. ξ (| dests := (λrip. case (dests ξ) rip of None ⇒ None
    | Some rsn ⇒ if rip ∈ vD (rt ξ) ∧ the (nhop (rt ξ) rip) = sip ξ
    ∧ sqn (rt ξ) rip < rsn then Some rsn else None) |)]
  [[ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)]
  [[ξ. ξ (| store := setRRF (store ξ) (dests ξ) |)]
  (
    ⟨ξ. dests ξ ≠ Map.empty⟩
    broadcast(λξ. rerr(dests ξ, ip ξ)). AODV()
  ⊕ ⟨ξ. dests ξ = Map.empty⟩
    AODV()
  ))"

```

```

declare ΓAODV.simps [simp del, code del]
lemmas ΓAODV.simps [simp, code] = ΓAODV.simps [simplified]

```

```

fun ΓAODV_skeleton
where
  "ΓAODV_skeleton PAodv = seqp_skeleton (ΓAODV PAodv)"
  | "ΓAODV_skeleton PNewPkt = seqp_skeleton (ΓAODV PNewPkt)"
  | "ΓAODV_skeleton PPkt = seqp_skeleton (ΓAODV PPkt)"
  | "ΓAODV_skeleton PRreq = seqp_skeleton (ΓAODV PRreq)"
  | "ΓAODV_skeleton PRrep = seqp_skeleton (ΓAODV PRrep)"
  | "ΓAODV_skeleton PRerr = seqp_skeleton (ΓAODV PRerr)"

```

```

lemma ΓAODV_skeleton_wf [simp]:
  "wellformed ΓAODV_skeleton"
  ⟨proof⟩

```

```

declare ΓAODV_skeleton.simps [simp del, code del]
lemmas ΓAODV_skeleton.simps [simp, code]
  = ΓAODV_skeleton.simps [simplified ΓAODV.simps seqp_skeleton.simps]

```

```

lemma aodv_proc_cases [dest]:
  fixes p pn
  shows "p ∈ ctermsl (ΓAODV pn) ⇒
    (p ∈ ctermsl (ΓAODV PAodv) ∨
     p ∈ ctermsl (ΓAODV PNewPkt) ∨
     p ∈ ctermsl (ΓAODV PPkt) ∨
     p ∈ ctermsl (ΓAODV PRreq) ∨
     p ∈ ctermsl (ΓAODV PRrep) ∨
     p ∈ ctermsl (ΓAODV PRerr))"
  ⟨proof⟩

```

```

definition σAODV :: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp) set"
where "σAODV i ≡ {(aodv_init i, ΓAODV PAodv)}"

```

```

abbreviation paodv
  :: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
  "paodv i ≡ (| init = σAODV i, trans = seqp_sos ΓAODV |)"

```

```

lemma aodv_trans: "trans (paodv i) = seqp_sos ΓAODV"
  ⟨proof⟩

```

```

lemma aadv_control_within [simp]: "control_within  $\Gamma_{AODV}$  (init (paadv i))"
  <proof>

lemma aadv_wf [simp]:
  "wellformed  $\Gamma_{AODV}$ "
  <proof>

lemmas aadv_labels_not_empty [simp] = labels_not_empty [OF aadv_wf]

lemma aadv_ex_label [intro]: " $\exists l. l \in \text{labels } \Gamma_{AODV} p$ "
  <proof>

lemma aadv_ex_labelE [elim]:
  assumes " $\forall l \in \text{labels } \Gamma_{AODV} p. P l p$ "
    and " $\exists p l. P l p \implies Q$ "
  shows "Q"
  <proof>

lemma aadv_simple_labels [simp]: "simple_labels  $\Gamma_{AODV}$ "
  <proof>

lemma  $\sigma_{AODV\_labels}$  [simp]: " $(\xi, p) \in \sigma_{AODV} i \implies \text{labels } \Gamma_{AODV} p = \{PAadv-:0\}$ "
  <proof>

lemma aadv_init_kD_empty [simp]:
  " $(\xi, p) \in \sigma_{AODV} i \implies kD (rt \xi) = \{\}$ "
  <proof>

lemma aadv_init_sip_not_ip [simp]: " $\neg(\text{sip (aadv\_init } i) = i)$ " <proof>

lemma aadv_init_sip_not_ip' [simp]:
  assumes " $(\xi, p) \in \sigma_{AODV} i$ "
  shows " $\text{sip } \xi \neq ip \xi$ "
  <proof>

lemma aadv_init_sip_not_i [simp]:
  assumes " $(\xi, p) \in \sigma_{AODV} i$ "
  shows " $\text{sip } \xi \neq i$ "
  <proof>

lemma clear_locals_sip_not_ip':
  assumes " $ip \xi = i$ "
  shows " $\neg(\text{sip (clear\_locals } \xi) = i)$ "
  <proof>

Stop the simplifier from descending into process terms.

declare seqp_congs [cong]

Configure the main invariant tactic for AODV.

declare
   $\Gamma_{AODV\_simps}$  [cterms_env]
  aadv_proc_cases [ctermsl_cases]
  seq_invariant_ctermsI [OF aadv_wf aadv_control_within aadv_simple_labels aadv_trans,
    cterms_intros]
  seq_step_invariant_ctermsI [OF aadv_wf aadv_control_within aadv_simple_labels aadv_trans,
    cterms_intros]

end

```

## 5.4 Invariant assumptions and properties

```

theory E_Aadv_Predicates
imports E_Aadv

```

begin

Definitions for expression assumptions on incoming messages and properties of outgoing messages.

abbreviation *not\_Pkt* :: "msg  $\Rightarrow$  bool"

where "not\_Pkt m  $\equiv$  case m of Pkt \_ \_ \_  $\Rightarrow$  False | \_  $\Rightarrow$  True"

definition *msg\_sender* :: "msg  $\Rightarrow$  ipc"

where "msg\_sender m  $\equiv$  case m of Rreq \_ \_ \_ \_ \_ ipc \_  $\Rightarrow$  ipc  
| Rrep \_ \_ \_ \_ ipc  $\Rightarrow$  ipc  
| Rerr \_ ipc  $\Rightarrow$  ipc  
| Pkt \_ \_ ipc  $\Rightarrow$  ipc"

lemma *msg\_sender\_simps* [simp]:

" $\wedge$ hops dip dsn dsk oip osn sip handled.  
msg\_sender (Rreq hops dip dsn dsk oip osn sip handled) = sip"  
" $\wedge$ hops dip dsn oip sip. msg\_sender (Rrep hops dip dsn oip sip) = sip"  
" $\wedge$ dests sip. msg\_sender (Rerr dests sip) = sip"  
" $\wedge$ dip sip. msg\_sender (Pkt d dip sip) = sip"  
<proof>

definition *msg\_zhops* :: "msg  $\Rightarrow$  bool"

where "msg\_zhops m  $\equiv$  case m of  
Rreq hopsc dipc \_ \_ oipc \_ sipc \_  $\Rightarrow$  hopsc = 0  $\longrightarrow$  oipc = sipc  
| Rrep hopsc dipc \_ \_ sipc  $\Rightarrow$  hopsc = 0  $\longrightarrow$  dipc = sipc  
| \_  $\Rightarrow$  True"

lemma *msg\_zhops\_simps* [simp]:

" $\wedge$ hops dip dsn dsk oip osn sip handled.  
msg\_zhops (Rreq hops dip dsn dsk oip osn sip handled) = (hops = 0  $\longrightarrow$  oip = sip)"  
" $\wedge$ hops dip dsn oip sip. msg\_zhops (Rrep hops dip dsn oip sip) = (hops = 0  $\longrightarrow$  dip = sip)"  
" $\wedge$ dests sip. msg\_zhops (Rerr dests sip) = True"  
" $\wedge$ dip. msg\_zhops (Newpkt d dip) = True"  
" $\wedge$ dip sip. msg\_zhops (Pkt d dip sip) = True"  
<proof>

definition *rreq\_rrep\_sn* :: "msg  $\Rightarrow$  bool"

where "rreq\_rrep\_sn m  $\equiv$  case m of Rreq \_ \_ \_ \_ \_ osnc \_ \_  $\Rightarrow$  osnc  $\geq$  1  
| Rrep \_ \_ dsnc \_ \_  $\Rightarrow$  dsnc  $\geq$  1  
| \_  $\Rightarrow$  True"

lemma *rreq\_rrep\_sn\_simps* [simp]:

" $\wedge$ hops dip dsn dsk oip osn sip handled.  
rreq\_rrep\_sn (Rreq hops dip dsn dsk oip osn sip handled) = (osn  $\geq$  1)"  
" $\wedge$ hops dip dsn oip sip. rreq\_rrep\_sn (Rrep hops dip dsn oip sip) = (dsn  $\geq$  1)"  
" $\wedge$ dests sip. rreq\_rrep\_sn (Rerr dests sip) = True"  
" $\wedge$ dip. rreq\_rrep\_sn (Newpkt d dip) = True"  
" $\wedge$ dip sip. rreq\_rrep\_sn (Pkt d dip sip) = True"  
<proof>

definition *rreq\_rrep\_fresh* :: "rt  $\Rightarrow$  msg  $\Rightarrow$  bool"

where "rreq\_rrep\_fresh crt m  $\equiv$  case m of Rreq hopsc \_ \_ \_ oipc osnc ipcc \_  $\Rightarrow$  (ipcc  $\neq$  oipc  $\longrightarrow$   
oipc  $\in$  kD(crt)  $\wedge$  (sqn crt oipc > osnc  
 $\vee$  (sqn crt oipc = osnc  
 $\wedge$  the (dhops crt oipc)  $\leq$  hopsc  
 $\wedge$  the (flag crt oipc) = val)))  
| Rreq hopsc dipc dsnc \_ ipcc  $\Rightarrow$  (ipcc  $\neq$  dipc  $\longrightarrow$   
dipc  $\in$  kD(crt)  
 $\wedge$  sqn crt dipc = dsnc  
 $\wedge$  the (dhops crt dipc) = hopsc  
 $\wedge$  the (flag crt dipc) = val)  
| \_  $\Rightarrow$  True"

lemma *rreq\_rrep\_fresh* [simp]:

" $\wedge$ hops dip dsn dsk oip osn sip handled.

```

rreq_rrep_fresh crt (Rreq hops dip dsn dsk oip osn sip handled) =
  (sip ≠ oip → oip ∈ kD(crt)
   ∧ (sqn crt oip > osn
      ∨ (sqn crt oip = osn
         ∧ the (dhops crt oip) ≤ hops
         ∧ the (flag crt oip) = val)))"
"∧ hops dip dsn oip sip. rreq_rrep_fresh crt (Rrep hops dip dsn oip sip) =
  (sip ≠ dip → dip ∈ kD(crt)
   ∧ sqn crt dip = dsn
   ∧ the (dhops crt dip) = hops
   ∧ the (flag crt dip) = val)"
"∧ dests sip.      rreq_rrep_fresh crt (Rerr dests sip) = True"
"∧ d dip.          rreq_rrep_fresh crt (Newpkt d dip)  = True"
"∧ d dip sip.     rreq_rrep_fresh crt (Pkt d dip sip)  = True"
⟨proof⟩

```

**definition** `rerr_invalid` :: "rt ⇒ msg ⇒ bool"

where "rerr\_invalid crt m ≡ case m of Rerr destsc \_ ⇒ (∀ ripc ∈ dom(destsc).  
 (ripc ∈ iD(crt) ∧ the (destsc ripc) = sqn crt ripc))  
 | \_ ⇒ True"

**lemma** `rerr_invalid [simp]`:

```

"∧ hops dip dsn dsk oip osn sip handled.
  rerr_invalid crt (Rreq hops dip dsn dsk oip osn sip handled) = True"
"∧ hops dip dsn oip sip. rerr_invalid crt (Rrep hops dip dsn oip sip) = True"
"∧ dests sip.          rerr_invalid crt (Rerr dests sip) = (∀ ripc ∈ dom(dests).
  ripc ∈ iD(crt) ∧ the (dests ripc) = sqn crt ripc)"
"∧ d dip.              rerr_invalid crt (Newpkt d dip)  = True"
"∧ d dip sip.         rerr_invalid crt (Pkt d dip sip)  = True"
⟨proof⟩

```

**definition**

`initmissing` :: "(nat ⇒ state option) × 'a ⇒ (nat ⇒ state) × 'a"

where

"initmissing σ = (λi. case (fst σ) i of None ⇒ aadv\_init i | Some s ⇒ s, snd σ)"

**lemma** `not_in_net_ips_fst_init_missing [simp]`:

```

assumes "i ∉ net_ips σ"
  shows "fst (initmissing (netgmap fst σ)) i = aadv_init i"
⟨proof⟩

```

**lemma** `fst_initmissing_netgmap_pair_fst [simp]`:

```

"fst (initmissing (netgmap (λ(p, q). (fst (id p), snd (id p), q)) s))
  = fst (initmissing (netgmap fst s))"
⟨proof⟩

```

We introduce a streamlined alternative to `initmissing` with `netgmap` to simplify invariant statements and thus facilitate their comprehension and presentation.

**lemma** `fst_initmissing_netgmap_default_aadv_init_netlift`:

```

"fst (initmissing (netgmap fst s)) = default aadv_init (netlift fst s)"
⟨proof⟩

```

**definition**

`netglobal` :: "((nat ⇒ state) ⇒ bool) ⇒ ((state × 'b) × 'c) net\_state ⇒ bool"

where

"netglobal P ≡ (λs. P (default aadv\_init (netlift fst s)))"

end

## 5.5 Quality relations between routes

theory `E_Fresher`

imports `E_Aadv_Data`

begin

## 5.5.1 Net sequence numbers

### On individual routes

#### definition

$nsqn_r :: "r \Rightarrow sqn"$

#### where

$nsqn_r r \equiv \text{if } \pi_4(r) = \text{val} \vee \pi_2(r) = 0 \text{ then } \pi_2(r) \text{ else } (\pi_2(r) - 1)$

#### lemma $nsqn_r\_def'$ :

$nsqn_r r = (\text{if } \pi_4(r) = \text{inv} \text{ then } \pi_2(r) - 1 \text{ else } \pi_2(r))$

$\langle \text{proof} \rangle$

#### lemma $nsqn_r\_zero$ [simp]:

$\bigwedge dsn dsk \text{ flag hops nhip. } nsqn_r (0, dsk, \text{flag}, \text{hops}, \text{nhip}) = 0$

$\langle \text{proof} \rangle$

#### lemma $nsqn_r\_val$ [simp]:

$\bigwedge dsn dsk \text{ hops nhip. } nsqn_r (dsn, dsk, \text{val}, \text{hops}, \text{nhip}) = dsn$

$\langle \text{proof} \rangle$

#### lemma $nsqn_r\_inv$ [simp]:

$\bigwedge dsn dsk \text{ hops nhip. } nsqn_r (dsn, dsk, \text{inv}, \text{hops}, \text{nhip}) = dsn - 1$

$\langle \text{proof} \rangle$

#### lemma $nsqn_r\_lte\_dsn$ [simp]:

$\bigwedge dsn dsk \text{ flag hops nhip. } nsqn_r (dsn, dsk, \text{flag}, \text{hops}, \text{nhip}) \leq dsn$

$\langle \text{proof} \rangle$

### On routes in routing tables

#### definition

$nsqn :: "rt \Rightarrow ip \Rightarrow sqn"$

#### where

$nsqn \equiv \lambda rt \text{ dip. case } \sigma_{\text{route}}(rt, \text{dip}) \text{ of None} \Rightarrow 0 \mid \text{Some } r \Rightarrow nsqn_r(r)$

#### lemma $nsqn\_sqn\_def$ :

$\bigwedge rt \text{ dip. } nsqn \text{ rt dip} = (\text{if } \text{flag } rt \text{ dip} = \text{Some } \text{val} \vee sqn \text{ rt dip} = 0$   
 $\text{then } sqn \text{ rt dip} \text{ else } sqn \text{ rt dip} - 1)$

$\langle \text{proof} \rangle$

#### lemma $\text{not\_in\_kD\_nsqn}$ [simp]:

assumes  $\text{"dip} \notin kD(rt)$

shows  $nsqn \text{ rt dip} = 0$

$\langle \text{proof} \rangle$

#### lemma $kD\_nsqn$ :

assumes  $\text{"dip} \in kD(rt)$

shows  $nsqn \text{ rt dip} = nsqn_r(\text{the } (\sigma_{\text{route}}(rt, \text{dip})))$

$\langle \text{proof} \rangle$

#### lemma $nsqn_r\_flag\_pred$ [simp, intro]:

fixes  $dsn dsk \text{ flag hops nhip pre}$

assumes  $P (nsqn_r (dsn, dsk, \text{val}, \text{hops}, \text{nhip}))$

and  $P (nsqn_r (dsn, dsk, \text{inv}, \text{hops}, \text{nhip}))$

shows  $P (nsqn_r (dsn, dsk, \text{flag}, \text{hops}, \text{nhip}))$

$\langle \text{proof} \rangle$

#### lemma $sqn\_nsqn$ :

$\bigwedge rt \text{ dip. } sqn \text{ rt dip} - 1 \leq nsqn \text{ rt dip}$

$\langle \text{proof} \rangle$

#### lemma $nsqn\_sqn$ : $nsqn \text{ rt dip} \leq sqn \text{ rt dip}$

*<proof>*

```
lemma val_nsqn_sqn [elim, simp]:  
  assumes "ip ∈ kD(rt)"  
    and "the (flag rt ip) = val"  
  shows "nsqn rt ip = sqn rt ip"  
<proof>
```

```
lemma vD_nsqn_sqn [elim, simp]:  
  assumes "ip ∈ vD(rt)"  
  shows "nsqn rt ip = sqn rt ip"  
<proof>
```

```
lemma inv_nsqn_sqn [elim, simp]:  
  assumes "ip ∈ kD(rt)"  
    and "the (flag rt ip) = inv"  
  shows "nsqn rt ip = sqn rt ip - 1"  
<proof>
```

```
lemma iD_nsqn_sqn [elim, simp]:  
  assumes "ip ∈ iD(rt)"  
  shows "nsqn rt ip = sqn rt ip - 1"  
<proof>
```

```
lemma nsqn_update_changed_kno_val [simp]: " $\wedge$ rt ip dsn dsk hops nhip.  
rt  $\neq$  update rt ip (dsn, kno, val, hops, nhip)  
 $\implies$  nsqn (update rt ip (dsn, kno, val, hops, nhip)) ip = dsn"  
<proof>
```

```
lemma nsqn_update_other [simp]:  
  fixes dsn dsk flag hops dip nhip pre rt ip  
  assumes "dip  $\neq$  ip"  
  shows "nsqn (update rt ip (dsn, dsk, flag, hops, nhip)) dip = nsqn rt dip"  
<proof>
```

```
lemma nsqn_invalidate_eq:  
  assumes "dip ∈ kD(rt)"  
    and "dests dip = Some rsn"  
  shows "nsqn (invalidate rt dests) dip = rsn - 1"  
<proof>
```

```
lemma nsqn_invalidate_other [simp]:  
  assumes "dip ∈ kD(rt)"  
    and "dip  $\notin$  dom dests"  
  shows "nsqn (invalidate rt dests) dip = nsqn rt dip"  
<proof>
```

## 5.5.2 Comparing routes

**definition**

```
fresher :: "r  $\Rightarrow$  r  $\Rightarrow$  bool" (" $\_ / \sqsubseteq \_$ ") [51, 51] 50
```

where

```
"fresher r r'  $\equiv$  ((nsqnr r < nsqnr r')  $\vee$  (nsqnr r = nsqnr r'  $\wedge$   $\pi_5$ (r)  $\geq$   $\pi_5$ (r')))"
```

```
lemma fresherI1 [intro]:  
  assumes "nsqnr r < nsqnr r'"  
  shows "r  $\sqsubseteq$  r'"  
<proof>
```

```
lemma fresherI2 [intro]:  
  assumes "nsqnr r = nsqnr r'"  
    and " $\pi_5$ (r)  $\geq$   $\pi_5$ (r)"  
  shows "r  $\sqsubseteq$  r'"  
<proof>
```



lemma fresherI [intro]:  
 assumes "(nsqn<sub>r</sub> r < nsqn<sub>r</sub> r') ∨ (nsqn<sub>r</sub> r = nsqn<sub>r</sub> r' ∧ π<sub>5</sub>(r) ≥ π<sub>5</sub>(r'))"  
 shows "r ⊆ r'"  
 ⟨proof⟩

lemma fresherE [elim]:  
 assumes "r ⊆ r'"  
 and "nsqn<sub>r</sub> r < nsqn<sub>r</sub> r' ⇒ P r r'"  
 and "nsqn<sub>r</sub> r = nsqn<sub>r</sub> r' ∧ π<sub>5</sub>(r) ≥ π<sub>5</sub>(r') ⇒ P r r'"  
 shows "P r r'"  
 ⟨proof⟩

lemma fresher\_refl [simp]: "r ⊆ r"  
 ⟨proof⟩

lemma fresher\_trans [elim, trans]:  
 "[[ x ⊆ y; y ⊆ z ]] ⇒ x ⊆ z"  
 ⟨proof⟩

lemma not\_fresher\_trans [elim, trans]:  
 "[[ ¬(x ⊆ y); ¬(z ⊆ x) ]] ⇒ ¬(z ⊆ y)"  
 ⟨proof⟩

lemma fresher\_dsn\_flag\_hops\_const [simp]:  
 fixes dsn dsk dsk' flag hops nhip nhip'  
 shows "(dsn, dsk, flag, hops, nhip) ⊆ (dsn, dsk', flag, hops, nhip)'"  
 ⟨proof⟩

### 5.5.3 Comparing routing tables

#### definition

rt\_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"

#### where

"rt\_fresher ≡ λdip rt rt'. (the (σ<sub>route</sub>(rt, dip))) ⊆ (the (σ<sub>route</sub>(rt', dip)))"

#### abbreviation

rt\_fresher\_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" ("(\_/ ⊆\_ \_)" [51, 999, 51] 50)

#### where

"rt1 ⊆<sub>i</sub> rt2 ≡ rt\_fresher i rt1 rt2"

#### lemma rt\_fresher\_def':

"(rt1 ⊆<sub>i</sub> rt2) = (nsqn<sub>r</sub> (the (rt1 i)) < nsqn<sub>r</sub> (the (rt2 i)) ∨  
 nsqn<sub>r</sub> (the (rt1 i)) = nsqn<sub>r</sub> (the (rt2 i)) ∧ π<sub>5</sub> (the (rt2 i)) ≤ π<sub>5</sub> (the (rt1 i)))"

⟨proof⟩

#### lemma single\_rt\_fresher [intro]:

assumes "the (rt1 ip) ⊆ the (rt2 ip)"

shows "rt1 ⊆<sub>ip</sub> rt2"

⟨proof⟩

#### lemma rt\_fresher\_single [intro]:

assumes "rt1 ⊆<sub>ip</sub> rt2"

shows "the (rt1 ip) ⊆ the (rt2 ip)"

⟨proof⟩

#### lemma rt\_fresher\_def2:

assumes "dip ∈ kD(rt1)"

and "dip ∈ kD(rt2)"

shows "(rt1 ⊆<sub>dip</sub> rt2) = (nsqn rt1 dip < nsqn rt2 dip  
 ∨ (nsqn rt1 dip = nsqn rt2 dip  
 ∧ the (dhops rt1 dip) ≥ the (dhops rt2 dip)))"

⟨proof⟩

```

lemma rt_fresherI1 [intro]:
  assumes "dip ∈ kD(rt1)"
    and "dip ∈ kD(rt2)"
    and "nsqn rt1 dip < nsqn rt2 dip"
  shows "rt1 ⊆dip rt2"
  ⟨proof⟩

lemma rt_fresherI2 [intro]:
  assumes "dip ∈ kD(rt1)"
    and "dip ∈ kD(rt2)"
    and "nsqn rt1 dip = nsqn rt2 dip"
    and "the (dhops rt1 dip) ≥ the (dhops rt2 dip)"
  shows "rt1 ⊆dip rt2"
  ⟨proof⟩

lemma rt_fresherE [elim]:
  assumes "rt1 ⊆dip rt2"
    and "dip ∈ kD(rt1)"
    and "dip ∈ kD(rt2)"
    and "[[ nsqn rt1 dip < nsqn rt2 dip ]] ⇒ P rt1 rt2 dip"
    and "[[ nsqn rt1 dip = nsqn rt2 dip;
      the (dhops rt1 dip) ≥ the (dhops rt2 dip) ]] ⇒ P rt1 rt2 dip"
  shows "P rt1 rt2 dip"
  ⟨proof⟩

lemma rt_fresher_refl [simp]: "rt ⊆dip rt"
  ⟨proof⟩

lemma rt_fresher_trans [elim, trans]:
  assumes "rt1 ⊆dip rt2"
    and "rt2 ⊆dip rt3"
  shows "rt1 ⊆dip rt3"
  ⟨proof⟩

lemma rt_fresher_if_Some [intro!]:
  assumes "the (rt dip) ⊆ r"
  shows "rt ⊆dip (λip. if ip = dip then Some r else rt ip)"
  ⟨proof⟩

definition rt_fresh_as :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_fresh_as ≡ λdip rt1 rt2. (rt1 ⊆dip rt2) ∧ (rt2 ⊆dip rt1)"

abbreviation
  rt_fresh_as_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" ("(_/ ≈ip _)" [51, 999, 51] 50)
where
  "rt1 ≈ip rt2 ≡ rt_fresh_as ip rt1 rt2"

lemma rt_fresh_as_refl [simp]: "∧rt dip. rt ≈dip rt"
  ⟨proof⟩

lemma rt_fresh_as_trans [simp, intro, trans]:
  "∧rt1 rt2 rt3 dip. [[ rt1 ≈dip rt2; rt2 ≈dip rt3 ]] ⇒ rt1 ≈dip rt3"
  ⟨proof⟩

lemma rt_fresh_asI [intro!]:
  assumes "rt1 ⊆dip rt2"
    and "rt2 ⊆dip rt1"
  shows "rt1 ≈dip rt2"
  ⟨proof⟩

lemma rt_fresh_as_fresherI [intro]:
  assumes "dip ∈ kD(rt1)"
    and "dip ∈ kD(rt2)"

```

```

    and "the (rt1 dip)  $\sqsubseteq$  the (rt2 dip)"
    and "the (rt2 dip)  $\sqsubseteq$  the (rt1 dip)"
  shows "rt1  $\approx_{\text{dip}}$  rt2"
<proof>

lemma nsqn_rt_fresh_asI:
  assumes "dip  $\in$  kD(rt)"
    and "dip  $\in$  kD(rt'"
    and "nsqn rt dip = nsqn rt' dip"
    and " $\pi_5(\text{the } (rt \text{ dip})) = \pi_5(\text{the } (rt' \text{ dip}))$ "
  shows "rt  $\approx_{\text{dip}}$  rt'"
<proof>

lemma rt_fresh_asE [elim]:
  assumes "rt1  $\approx_{\text{dip}}$  rt2"
    and "[[ rt1  $\sqsubseteq_{\text{dip}}$  rt2; rt2  $\sqsubseteq_{\text{dip}}$  rt1 ]  $\implies$  P rt1 rt2 dip]"
  shows "P rt1 rt2 dip"
<proof>

lemma rt_fresh_asD1 [dest]:
  assumes "rt1  $\approx_{\text{dip}}$  rt2"
  shows "rt1  $\sqsubseteq_{\text{dip}}$  rt2"
<proof>

lemma rt_fresh_asD2 [dest]:
  assumes "rt1  $\approx_{\text{dip}}$  rt2"
  shows "rt2  $\sqsubseteq_{\text{dip}}$  rt1"
<proof>

lemma rt_fresh_as_sym:
  assumes "rt1  $\approx_{\text{dip}}$  rt2"
  shows "rt2  $\approx_{\text{dip}}$  rt1"
<proof>

lemma not_rt_fresh_asI1 [intro]:
  assumes " $\neg$  (rt1  $\sqsubseteq_{\text{dip}}$  rt2)"
  shows " $\neg$  (rt1  $\approx_{\text{dip}}$  rt2)"
<proof>

lemma not_rt_fresh_asI2 [intro]:
  assumes " $\neg$  (rt2  $\sqsubseteq_{\text{dip}}$  rt1)"
  shows " $\neg$  (rt1  $\approx_{\text{dip}}$  rt2)"
<proof>

lemma not_single_rt_fresher [elim]:
  assumes " $\neg$ (the (rt1 ip)  $\sqsubseteq$  the (rt2 ip))"
  shows " $\neg$ (rt1  $\sqsubseteq_{\text{ip}}$  rt2)"
<proof>

lemmas not_single_rt_fresh_asI1 [intro] = not_rt_fresh_asI1 [OF not_single_rt_fresher]
lemmas not_single_rt_fresh_asI2 [intro] = not_rt_fresh_asI2 [OF not_single_rt_fresher]

lemma not_rt_fresher_single [elim]:
  assumes " $\neg$ (rt1  $\sqsubseteq_{\text{ip}}$  rt2)"
  shows " $\neg$ (the (rt1 ip)  $\sqsubseteq$  the (rt2 ip))"
<proof>

lemma rt_fresh_as_nsqnr:
  assumes "dip  $\in$  kD(rt1)"
    and "dip  $\in$  kD(rt2)"
    and "rt1  $\approx_{\text{dip}}$  rt2"
  shows "nsqnr (the (rt2 dip)) = nsqnr (the (rt1 dip))"
<proof>

```

```

lemma rt_fresher_mapupd [intro!]:
  assumes "dip ∈ kD(rt)"
    and "the (rt dip) ⊆ r"
  shows "rt ⊆dip rt(dip ↦ r)"
  ⟨proof⟩

```

```

lemma rt_fresher_map_update_other [intro!]:
  assumes "dip ∈ kD(rt)"
    and "dip ≠ ip"
  shows "rt ⊆dip rt(ip ↦ r)"
  ⟨proof⟩

```

```

lemma rt_fresher_update_other [simp]:
  assumes inkD: "dip ∈ kD(rt)"
    and "dip ≠ ip"
  shows "rt ⊆dip update rt ip r"
  ⟨proof⟩

```

```

theorem rt_fresher_update [simp]:
  assumes "dip ∈ kD(rt)"
    and "the (dhops rt dip) ≥ 1"
    and "update_arg_wf r"
  shows "rt ⊆dip update rt ip r"
  ⟨proof⟩

```

```

theorem rt_fresher_invalidate [simp]:
  assumes "dip ∈ kD(rt)"
    and indests: "∀ rip ∈ dom(dests). rip ∈ vD(rt) ∧ sqn rt rip < the (dests rip)"
  shows "rt ⊆dip invalidate rt dests"
  ⟨proof⟩

```

```

lemma nsqnr_invalidate [simp]:
  assumes "dip ∈ kD(rt)"
    and "dip ∈ dom(dests)"
  shows "nsqnr (the (invalidate rt dests dip)) = the (dests dip) - 1"
  ⟨proof⟩

```

```

lemma rt_fresh_as_inc_invalidate [simp]:
  assumes "dip ∈ kD(rt)"
    and "∀ rip ∈ dom(dests). rip ∈ vD(rt) ∧ the (dests rip) = inc (sqn rt rip)"
  shows "rt ≈dip invalidate rt dests"
  ⟨proof⟩

```

```

lemmas rt_fresher_inc_invalidate [simp] = rt_fresh_as_inc_invalidate [THEN rt_fresh_asD1]

```

#### 5.5.4 Strictly comparing routing tables

```

definition rt_strictly_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"

```

where

```

"rt_strictly_fresher ≡ λdip rt1 rt2. (rt1 ⊆dip rt2) ∧ ¬(rt1 ≈dip rt2)"

```

abbreviation

```

rt_strictly_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" ("(_/ ⊆_ _)" [51, 999, 51] 50)

```

where

```

"rt1 ⊆i rt2 ≡ rt_strictly_fresher i rt1 rt2"

```

```

lemma rt_strictly_fresher_def'':

```

```

"rt1 ⊆i rt2 = ((rt1 ⊆i rt2) ∧ ¬(rt2 ⊆i rt1))"

```

⟨proof⟩

```

lemma rt_strictly_fresherI' [intro]:

```

```

  assumes "rt1 ⊆i rt2"

```

```

    and "¬(rt2 ⊆i rt1)"

```

```

  shows "rt1 ⊆i rt2"

```

$\langle proof \rangle$

lemma *rt\_strictly\_fresherE'* [elim]:  
 assumes "rt1  $\sqsubseteq_i$  rt2"  
 and "[[ rt1  $\sqsubseteq_i$  rt2;  $\neg$ (rt2  $\sqsubseteq_i$  rt1) ]  $\implies$  P rt1 rt2 i"  
 shows "P rt1 rt2 i"  
 $\langle proof \rangle$

lemma *rt\_strictly\_fresherI* [intro]:  
 assumes "rt1  $\sqsubseteq_i$  rt2"  
 and " $\neg$ (rt1  $\approx_i$  rt2)"  
 shows "rt1  $\sqsubseteq_i$  rt2"  
 $\langle proof \rangle$

lemmas *rt\_strictly\_fresher\_singleI* [elim] = *rt\_strictly\_fresherI* [OF single\_rt\_fresher]

lemma *rt\_strictly\_fresherE* [elim]:  
 assumes "rt1  $\sqsubseteq_i$  rt2"  
 and "[[ rt1  $\sqsubseteq_i$  rt2;  $\neg$ (rt1  $\approx_i$  rt2) ]  $\implies$  P rt1 rt2 i"  
 shows "P rt1 rt2 i"  
 $\langle proof \rangle$

lemma *rt\_strictly\_fresher\_def'*:  
 "rt1  $\sqsubseteq_i$  rt2 =  
 (nsqn<sub>r</sub> (the (rt1 i)) < nsqn<sub>r</sub> (the (rt2 i))  
 $\vee$  (nsqn<sub>r</sub> (the (rt1 i)) = nsqn<sub>r</sub> (the (rt2 i))  $\wedge$   $\pi_5$ (the (rt1 i)) >  $\pi_5$ (the (rt2 i))))"  
 $\langle proof \rangle$

lemma *rt\_strictly\_fresher\_fresherD* [dest]:  
 assumes "rt1  $\sqsubseteq_{dip}$  rt2"  
 shows "the (rt1 dip)  $\sqsubseteq$  the (rt2 dip)"  
 $\langle proof \rangle$

lemma *rt\_strictly\_fresher\_not\_fresh\_asD* [dest]:  
 assumes "rt1  $\sqsubseteq_{dip}$  rt2"  
 shows " $\neg$  rt1  $\approx_{dip}$  rt2"  
 $\langle proof \rangle$

lemma *rt\_strictly\_fresher\_trans* [elim, trans]:  
 assumes "rt1  $\sqsubseteq_{dip}$  rt2"  
 and "rt2  $\sqsubseteq_{dip}$  rt3"  
 shows "rt1  $\sqsubseteq_{dip}$  rt3"  
 $\langle proof \rangle$

lemma *rt\_strictly\_fresher\_irefl* [simp]: " $\neg$  (rt  $\sqsubseteq_{dip}$  rt)"  
 $\langle proof \rangle$

lemma *rt\_fresher\_trans\_rt\_strictly\_fresher* [elim, trans]:  
 assumes "rt1  $\sqsubseteq_{dip}$  rt2"  
 and "rt2  $\sqsubseteq_{dip}$  rt3"  
 shows "rt1  $\sqsubseteq_{dip}$  rt3"  
 $\langle proof \rangle$

lemma *rt\_fresher\_trans\_rt\_strictly\_fresher'* [elim, trans]:  
 assumes "rt1  $\sqsubseteq_{dip}$  rt2"  
 and "rt2  $\sqsubseteq_{dip}$  rt3"  
 shows "rt1  $\sqsubseteq_{dip}$  rt3"  
 $\langle proof \rangle$

lemma *rt\_fresher\_imp\_nsqn\_le*:  
 assumes "rt1  $\sqsubseteq_{ip}$  rt2"  
 and "ip  $\in$  kD rt1"  
 and "ip  $\in$  kD rt2"  
 shows "nsqn rt1 ip  $\leq$  nsqn rt2 ip"

```

⟨proof⟩

lemma rt_strictly_fresher_ltI [intro]:
  assumes "dip ∈ kD(rt1)"
    and "dip ∈ kD(rt2)"
    and "nsqn rt1 dip < nsqn rt2 dip"
  shows "rt1 ⊑dip rt2"
⟨proof⟩

lemma rt_strictly_fresher_eqI [intro]:
  assumes "i ∈ kD(rt1)"
    and "i ∈ kD(rt2)"
    and "nsqn rt1 i = nsqn rt2 i"
    and "π5(the (rt2 i)) < π5(the (rt1 i))"
  shows "rt1 ⊑i rt2"
⟨proof⟩

lemma invalidate_rtsf_left [simp]:
  "∧ dests dip rt rt'. dests dip = None ⇒ (invalidate rt dests ⊑dip rt') = (rt ⊑dip rt')"
⟨proof⟩

lemma vD_invalidate_rt_strictly_fresher [simp]:
  assumes "dip ∈ vD(invalidate rt1 dests)"
  shows "(invalidate rt1 dests ⊑dip rt2) = (rt1 ⊑dip rt2)"
⟨proof⟩

lemma rt_strictly_fresher_update_other [elim!]:
  "∧ dip ip rt r rt'. [ dip ≠ ip; rt ⊑dip rt' ] ⇒ update rt ip r ⊑dip rt'"
⟨proof⟩

lemma lt_sqn_imp_update_strictly_fresher:
  assumes "dip ∈ vD (rt2 nhip)"
    and *: "osn < sqn (rt2 nhip) dip"
    and **: "rt ≠ update rt dip (osn, kno, val, hops, nhip)"
  shows "update rt dip (osn, kno, val, hops, nhip) ⊑dip rt2 nhip"
⟨proof⟩

lemma dhops_le_hops_imp_update_strictly_fresher:
  assumes "dip ∈ vD(rt2 nhip)"
    and sqn: "sqn (rt2 nhip) dip = osn"
    and hop: "the (dhops (rt2 nhip) dip) ≤ hops"
    and **: "rt ≠ update rt dip (osn, kno, val, Suc hops, nhip)"
  shows "update rt dip (osn, kno, val, Suc hops, nhip) ⊑dip rt2 nhip"
⟨proof⟩

lemma nsqn_invalidate:
  assumes "dip ∈ kD(rt)"
    and "∀ ip ∈ dom(dests). ip ∈ vD(rt) ∧ the (dests ip) = inc (sqn rt ip)"
  shows "nsqn (invalidate rt dests) dip = nsqn rt dip"
⟨proof⟩

```

end

## 5.6 Invariant proofs on individual processes

```

theory E_Seq_Invariants
imports "../.../AWN/Invariants" E_Aadv E_Aadv_Data E_Aadv_Predicates E_Fresher
begin

```

The proposition numbers are taken from the December 2013 version of the Fehnker et al technical report.

Proposition 7.2

```

lemma sequence_number_increases:
  "paadv i ⊨A onll ΓAODV (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"

```

*<proof>*

**lemma sequence\_number\_one\_or\_bigger:**

"paadv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_). 1 \leq sn \xi$ )"  
*<proof>*

We can get rid of the onl/onll if desired...

**lemma sequence\_number\_increases':**

"paadv i  $\models_A$  ( $\lambda((\xi, \_), \_, (\xi', \_)). sn \xi \leq sn \xi'$ )"  
*<proof>*

**lemma sequence\_number\_one\_or\_bigger':**

"paadv i  $\models$  ( $\lambda(\xi, \_). 1 \leq sn \xi$ )"  
*<proof>*

**lemma sip\_in\_kD:**

"paadv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, 1). 1 \in (\{PAadv-:7\} \cup \{PAadv-:5\} \cup \{PRrep-:0..PRrep-:4\}$   
 $\cup \{PRreq-:0..PRreq-:3\}) \longrightarrow sip \xi \in kD (rt \xi)$ )"  
*<proof>*

Proposition 7.38

**lemma includes\_nhip:**

"paadv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, 1). \forall dip \in kD(rt \xi). the (nhop (rt \xi) dip) \in kD(rt \xi)$ )"  
*<proof>*

Proposition 7.4

**lemma known\_destinations\_increase:**

"paadv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), \_, (\xi', \_)). kD (rt \xi) \subseteq kD (rt \xi')$ )"  
*<proof>*

Proposition 7.5

**lemma rreqs\_increase:**

"paadv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), \_, (\xi', \_)). rreqs \xi \subseteq rreqs \xi'$ )"  
*<proof>*

**lemma dests\_bigger\_than\_sqn:**

"paadv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, 1). 1 \in \{PAadv-:15..PAadv-:17\}$   
 $\cup \{PPkt-:7..PPkt-:9\}$   
 $\cup \{PRreq-:11..PRreq-:13\}$   
 $\cup \{PRreq-:20..PRreq-:22\}$   
 $\cup \{PRrep-:7..PRrep-:9\}$   
 $\cup \{PRerr-:1..PRerr-:4\} \cup \{PRerr-:6\}$   
 $\longrightarrow (\forall ip \in dom(dests \xi). ip \in kD(rt \xi) \wedge sqn (rt \xi) ip \leq the (dests \xi ip))$ )"  
*<proof>*

Proposition 7.6

**lemma sqns\_increase:**

"paadv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), \_, (\xi', \_)). \forall ip. sqn (rt \xi) ip \leq sqn (rt \xi') ip$ )"  
*<proof>*

Proposition 7.7

**lemma ip\_constant:**

"paadv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_). ip \xi = i$ )"  
*<proof>*

Proposition 7.8

**lemma sender\_ip\_valid':**

"paadv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), a, \_). anycast (\lambda m. not\_Pkt m \longrightarrow msg\_sender m = ip \xi) a$ )"  
*<proof>*

**lemma sender\_ip\_valid:**

"paadv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda((\xi, \_), a, \_). anycast (\lambda m. not\_Pkt m \longrightarrow msg\_sender m = i) a$ )"

*<proof>*

**lemma received\_msg\_inv:**

"paadv i  $\models$  (recvm<sub>sg</sub> P  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). l \in \{PAadv-:1\} \rightarrow P$  (msg  $\xi$ ))"  
*<proof>*

Proposition 7.9

**lemma sip\_not\_ip':**

"paadv i  $\models$  (recvm<sub>sg</sub> ( $\lambda m. not\_Pkt$  m  $\rightarrow$  msg\_sender m  $\neq$  i)  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_). sip$   $\xi \neq ip$   $\xi$ )"  
*<proof>*

**lemma sip\_not\_ip:**

"paadv i  $\models$  (recvm<sub>sg</sub> ( $\lambda m. not\_Pkt$  m  $\rightarrow$  msg\_sender m  $\neq$  i)  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_). sip$   $\xi \neq i$ )"  
*<proof>*

Neither *sip\_not\_ip'* nor *sip\_not\_ip* is needed to show loop freedom.

Proposition 7.10

**lemma hop\_count\_positive:**

"paadv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, \_). \forall ip \in kD$  (rt  $\xi$ ). the (dhops (rt  $\xi$ ) ip)  $\geq$  1)"  
*<proof>*

**lemma rreq\_dip\_in\_vD\_dip\_eq\_ip:**

"paadv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). (l \in \{PRreq-:16..PRreq-:17\} \rightarrow dip$   $\xi \in vD$ (rt  $\xi$ ))  
 $\wedge (l \in \{PRreq-:6, PRreq-:7\} \rightarrow dip$   $\xi = ip$   $\xi$ )  
 $\wedge (l \in \{PRreq-:15..PRreq-:17\} \rightarrow dip$   $\xi \neq ip$   $\xi$ ))"  
*<proof>*

**lemma rrep\_dip\_in\_vD:**

"paadv i  $\models$  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). (l \in \{PRrep-:4\} \rightarrow dip$   $\xi \in vD$ (rt  $\xi$ )))"  
*<proof>*

Proposition 7.11

**lemma anycast\_msg\_zhops:**

" $\bigwedge rreqid$  dip dsn dsk oip osn sip.  
paadv i  $\models_A$  onll  $\Gamma_{AODV}$  ( $\lambda(\_, a, \_). anycast$  msg\_zhops a)"  
*<proof>*

**lemma hop\_count\_zero\_oip\_dip\_sip:**

"paadv i  $\models$  (recvm<sub>sg</sub> msg\_zhops  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l).$   
 $(l \in \{PAadv-:4..PAadv-:5\} \cup \{PRreq-:n/n. True\} \rightarrow$   
 $(hops$   $\xi = 0 \rightarrow oip$   $\xi = sip$   $\xi))$   
 $\wedge$   
 $((l \in \{PAadv-:6..PAadv-:7\} \cup \{PRrep-:n/n. True\} \rightarrow$   
 $(hops$   $\xi = 0 \rightarrow dip$   $\xi = sip$   $\xi))))$ "  
*<proof>*

**lemma osn\_rreq:**

"paadv i  $\models$  (recvm<sub>sg</sub> rreq\_rrep\_sn  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l).$   
 $l \in \{PAadv-:4, PAadv-:5\} \cup \{PRreq-:n/n. True\} \rightarrow 1 \leq osn$   $\xi$ )"  
*<proof>*

**lemma osn\_rreq':**

"paadv i  $\models$  (recvm<sub>sg</sub> ( $\lambda m. rreq\_rrep\_sn$  m  $\wedge$  msg\_zhops m)  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l).$   
 $l \in \{PAadv-:4, PAadv-:5\} \cup \{PRreq-:n/n. True\} \rightarrow 1 \leq osn$   $\xi$ )"  
*<proof>*

**lemma dsn\_rrep:**

"paadv i  $\models$  (recvm<sub>sg</sub> rreq\_rrep\_sn  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l).$   
 $l \in \{PAadv-:6, PAadv-:7\} \cup \{PRrep-:n/n. True\} \rightarrow 1 \leq dsn$   $\xi$ )"  
*<proof>*

**lemma dsn\_rrep':**

"paadv i  $\models$  (recvm<sub>sg</sub> ( $\lambda m. rreq\_rrep\_sn$  m  $\wedge$  msg\_zhops m)  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l).$



$1 \in \{PAadv\!-\!:6, PAadv\!-\!:7\} \cup \{PRrep\!-\!:n/n. True\} \longrightarrow 1 \leq dsn \xi$ "

*<proof>*

**lemma** *hop\_count\_zero\_oip\_dip\_sip'*:

"paadv i  $\models$  (recvmsg ( $\lambda m. rreq_rrep\_sn m \wedge msg\_zhops m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV} (\lambda(\xi, l).$   
 $(1 \in \{PAadv\!-\!:4..PAadv\!-\!:5\} \cup \{PRreq\!-\!:n/n. True\} \longrightarrow$   
 $(hops \xi = 0 \longrightarrow oip \xi = sip \xi))$   
 $\wedge$   
 $((1 \in \{PAadv\!-\!:6..PAadv\!-\!:7\} \cup \{PRrep\!-\!:n/n. True\} \longrightarrow$   
 $(hops \xi = 0 \longrightarrow dip \xi = sip \xi)))$ )"

*<proof>*

Proposition 7.12

**lemma** *zero\_seq\_unk\_hops\_one'*:

"paadv i  $\models$  (recvmsg ( $\lambda m. rreq_rrep\_sn m \wedge msg\_zhops m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV} (\lambda(\xi, \_).$   
 $\forall dip \in kD(rt \xi). (sqn (rt \xi) dip = 0 \longrightarrow sqnf (rt \xi) dip = unk$   
 $\wedge (sqnf (rt \xi) dip = unk \longrightarrow the (dhops (rt \xi) dip) = 1)$   
 $\wedge (the (dhops (rt \xi) dip) = 1 \longrightarrow the (nhop (rt \xi) dip) = dip))$ )"

*<proof>*

**lemma** *zero\_seq\_unk\_hops\_one*:

"paadv i  $\models$  (recvmsg ( $\lambda m. rreq_rrep\_sn m \wedge msg\_zhops m$ )  $\rightarrow$ ) onl  $\Gamma_{AODV} (\lambda(\xi, \_).$   
 $\forall dip \in kD(rt \xi). (sqn (rt \xi) dip = 0 \longrightarrow (sqnf (rt \xi) dip = unk$   
 $\wedge the (dhops (rt \xi) dip) = 1$   
 $\wedge the (nhop (rt \xi) dip) = dip))$ )"

*<proof>*

**lemma** *kD\_unk\_or\_atleast\_one*:

"paadv i  $\models$  (recvmsg rreq\_rrep\_sn  $\rightarrow$ ) onl  $\Gamma_{AODV} (\lambda(\xi, l).$   
 $\forall dip \in kD(rt \xi). \pi_3(the (rt \xi dip)) = unk \vee 1 \leq \pi_2(the (rt \xi dip))$ )"

*<proof>*

Proposition 7.13

**lemma** *rreq\_rrep\_sn\_any\_step\_invariant*:

"paadv i  $\models_A$  (recvmsg rreq\_rrep\_sn  $\rightarrow$ ) onll  $\Gamma_{AODV} (\lambda(\_, a, \_). anycast rreq_rrep\_sn a)$ "  
*<proof>*

Proposition 7.14

**lemma** *rreq\_rrep\_fresh\_any\_step\_invariant*:

"paadv i  $\models_A$  onll  $\Gamma_{AODV} (\lambda((\xi, \_), a, \_). anycast (rreq_rrep\_fresh (rt \xi)) a)$ "  
*<proof>*

Proposition 7.15

**lemma** *rerr\_invalid\_any\_step\_invariant*:

"paadv i  $\models_A$  onll  $\Gamma_{AODV} (\lambda((\xi, \_), a, \_). anycast (rerr\_invalid (rt \xi)) a)$ "  
*<proof>*

Proposition 7.16

Some well-definedness obligations are irrelevant for the Isabelle development:

1. In each routing table there is at most one entry for each destination: guaranteed by type.
2. In each store of queued data packets there is at most one data queue for each destination: guaranteed by structure.
3. Whenever a set of pairs (*rip*, *rsn*) is assigned to the variable *destds* of type  $ip \rightarrow sqn$ , or to the first argument of the function *rerr*, this set is a partial function, i.e., there is at most one entry (*rip*, *rsn*) for each destination *rip*: guaranteed by type.

**lemma** *destds\_vD\_inc\_sqn*:

"paadv i  $\models$   
onl  $\Gamma_{AODV} (\lambda(\xi, l). (1 \in \{PAadv\!-\!:15, PPkt\!-\!:7, PRreq\!-\!:11, PRreq\!-\!:20, PRrep\!-\!:7\}$

$$\begin{aligned} &\rightarrow (\forall ip \in \text{dom}(\text{dests } \xi). ip \in vD(\text{rt } \xi) \wedge \text{the } (\text{dests } \xi \text{ ip}) = \text{inc } (\text{sqn } (\text{rt } \xi) \text{ ip})) \\ &\wedge (1 = \text{PRerr} - : 1 \\ &\rightarrow (\forall ip \in \text{dom}(\text{dests } \xi). ip \in vD(\text{rt } \xi) \wedge \text{the } (\text{dests } \xi \text{ ip}) > \text{sqn } (\text{rt } \xi) \text{ ip}))" \end{aligned}$$

*<proof>*

Proposition 7.27

**lemma** *route\_tables\_fresher*:

$$\text{"paadv } i \models_{\mathcal{A}} (\text{recvmgs } r\text{req\_rrep\_sn} \rightarrow) \text{ onll } \Gamma_{AODV} (\lambda((\xi, \_), \_, (\xi', \_)). \forall \text{dip} \in kD(\text{rt } \xi). \text{rt } \xi \sqsubseteq_{\text{dip}} \text{rt } \xi'))"$$

*<proof>*

end

## 5.7 The quality increases predicate

**theory** *E\_Quality\_Increases*

**imports** *E\_Aadv\_Predicates E\_Fresher*

**begin**

**definition** *quality\_increases* :: "state  $\Rightarrow$  state  $\Rightarrow$  bool"

where "quality\_increases  $\xi \xi'$   $\equiv$  ( $\forall \text{dip} \in kD(\text{rt } \xi). \text{dip} \in kD(\text{rt } \xi') \wedge \text{rt } \xi \sqsubseteq_{\text{dip}} \text{rt } \xi'$ )  
 $\wedge$  ( $\forall \text{dip}. \text{sqn } (\text{rt } \xi) \text{ dip} \leq \text{sqn } (\text{rt } \xi') \text{ dip}$ )"

**lemma** *quality\_increasesI* [intro!]:

assumes " $\wedge \text{dip}. \text{dip} \in kD(\text{rt } \xi) \implies \text{dip} \in kD(\text{rt } \xi')$ "  
and " $\wedge \text{dip}. \llbracket \text{dip} \in kD(\text{rt } \xi); \text{dip} \in kD(\text{rt } \xi') \rrbracket \implies \text{rt } \xi \sqsubseteq_{\text{dip}} \text{rt } \xi'$ "  
and " $\wedge \text{dip}. \text{sqn } (\text{rt } \xi) \text{ dip} \leq \text{sqn } (\text{rt } \xi') \text{ dip}$ "  
shows "quality\_increases  $\xi \xi'$ "

*<proof>*

**lemma** *quality\_increasesE* [elim]:

fixes *dip*  
assumes "quality\_increases  $\xi \xi'$ "  
and " $\text{dip} \in kD(\text{rt } \xi)$ "  
and " $\llbracket \text{dip} \in kD(\text{rt } \xi'); \text{rt } \xi \sqsubseteq_{\text{dip}} \text{rt } \xi'; \text{sqn } (\text{rt } \xi) \text{ dip} \leq \text{sqn } (\text{rt } \xi') \text{ dip} \rrbracket \implies R \text{ dip } \xi \xi'$ "  
shows " $R \text{ dip } \xi \xi'$ "

*<proof>*

**lemma** *quality\_increases\_rt\_fresherD* [dest]:

fixes *ip*  
assumes "quality\_increases  $\xi \xi'$ "  
and " $ip \in kD(\text{rt } \xi)$ "  
shows " $\text{rt } \xi \sqsubseteq_{ip} \text{rt } \xi'$ "

*<proof>*

**lemma** *quality\_increases\_sqnE* [elim]:

fixes *dip*  
assumes "quality\_increases  $\xi \xi'$ "  
and " $\text{sqn } (\text{rt } \xi) \text{ dip} \leq \text{sqn } (\text{rt } \xi') \text{ dip} \implies R \text{ dip } \xi \xi'$ "  
shows " $R \text{ dip } \xi \xi'$ "

*<proof>*

**lemma** *quality\_increases\_refl* [intro, simp]: "quality\_increases  $\xi \xi$ "

*<proof>*

**lemma** *strictly\_fresher\_quality\_increases\_right* [elim]:

fixes  $\sigma \sigma'$  *dip*  
assumes " $\text{rt } (\sigma \text{ i}) \sqsubseteq_{\text{dip}} \text{rt } (\sigma \text{ nhip})$ "  
and *qinc*: "quality\_increases  $(\sigma \text{ nhip}) (\sigma' \text{ nhip})$ "  
and " $\text{dip} \in kD(\text{rt } (\sigma \text{ nhip}))$ "  
shows " $\text{rt } (\sigma \text{ i}) \sqsubseteq_{\text{dip}} \text{rt } (\sigma' \text{ nhip})$ "

*<proof>*

**lemma** *kD\_quality\_increases* [elim]:

```

assumes "i∈kD(rt ξ)"
  and "quality_increases ξ ξ'"
  shows "i∈kD(rt ξ')"
⟨proof⟩

lemma kD_nsqn_quality_increases [elim]:
  assumes "i∈kD(rt ξ)"
  and "quality_increases ξ ξ'"
  shows "i∈kD(rt ξ') ∧ nsqn (rt ξ) i ≤ nsqn (rt ξ') i"
⟨proof⟩

lemma nsqn_quality_increases [elim]:
  assumes "i∈kD(rt ξ)"
  and "quality_increases ξ ξ'"
  shows "nsqn (rt ξ) i ≤ nsqn (rt ξ') i"
⟨proof⟩

lemma kD_nsqn_quality_increases_trans [elim]:
  assumes "i∈kD(rt ξ)"
  and "s ≤ nsqn (rt ξ) i"
  and "quality_increases ξ ξ'"
  shows "i∈kD(rt ξ') ∧ s ≤ nsqn (rt ξ') i"
⟨proof⟩

lemma nsqn_quality_increases_nsqn_lt_lt [elim]:
  assumes "i∈kD(rt ξ)"
  and "quality_increases ξ ξ'"
  and "s < nsqn (rt ξ) i"
  shows "s < nsqn (rt ξ') i"
⟨proof⟩

lemma nsqn_quality_increases_dhops [elim]:
  assumes "i∈kD(rt ξ)"
  and "quality_increases ξ ξ'"
  and "nsqn (rt ξ) i = nsqn (rt ξ') i"
  shows "the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i)"
⟨proof⟩

lemma nsqn_quality_increases_nsqn_eq_le [elim]:
  assumes "i∈kD(rt ξ)"
  and "quality_increases ξ ξ'"
  and "s = nsqn (rt ξ) i"
  shows "s < nsqn (rt ξ') i ∨ (s = nsqn (rt ξ') i ∧ the (dhops (rt ξ) i) ≥ the (dhops (rt ξ') i))"
⟨proof⟩

lemma quality_increases_rreq_rrep_props [elim]:
  fixes sn ip hops sip
  assumes qinc: "quality_increases (σ sip) (σ' sip)"
  and "1 ≤ sn"
  and *: "ip∈kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip
  ∧ (nsqn (rt (σ sip)) ip = sn
  → (the (dhops (rt (σ sip)) ip) ≤ hops
  ∨ the (flag (rt (σ sip)) ip) = inv))"
  shows "ip∈kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip
  ∧ (nsqn (rt (σ' sip)) ip = sn
  → (the (dhops (rt (σ' sip)) ip) ≤ hops
  ∨ the (flag (rt (σ' sip)) ip) = inv))"
  (is "_ ∧ ?nsqnafter")
⟨proof⟩

lemma quality_increases_rreq_rrep_props':
  fixes sn ip hops sip
  assumes "∀j. quality_increases (σ j) (σ' j)"
  and "1 ≤ sn"

```

and \*: "ip ∈ kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip  
 ∧ (nsqn (rt (σ sip)) ip = sn  
 → (the (dhops (rt (σ sip)) ip) ≤ hops  
 ∨ the (flag (rt (σ sip)) ip) = inv))"  
 shows "ip ∈ kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip  
 ∧ (nsqn (rt (σ' sip)) ip = sn  
 → (the (dhops (rt (σ' sip)) ip) ≤ hops  
 ∨ the (flag (rt (σ' sip)) ip) = inv))"

⟨proof⟩

lemma rteq\_quality\_increases:

assumes "∀ j. j ≠ i → quality\_increases (σ j) (σ' j)"  
 and "rt (σ' i) = rt (σ i)"  
 shows "∀ j. quality\_increases (σ j) (σ' j)"

⟨proof⟩

definition msg\_fresh :: "(ip ⇒ state) ⇒ msg ⇒ bool"

where "msg\_fresh σ m ≡

case m of Rreq hops \_ \_ oipc osnc sipc \_ ⇒ osnc ≥ 1 ∧ (sipc ≠ oipc →  
 oipc ∈ kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) oipc ≥ osnc  
 ∧ (nsqn (rt (σ sipc)) oipc = osnc  
 → (hops ≥ the (dhops (rt (σ sipc)) oipc)  
 ∨ the (flag (rt (σ sipc)) oipc) = inv)))  
 | Rrep hopsc dipc dsnc \_ sipc ⇒ dsnc ≥ 1 ∧ (sipc ≠ dipc →  
 dipc ∈ kD(rt (σ sipc)) ∧ nsqn (rt (σ sipc)) dipc ≥ dsnc  
 ∧ (nsqn (rt (σ sipc)) dipc = dsnc  
 → (hops ≥ the (dhops (rt (σ sipc)) dipc)  
 ∨ the (flag (rt (σ sipc)) dipc) = inv)))  
 | Rerr destsc sipc ⇒ (∀ ripc ∈ dom(destsc). (ripc ∈ kD(rt (σ sipc))  
 ∧ the (destsc ripc) - 1 ≤ nsqn (rt (σ sipc)) ripc))  
 | \_ ⇒ True"

lemma msg\_fresh [simp]:

"∧ hops dip dsn dsk oip osn sip handled.

msg\_fresh σ (Rreq hops dip dsn dsk oip osn sip handled) =  
 (osn ≥ 1 ∧ (sip ≠ oip → oip ∈ kD(rt (σ sip))  
 ∧ nsqn (rt (σ sip)) oip ≥ osn  
 ∧ (nsqn (rt (σ sip)) oip = osn  
 → (hops ≥ the (dhops (rt (σ sip)) oip)  
 ∨ the (flag (rt (σ sip)) oip) = inv))))"

"∧ hops dip dsn oip sip. msg\_fresh σ (Rrep hops dip dsn oip sip) =  
 (dsn ≥ 1 ∧ (sip ≠ dip → dip ∈ kD(rt (σ sip))  
 ∧ nsqn (rt (σ sip)) dip ≥ dsn  
 ∧ (nsqn (rt (σ sip)) dip = dsn  
 → (hops ≥ the (dhops (rt (σ sip)) dip)  
 ∨ the (flag (rt (σ sip)) dip) = inv))))"

"∧ dests sip. msg\_fresh σ (Rerr dests sip) =  
 (∀ ripc ∈ dom(dests). (ripc ∈ kD(rt (σ sip))  
 ∧ the (dests ripc) - 1 ≤ nsqn (rt (σ sip)) ripc))"

"∧ d dip. msg\_fresh σ (Newpkt d dip) = True"

"∧ d dip sip. msg\_fresh σ (Pkt d dip sip) = True"

⟨proof⟩

lemma msg\_fresh\_inc\_sn [simp, elim]:

"msg\_fresh σ m ⇒ rreq\_rrep\_sn m"

⟨proof⟩

lemma recv\_msg\_fresh\_inc\_sn [simp, elim]:

"orecvmsg (msg\_fresh) σ m ⇒ recvmmsg rreq\_rrep\_sn m"

⟨proof⟩

lemma rreq\_nsqn\_is\_fresh [simp]:

fixes σ msg hops dip dsn dsk oip osn sip handled

assumes "rreq\_rrep\_fresh (rt (σ sip)) (Rreq hops dip dsn dsk oip osn sip handled)"

```

    and "rreq_rrep_sn (Rreq hops dip dsn dsk oip osn sip handled)"
shows "msg_fresh  $\sigma$  (Rreq hops dip dsn dsk oip osn sip handled)"
    (is "msg_fresh  $\sigma$  ?msg")
<proof>

```

```

lemma rrep_nsqn_is_fresh [simp]:
  fixes  $\sigma$  msg hops dip dsn oip sip
  assumes "rreq_rrep_fresh (rt ( $\sigma$  sip)) (Rrep hops dip dsn oip sip)"
    and "rreq_rrep_sn (Rrep hops dip dsn oip sip)"
  shows "msg_fresh  $\sigma$  (Rrep hops dip dsn oip sip)"
    (is "msg_fresh  $\sigma$  ?msg")
<proof>

```

```

lemma rerr_nsqn_is_fresh [simp]:
  fixes  $\sigma$  msg dests sip
  assumes "rerr_invalid (rt ( $\sigma$  sip)) (Rerr dests sip)"
  shows "msg_fresh  $\sigma$  (Rerr dests sip)"
    (is "msg_fresh  $\sigma$  ?msg")
<proof>

```

```

lemma quality_increases_msg_fresh [elim]:
  assumes qinc: " $\forall j$ . quality_increases ( $\sigma$  j) ( $\sigma'$  j)"
    and "msg_fresh  $\sigma$  m"
  shows "msg_fresh  $\sigma'$  m"
<proof>

```

end

## 5.8 The ‘open’ AODV model

```

theory E_OAodv
imports E_Aodv "../..../AWN/OAWN_SOS_Labels" "../..../AWN/OAWN_Convert"
begin

```

Definitions for stating and proving global network properties over individual processes.

```

definition  $\sigma_{AODV}'$  :: " $((ip \Rightarrow state) \times ((state, msg, pseqp, pseqp label) seqp)) set$ "
where " $\sigma_{AODV}' \equiv \{(\lambda i. aodv\_init\ i, \Gamma_{AODV}\ PAodv)\}$ "

```

```

abbreviation opaodv
  :: " $ip \Rightarrow ((ip \Rightarrow state) \times (state, msg, pseqp, pseqp label) seqp, msg seq\_action) automaton$ "
where
  " $opaodv\ i \equiv (\mid\ init = \sigma_{AODV}', trans = oseqp\_sos\ \Gamma_{AODV}\ i\ \mid)$ "

```

```

lemma initiali_aodv [intro!, simp]: "initiali i (init (opaodv i)) (init (paodv i))"
<proof>

```

```

lemma oaodv_control_within [simp]: "control_within  $\Gamma_{AODV}$  (init (opaodv i))"
<proof>

```

```

lemma  $\sigma_{AODV}'$ _labels [simp]: " $(\sigma, p) \in \sigma_{AODV}' \implies labels\ \Gamma_{AODV}\ p = \{PAodv-:0\}$ "
<proof>

```

```

lemma oaodv_init_kD_empty [simp]:
  " $(\sigma, p) \in \sigma_{AODV}' \implies kD\ (rt\ (\sigma\ i)) = \{\}$ "
<proof>

```

```

lemma oaodv_init_vD_empty [simp]:
  " $(\sigma, p) \in \sigma_{AODV}' \implies vD\ (rt\ (\sigma\ i)) = \{\}$ "
<proof>

```

```

lemma oaodv_trans: "trans (opaodv i) = oseqp_sos  $\Gamma_{AODV}$  i"
<proof>

```

declare

```

oseq_invariant_ctermsI [OF aadv_wf oaadv_control_within aadv_simple_labels oaadv_trans, cterms_intros]
oseq_step_invariant_ctermsI [OF aadv_wf oaadv_control_within aadv_simple_labels oaadv_trans, cterms_intros]

```

end

## 5.9 Global invariant proofs over sequential processes

```

theory E_Global_Invariants
imports E_Seq_Invariants
        E_Aadv_Predicates
        E_Fresher
        E_Quality_Increases
        "../.../AWN/OAWN_Convert"
        E_OAadv

begin

lemma other_quality_increases [elim]:
  assumes "other quality_increases I  $\sigma$   $\sigma'$ "
  shows " $\forall j$ . quality_increases ( $\sigma$  j) ( $\sigma'$  j)"
  <proof>

lemma weaken_otherwith [elim]:
  fixes m
  assumes *: "otherwith P I (orecvmsg Q)  $\sigma$   $\sigma'$  a"
  and weakenP: " $\bigwedge \sigma m$ . P  $\sigma$  m  $\implies$  P'  $\sigma$  m"
  and weakenQ: " $\bigwedge \sigma m$ . Q  $\sigma$  m  $\implies$  Q'  $\sigma$  m"
  shows "otherwith P' I (orecvmsg Q')  $\sigma$   $\sigma'$  a"
  <proof>

lemma oreceived_msg_inv:
  assumes other: " $\bigwedge \sigma \sigma' m$ .  $\llbracket P \sigma m$ ; other Q {i}  $\sigma \sigma'$   $\rrbracket \implies P \sigma' m$ "
  and local: " $\bigwedge \sigma m$ . P  $\sigma$  m  $\implies$  P ( $\sigma(i := \sigma i(\text{msg} := m))$ ) m"
  shows "opaadv i  $\models$  (otherwith Q {i} (orecvmsg P), other Q {i}  $\rightarrow$ )
        onl  $\Gamma_{AADV}$  ( $\lambda(\sigma, l)$ .  $l \in \{\text{PAadv}:1\} \longrightarrow P \sigma (\text{msg} (\sigma i))$ )"
  <proof>

(Equivalent to) Proposition 7.27

lemma local_quality_increases:
  "paadv i  $\models_A$  (recvmsg rreq_rrep_sn  $\rightarrow$ ) onll  $\Gamma_{AADV}$  ( $\lambda((\xi, \_), \_, (\xi', \_))$ . quality_increases  $\xi \xi'$ )"
  <proof>

lemmas olocal_quality_increases =
  open_seq_step_invariant [OF local_quality_increases initiali_aadv oaadv_trans aadv_trans,
    simplified seqll_onll_swap]

lemma oquality_increases:
  "opaadv i  $\models_A$  (otherwith quality_increases {i} (orecvmsg ( $\lambda\_$ . rreq_rrep_sn)),
    other quality_increases {i}  $\rightarrow$ )
    onll  $\Gamma_{AADV}$  ( $\lambda((\sigma, \_), \_, (\sigma', \_))$ .  $\forall j$ . quality_increases ( $\sigma$  j) ( $\sigma'$  j))"
  (is " $\_ \models_A$  (?S,  $\_ \rightarrow$ )  $\_$ ")
  <proof>

lemma rreq_rrep_nsqn_fresh_any_step_invariant:
  "opaadv i  $\models_A$  (act (recvmsg rreq_rrep_sn), other A {i}  $\rightarrow$ )
    onll  $\Gamma_{AADV}$  ( $\lambda((\sigma, \_), a, \_)$ . anycast (msg_fresh  $\sigma$ ) a)"
  <proof>

lemma oreceived_rreq_rrep_nsqn_fresh_inv:
  "opaadv i  $\models$  (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i}  $\rightarrow$ )
    onl  $\Gamma_{AADV}$  ( $\lambda(\sigma, l)$ .  $l \in \{\text{PAadv}:1\} \longrightarrow \text{msg\_fresh } \sigma (\text{msg} (\sigma i))$ )"
  <proof>

lemma oquality_increases_nsqn_fresh:

```

"opaadv i  $\models_A$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onll  $\Gamma_{AODV}$  ( $\lambda((\sigma, \_), \_, (\sigma', \_)). \forall j. \text{quality\_increases } (\sigma j) (\sigma' j)$ )")  
 <proof>

lemma oosn\_rreq:

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onl  $\Gamma_{AODV}$  (seq1 i ( $\lambda(\xi, l). l \in \{PAadv-:4, PAadv-:5\} \cup \{PRreq-:n \mid n. \text{True}\} \rightarrow 1 \leq \text{osn } \xi$ ))")  
 <proof>

lemma rreq\_sip:

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onl  $\Gamma_{AODV}$  ( $\lambda(\sigma, l).$   
 ( $l \in \{PAadv-:4, PAadv-:5, PRreq-:0, PRreq-:2\} \wedge \text{sip } (\sigma i) \neq \text{oip } (\sigma i)$ )  
 $\rightarrow \text{oip } (\sigma i) \in \text{kD}(\text{rt } (\sigma (\text{sip } (\sigma i))))$   
 $\wedge \text{nsqn } (\text{rt } (\sigma (\text{sip } (\sigma i)))) (\text{oip } (\sigma i)) \geq \text{osn } (\sigma i)$   
 $\wedge (\text{nsqn } (\text{rt } (\sigma (\text{sip } (\sigma i)))) (\text{oip } (\sigma i)) = \text{osn } (\sigma i)$   
 $\rightarrow (\text{hops } (\sigma i) \geq \text{the } (\text{dhops } (\text{rt } (\sigma (\text{sip } (\sigma i)))) (\text{oip } (\sigma i)))$   
 $\vee \text{the } (\text{flag } (\text{rt } (\sigma (\text{sip } (\sigma i)))) (\text{oip } (\sigma i))) = \text{inv}))$ )")  
 (is "\_  $\models$  (?S, ?U  $\rightarrow$ ) \_")  
 <proof>

lemma odsn\_rrep:

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onl  $\Gamma_{AODV}$  (seq1 i ( $\lambda(\xi, l). l \in \{PAadv-:6, PAadv-:7\} \cup \{PRrep-:n \mid n. \text{True}\} \rightarrow 1 \leq \text{dsn } \xi$ ))")  
 <proof>

lemma rrep\_sip:

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onl  $\Gamma_{AODV}$  ( $\lambda(\sigma, l).$   
 ( $l \in \{PAadv-:6, PAadv-:7, PRrep-:0, PRrep-:1\} \wedge \text{sip } (\sigma i) \neq \text{dip } (\sigma i)$ )  
 $\rightarrow \text{dip } (\sigma i) \in \text{kD}(\text{rt } (\sigma (\text{sip } (\sigma i))))$   
 $\wedge \text{nsqn } (\text{rt } (\sigma (\text{sip } (\sigma i)))) (\text{dip } (\sigma i)) \geq \text{dsn } (\sigma i)$   
 $\wedge (\text{nsqn } (\text{rt } (\sigma (\text{sip } (\sigma i)))) (\text{dip } (\sigma i)) = \text{dsn } (\sigma i)$   
 $\rightarrow (\text{hops } (\sigma i) \geq \text{the } (\text{dhops } (\text{rt } (\sigma (\text{sip } (\sigma i)))) (\text{dip } (\sigma i)))$   
 $\vee \text{the } (\text{flag } (\text{rt } (\sigma (\text{sip } (\sigma i)))) (\text{dip } (\sigma i))) = \text{inv}))$ )")  
 (is "\_  $\models$  (?S, ?U  $\rightarrow$ ) \_")  
 <proof>

lemma rerr\_sip:

"opaadv i  $\models$  (otherwith quality\_increases {i} (orecvmsg msg\_fresh),  
 other quality\_increases {i}  $\rightarrow$ )  
 onl  $\Gamma_{AODV}$  ( $\lambda(\sigma, l).$   
 ( $l \in \{PAadv-:8, PAadv-:9, PRerr-:0, PRerr-:1\}$   
 $\rightarrow (\forall \text{ripc} \in \text{dom}(\text{dests } (\sigma i)). \text{ripc} \in \text{kD}(\text{rt } (\sigma (\text{sip } (\sigma i)))) \wedge$   
 $\text{the } (\text{dests } (\sigma i) \text{ ripc}) - 1 \leq \text{nsqn } (\text{rt } (\sigma (\text{sip } (\sigma i)))) \text{ ripc}))$ )")  
 (is "\_  $\models$  (?S, ?U  $\rightarrow$ ) \_")  
 <proof>

lemma prerr\_guard: "paadv i  $\models$

onl  $\Gamma_{AODV}$  ( $\lambda(\xi, l). (l = \text{PRerr-:1}$   
 $\rightarrow (\forall \text{ip} \in \text{dom}(\text{dests } \xi). \text{ip} \in \text{vD}(\text{rt } \xi)$   
 $\wedge \text{the } (\text{nhop } (\text{rt } \xi) \text{ ip}) = \text{sip } \xi$   
 $\wedge \text{sqn } (\text{rt } \xi) \text{ ip} < \text{the } (\text{dests } \xi \text{ ip})))$ )")

<proof>

lemmas odests\_vD\_inc\_sqn =

open\_seq\_invariant [OF dests\_vD\_inc\_sqn initiali\_aadv oaadv\_trans aadv\_trans,  
 simplified seq1\_onl\_swap,  
 THEN oinvariant\_anyact]

```

lemmas oprerr_guard =
  open_seq_invariant [OF prerr_guard initiali_aodv oaodv_trans aodv_trans,
    simplified seq1_onl_swap,
    THEN oinvariant_anyact]

```

Proposition 7.28

```

lemma seq_compare_next_hop':
  "opaodv i  $\models$  (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i}  $\rightarrow$ ) onl  $\Gamma_{AODV}$  ( $\lambda(\sigma, \_)$ .
     $\forall$ dip. let nhip = the (nhop (rt ( $\sigma$  i)) dip)
      in dip  $\in$  kD(rt ( $\sigma$  i))  $\wedge$  nhip  $\neq$  dip  $\rightarrow$ 
        dip  $\in$  kD(rt ( $\sigma$  nhip))  $\wedge$  nsqn (rt ( $\sigma$  i)) dip  $\leq$  nsqn (rt ( $\sigma$  nhip)) dip)"
  (is "_  $\models$  (?S, ?U  $\rightarrow$ ) _")
  <proof>

```

Proposition 7.30

```

lemmas okD_unk_or_atleast_one =
  open_seq_invariant [OF kD_unk_or_atleast_one initiali_aodv,
    simplified seq1_onl_swap]

```

```

lemmas ozero_seq_unk_hops_one =
  open_seq_invariant [OF zero_seq_unk_hops_one initiali_aodv,
    simplified seq1_onl_swap]

```

```

lemma oreachable_fresh_okD_unk_or_atleast_one:
  fixes dip
  assumes "( $\sigma, p$ )  $\in$  oreachable (opaodv i)
    (otherwith (op=) {i} (orecvmsg ( $\lambda\sigma m$ . msg_fresh  $\sigma$  m
       $\wedge$  msg_zhops m)))
    (other quality_increases {i})"
  and "dip  $\in$  kD(rt ( $\sigma$  i))"
  shows " $\pi_3$ (the (rt ( $\sigma$  i) dip)) = unk  $\vee$  1  $\leq$   $\pi_2$ (the (rt ( $\sigma$  i) dip))"
  (is "?P dip")
  <proof>

```

```

lemma oreachable_fresh_ozero_seq_unk_hops_one:
  fixes dip
  assumes "( $\sigma, p$ )  $\in$  oreachable (opaodv i)
    (otherwith (op=) {i} (orecvmsg ( $\lambda\sigma m$ . msg_fresh  $\sigma$  m
       $\wedge$  msg_zhops m)))
    (other quality_increases {i})"
  and "dip  $\in$  kD(rt ( $\sigma$  i))"
  shows "sqn (rt ( $\sigma$  i)) dip = 0  $\rightarrow$ 
    sqnf (rt ( $\sigma$  i)) dip = unk
     $\wedge$  the (dhops (rt ( $\sigma$  i)) dip) = 1
     $\wedge$  the (nhop (rt ( $\sigma$  i)) dip) = dip"
  (is "?P dip")
  <proof>

```

```

lemma seq_nhop_quality_increases':
  shows "opaodv i  $\models$  (otherwith (op=) {i}
    (orecvmsg ( $\lambda\sigma m$ . msg_fresh  $\sigma$  m  $\wedge$  msg_zhops m)),
    other quality_increases {i}  $\rightarrow$ )
    onl  $\Gamma_{AODV}$  ( $\lambda(\sigma, \_)$ .  $\forall$ dip. let nhip = the (nhop (rt ( $\sigma$  i)) dip)
      in dip  $\in$  vD (rt ( $\sigma$  i))  $\cap$  vD (rt ( $\sigma$  nhip))
         $\wedge$  nhip  $\neq$  dip
         $\rightarrow$  (rt ( $\sigma$  i))  $\sqsubset_{dip}$  (rt ( $\sigma$  nhip)))"
  (is "_  $\models$  (?S i, _  $\rightarrow$ ) _")
  <proof>

```

```

lemma seq_compare_next_hop:
  fixes w
  shows "opaodv i  $\models$  (otherwith (op=) {i} (orecvmsg msg_fresh),

```



```

other quality_increases {i} →)
global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
in dip ∈ kD(rt (σ i)) ∧ nhip ≠ dip →
dip ∈ kD(rt (σ nhip))
∧ nsqn (rt (σ i)) dip ≤ nsqn (rt (σ nhip)) dip)"

```

⟨proof⟩

**lemma seq\_nhop\_quality\_increases:**

```

shows "opaadv i ⊨ (otherwith (op=) {i}
(orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
other quality_increases {i} →)
global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
→ (rt (σ i)) ⊑dip (rt (σ nhip)))"

```

⟨proof⟩

end

## 5.10 Routing graphs and loop freedom

```

theory E_Loop_Freedom
imports E_Aadv_Predicates E_Fresher
begin

```

Define the central theorem that relates an invariant over network states to the absence of loops in the associate routing graph.

**definition**

```

rt_graph :: "(ip ⇒ state) ⇒ ip ⇒ ip rel"
where
"rt_graph σ = (λdip.
{(ip, ip') | ip ip' dsn dsk hops.
ip ≠ dip ∧ rt (σ ip) dip = Some (dsn, dsk, val, hops, ip')})"

```

Given the state of a network  $\sigma$ , a routing graph for a given destination  $dip$  abstracts the details of routing tables into nodes (ip addresses) and vertices (valid routes between ip addresses).

**lemma rt\_graphE [elim]:**

```

fixes n dip ip ip'
assumes "(ip, ip') ∈ rt_graph σ dip"
shows "ip ≠ dip ∧ (∃r. rt (σ ip) = r
∧ (∃dsn dsk hops. r dip = Some (dsn, dsk, val, hops, ip')))"

```

⟨proof⟩

**lemma rt\_graph\_vD [dest]:**

```

"∧ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⇒ dip ∈ vD(rt (σ ip))"

```

⟨proof⟩

**lemma rt\_graph\_vD\_trans [dest]:**

```

"∧ip ip' σ dip. (ip, ip') ∈ (rt_graph σ dip)+ ⇒ dip ∈ vD(rt (σ ip))"

```

⟨proof⟩

**lemma rt\_graph\_not\_dip [dest]:**

```

"∧ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⇒ ip ≠ dip"

```

⟨proof⟩

**lemma rt\_graph\_not\_dip\_trans [dest]:**

```

"∧ip ip' σ dip. (ip, ip') ∈ (rt_graph σ dip)+ ⇒ ip ≠ dip"

```

⟨proof⟩

NB: the property below cannot be lifted to the transitive closure

**lemma rt\_graph\_nhip\_is\_nhop [dest]:**

```

"∧ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⇒ ip' = the (nhop (rt (σ ip)) dip)"

```

⟨proof⟩

```

theorem inv_to_loop_freedom:
  assumes "∀ i dip. let nhip = the (nhop (rt (σ i)) dip)
              in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
              → (rt (σ i)) ⊆dip (rt (σ nhip))"
  shows "∀ dip. irrefl ((rt_graph σ dip)+)"
  ⟨proof⟩
end

```

## 5.11 Lift and transfer invariants to show loop freedom

```

theory E_Aodv_Loop_Freedom
imports "../.../AWN/OClosed_Transfer" "../.../AWN/Qmsg_Lifting" E_Global_Invariants E_Loop_Freedom
begin

```

### 5.11.1 Lift to parallel processes with queues

```

lemma par_step_no_change_on_send_or_receive:
  fixes σ s a σ' s'
  assumes "((σ, s), a, (σ', s')) ∈ oparp_sos i (oseqp_sos ΓAODV i) (seqp_sos ΓQMSG)"
  and "a ≠ τ"
  shows "σ' i = σ i"
  ⟨proof⟩

```

```

lemma par_nhop_quality_increases:
  shows "opaodv i ⟨⟨i qmsg ⊢ (otherwith (op=) {i} (orecvmsg (λσ m.
    msg_fresh σ m ∧ msg_zhops m)),
    other quality_increases {i} →)
    global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)
    in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
    → (rt (σ i)) ⊆dip (rt (σ nhip)))⟩"
  ⟨proof⟩

```

```

lemma par_rreq_rrep_sn_quality_increases:
  "opaodv i ⟨⟨i qmsg ⊢A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
    globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))⟩"
  ⟨proof⟩

```

```

lemma par_rreq_rrep_nsqn_fresh_any_step:
  shows "opaodv i ⟨⟨i qmsg ⊢A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ,
    other (λ_ _. True) {i} →)
    globala (λ(σ, a, σ'). anycast (msg_fresh σ) a)⟩"
  ⟨proof⟩

```

```

lemma par_anycast_msg_zhops:
  shows "opaodv i ⟨⟨i qmsg ⊢A (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
    globala (λ(_, a, _). anycast msg_zhops a)⟩"
  ⟨proof⟩

```

### 5.11.2 Lift to nodes

```

lemma node_step_no_change_on_send_or_receive:
  assumes "((σ, NodeS i P R), a, (σ', NodeS i' P' R')) ∈ onode_sos
              (oparp_sos i (oseqp_sos ΓAODV i) (seqp_sos ΓQMSG))"
  and "a ≠ τ"
  shows "σ' i = σ i"
  ⟨proof⟩

```

```

lemma node_nhop_quality_increases:
  shows "⟨ i : opaodv i ⟨⟨i qmsg : R ⟩o ⊢
    (otherwith (op=) {i}
    (oarrivmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
    other quality_increases {i}
    →) global (λσ. ∀dip. let nhip = the (nhop (rt (σ i)) dip)

```

$$\text{in dip} \in \text{vD} (\text{rt} (\sigma \text{ i})) \cap \text{vD} (\text{rt} (\sigma \text{ nhip})) \wedge \text{nhip} \neq \text{dip} \\ \longrightarrow (\text{rt} (\sigma \text{ i})) \sqsubset_{\text{dip}} (\text{rt} (\sigma \text{ nhip}))"$$

*<proof>*

**lemma node\_quality\_increases:**

$$\langle i : \text{opaadv } i \langle \langle_i \text{qmsg} : R \rangle_o \models_A (\lambda \sigma \_ . \text{oarrivemsg} (\lambda \_ . \text{rreq\_rrep\_sn}) \sigma, \\ \text{other} (\lambda \_ \_ . \text{True}) \{i\} \rightarrow) \\ \text{globala} (\lambda (\sigma, \_ , \sigma') . \text{quality\_increases} (\sigma \text{ i}) (\sigma' \text{ i})) \rangle"$$

*<proof>*

**lemma node\_rreq\_rrep\_nsqn\_fresh\_any\_step:**

$$\text{shows} \langle i : \text{opaadv } i \langle \langle_i \text{qmsg} : R \rangle_o \models_A \\ (\lambda \sigma \_ . \text{oarrivemsg} (\lambda \_ . \text{rreq\_rrep\_sn}) \sigma, \text{other} (\lambda \_ \_ . \text{True}) \{i\} \rightarrow) \\ \text{globala} (\lambda (\sigma, a, \sigma') . \text{castmsg} (\text{msg\_fresh } \sigma) a) \rangle"$$

*<proof>*

**lemma node\_anycast\_msg\_zhops:**

$$\text{shows} \langle i : \text{opaadv } i \langle \langle_i \text{qmsg} : R \rangle_o \models_A \\ (\lambda \sigma \_ . \text{oarrivemsg} (\lambda \_ . \text{rreq\_rrep\_sn}) \sigma, \text{other} (\lambda \_ \_ . \text{True}) \{i\} \rightarrow) \\ \text{globala} (\lambda (\_, a, \_) . \text{castmsg } \text{msg\_zhops } a) \rangle"$$

*<proof>*

**lemma node\_silent\_change\_only:**

$$\text{shows} \langle i : \text{opaadv } i \langle \langle_i \text{qmsg} : R_i \rangle_o \models_A (\lambda \sigma \_ . \text{oarrivemsg} (\lambda \_ \_ . \text{True}) \sigma, \\ \text{other} (\lambda \_ \_ . \text{True}) \{i\} \rightarrow) \\ \text{globala} (\lambda (\sigma, a, \sigma') . a \neq \tau \longrightarrow \sigma' \text{ i} = \sigma \text{ i}) \rangle"$$

*<proof>*

### 5.11.3 Lift to partial networks

**lemma arrive\_rreq\_rrep\_nsqn\_fresh\_inc\_sn [simp]:**

$$\text{assumes} \text{"oarrivemsg} (\lambda \sigma m . \text{msg\_fresh } \sigma m \wedge P \sigma m) \sigma m"$$

$$\text{shows} \text{"oarrivemsg} (\lambda \_ . \text{rreq\_rrep\_sn}) \sigma m"$$

*<proof>*

**lemma opnet\_nhop\_quality\_increases:**

$$\text{shows} \text{"opnet} (\lambda i . \text{opaadv } i \langle \langle_i \text{qmsg} \rangle p \models \\ (\text{otherwith} (\text{op} =) (\text{net\_tree\_ips } p) \\ (\text{oarrivemsg} (\lambda \sigma m . \text{msg\_fresh } \sigma m \wedge \text{msg\_zhops } m)), \\ \text{other } \text{quality\_increases} (\text{net\_tree\_ips } p) \rightarrow) \\ \text{global} (\lambda \sigma . \forall i \in \text{net\_tree\_ips } p . \forall \text{dip} . \\ \text{let } \text{nhip} = \text{the} (\text{nhop} (\text{rt} (\sigma \text{ i})) \text{dip}) \\ \text{in } \text{dip} \in \text{vD} (\text{rt} (\sigma \text{ i})) \cap \text{vD} (\text{rt} (\sigma \text{ nhip})) \wedge \text{nhip} \neq \text{dip} \\ \longrightarrow (\text{rt} (\sigma \text{ i})) \sqsubset_{\text{dip}} (\text{rt} (\sigma \text{ nhip})) \rangle"$$

*<proof>*

### 5.11.4 Lift to closed networks

**lemma onet\_nhop\_quality\_increases:**

$$\text{shows} \text{"oclosed} (\text{opnet} (\lambda i . \text{opaadv } i \langle \langle_i \text{qmsg} \rangle p) \\ \models (\lambda \_ \_ \_ . \text{True}, \text{other } \text{quality\_increases} (\text{net\_tree\_ips } p) \rightarrow) \\ \text{global} (\lambda \sigma . \forall i \in \text{net\_tree\_ips } p . \forall \text{dip} . \\ \text{let } \text{nhip} = \text{the} (\text{nhop} (\text{rt} (\sigma \text{ i})) \text{dip}) \\ \text{in } \text{dip} \in \text{vD} (\text{rt} (\sigma \text{ i})) \cap \text{vD} (\text{rt} (\sigma \text{ nhip})) \wedge \text{nhip} \neq \text{dip} \\ \longrightarrow (\text{rt} (\sigma \text{ i})) \sqsubset_{\text{dip}} (\text{rt} (\sigma \text{ nhip})) \rangle"$$

*(is " \_  $\models$  ( \_ , ?U  $\rightarrow$ ) ?inv")*

*<proof>*

### 5.11.5 Transfer into the standard model

**interpretation aadv\_openproc: openproc paadv opaadv id**

**rewrites "aadv\_openproc.initmissing = initmissing"**

*<proof>*

**interpretation aadv\_openproc\_par\_qmsg: openproc\_parq paadv opaadv id qmsg**

```
rewrites "aadv_openproc_par_qmsg.netglobal = netglobal"
  and "aadv_openproc_par_qmsg.initmissing = initmissing"
⟨proof⟩
```

**lemma** *net\_nhop\_quality\_increases*:

```
assumes "wf_net_tree n"
shows "closed (pnet (λi. paadv i ⟨⟨ qmsg⟩ n) ≡ netglobal
  (λσ. ∀i dip. let nhip = the (nhop (rt (σ i)) dip)
    in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
    → (rt (σ i)) ⊆dip (rt (σ nhip))))"
  (is "_ ≡ netglobal (λσ. ∀i. ?inv σ i)")
⟨proof⟩
```

### 5.11.6 Loop freedom of AODV

**theorem** *aadv\_loop\_freedom*:

```
assumes "wf_net_tree n"
shows "closed (pnet (λi. paadv i ⟨⟨ qmsg⟩ n) ≡ netglobal (λσ. ∀dip. irrefl ((rt_graph σ dip)+))"
⟨proof⟩
```

**end**

# Bibliography

- [1] T. Bourke. Mechanization of the Algebra for Wireless Networks (AWN). *Archive of Formal Proofs*, 2014. <http://isa-afp.org/entries/AWN.shtml>.
- [2] T. Bourke, R. J. van Glabbeek, and P. Höfner. A mechanized proof of loop freedom of the (untimed) AODV routing protocol. In F. Cassez and J.-F. Raskin, editors, *Proceedings of the 12th International Conference on Automated Technology for Verification and Analysis (ATVA 2014)*, volume 8837 of *Lecture Notes in Computer Science*, pages 47–63, Sydney, Australia, Nov. 2014. Springer.
- [3] T. Bourke, R. J. van Glabbeek, and P. Höfner. Showing invariance compositionally for a process algebra for network protocols. In G. Klein and R. Gamboa, editors, *Proceedings of the 5th International Conference on Interactive Theorem Proving (ITP 2014)*, volume 8558 of *Lecture Notes in Computer Science*, pages 144–159, Vienna, Austria, July 2014. Springer.
- [4] A. Fehnker, R. J. van Glabbeek, P. Höfner, A. McIver, M. Portmann, and W. L. Tan. A process algebra for wireless mesh networks used for modelling, verifying and analysing AODV. Technical Report 5513, NICTA, 2013.
- [5] S. Miskovic and E. W. Knightly. Routing primitives for wireless mesh networks: Design, analysis and experiments. In *Conference on Information Communications (INFOCOM '10)*, pages 2793–2801. IEEE, 2010.
- [6] C. E. Perkins, E. M. Belding-Royer, and S. R. Das. Ad hoc on-demand distance vector (AODV) routing. RFC 3561 (Experimental), Network Working Group, 2003.