

# Loop freedom of the (untimed) AODV routing protocol

Timothy Bourke<sup>1</sup>                  Peter Höfner<sup>2</sup>

March 17, 2025

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

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

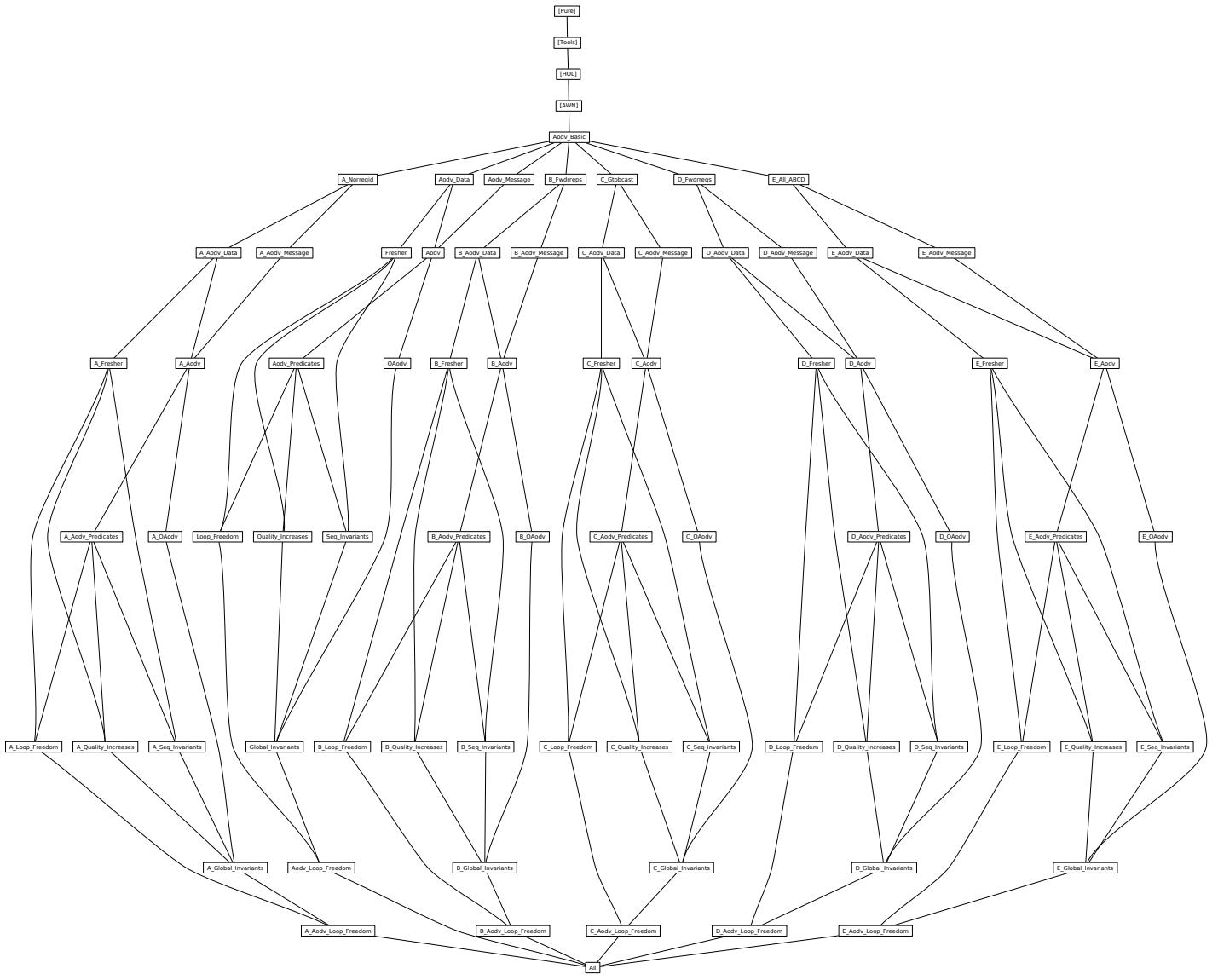
## Abstract

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show that Isabelle/HOL can re-establish most proof obligations automatically and identify exactly the steps that are no longer valid.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 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 ≡ Known"
abbreviation unk where "unk ≡ Unknown"

datatype p = NoRequestRequired | RequestRequired
abbreviation noreq where "noreq ≡ NoRequestRequired"
abbreviation req where "req ≡ RequestRequired"

datatype f = Valid | Invalid
abbreviation val where "val ≡ Valid"
abbreviation inv where "inv ≡ Invalid"

lemma not_ks [simp]:
  "(x ≠ kno) = (x = unk)"
  "(x ≠ unk) = (x = kno)"
  ⟨proof⟩

lemma not_ps [simp]:
  "(x ≠ noreq) = (x = req)"
  "(x ≠ req) = (x = noreq)"
  ⟨proof⟩

lemma not_ffs [simp]:
  "(x ≠ val) = (x = inv)"
  "(x ≠ 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 ⇒ sqn"
  where "inc sn ≡ if sn = 0 then sn else sn + 1"

lemma less_than_inc [simp]: "x ≤ inc x"
  ⟨proof⟩

lemma inc_minus_suc_0 [simp]:
  "inc x - Suc 0 = x"
  ⟨proof⟩

lemma inc_never_one' [simp, intro]: "inc x ≠ Suc 0"
  ⟨proof⟩

lemma inc_never_one [simp, intro]: "inc x ≠ 1"
  ⟨proof⟩
```

## 0.2.2 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 × k × f × nat × ip × ip set"

definition proj2 :: "r ⇒ sqn" (<π2>)
  where "π2 ≡ λ(dsн, _, _, _, _, _). 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(_, _, _, _, nhop, _). nhop"

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

lemma projs [simp]:
  "π2(dsн, dsk, flag, hops, nhop, pre) = dsn"
  "π3(dsн, dsk, flag, hops, nhop, pre) = dsk"
  "π4(dsн, dsk, flag, hops, nhop, pre) = flag"
  "π5(dsн, dsk, flag, hops, nhop, pre) = hops"
  "π6(dsн, dsk, flag, hops, nhop, pre) = nhop"
  "π7(dsн, dsk, flag, hops, nhop, 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]:
  " $\bigwedge ip\ rt. ip \in vD\ rt \implies ip \in kD\ rt$ "  

  " $\bigwedge ip\ rt. ip \in iD\ rt \implies ip \in kD\ rt$ "
  ⟨proof⟩

lemma kD_Some [dest]:
  fixes dip rt
  assumes "dip ∈ kD rt"
  shows "∃ dsn dsk flag hops nhip 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)"

```

$\langle proof \rangle$

```
lemma inv_is_id [elim]:  
  fixes ip rt  
  assumes "ip ∈ kD(rt)"  
    and "the (flag rt ip) = inv"  
  shows "ip ∈ iD(rt)"  
 $\langle proof \rangle$ 
```

```
lemma iD_flag_is_inv [elim, simp]:  
  fixes ip rt  
  assumes "ip ∈ iD(rt)"  
  shows "the (flag rt ip) = inv"  
 $\langle proof \rangle$ 
```

```
lemma kD_but_not_vD_is_id [elim]:  
  fixes ip rt  
  assumes "ip ∈ kD(rt)"  
    and "ip ∉ vD(rt)"  
  shows "ip ∈ iD(rt)"  
 $\langle proof \rangle$ 
```

```
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"  
 $\langle proof \rangle$ 
```

```
lemma proj5_eq_dhops: "¬ dip rt. dip ∈ kD(rt) ⟹ π_5(the (rt dip)) = the (dhops rt dip)"  
 $\langle proof \rangle$ 
```

```
lemma proj4_eq_flag: "¬ dip rt. dip ∈ kD(rt) ⟹ π_4(the (rt dip)) = the (flag rt dip)"  
 $\langle proof \rangle$ 
```

```
lemma proj2_eq_sqn: "¬ dip rt. dip ∈ kD(rt) ⟹ π_2(the (rt dip)) = sqn rt dip"  
 $\langle proof \rangle$ 
```

```
lemma kD_sqnf_is_proj3 [simp]:  
  "¬ ip rt. ip ∈ kD(rt) ⟹ sqnf rt ip = π_3(the (rt ip))"  
 $\langle proof \rangle$ 
```

```
lemma vD_flag_val [simp]:  
  "¬ dip rt. dip ∈ vD(rt) ⟹ the (flag rt dip) = val"  
 $\langle proof \rangle$ 
```

```
lemma kD_update [simp]:  
  "¬ rt nip v. kD(rt(nip ↦ v)) = insert nip (kD rt)"  
 $\langle proof \rangle$ 
```

```
lemma kD_empty [simp]: "kD Map.empty = {}"  
 $\langle proof \rangle$ 
```

```
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'"  
 $\langle proof \rangle$ 
```

## 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 ⇒ 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, f1, hops, nhip, pre) npre = (dsn, dsk, f1, 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) ⇒ sqnf (the (addpreRT (rt) ip npre)) dip = sqnf (rt) 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 ns rt ip
  defines "s ≡ the σroute(rt, ip)"
  and "nr ≡ addpre r (π7(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 $rt\ ip = Some\ inv]$	$\implies update\ rt\ ip\ r = rt\ (ip \mapsto nr)$ "
" $[ip \in kD(rt); \pi_3(r) = unk; (\pi_2(r) = 0) = (\pi_3(r) = unk)]$	$\implies update\ rt\ ip\ r = rt\ (ip \mapsto nr')$ "
" $[ip \in kD(rt); sqn\ rt\ ip \geq \pi_2(r); \pi_3(r) = kno;$ $sqn\ rt\ ip = \pi_2(r) \implies the\ (dhops\ rt\ ip) \leq \pi_5(r) \wedge the\ (flag\ rt\ ip) = val]$	$\implies update\ rt\ ip\ r = rt\ (ip \mapsto ns)$ "

$\langle proof \rangle$

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 " $(P(update\ rt\ ip\ r))$ "

$\langle proof \rangle$

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 " $(P(update\ rt\ ip\ r))$ "

$\langle proof \rangle$

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

fixes rt nip dsn dsk flag hops nhip pre  
 shows " $kD(update\ rt\ nip\ (dsn, dsk, flag, hops, nhip, pre)) = insert\ nip\ (kD\ rt)$ "  
 $\langle proof \rangle$

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"  
 $\langle proof \rangle$

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:
 "¬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"

```

$\langle proof \rangle$

```
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')"
```

$\langle proof \rangle$

## 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]:
```

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

```
lemma proj5_invalidate [simp]:
```

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

```
lemma proj6_invalidate [simp]:
```

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

```
lemma proj7_invalidate [simp]:
```

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

```
lemma invalidate_kD_inv [simp]:
```

```
  " $\bigwedge rt\ dests.$  kD (invalidate rt dests) = kD rt"  
 $\langle proof \rangle$ 
```

```
lemma invalidate_sqn:
```

```
  fixes rt dip dests  
  assumes " $\forall rsn.$  dests dip = Some rsn  $\longrightarrow$  sqn rt dip  $\leq$  rsn"  
  shows "sqn rt dip  $\leq$  sqn (invalidate rt dests) dip"  
 $\langle proof \rangle$ 
```

```
lemma sqn_invalidate_in_dests [simp]:
```

```
  fixes dests ipa rsn rt  
  assumes "dests ipa = Some rsn"  
  and "ipa \in kD(rt)"  
  shows "sqn (invalidate rt dests) ipa = rsn"  
 $\langle proof \rangle$ 
```

```
lemma dhops_invalidate [simp]:
```

```
  " $\bigwedge dip.$  the (dhops (invalidate rt dests) dip) = the (dhops rt dip)"  
 $\langle proof \rangle$ 
```

```
lemma sqnf_invalidate [simp]:
```

```
  " $\bigwedge dip.$  sqnf (invalidate (rt \xi) (dests \xi)) dip = sqnf (rt \xi) dip"  
 $\langle proof \rangle$ 
```

```
lemma nhop_invalidate [simp]:
```

```
  " $\bigwedge dip.$  the (nhop (invalidate (rt \xi) (dests \xi)) dip) = the (nhop (rt \xi) dip)"  
 $\langle proof \rangle$ 
```

```
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 ∈ 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 ⇒ 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))
         ∧ pre = π₇(the (rt dip))"
⟨proof⟩

lemma proj3_inv: "¬(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⟩

```

## 0.2.5 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"

```

## 0.2.6 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))"

```

### 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.  $\sigma_{p\text{-}flag}(store, dip)$  is partial.
4. The routing table (*rt*) is modelled as a map ( $ip \Rightarrow r\ option$ ) 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

## 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 → sqn" ip
  | Newpkt data ip
  | Pkt data ip ip

instantiation msg :: msg
begin
  definition newpkt_def [simp]: "newpkt ≡ λ(d, dip). Newpkt d dip"
  definition eq_newpkt_def: "eq_newpkt m ≡ case m of Newpkt d dip ⇒ True | _ ⇒ 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"
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 "aodv_init i ≡ ()"
  ip = i,
  sn = 1,
  rt = Map.empty,
  rreqs = {},
  store = Map.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⟩

```

#### 0.4.2 Auxilliary message handling definitions

```

definition is_newpkt
where "is_newpkt  $\xi$  ≡ case msg  $\xi$  of
      Newpkt data' dip' ⇒ { $\xi$ (data := data', dip := dip') }
      | _ ⇒ {}"

definition is_pkt
where "is_pkt  $\xi$  ≡ case msg  $\xi$  of
      Pkt data' dip' oip' ⇒ { $\xi$ (data := data', dip := dip', oip := oip') }
      | _ ⇒ {}"

definition is_rreq
where "is_rreq  $\xi$  ≡ case msg  $\xi$  of
      Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip' ⇒
          { $\xi$ (hops := hops', rreqid := rreqid', dip := dip', dsn := dsn',
           dsk := dsk', oip := oip', osn := osn', sip := sip') }
      | _ ⇒ {}"

lemma is_rreq_asm [dest!]:
assumes " $\xi'$  ∈ is_rreq  $\xi$ "
shows "( $\exists$  hops' rreqid' dip' dsn' dsk' oip' osn' sip'.
      msg  $\xi$  = Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip' ∧
       $\xi'$  =  $\xi$ (hops := hops', rreqid := rreqid', dip := dip', dsn := dsn',
           dsk := dsk', oip := oip', osn := osn', sip := sip'))"
⟨proof⟩

definition is_rrep
where "is_rrep  $\xi$  ≡ case msg  $\xi$  of
      Rrep hops' dip' dsn' oip' sip' ⇒
          { $\xi$ (hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip') }
      | _ ⇒ {}"

lemma is_rrep_asm [dest!]:
assumes " $\xi'$  ∈ is_rrep  $\xi$ "
shows "( $\exists$  hops' dip' dsn' oip' sip'.
      msg  $\xi$  = Rrep hops' dip' dsn' oip' sip' ∧
       $\xi'$  =  $\xi$ (hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip'))"
⟨proof⟩

definition is_rerr
where "is_rerr  $\xi$  ≡ case msg  $\xi$  of
      Rerr dests' sip' ⇒ { $\xi$ (dests := dests', sip := sip') }
      | _ ⇒ {}"

lemma is_rerr_asm [dest!]:
assumes " $\xi'$  ∈ is_rerr  $\xi$ "
shows "( $\exists$  dests' sip'.
      msg  $\xi$  = Rerr dests' sip' ∧
       $\xi'$  =  $\xi$ (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]:
" $\xi'$  ∈ is_rerr  $\xi$  ⇒ ip  $\xi'$  = ip  $\xi$ "
" $\xi'$  ∈ is_rrep  $\xi$  ⇒ ip  $\xi'$  = ip  $\xi$ "
" $\xi'$  ∈ is_rreq  $\xi$  ⇒ ip  $\xi'$  = ip  $\xi$ "
" $\xi'$  ∈ is_pkt  $\xi$  ⇒ ip  $\xi'$  = ip  $\xi$ "
" $\xi'$  ∈ is_newpkt  $\xi$  ⇒ ip  $\xi'$  = ip  $\xi$ "
⟨proof⟩

lemma is_msg_inv_sn [simp]:
" $\xi'$  ∈ is_rerr  $\xi$  ⇒ sn  $\xi'$  = sn  $\xi$ "

```

```

" $\xi' \in \text{is\_rrep } \xi \implies \text{sn } \xi' = \text{sn } \xi"$ 
" $\xi' \in \text{is\_rreq } \xi \implies \text{sn } \xi' = \text{sn } \xi"$ 
" $\xi' \in \text{is\_pkt } \xi \implies \text{sn } \xi' = \text{sn } \xi"$ 
" $\xi' \in \text{is\_newpkt } \xi \implies \text{sn } \xi' = \text{sn } \xi"$ 
⟨proof⟩

```

```

lemma is_msg_inv_rt [simp]:
  " $\xi' \in \text{is\_rerr } \xi \implies \text{rt } \xi' = \text{rt } \xi"$ 
  " $\xi' \in \text{is\_rrep } \xi \implies \text{rt } \xi' = \text{rt } \xi"$ 
  " $\xi' \in \text{is\_rreq } \xi \implies \text{rt } \xi' = \text{rt } \xi"$ 
  " $\xi' \in \text{is\_pkt } \xi \implies \text{rt } \xi' = \text{rt } \xi"$ 
  " $\xi' \in \text{is\_newpkt } \xi \implies \text{rt } \xi' = \text{rt } \xi"$ 
⟨proof⟩

```

```

lemma is_msg_inv_rreqs [simp]:
  " $\xi' \in \text{is\_rerr } \xi \implies \text{rreqs } \xi' = \text{rreqs } \xi"$ 
  " $\xi' \in \text{is\_rrep } \xi \implies \text{rreqs } \xi' = \text{rreqs } \xi"$ 
  " $\xi' \in \text{is\_rreq } \xi \implies \text{rreqs } \xi' = \text{rreqs } \xi"$ 
  " $\xi' \in \text{is\_pkt } \xi \implies \text{rreqs } \xi' = \text{rreqs } \xi"$ 
  " $\xi' \in \text{is\_newpkt } \xi \implies \text{rreqs } \xi' = \text{rreqs } \xi"$ 
⟨proof⟩

```

```

lemma is_msg_inv_store [simp]:
  " $\xi' \in \text{is\_rerr } \xi \implies \text{store } \xi' = \text{store } \xi"$ 
  " $\xi' \in \text{is\_rrep } \xi \implies \text{store } \xi' = \text{store } \xi"$ 
  " $\xi' \in \text{is\_rreq } \xi \implies \text{store } \xi' = \text{store } \xi"$ 
  " $\xi' \in \text{is\_pkt } \xi \implies \text{store } \xi' = \text{store } \xi"$ 
  " $\xi' \in \text{is\_newpkt } \xi \implies \text{store } \xi' = \text{store } \xi"$ 
⟨proof⟩

```

```

lemma is_msg_inv_sip [simp]:
  " $\xi' \in \text{is\_pkt } \xi \implies \text{sip } \xi' = \text{sip } \xi"$ 
  " $\xi' \in \text{is\_newpkt } \xi \implies \text{sip } \xi' = \text{sip } \xi"$ 
⟨proof⟩

```

### 0.4.3 The protocol process

```

datatype pseqp =
  PAodv
  | PNewPkt
  | PPkt
  | PRreq
  | PRrep
  | PRerr

fun nat_of_seqp :: "pseqp ⇒ nat"
where
  "nat_of_seqp PAodv = 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 ≡ \_. [clear_locals] call(PAodv)"

```

abbreviation  $\text{PKT}$

where

" $\text{PKT}$  args  $\equiv$

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

abbreviation  $\text{NEWPKT}$

where

" $\text{NEWPKT}$  args  $\equiv$

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

abbreviation  $\text{RREQ}$

where

" $\text{RREQ}$  args  $\equiv$

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

abbreviation  $\text{RREP}$

where

" $\text{RREP}$  args  $\equiv$

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

abbreviation  $\text{RERR}$

where

" $\text{RERR}$  args  $\equiv$

```
[\xi. let (dests, sip) = args \xi in
  (clear_locals \xi) () dests := dests, sip := sip ())
  call(PRerr)"
```

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

where

```
"\Gamma_{AODV} PAodv = labelled PAodv (
  receive(\lambda msg' \xi. \xi () msg := msg' ()).
  ( \langle \text{is\_newpkt} \rangle \text{ NEWPKT}(\lambda \xi. (\text{data } \xi, \text{ ip } \xi))
  \oplus \langle \text{is\_pkt} \rangle \text{ PKT}(\lambda \xi. (\text{data } \xi, \text{ dip } \xi, \text{ oip } \xi))
  \oplus \langle \text{is\_rreq} \rangle
    [\xi. \xi () \text{ rt} := \text{update} (\text{rt } \xi) (\text{sip } \xi) (0, \text{ unk}, \text{ val}, 1, \text{ sip } \xi, \{\}) ()]
    \text{RREQ}(\lambda \xi. (\text{hops } \xi, \text{ rreqid } \xi, \text{ dip } \xi, \text{ dsn } \xi, \text{ dsk } \xi, \text{ oip } \xi, \text{ osn } \xi, \text{ sip } \xi))
  \oplus \langle \text{is\_rrep} \rangle
    [\xi. \xi () \text{ rt} := \text{update} (\text{rt } \xi) (\text{sip } \xi) (0, \text{ unk}, \text{ val}, 1, \text{ sip } \xi, \{\}) ()]
    \text{RREP}(\lambda \xi. (\text{hops } \xi, \text{ dip } \xi, \text{ dsn } \xi, \text{ oip } \xi, \text{ sip } \xi))
  \oplus \langle \text{is\_rerr} \rangle
    [\xi. \xi () \text{ rt} := \text{update} (\text{rt } \xi) (\text{sip } \xi) (0, \text{ unk}, \text{ val}, 1, \text{ sip } \xi, \{\}) ()]
    \text{RERR}(\lambda \xi. (\text{dests } \xi, \text{ sip } \xi))
)
\oplus \langle \lambda \xi. \{ \xi | \text{ dip} := \text{dip } \} | \text{ dip}. \text{ dip} \in \text{qD(store } \xi) \cap \text{vD(rt } \xi) \}
  [\xi. \xi () \text{ data} := \text{hd}(\sigma_{\text{queue}}(\text{store } \xi, \text{ dip } \xi)) ()]
  \text{unicast}(\lambda \xi. \text{the} (\text{nhop} (\text{rt } \xi) (\text{dip } \xi)), \lambda \xi. \text{pkt}(\text{data } \xi, \text{ dip } \xi, \text{ ip } \xi)).
  [\xi. \xi () \text{ store} := \text{the} (\text{drop} (\text{dip } \xi) (\text{store } \xi)) ())
  \text{AODV}()
  \triangleright [\xi. \xi () \text{ dests} := (\lambda \text{rip}. \text{if} (\text{rip} \in \text{vD(rt } \xi) \wedge \text{nhop(rt } \xi) \text{ rip} = \text{nhop(rt } \xi) \text{ (dip } \xi)) \text{ then Some} (\text{inc} (\text{sqn(rt } \xi) \text{ rip})) \text{ else None}) ()]
  [\xi. \xi () \text{ rt} := \text{invalidate} (\text{rt } \xi) (\text{dests } \xi)) ()
  [\xi. \xi () \text{ store} := \text{setRRF} (\text{store } \xi) (\text{dests } \xi)) ()]
```

```

 $\llbracket \xi. \xi \mid pre := \bigcup \{ \text{the } (\text{precs } (rt \xi) rip) \mid rip \in \text{dom } (\text{dests } \xi) \} \rrbracket$ 
 $\llbracket \xi. \xi \mid \text{dests} := (\lambda rip. \text{if } ((\text{dests } \xi) rip \neq \text{None} \wedge \text{the } (\text{precs } (rt \xi) rip) \neq \{\})$ 
 $\quad \text{then } (\text{dests } \xi) rip \text{ else } \text{None}) \rrbracket \rrbracket$ 
 $\text{groupcast}(\lambda \xi. \text{pre } \xi, \lambda \xi. \text{rerr}(\text{dests } \xi, ip \xi)). AODV()$ 
 $\oplus \langle \lambda \xi. \{ \xi \mid dip := dip \}$ 
 $\quad | dip. dip \in qD(store \xi) - vD(rt \xi) \wedge \text{the } (\sigma_{p\text{-flag}}(store \xi, dip)) = req \rangle$ 
 $\llbracket \xi. \xi \mid store := \text{unsetRRF } (store \xi) (dip \xi) \rrbracket$ 
 $\llbracket \xi. \xi \mid sn := inc (sn \xi) \rrbracket$ 
 $\llbracket \xi. \xi \mid rreqid := nrreqid (rreqs \xi) (ip \xi) \rrbracket$ 
 $\llbracket \xi. \xi \mid rreqs := rreqs \xi \cup \{(ip \xi, rreqid \xi)\} \rrbracket$ 
 $\text{broadcast}(\lambda \xi. \text{rreq}(0, rreqid \xi, dip \xi, sqn (rt \xi) (dip \xi), sqnf (rt \xi) (dip \xi),$ 
 $\quad ip \xi, sn \xi, ip \xi)). AODV()")$ 

| " $\Gamma_{AODV} PNewPkt = \text{labelled } PNewPkt$  (
 $\langle \xi. dip \xi = ip \xi \rangle$ 
 $\quad \text{deliver}(\lambda \xi. \text{data } \xi). AODV()$ 
 $\oplus \langle \xi. dip \xi \neq ip \xi \rangle$ 
 $\quad \llbracket \xi. \xi \mid store := \text{add } (\text{data } \xi) (dip \xi) (store \xi) \rrbracket$ 
 $\quad AODV()")$ 

| " $\Gamma_{AODV} PPkt = \text{labelled } PPkt$  (
 $\langle \xi. dip \xi = ip \xi \rangle$ 
 $\quad \text{deliver}(\lambda \xi. \text{data } \xi). AODV()$ 
 $\oplus \langle \xi. dip \xi \neq ip \xi \rangle$ 
 $\quad ($ 
 $\quad \langle \xi. dip \xi \in vD (rt \xi) \rangle$ 
 $\quad \text{unicast}(\lambda \xi. \text{the } (\text{nhop } (rt \xi) (dip \xi)), \lambda \xi. \text{pkt}(\text{data } \xi, dip \xi, oip \xi)). AODV()$ 
 $\quad \triangleright$ 
 $\quad \llbracket \xi. \xi \mid \text{dests} := (\lambda rip. \text{if } (rip \in vD (rt \xi) \wedge \text{nhop } (rt \xi) rip = \text{nhop } (rt \xi) (dip \xi))$ 
 $\quad \quad \text{then Some } (\text{inc } (\text{sqn } (rt \xi) rip)) \text{ else } \text{None}) \rrbracket$ 
 $\quad \llbracket \xi. \xi \mid rt := \text{invalidate } (rt \xi) (\text{dests } \xi) \rrbracket$ 
 $\quad \llbracket \xi. \xi \mid store := \text{setRRF } (store \xi) (\text{dests } \xi) \rrbracket$ 
 $\quad \llbracket \xi. \xi \mid pre := \bigcup \{ \text{the } (\text{precs } (rt \xi) rip) \mid rip. rip \in \text{dom } (\text{dests } \xi) \} \rrbracket$ 
 $\quad \llbracket \xi. \xi \mid \text{dests} := (\lambda rip. \text{if } ((\text{dests } \xi) rip \neq \text{None} \wedge \text{the } (\text{precs } (rt \xi) rip) \neq \{\})$ 
 $\quad \quad \text{then } (\text{dests } \xi) rip \text{ else } \text{None}) \rrbracket$ 
 $\quad \text{groupcast}(\lambda \xi. \text{pre } \xi, \lambda \xi. \text{rerr}(\text{dests } \xi, ip \xi)). AODV()$ 
 $\oplus \langle \xi. dip \xi \notin vD (rt \xi) \rangle$ 
 $\quad ($ 
 $\quad \langle \xi. dip \xi \in iD (rt \xi) \rangle$ 
 $\quad \text{groupcast}(\lambda \xi. \text{the } (\text{precs } (rt \xi) (dip \xi)),$ 
 $\quad \quad \lambda \xi. \text{rerr}([dip \xi \mapsto \text{sqn } (rt \xi) (dip \xi)], ip \xi)). AODV()$ 
 $\oplus \langle \xi. dip \xi \notin iD (rt \xi) \rangle$ 
 $\quad AODV()$ 
 $\quad )$ 
 $\quad ))$ 
 $\quad )$ 
 $\quad ))"$ 

| " $\Gamma_{AODV} PRreq = \text{labelled } PRreq$  (
 $\langle \xi. (oip \xi, rreqid \xi) \in rreqs \xi \rangle$ 
 $\quad AODV()$ 
 $\oplus \langle \xi. (oip \xi, rreqid \xi) \notin rreqs \xi \rangle$ 
 $\quad \llbracket \xi. \xi \mid rt := \text{update } (rt \xi) (oip \xi) (osn \xi, kno, val, hops \xi + 1, sip \xi, \{\}) \rrbracket$ 
 $\quad \llbracket \xi. \xi \mid rreqs := rreqs \xi \cup \{(oip \xi, rreqid \xi)\} \rrbracket$ 
 $\quad ($ 
 $\quad \langle \xi. dip \xi = ip \xi \rangle$ 
 $\quad \llbracket \xi. \xi \mid sn := \max (sn \xi) (dsn \xi) \rrbracket$ 
 $\quad \text{unicast}(\lambda \xi. \text{the } (\text{nhop } (rt \xi) (oip \xi)), \lambda \xi. \text{rrep}(0, dip \xi, sn \xi, oip \xi, ip \xi)). AODV()$ 
 $\quad \triangleright$ 
 $\quad \llbracket \xi. \xi \mid \text{dests} := (\lambda rip. \text{if } (rip \in vD (rt \xi) \wedge \text{nhop } (rt \xi) rip = \text{nhop } (rt \xi) (oip \xi))$ 
 $\quad \quad \text{then Some } (\text{inc } (\text{sqn } (rt \xi) rip)) \text{ else } \text{None}) \rrbracket$ 
 $\quad \llbracket \xi. \xi \mid rt := \text{invalidate } (rt \xi) (\text{dests } \xi) \rrbracket$ 
 $\quad \llbracket \xi. \xi \mid store := \text{setRRF } (store \xi) (\text{dests } \xi) \rrbracket$ 
 $\quad \llbracket \xi. \xi \mid pre := \bigcup \{ \text{the } (\text{precs } (rt \xi) rip) \mid rip. rip \in \text{dom } (\text{dests } \xi) \} \rrbracket$ 
 $\quad \llbracket \xi. \xi \mid \text{dests} := (\lambda rip. \text{if } ((\text{dests } \xi) rip \neq \text{None} \wedge \text{the } (\text{precs } (rt \xi) rip) \neq \{\})$ 
 $\quad \quad \text{then } (\text{dests } \xi) rip \text{ else } \text{None}) \rrbracket$ 

```

```

groupcast( $\lambda \xi. \text{pre } \xi, \lambda \xi. \text{rerr}(\text{dests } \xi, \text{ip } \xi))$ .AODV()
 $\oplus \langle \xi. \text{dip } \xi \neq \text{ip } \xi \rangle$ 
(
   $\langle \xi. \text{dip } \xi \in vD(\text{rt } \xi) \wedge \text{dsn } \xi \leq \text{sqn }(\text{rt } \xi) (\text{dip } \xi) \wedge \text{sqnf }(\text{rt } \xi) (\text{dip } \xi) = \text{kno} \rangle$ 
   $\llbracket \xi. \xi \mid \text{rt} := \text{the}(\text{addpreRT}(\text{rt } \xi) (\text{dip } \xi) \backslash \text{sip } \xi) \rrbracket$ 
   $\llbracket \xi. \xi \mid \text{rt} := \text{the}(\text{addpreRT}(\text{rt } \xi) (\text{oip } \xi) \{ \text{the}(\text{nhop}(\text{rt } \xi) (\text{dip } \xi)) \}) \rrbracket$ 
  unicast( $\lambda \xi. \text{the}(\text{nhop}(\text{rt } \xi) (\text{oip } \xi)), \lambda \xi. \text{rrep}(\text{the}(\text{dhops}(\text{rt } \xi) (\text{dip } \xi)), \text{dip } \xi,$ 
           $\text{sqn }(\text{rt } \xi) (\text{dip } \xi), \text{oip } \xi, \text{ip } \xi)$ ).
  AODV()
)
 $\triangleright$ 
   $\langle \xi. \xi \mid \text{dests} := (\lambda \text{rip}. \text{if } (\text{rip} \in vD(\text{rt } \xi) \wedge \text{nhop}(\text{rt } \xi) \text{rip} = \text{nhop}(\text{rt } \xi) (\text{oip } \xi))$ 
     $\text{then Some}(\text{inc}(\text{sqn }(\text{rt } \xi) \text{rip})) \text{else None}) \rrbracket$ 
   $\llbracket \xi. \xi \mid \text{rt} := \text{invalidate}(\text{rt } \xi) (\text{dests } \xi) \rrbracket$ 
   $\llbracket \xi. \xi \mid \text{store} := \text{setRRF}(\text{store } \xi) (\text{dests } \xi) \rrbracket$ 
   $\llbracket \xi. \xi \mid \text{pre} := \bigcup \{ \text{the}(\text{precs}(\text{rt } \xi) \text{rip}) \mid \text{rip} \in \text{dom}(\text{dests } \xi) \} \rrbracket$ 
   $\llbracket \xi. \xi \mid \text{dests} := (\lambda \text{rip}. \text{if } ((\text{dests } \xi) \text{rip} \neq \text{None} \wedge \text{the}(\text{precs}(\text{rt } \xi) \text{rip}) \neq \{\})$ 
     $\text{then}(\text{dests } \xi) \text{rip} \text{else None}) \rrbracket$ 
  groupcast( $\lambda \xi. \text{pre } \xi, \lambda \xi. \text{rerr}(\text{dests } \xi, \text{ip } \xi))$ .AODV()
 $\oplus \langle \xi. \text{dip } \xi \notin vD(\text{rt } \xi) \vee \text{sqn }(\text{rt } \xi) (\text{dip } \xi) < \text{dsn } \xi \vee \text{sqnf }(\text{rt } \xi) (\text{dip } \xi) = \text{unk} \rangle$ 
  broadcast( $\lambda \xi. \text{rreq}(\text{hops } \xi + 1, \text{rreqid } \xi, \text{dip } \xi, \max(\text{sqn }(\text{rt } \xi) (\text{dip } \xi)) (\text{dsn } \xi),$ 
             $\text{dsk } \xi, \text{oip } \xi, \text{osn } \xi, \text{ip } \xi)$ ).
  AODV()
)
)
))

| " $\Gamma_{AODV}$  PRrep = labelled PRrep (
   $\langle \xi. \text{rt } \xi \neq \text{update}(\text{rt } \xi) (\text{dip } \xi) (\text{dsn } \xi, \text{kno}, \text{val}, \text{hops } \xi + 1, \text{sip } \xi, \{\}) \rangle$ 
(
   $\llbracket \xi. \xi \mid \text{rt} := \text{update}(\text{rt } \xi) (\text{dip } \xi) (\text{dsn } \xi, \text{kno}, \text{val}, \text{hops } \xi + 1, \text{sip } \xi, \{\}) \rrbracket$ 
   $\langle \xi. \text{oip } \xi = \text{ip } \xi \rangle$ 
  AODV()
   $\oplus \langle \xi. \text{oip } \xi \neq \text{ip } \xi \rangle$ 
(
     $\langle \xi. \text{oip } \xi \in vD(\text{rt } \xi) \rangle$ 
     $\llbracket \xi. \xi \mid \text{rt} := \text{the}(\text{addpreRT}(\text{rt } \xi) (\text{dip } \xi) \{ \text{the}(\text{nhop}(\text{rt } \xi) (\text{oip } \xi)) \}) \rrbracket$ 
     $\llbracket \xi. \xi \mid \text{rt} := \text{the}(\text{addpreRT}(\text{rt } \xi) (\text{the}(\text{nhop}(\text{rt } \xi) (\text{dip } \xi)))$ 
       $\{ \text{the}(\text{nhop}(\text{rt } \xi) (\text{oip } \xi)) \}) \rrbracket$ 
    unicast( $\lambda \xi. \text{the}(\text{nhop}(\text{rt } \xi) (\text{oip } \xi)), \lambda \xi. \text{rrep}(\text{hops } \xi + 1, \text{dip } \xi, \text{dsn } \xi, \text{oip } \xi, \text{ip } \xi)$ ).
    AODV()
)
 $\triangleright$ 
   $\llbracket \xi. \xi \mid \text{dests} := (\lambda \text{rip}. \text{if } (\text{rip} \in vD(\text{rt } \xi) \wedge \text{nhop}(\text{rt } \xi) \text{rip} = \text{nhop}(\text{rt } \xi) (\text{oip } \xi))$ 
     $\text{then Some}(\text{inc}(\text{sqn }(\text{rt } \xi) \text{rip})) \text{else None}) \rrbracket$ 
   $\llbracket \xi. \xi \mid \text{rt} := \text{invalidate}(\text{rt } \xi) (\text{dests } \xi) \rrbracket$ 
   $\llbracket \xi. \xi \mid \text{store} := \text{setRRF}(\text{store } \xi) (\text{dests } \xi) \rrbracket$ 
   $\llbracket \xi. \xi \mid \text{pre} := \bigcup \{ \text{the}(\text{precs}(\text{rt } \xi) \text{rip}) \mid \text{rip} \in \text{dom}(\text{dests } \xi) \} \rrbracket$ 
   $\llbracket \xi. \xi \mid \text{dests} := (\lambda \text{rip}. \text{if } ((\text{dests } \xi) \text{rip} \neq \text{None} \wedge \text{the}(\text{precs}(\text{rt } \xi) \text{rip}) \neq \{\})$ 
     $\text{then}(\text{dests } \xi) \text{rip} \text{else None}) \rrbracket$ 
  groupcast( $\lambda \xi. \text{pre } \xi, \lambda \xi. \text{rerr}(\text{dests } \xi, \text{ip } \xi))$ .AODV()
   $\oplus \langle \xi. \text{oip } \xi \notin vD(\text{rt } \xi) \rangle$ 
  AODV()
)
)
 $\oplus \langle \xi. \text{rt } \xi = \text{update}(\text{rt } \xi) (\text{dip } \xi) (\text{dsn } \xi, \text{kno}, \text{val}, \text{hops } \xi + 1, \text{sip } \xi, \{\}) \rangle$ 
  AODV()
)

| " $\Gamma_{AODV}$  PRerr = labelled PRerr (
   $\llbracket \xi. \xi \mid \text{dests} := (\lambda \text{rip}. \text{case}(\text{dests } \xi) \text{rip} \text{of} \text{None} \Rightarrow \text{None}$ 
     $\mid \text{Some} \text{rsn} \Rightarrow \text{if} \text{rip} \in vD(\text{rt } \xi) \wedge \text{the}(\text{nhop}(\text{rt } \xi) \text{rip}) = \text{sip } \xi$ 
       $\wedge \text{sqn }(\text{rt } \xi) \text{rip} < \text{rsn} \text{then Some} \text{rsn} \text{else None}) \rrbracket$ 
   $\llbracket \xi. \xi \mid \text{rt} := \text{invalidate}(\text{rt } \xi) (\text{dests } \xi) \rrbracket$ 
   $\llbracket \xi. \xi \mid \text{store} := \text{setRRF}(\text{store } \xi) (\text{dests } \xi) \rrbracket$ 

```

```

 $\llbracket \xi. \xi \mid pre := \bigcup \{ \text{the } (\text{precs } (rt \xi) rip) \mid rip \in \text{dom } (\text{dests } \xi) \} \rrbracket$ 
 $\llbracket \xi. \xi \mid \text{dests} := (\lambda rip. \text{if } ((\text{dests } \xi) rip \neq \text{None} \wedge \text{the } (\text{precs } (rt \xi) rip) \neq \{\})$ 
 $\quad \text{then } (\text{dests } \xi) rip \text{ else None}) \rrbracket$ 
 $\text{groupcast}(\lambda \xi. \text{pre } \xi, \lambda \xi. \text{rerr}(\text{dests } \xi, ip \xi)). \text{AODV}()$ "
```

declare  $\Gamma_{AODV}.\text{simp}$  [simp del, code del]  
lemmas  $\Gamma_{AODV\_simp}$  [simp, code] =  $\Gamma_{AODV}.\text{simp}$  [simplified]

fun  $\Gamma_{AODV\_skeleton}$   
where

- "  $\Gamma_{AODV\_skeleton} PAodv = \text{seqp\_skeleton } (\Gamma_{AODV} PAodv)$ "
- "  $\Gamma_{AODV\_skeleton} PNewPkt = \text{seqp\_skeleton } (\Gamma_{AODV} PNewPkt)$ "
- "  $\Gamma_{AODV\_skeleton} PPkt = \text{seqp\_skeleton } (\Gamma_{AODV} PPkt)$ "
- "  $\Gamma_{AODV\_skeleton} PRreq = \text{seqp\_skeleton } (\Gamma_{AODV} PRreq)$ "
- "  $\Gamma_{AODV\_skeleton} PRrep = \text{seqp\_skeleton } (\Gamma_{AODV} PRrep)$ "
- "  $\Gamma_{AODV\_skeleton} PRerr = \text{seqp\_skeleton } (\Gamma_{AODV} PRerr)$ "

lemma  $\Gamma_{AODV\_skeleton\_wf}$  [simp]:  
"wellformed  $\Gamma_{AODV\_skeleton}$ "  
*(proof)*

declare  $\Gamma_{AODV\_skeleton}.\text{simp}$  [simp del, code del]  
lemmas  $\Gamma_{AODV\_skeleton\_simp}$  [simp, code]  
=  $\Gamma_{AODV\_skeleton}.\text{simp}$  [simplified  $\Gamma_{AODV}.\text{simp}$  seqp\_skeleton.simp]

lemma aodv\_proc\_cases [dest]:  
fixes  $p pn$   
shows " $p \in \text{ctermstl } (\Gamma_{AODV} pn) \Rightarrow$   
 $(p \in \text{ctermstl } (\Gamma_{AODV} PAodv) \vee$   
 $p \in \text{ctermstl } (\Gamma_{AODV} PNewPkt) \vee$   
 $p \in \text{ctermstl } (\Gamma_{AODV} PPkt) \vee$   
 $p \in \text{ctermstl } (\Gamma_{AODV} PRreq) \vee$   
 $p \in \text{ctermstl } (\Gamma_{AODV} PRrep) \vee$   
 $p \in \text{ctermstl } (\Gamma_{AODV} PRerr))$ "  
*(proof)*

definition  $\sigma_{AODV} :: "ip \Rightarrow (\text{state} \times (\text{state}, \text{msg}, \text{pseqp}, \text{pseqp label}) \text{ seqp}) \text{ set}$ "  
where " $\sigma_{AODV} i \equiv \{(aodv\_init i, \Gamma_{AODV} PAodv)\}"$

abbreviation paodv  
:: "ip  $\Rightarrow (\text{state} \times (\text{state}, \text{msg}, \text{pseqp}, \text{pseqp label}) \text{ seqp}, \text{msg seq\_action}) \text{ automaton}$ "  
where  
"paodv i  $\equiv \mid \text{init} = \sigma_{AODV} i, \text{trans} = \text{seqp\_sos } \Gamma_{AODV} \mid$ "

lemma aodv\_trans: "trans (paodv i) = seqp\_sos  $\Gamma_{AODV}$ "  
*(proof)*

lemma aodv\_control\_within [simp]: "control\_within  $\Gamma_{AODV} (\text{init } (\text{paodv } i))$ "  
*(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]: " $\exists l. l \in \text{labels } \Gamma_{AODV} p$ "  
*(proof)*

lemma aodv\_ex\_labelE [elim]:  
assumes " $\forall l \in \text{labels } \Gamma_{AODV} p. P l p$ "  
and " $\exists p l. P l p \Rightarrow Q$ "  
shows "Q"  
*(proof)*

```

lemma aodv_simple_labels [simp]: "simple_labels ΓAODV"  

  ⟨proof⟩  
  

lemma σAODV_labels [simp]: "(ξ, p) ∈ σAODV i ⟹ labels ΓAODV p = {PAodv-:0}"  

  ⟨proof⟩  
  

lemma aodv_init_kD_empty [simp]:  

  "(ξ, p) ∈ σAODV i ⟹ kD (rt ξ) = {}"  

  ⟨proof⟩  
  

lemma aodv_init_sip_not_ip [simp]: "¬(sip (aodv_init i) = i)" ⟨proof⟩  
  

lemma aodv_init_sip_not_ip' [simp]:  

  assumes "(ξ, p) ∈ σAODV i"  

  shows "sip ξ ≠ ip ξ"  

  ⟨proof⟩  
  

lemma aodv_init_sip_not_i [simp]:  

  assumes "(ξ, p) ∈ σAODV 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 [cterms_env]  

  aodv_proc_cases [ctermsl_cases]  

  seq_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,  

    cterms_intros]  

  seq_step_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,  

    cterms_intros]  
  

end

```

## 0.5 Invariant assumptions and properties

```
theory Aodv_Predicates
imports 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 ⇒ ip"  

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 rreqid dip dsn dsk oip osn sip.  

    msg_sender (Rreq hops rreqid 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)"

```

```

"\ $\wedge d \text{ dip } sip.$  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]:
"\ $\wedge \text{hops rreqid dip dsn dsk oip osn sip}.$ 
  msg_zhops (Rreq hops rreqid dip dsn dsk oip osn sip) = (hops = 0 → oip = sip)"
"\ $\wedge \text{hops dip dsn oip sip}.$  msg_zhops (Rrep hops dip dsn oip sip) = (hops = 0 → dip = sip)"
"\ $\wedge \text{dests sip}.$  msg_zhops (Rerr dests sip) = True"
"\ $\wedge d \text{ dip}.$  msg_zhops (Newpkt d dip) = True"
"\ $\wedge d \text{ dip sip}.$  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]:
"\ $\wedge \text{hops rreqid dip dsn dsk oip osn sip}.$ 
  rreq_rrep_sn (Rreq hops rreqid dip dsn dsk oip osn sip) = (osn ≥ 1)"
"\ $\wedge \text{hops dip dsn oip sip}.$  rreq_rrep_sn (Rrep hops dip dsn oip sip) = (dsn ≥ 1)"
"\ $\wedge \text{dests sip}.$  rreq_rrep_sn (Rerr dests sip) = True"
"\ $\wedge d \text{ dip}.$  rreq_rrep_sn (Newpkt d dip) = True"
"\ $\wedge d \text{ dip sip}.$  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]:
"\ $\wedge \text{hops rreqid dip dsn dsk oip osn sip}.$ 
  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)))"
"\ $\wedge \text{hops dip dsn oip sip}.$  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)"
"\ $\wedge \text{dests sip}.$  rreq_rrep_fresh crt (Rerr dests sip) = True"
"\ $\wedge d \text{ dip}.$  rreq_rrep_fresh crt (Newpkt d dip) = True"
"\ $\wedge d \text{ dip sip}.$  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 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 ⇒ aodv_init i | Some s ⇒ s, snd σ)"

lemma not_in_net ips fst init_missing [simp]:

```

```

assumes "i ∈ net_ip"
shows "fst (initmissing (netgmap fst σ)) i = aodv_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

```

## 0.6 Quality relations between routes

```
theory Fresher
imports Aodv_Data
begin
```

### 0.6.1 Net sequence numbers

On individual routes

## definition

**where**      $"nsqn_r\ r \equiv if\ \pi_4(r) = val\ \vee\ \pi_2(r) = 0\ then\ \pi_2(r)\ else\ (\pi_2(r) - 1)"$

```

lemma nsqn_r_def' :
  "nsqn_r r = (if π₄(r) = inv then π₂(r) - 1 else π₂(r))"

⟨proof⟩

```

```
lemma nsqn_r_zero [simp]:
  "⟨proof⟩
  ∀ dsn dsk flag hops nhip pre. nsqn_r (0, dsk, flag, hops, nhip, pre) = 0"
```

```

lemma nsqn_r_val [simp]:
  " $\wedge_{dsn\ dsk\ hops\ nhip\ pre.} nsqn_r(dsn, dsk, val, hops, nhip, pre) = dsn$ "
  ⟨proof⟩

lemma nsqn_r_inv [simp]:
  " $\wedge_{dsn\ dsk\ hops\ nhip\ pre.} nsqn_r(dsn, dsk, inv, hops, nhip, pre) = dsn - 1$ "
  ⟨proof⟩

lemma nsqn_r_lte_dsn [simp]:
  " $\wedge_{dsn\ dsk\ flag\ hops\ nhip\ pre.} nsqn_r(dsn, dsk, flag, hops, nhip, pre) \leq dsn$ "
  ⟨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:
  " $\wedge_{rt\ dip.} nsqn\ rt\ dip = (\text{if } flag\ rt\ dip = \text{Some val} \vee sqn\ rt\ dip = 0$ 
    $\text{then } sqn\ rt\ dip \text{ else } sqn\ rt\ dip - 1)$ "
  ⟨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(σ_{route}(rt, dip)))"
  ⟨proof⟩

lemma nsqn_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]:
  " $\wedge_{rt\ dip\ npre\ dip'.} dip \in kD(rt) \implies$ 
    $nsqn_r(\text{the}(\text{the}(\text{addpreRT}\ rt\ dip\ npre)\ dip')) = nsqn_r(\text{the}(rt\ dip'))$ "
  ⟨proof⟩

lemma sqn_nsqn:
  " $\wedge_{rt\ dip.} sqn\ rt\ dip - 1 \leq nsqn\ rt\ dip$ "
  ⟨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' ≡ ((nsqn_r r < nsqn_r r') ∨ (nsqn_r r = nsqn_r r' ∧ π5(r) ≥ π5(r')))"

lemma fresherI1 [intro]:
  assumes "nsqn_r r < nsqn_r r'"
  shows "r ≤ r"
  ⟨proof⟩

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

lemma fresherI [intro]:
  assumes "(nsqn_r r < nsqn_r r') ∨ (nsqn_r r = nsqn_r 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"
⟨proof⟩
```

```

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" (<_/_ ≈ _ _)> [51, 999, 51] 50
where
"rt1 ≈i rt2 ≡ rt_fresh_as i 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 ≈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_nsqr:
  assumes "dip ∈ kD(rt1)"
    and "dip ∈ kD(rt2)"
    and "rt1 ≈dip rt2"
  shows "nsqr (the (rt2 dip)) = nsqr (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 nsqn_r_invalidate [simp]:
  assumes "dip ∈ kD(rt)"
    and "dip ∈ dom(dests)"
  shows "nsqn_r (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]

0.6.4 Strictly comparing routing tables

definition rt_strictly_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_strictly_fresher ≡ λ dip 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 =
   (nsqn_r (the (rt1 i)) < nsqn_r (the (rt2 i))
    ∨ (nsqn_r (the (rt1 i)) = nsqn_r (the (rt2 i)) ∧ π_5(the (rt1 i)) > π_5(the (rt2 i))))"
  ⟨proof⟩

lemma rt_strictly_fresher_fresherD [dest]:
  assumes "rt1 ⊑_{dip} rt2"
  shows "the (rt1 dip) ⊑ the (rt2 dip)"
  ⟨proof⟩

lemma rt_strictly_fresher_not_fresh_asD [dest]:
  assumes "rt1 ⊑_{dip} rt2"
  shows "¬ rt1 ≈_{dip} rt2"
  ⟨proof⟩

lemma rt_strictly_fresher_trans [elim, trans]:
  assumes "rt1 ⊑_{dip} rt2"
    and "rt2 ⊑_{dip} rt3"
  shows "rt1 ⊑_{dip} rt3"
  ⟨proof⟩

lemma rt_strictly_fresher_irrefl [simp]: "¬ (rt ⊑_{dip} rt)"
  ⟨proof⟩

lemma rt_fresher_trans_rt_strictly_fresher [elim, trans]:
  assumes "rt1 ⊑_{dip} rt2"
    and "rt2 ⊑_{dip} rt3"
  shows "rt1 ⊑_{dip} rt3"
  ⟨proof⟩

lemma rt_fresher_trans_rt_strictly_fresher' [elim, trans]:
  assumes "rt1 ⊑_{dip} rt2"
    and "rt2 ⊑_{dip} rt3"
  shows "rt1 ⊑_{dip} rt3"
  ⟨proof⟩

lemma rt_fresher_imp_nsqn_le:
  assumes "rt1 ⊑_{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  $\Gamma_{AODV}(\lambda(\xi, 1). 1 \in (\{PAodv-:7\} \cup \{PAodv-:5\} \cup \{PRrep-:0..PRrep-:1\}$   

 $\cup \{PRreq-:0..PRreq-:3\}) \longrightarrow sip \xi \in kD(rt \xi))"$ 
```

*(proof)*

```
lemma rrep_1_update_changes:
```

```
"paodv i  $\models$  onl  $\Gamma_{AODV}(\lambda(\xi, 1). (1 = PRrep-:1 \longrightarrow$   

 $rt \xi \neq update(rt \xi) (dip \xi) (dsn \xi, kno, val, hops \xi + 1, sip \xi, \{}))")$ 
```

*(proof)*

```
lemma addpreRT_partly_welldefined:
```

```
"paodv i  $\models$   

onl  $\Gamma_{AODV}(\lambda(\xi, 1). (1 \in \{PRreq-:16..PRreq-:18\} \cup \{PRrep-:2..PRrep-:6\} \longrightarrow dip \xi \in kD(rt \xi)) \wedge$   

 $(1 \in \{PRreq-:3..PRreq-:17\} \longrightarrow oip \xi \in kD(rt \xi)))")$ 
```

*(proof)*

Proposition 7.38

```
lemma includes_nhip:
```

```
"paodv i  $\models$  onl  $\Gamma_{AODV}(\lambda(\xi, 1). \forall dip \in kD(rt \xi). \text{the}(nhop(rt \xi) dip) \in kD(rt \xi))")$ 
```

*(proof)*

Proposition 7.22: needed in Proposition 7.4

```
lemma addpreRT_welldefined:
```

```
"paodv i  $\models$  onl  $\Gamma_{AODV}(\lambda(\xi, 1). (1 \in \{PRreq-:16..PRreq-:18\} \longrightarrow dip \xi \in kD(rt \xi)) \wedge$   

 $(1 = PRreq-:17 \longrightarrow oip \xi \in kD(rt \xi)) \wedge$   

 $(1 = PRrep-:5 \longrightarrow dip \xi \in kD(rt \xi)) \wedge$   

 $(1 = PRrep-:6 \longrightarrow (\text{the}(nhop(rt \xi) (dip \xi))) \in kD(rt \xi)))")$ 
```

*(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 ≡ onl ΓAODV (λ(ξ, 1). 1 ∈ {PAodv-:15..PAodv-:19}
    ∪ {PPkt-:7..PPkt-:11}
    ∪ {PRreq-:9..PRreq-:13}
    ∪ {PRreq-:21..PRreq-:25}
    ∪ {PRrep-:10..PRrep-:14}
    ∪ {PRerr-:1..PRerr-:5})
    → (forall ip ∈ dom(dests ξ). ip ∈ kD(rt ξ) ∧ sqn(rt ξ) ip ≤ the(dests ξ ip)))"
  ⟨proof⟩

```

Proposition 7.6

```

lemma sqns_increase:
  "paodv i ≡A onll ΓAODV (λ((ξ, _), _, (ξ', _)). ∀ ip. sqn(rt ξ) ip ≤ sqn(rt ξ') ip)"
  ⟨proof⟩

```

Proposition 7.7

```

lemma ip_constant:
  "paodv i ≡ onl ΓAODV (λ(ξ, _). ip ξ = i)"
  ⟨proof⟩

```

Proposition 7.8

```

lemma sender_ip_valid':
  "paodv i ≡A onll ΓAODV (λ((ξ, _), a, _). anycast(λm. not_Pkt m → msg_sender m = ip ξ) a)"
  ⟨proof⟩

```

```

lemma sender_ip_valid:
  "paodv i ≡A onll ΓAODV (λ((ξ, _), a, _). anycast(λm. not_Pkt m → msg_sender m = i) a)"
  ⟨proof⟩

```

```

lemma received_msg_inv:
  "paodv i ≡ (recvmsg P →) onl ΓAODV (λ(ξ, 1). 1 ∈ {PAodv-:1} → P(msg ξ))"
  ⟨proof⟩

```

Proposition 7.9

```

lemma sip_not_ip':
  "paodv i ≡ (recvmsg (λm. not_Pkt m → msg_sender m ≠ i) →) onl ΓAODV (λ(ξ, _). sip ξ ≠ ip ξ)"
  ⟨proof⟩

```

```

lemma sip_not_ip:
  "paodv i ≡ (recvmsg (λm. not_Pkt m → msg_sender m ≠ i) →) onl ΓAODV (λ(ξ, _). sip ξ ≠ 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 ≡ onl ΓAODV (λ(ξ, _). ∀ ip ∈ kD(rt ξ). the(dhops(rt ξ) ip) ≥ 1)"
  ⟨proof⟩

```

```

lemma rreq_dip_in_vD_dip_eq_ip:
  "paodv i ≡ onl ΓAODV (λ(ξ, 1). (1 ∈ {PRreq-:16..PRreq-:18} → dip ξ ∈ vD(rt ξ))
    ∧ (1 ∈ {PRreq-:5, PRreq-:6} → dip ξ = ip ξ)
    ∧ (1 ∈ {PRreq-:15..PRreq-:18} → dip ξ ≠ ip ξ))"
  ⟨proof⟩

```

Proposition 7.11

```

lemma anycast_msg_zhops:
  "¬ ∃ rreqid dip dsn dsk oip osn sip.
    paodv i ≡A onll ΓAODV (λ(_, a, _). anycast msg_zhops a)"
  ⟨proof⟩

```

```

lemma hop_count_zero_oip_dip_sip:

```

```

"paodv i ⊨ (recvmsg msg_zhops → onl ΓAODV (λ(ξ, 1).
    (1 ∈ {PAodv-:4..PAodv-:5} ∪ {PRreq-:n/n. True} →
        (hops ξ = 0 → oip ξ = sip ξ))
    ∧
    ((1 ∈ {PAodv-:6..PAodv-:7} ∪ {PRrep-:n/n. True} →
        (hops ξ = 0 → dip ξ = sip ξ))))"
⟨proof⟩

lemma osn_rreq:
"paodv i ⊨ (recvmsg rreq_rrep_sn → onl ΓAODV (λ(ξ, 1).
    1 ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n/n. True} → 1 ≤ osn ξ))"
⟨proof⟩

lemma osn_rreq':
"paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) → onl ΓAODV (λ(ξ, 1).
    1 ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n/n. True} → 1 ≤ osn ξ))"
⟨proof⟩

lemma dsn_rrep:
"paodv i ⊨ (recvmsg rreq_rrep_sn → onl ΓAODV (λ(ξ, 1).
    1 ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n/n. True} → 1 ≤ dsn ξ))"
⟨proof⟩

lemma dsn_rrep':
"paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) → onl ΓAODV (λ(ξ, 1).
    1 ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n/n. True} → 1 ≤ dsn ξ))"
⟨proof⟩

lemma hop_count_zero_oip_dip_sip':
"paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) → onl ΓAODV (λ(ξ, 1).
    (1 ∈ {PAodv-:4..PAodv-:5} ∪ {PRreq-:n/n. True} →
        (hops ξ = 0 → oip ξ = sip ξ))
    ∧
    ((1 ∈ {PAodv-:6..PAodv-:7} ∪ {PRrep-:n/n. True} →
        (hops ξ = 0 → dip ξ = sip ξ))))"
⟨proof⟩

```

Proposition 7.12

```

lemma zero_seq_unk_hops_one':
"paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) → onl ΓAODV (λ(ξ, _).
    ∀ dip ∈ kD(rt ξ). (sqn(rt ξ) dip = 0 → sqnf(rt ξ) dip = unk)
    ∧ (sqnf(rt ξ) dip = unk → the(dhops(rt ξ) dip) = 1)
    ∧ (the(dhops(rt ξ) dip) = 1 → the(nhop(rt ξ) dip) = dip)))"
⟨proof⟩

lemma zero_seq_unk_hops_one:
"paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) → onl ΓAODV (λ(ξ, _).
    ∀ dip ∈ kD(rt ξ). (sqn(rt ξ) dip = 0 → (sqnf(rt ξ) dip = unk
    ∧ the(dhops(rt ξ) dip) = 1
    ∧ the(nhop(rt ξ) dip) = dip)))"
⟨proof⟩

```

```

lemma kD_unk_or_atleast_one:
"paodv i ⊨ (recvmsg rreq_rrep_sn → onl ΓAODV (λ(ξ, 1).
    ∀ dip ∈ kD(rt ξ). π3(the(rt ξ dip)) = unk ∨ 1 ≤ π2(the(rt ξ dip))))"
⟨proof⟩

```

Proposition 7.13

```

lemma rreq_rrep_sn_any_step_invariant:
"paodv i ⊨A (recvmsg rreq_rrep_sn → onll ΓAODV (λ( _, a, _). anycast rreq_rrep_sn a))"
⟨proof⟩

```

Proposition 7.14

```

lemma rreq_rrep_fresh_any_step_invariant:

```

```
"paodv i ⊨_A onll ΓAODV (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a)"  

⟨proof⟩
```

Proposition 7.15

```
lemma rerr_invalid_any_step_invariant:
```

```
"paodv i ⊨_A onll ΓAODV (λ((ξ, _), a, _). anycast (rerr_invalid (rt ξ)) 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:
```

```
"paodv i ⊨  

  onl ΓAODV (λ(ξ, 1). (1 ∈ {PAodv-:15, PPkt-:7, PRreq-:9, PRreq-:21, PRrep-:10}  

    → ( ∀ ip ∈ dom(dests ξ). ip ∈ vD(rt ξ) ∧ the (dests ξ ip) = inc (sqn (rt ξ) ip)))  

  ∧ (1 = PRerr-:1  

    → ( ∀ ip ∈ dom(dests ξ). ip ∈ vD(rt ξ) ∧ the (dests ξ ip) > sqn (rt ξ) ip)))"  

⟨proof⟩
```

Proposition 7.27

```
lemma route_tables_fresher:
```

```
"paodv i ⊨_A (recvmsg rreq_rrep_sn → onll ΓAODV (λ((ξ, _), _, (ξ', _)).  

  ∀ dip ∈ kD(rt ξ). rt ξ ⊑dip rt ξ'))"  

⟨proof⟩
```

end

## 0.8 The quality increases predicate

```
theory Quality_Increases
imports Aodv_Predicates Fresher
begin
```

```
definition quality_increases :: "state ⇒ state ⇒ bool"
where "quality_increases ξ ξ' ≡ ( ∀ dip ∈ kD(rt ξ). dip ∈ kD(rt ξ') ∧ rt ξ ⊑dip rt ξ')
  ∧ ( ∀ dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip)"
```

```
lemma quality_increasesI [intro!]:
assumes " ∀ dip. dip ∈ kD(rt ξ) ⇒ dip ∈ kD(rt ξ')"
and " ∀ dip. [ dip ∈ kD(rt ξ); dip ∈ kD(rt ξ') ] ⇒ rt ξ ⊑dip rt ξ'"
and " ∀ dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip"
shows "quality_increases ξ ξ"
⟨proof⟩
```

```
lemma quality_increasesE [elim]:
fixes dip
assumes "quality_increases ξ ξ'"
and "dip ∈ kD(rt ξ)"
and "[ dip ∈ kD(rt ξ'); rt ξ ⊑dip rt ξ'; sqn (rt ξ) dip ≤ sqn (rt ξ') dip ] ⇒ R dip ξ ξ'"
shows "R dip ξ ξ"
⟨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)

```

```

" \ A 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)))"
" \ A 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))"
" \ A d dip. msg_fresh σ (Newpkt d dip) = True"
" \ A 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 ⇒ recvmsg 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' ≡ {λ i. aodv_init i, ΓAODV PAodv}"

```

```
abbreviation opaodv
```

```

:: "ip ⇒ ((ip ⇒ state) × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
"opaodv i ≡ () init = σAODV', trans = oseqp_sos ΓAODV i ()"

lemma initiali_aodv [intro!, simp]: "initiali i (init (opaodv i)) (init (paodv i))" 
⟨proof⟩

lemma oaodv_control_within [simp]: "control_within ΓAODV (init (opaodv i))" 
⟨proof⟩

lemma σAODV'_labels [simp]: "(σ, p) ∈ σAODV' ⇒ labels ΓAODV p = {PAodv-:0}" 
⟨proof⟩

lemma oaodv_init_kD_empty [simp]:
"(σ, p) ∈ σAODV' ⇒ kD (rt (σ i)) = {}" 
⟨proof⟩

lemma oaodv_init_vD_empty [simp]:
"(σ, p) ∈ σAODV' ⇒ vD (rt (σ i)) = {}" 
⟨proof⟩

lemma oaodv_trans: "trans (opaodv i) = oseqp_sos ΓAODV 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

```

## 0.10 Global invariant proofs over sequential processes

```

theory Global_Invariants
imports Seq_Invariants
  Aodv_Predicates
  Fresher
  Quality_Increases
  AWN.OAWN_Convert
  OAodv
begin

lemma other_quality_increases [elim]:
assumes "other quality_increases I σ σ'"
shows "∀j. quality_increases (σ j) (σ' j)" 
⟨proof⟩

lemma weaken_otherwith [elim]:
fixes m
assumes *: "otherwith P I (orecvmsg Q) σ σ' a"
  and weakenP: "∀σ m. P σ m ⇒ P' σ m"
  and weakenQ: "∀σ m. Q σ m ⇒ Q' σ m"
shows "otherwith P' I (orecvmsg Q') σ σ' a" 
⟨proof⟩

lemma oreceived_msg_inv:
assumes other: "∀σ σ' m. [ P σ m; other Q {i} σ σ' ] ⇒ P σ' m"
  and local: "∀σ m. P σ m ⇒ P (σ(i := σ i (msg := m))) m"
shows "opaodv i ≡ (otherwith Q {i} (orecvmsg P), other Q {i} →)
          onl ΓAODV (λ(σ, l). l ∈ {PAodv-:1} → P σ (msg (σ i)))" 
⟨proof⟩

```

(Equivalent to) Proposition 7.27

```

lemma local_quality_increases:
"paodv i ≡_A (recvmsg rreq_rrep_sn →) onll ΓAODV (λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ' )"

```

$\langle proof \rangle$

```

lemmas olocal_quality_increases =
  open_seq_stepInvariant [OF local_quality_increases initiali_aodv oaodv_trans aodv_trans,
    simplified seq1l_onll_swap]

lemma oquality_increases:
  "opaodv i ⊨_A (otherwith quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)),
    other quality_increases {i} →)
    onll ΓAODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j))"
  (is "_ ⊨_A (?S, _ →) _")
  ⟨proof⟩

lemma rreq_rrep_nsqn_fresh_any_step_invariant:
  "opaodv i ⊨_A (act (recvmsg rreq_rrep_sn), other A {i} →)
    onll ΓAODV (λ((σ, _), a, _). anycast (msg_fresh σ) a)"
  ⟨proof⟩

lemma oreceived_rreq_rrep_nsqn_fresh_inv:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i} →)
    onl ΓAODV (λ(σ, l). l ∈ {PAodv-:1} → msg_fresh σ (msg (σ i)))"
  ⟨proof⟩

lemma oquality_increases_nsqn_fresh:
  "opaodv i ⊨_A (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i} →)
    onll ΓAODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j))"
  ⟨proof⟩

lemma oosn_rreq:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i} →)
    onl ΓAODV (seql i (λ(ξ, l). l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n | n. True} → 1 ≤ osn ξ))"
  ⟨proof⟩

lemma rreq_sip:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i} →)
    onl ΓAODV (λ(σ, l).
      (l ∈ {PAodv-:4, PAodv-:5, PRreq-:0, PRreq-:2} ∧ sip (σ i) ≠ oip (σ i))
      → oip (σ i) ∈ kD(rt (σ (sip (σ i))))
      ∧ nsqn (rt (σ (sip (σ i)))) (oip (σ i)) ≥ osn (σ i)
      ∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i)) = osn (σ i))
      → (hops (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (oip (σ i)))
        ∨ the (flag (rt (σ (sip (σ i)))) (oip (σ i))) = inv)))"
  (is "_ ⊨_A (?S, ?U →) _")
  ⟨proof⟩

lemma odsn_rrep:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i} →)
    onl ΓAODV (seql i (λ(ξ, l). l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n | n. True} → 1 ≤ dsn ξ))"
  ⟨proof⟩

lemma rrep_sip:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i} →)
    onl ΓAODV (λ(σ, l).
      (l ∈ {PAodv-:6, PAodv-:7, PRrep-:0, PRrep-:1} ∧ sip (σ i) ≠ dip (σ i))
      → dip (σ i) ∈ kD(rt (σ (sip (σ 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:
"opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} →)
onl ΓAODV (λ(σ, 1).
  l ∈ {PAodv-:8, PAodv-: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: "paodv i ⊨
  onl ΓAODV (λ(ξ, 1). (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 oaddpreRT_welldefined =
  open_seq_invariant [OF addpreRT_welldefined initiali_aodv oaodv_trans aodv_trans,
  simplified seql_onl_swap,
  THEN oinvariant_anyact]

lemmas odests_vD_inc_sqn =
  open_seq_invariant [OF dests_vD_inc_sqn initiali_aodv oaodv_trans aodv_trans,
  simplified seql_onl_swap,
  THEN oinvariant_anyact]

lemmas oprerr_guard =
  open_seq_invariant [OF prerr_guard initiali_aodv oaodv_trans aodv_trans,
  simplified seql_onl_swap,
  THEN oinvariant_anyact]

```

Proposition 7.28

```

lemma seq_compare_next_hop':
"opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} →) onl ΓAODV (λ(σ, _).
  ∀ 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)"
(is "_" ⊨ (?S, ?U →) _")
⟨proof⟩

```

Proposition 7.30

```

lemmas okD_unk_or_atleast_one =
  open_seq_invariant [OF kD_unk_or_atleast_one initiali_aodv,
  simplified seql_onl_swap]

```

```

lemmas ozero_seq_unk_hops_one =
  open_seq_invariant [OF zero_seq_unk_hops_one initiali_aodv,
  simplified seql_onl_swap]

```

```

lemma oreachable_fresh_okD_unk_or_atleast_one:
  fixes dip
  assumes "(σ, p) ∈ oreachable (opaodv i)
            (otherwith ((=)) {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")

```

$\langle proof \rangle$

```

lemma oreachable_fresh_ozero_seq_unk_hops_one:
  fixes dip
  assumes "( $\sigma$ ,  $p$ ) \in \text{oreachable} (\text{opaodv } i)
           (\text{otherwith } ((=)) \{i\} (\text{orecvmsg } (\lambda \sigma m. \text{msg\_fresh } \sigma m
                                         \wedge \text{msg\_zhops } m)))
           (\text{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 \text{the (dhops (rt (\sigma i)) dip)} = 1
         \wedge \text{the (nhop (rt (\sigma i)) dip)} = dip"
    (is "?P dip")
  ⟨proof⟩

lemma seq_nhop_quality_increases':
  shows "\text{opaodv } i \models (\text{otherwith } ((=)) \{i\}
                                (\text{orecvmsg } (\lambda \sigma m. \text{msg\_fresh } \sigma m \wedge \text{msg\_zhops } m)),
                                \text{other quality\_increases } \{i\} \rightarrow
                                \text{onl } \Gamma_{AODV} (\lambda (\sigma, _). \forall dip. \text{let nhip} = \text{the (nhop (rt (\sigma i)) dip)}
                                         \text{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 (?S i, \_ \rightarrow) \_")
  ⟨proof⟩

lemma seq_nhop_quality_increases:
  shows "\text{opaodv } i \models (\text{otherwith } ((=)) \{i\}
                                (\text{orecvmsg } (\lambda \sigma m. \text{msg\_fresh } \sigma m \wedge \text{msg\_zhops } m)),
                                \text{other quality\_increases } \{i\} \rightarrow
                                \text{global } (\lambda \sigma. \forall dip. \text{let nhip} = \text{the (nhop (rt (\sigma i)) dip)}
                                         \text{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)))"
  ⟨proof⟩
end

```

## 0.11 Routing graphs and loop freedom

```

theory Loop_Freedom
imports Aodv_Predicates 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') / 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]:

```

```

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

⟨proof⟩

lemma rt_graph_vD_trans [dest]:  

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

⟨proof⟩

lemma rt_graph_not_dip [dest]:  

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

⟨proof⟩

lemma rt_graph_not_dip_trans [dest]:  

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

⟨proof⟩

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

lemma rt_graph_nhop_is_nhop [dest]:  

"\ $\bigwedge ip' \sigma dip. (ip, ip') \in rt\_graph \sigma dip \implies ip' = \text{the } (nhop (rt (\sigma ip)) dip)$ "  

⟨proof⟩

theorem inv_to_loop_freedom:  

assumes "¬ $\forall i dip. \text{let nhop} = \text{the } (nhop (rt (\sigma i)) dip)$   

          in  $dip \in vD(rt (\sigma i)) \cap vD(rt (\sigma nhop)) \wedge nhop \neq dip$   

          →  $(rt (\sigma i)) \sqsubset_{dip} (rt (\sigma nhop))$ "  

shows "¬ $\forall dip. \text{irrefl } ((rt\_graph \sigma dip)^+)$ "  

⟨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 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 (=) {i}) (orecvmsg (λσ m.  

msg_fresh σ m ∧ msg_zhops m)),  

other quality_increases {i} →)  

global (λσ. ∀ dip. let nhop = the (nhop (rt (σ i)) dip)  

in dip ∈ vD(rt (σ i)) ∩ vD(rt (σ nhop)) ∧ nhop ≠ dip  

→ (rt (σ i)) ⊂_{dip} (rt (σ nhop)))"  

⟨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⟩

```

### 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 : opaodv i ⟨⟨i qmsg : R⟩⟩_o ≡
          (otherwith ((=)) {i})
            (oarriavemsg (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 : opaodv i ⟨⟨i qmsg : R⟩⟩_o ≡_A (λσ _. oarriavemsg (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 : opaodv i ⟨⟨i qmsg : R⟩⟩_o ≡_A
          (λσ _. oarriavemsg (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 : opaodv i ⟨⟨i qmsg : R⟩⟩_o ≡_A
          (λσ _. oarriavemsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
          globala (λ(_ _, a, _) . castmsg msg_zhops a)"
  ⟨proof⟩

```

```

lemma node_silent_change_only:
  shows "⟨ i : opaodv i ⟨⟨i qmsg : Ri⟩⟩_o ≡_A (λσ _. oarriavemsg (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 "oarriavemsg (λσ m. msg_fresh σ m ∧ P σ m) σ m"
  shows "oarriavemsg (λ_. rreq_rrep_sn) σ m"
  ⟨proof⟩

```

```

lemma opnet_nhop_quality_increases:
  shows "opnet (λi. opaodv i ⟨⟨i qmsg⟩⟩ p ≡
              (otherwith ((=)) (net_tree_ips p)
                (oarriavemsg (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)*

#### 0.12.4 Lift to closed networks

```
lemma onet_nhop_quality_increases:
  shows "oclosed (opnet (λi. opaodv i ⟨⟨i qmsg⟩⟩ p)
    |= (λ_ __. True, other_quality_increases (net_tree_ips p) →)
    global (λσ. ∀i∈net_tree_ips p. ∀dip.
      let nhop = the (nhop (rt (σ i)) dip)
      in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhop)) ∧ nhop ≠ dip
      → (rt (σ i)) ⊑_dip (rt (σ nhop)))"
  (is "_ |= (_ , ?U →) ?inv")
  ⟨proof⟩
```

#### 0.12.5 Transfer into the standard model

```
interpretation aodv_openproc: openproc paodv opaodv id
  rewrites "aodv_openproc.initmissing = initmissing"
  ⟨proof⟩
```

```
interpretation aodv_openproc_par_qmsg: openproc_paq paodv opaodv id qmsg
  rewrites "aodv_openproc_par_qmsg.netglobal = netglobal"
  and "aodv_openproc_par_qmsg.initmissing = initmissing"
  ⟨proof⟩
```

```
lemma net_nhop_quality_increases:
  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paodv i ⟨⟨ qmsg⟩⟩ n) |= netglobal
    (λσ. ∀i dip. let nhop = the (nhop (rt (σ i)) dip)
      in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhop)) ∧ nhop ≠ dip
      → (rt (σ i)) ⊑_dip (rt (σ nhop)))"
  (is "_ |= netglobal (λσ. ∀i. ?inv σ i)")
  ⟨proof⟩
```

#### 0.12.6 Loop freedom of AODV

```
theorem aodv_loop_freedonm:
  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paodv i ⟨⟨ qmsg⟩⟩ n) |= netglobal (λσ. ∀dip. irrefl ((rt_graph σ dip) +)))"
  ⟨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 ⇒ sqn"
  where "inc sn ≡ if sn = 0 then sn else sn + 1"
```

```
lemma less_than_inc [simp]: "x ≤ inc x"
  ⟨proof⟩
```

```
lemma inc_minus_suc_0 [simp]:
  "inc x - Suc 0 = x"
  ⟨proof⟩
```

```
lemma inc_never_one' [simp, intro]: "inc x ≠ Suc 0"
  ⟨proof⟩
```

```
lemma inc_never_one [simp, intro]: "inc x ≠ 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 × 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 " $\pi_4 \equiv \lambda(\_, \_, \_, \_, \_, \_). \text{flag}$ "
definition proj5 :: "r ⇒ nat" (< $\pi_5$ >)
  where " $\pi_5 \equiv \lambda(\_, \_, \_, \_, \_, \_). \text{hops}$ " 

definition proj6 :: "r ⇒ ip" (< $\pi_6$ >)
  where " $\pi_6 \equiv \lambda(\_, \_, \_, \_, \_, \_). \text{nhip}$ " 

definition proj7 :: "r ⇒ ip set" (< $\pi_7$ >)
  where " $\pi_7 \equiv \lambda(\_, \_, \_, \_, \_, \_). \text{pre}$ " 

lemma projs [simp]:
  " $\pi_2(\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}, \text{pre}) = \text{dsn}$ "
  " $\pi_3(\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}, \text{pre}) = \text{dsk}$ "
  " $\pi_4(\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}, \text{pre}) = \text{flag}$ "
  " $\pi_5(\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}, \text{pre}) = \text{hops}$ "
  " $\pi_6(\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}, \text{pre}) = \text{nhip}$ "
  " $\pi_7(\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}, \text{pre}) = \text{pre}$ "
  ⟨proof⟩

lemma proj3_pred [intro]: " $\llbracket P \text{ kno}; P \text{ unk} \rrbracket \implies P (\pi_3 \ x)$ "
  ⟨proof⟩

lemma proj4_pred [intro]: " $\llbracket P \text{ val}; P \text{ inv} \rrbracket \implies P (\pi_4 \ x)$ "
  ⟨proof⟩

lemma proj6_pair_snd [simp]:
  fixes dsn' r
  shows " $\pi_6(\text{dsn}', \text{snd}(r)) = \pi_6(r)$ "
  ⟨proof⟩

```

### 1.1.3 Routing Tables

Routing tables map ip addresses to route entries.

```

type_synonym rt = "ip → r"

syntax
  "_Sigma_route" :: "rt ⇒ ip → r"  (< $\sigma_{\text{route}}$ , (, )>)

translations
  " $\sigma_{\text{route}}(rt, dip)$ " => "rt dip"

definition sqn :: "rt ⇒ ip ⇒ sqn"
  where "sqn rt dip ≡ case  $\sigma_{\text{route}}(rt, dip)$  of Some r ⇒  $\pi_2(r)$  | None ⇒ 0"

definition sqnf :: "rt ⇒ ip ⇒ k"
  where "sqnf rt dip ≡ case  $\sigma_{\text{route}}(rt, dip)$  of Some r ⇒  $\pi_3(r)$  | None ⇒ unk"

abbreviation flag :: "rt ⇒ ip → f"
  where "flag rt dip ≡ map_option  $\pi_4(\sigma_{\text{route}}(rt, dip))$ "

abbreviation dhops :: "rt ⇒ ip → nat"
  where "dhops rt dip ≡ map_option  $\pi_5(\sigma_{\text{route}}(rt, dip))$ "

abbreviation nhop :: "rt ⇒ ip → ip"
  where "nhop rt dip ≡ map_option  $\pi_6(\sigma_{\text{route}}(rt, dip))$ "

abbreviation precs :: "rt ⇒ ip → ip set"
  where "precs rt dip ≡ map_option  $\pi_7(\sigma_{\text{route}}(rt, dip))$ "

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.
    σ_route(rt, dip) = Some (dsn, dsk, flag, hops, nhip, pre)"
  ⟨proof⟩

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

lemma vD_Some [dest]:
  fixes dip rt
  assumes "dip ∈ vD rt"
  shows "∃ dsn dsk hops nhip pre.
    σ_route(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.
    σ_route(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⟩

```

#### 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]:
  " $\bigwedge ip \in kD(rt) \Rightarrow ip \in kD(rt)$ "  $\Rightarrow$  sqnf (the (addpreRT (rt) ip npre)) dip = sqnf (rt) 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: " $\llbracket ip \in kD(rt); sqn rt ip < \pi_2(r) \rrbracket$ 
 $\implies P(rt(ip \mapsto addpre r(\pi_7(\sigma_{route}(rt, ip)))))$ "
and c3: " $\llbracket ip \in kD(rt); sqn rt ip = \pi_2(r); the(dhops rt ip) > \pi_5(r) \rrbracket$ 
 $\implies P(rt(ip \mapsto addpre r(\pi_7(\sigma_{route}(rt, ip)))))$ "
and c4: " $\llbracket ip \in kD(rt); sqn rt ip = \pi_2(r); the(flag rt ip) = inv \rrbracket$ 
 $\implies P(rt(ip \mapsto addpre r(\pi_7(\sigma_{route}(rt, ip)))))$ "
and c5: " $\llbracket ip \in kD(rt); \pi_3(r) = unk \rrbracket$ 
 $\implies P(rt(ip \mapsto (\pi_2(\sigma_{route}(rt, ip)), \pi_3(r),$ 
 $\pi_4(r), \pi_5(r), \pi_6(r), \pi_7(addpre r(\pi_7(\sigma_{route}(rt, ip)))))))$ "
and c6: " $\llbracket ip \in kD(rt); sqn rt ip \geq \pi_2(r); \pi_3(r) = kno;$ 
 $sqn rt ip = \pi_2(r) \implies the(dhops rt ip) \leq \pi_5(r) \wedge the(flag rt ip) = val \rrbracket$ 
 $\implies P(rt(ip \mapsto addpre(the(\sigma_{route}(rt, ip))(\pi_7(r)))))$ "
shows "(P(update rt ip r))"
⟨proof⟩

```

```

lemma update_cases_kD:
assumes "( $\pi_2(r) = 0$ ) = ( $\pi_3(r) = unk$ )"
and "ip ∈ kD(rt)"
and c2: "sqn rt ip <  $\pi_2(r)$   $\implies P(rt(ip \mapsto addpre r(\pi_7(\sigma_{route}(rt, ip)))))$ "
and c3: " $\llbracket sqn rt ip = \pi_2(r); the(dhops rt ip) > \pi_5(r) \rrbracket$ 
 $\implies P(rt(ip \mapsto addpre r(\pi_7(\sigma_{route}(rt, ip)))))$ "
and c4: " $\llbracket sqn rt ip = \pi_2(r); the(flag rt ip) = inv \rrbracket$ 
 $\implies P(rt(ip \mapsto addpre r(\pi_7(\sigma_{route}(rt, ip)))))$ "
and c5: " $\pi_3(r) = unk \implies P(rt(ip \mapsto (\pi_2(\sigma_{route}(rt, ip)), \pi_3(r),$ 
 $\pi_4(r), \pi_5(r), \pi_6(r),$ 
 $\pi_7(addpre r(\pi_7(\sigma_{route}(rt, ip)))))))$ "
and c6: " $\llbracket sqn rt ip \geq \pi_2(r); \pi_3(r) = kno;$ 
 $sqn rt ip = \pi_2(r) \implies the(dhops rt ip) \leq \pi_5(r) \wedge the(flag rt ip) = val \rrbracket$ 
 $\implies P(rt(ip \mapsto addpre(the(\sigma_{route}(rt, ip))(\pi_7(r)))))$ "
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 ≠ 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  $\pi_2(v)$  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 ≤ 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: " $\forall ip \in kD \ rt. \ the(dhops \ rt \ ip) \geq 1$ "  

  and ip: "(ip = rip \wedge Suc 0 \leq hops) \vee (ip \neq rip \wedge ip \in kD \ 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]:  

  " $\bigwedge rt \ ip \ dsn \ dsk \ flag \ hops \ nhip \ pre. \ sqn(rt(ip \mapsto v)) \ ip = \pi_2(v)$ "  

⟨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]:  

  " $\bigwedge rt \ dip \ ip \ dsn \ hops \ npre. \ the(nhop(update rt dip (dsn, unk, val, hops, ip, npre))) \ dip = ip$ "  

⟨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:  

  " $\bigwedge rt \ ip \ dsn \ dsk \ flag \ hops \ sip.$   

  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:

```

```

"\ $\wedge_{rt \in \text{rt\_set}} \forall dip \in \text{ip\_set} \forall dsn \in \text{dsn\_set} \forall hops \in \text{hops\_set} \forall npre \in \text{npre\_set}$ .  $1 \leq dsn \implies 1 \leq (\text{sqn}(\text{update}(rt, dip, (dsn, kno, val, hops, ip, npre)), dip))$ " dip)"  

⟨proof⟩

lemma proj3_update [simp]: "¬ update(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]: "¬ update(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]: "¬ update(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]: "¬ update(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]:  

  "¬ update(rt, dip, dsn, dsk, hops, nhip, pre) =  

  dip ∈ vD(update(rt, dip, (dsn, dsk, val, hops, nhip, pre))) ⇒ (dip ∈ vD(rt) ∨ dip = dip)"  

⟨proof⟩

```

## Invalidating route entries

```

definition invalidate :: "rt ⇒ (ip → sqn) ⇒ rt"  

where "invalidate rt dests ≡  

  λip. case (rt ip, dests ip) of  

    (None, _) ⇒ None  

  | (Some s, None) ⇒ Some s  

  | (Some (_, dsk, _, hops, nhip, pre), Some rsn) ⇒  

    Some (rsn, dsk, inv, hops, nhip, pre)"

```

```
lemma proj3_invalidate [simp]:
```

```

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

⟨proof⟩

lemma proj5_invalidate [simp]:  

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

⟨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⟩

lemma proj7_invalidate [simp]:  

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

⟨proof⟩

1.1.5 Route Requests

lemma invalidate_kD_inv [simp]:  

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

⟨proof⟩

lemma invalidate_sqn:  

fixes rt dip dests  

assumes "¬ $\exists rsn. dests dip = Some rsn \rightarrow sqn rt dip \leq rsn$ "  

shows "sqn rt dip \leq sqn (invalidate rt dests) dip"  

⟨proof⟩

lemma sqn_invalidate_in_dests [simp]:  

fixes dests ipa rsn rt  

assumes "dests ipa = Some rsn"  

and "ipa \in kD(rt)"  

shows "sqn (invalidate rt dests) ipa = rsn"  

⟨proof⟩

lemma dhops_invalidate [simp]:  

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

⟨proof⟩

lemma sqnf_invalidate [simp]:  

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

⟨proof⟩

lemma nhop_invalidate [simp]:  

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

⟨proof⟩

lemma invalidate_other [simp]:  

fixes rt dests dip  

assumes "dip \notin \text{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:  

"\ $\bigwedge dip rt \text{ dests}. dip \in vD(\text{invalidate } rt \text{ dests}) \Rightarrow dip \in vD(rt) \wedge dests dip = None$ "  

⟨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 ⇒ 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))
         ∧ pre = π7(the (rt dip))"

⟨proof⟩

```

**lemma** proj3\_inv: " $\bigwedge dip\ rt\ dests.\ dip \in kD(rt)$   
 $\qquad\qquad\qquad \implies \pi_3(\text{the } (\text{invalidate } rt\ dests\ dip)) = \pi_3(\text{the } (rt\ dip))$ "  
*(proof)*

```

lemma dests_iD_invalidate [simp]:
  assumes "dests ip = Some rsn"
          and "ip ∈ kD(rt)"
  shows "ip ∈ iD(invalidate rt dests)"
  ⟨proof⟩

```

### 1.1.6 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**  $qD :: "store \Rightarrow ip\ set"$   
**where** " $qD \equiv \text{dom}$ "

```

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 ≡ λip. if dests ip = None then store ip
                                             else map_option (λ( ., a). (req, a)) (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.  $\sigma_{p\text{-}flag}(store, dip)$  is partial.
4. The routing table ( $rt$ ) is modelled as a map ( $ip \Rightarrow r\ option$ ) 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 → sqn" ip
  | Newpkt data ip
  | Pkt data ip ip

instantiation msg :: msg
begin
  definition newpkt_def [simp]: "newpkt ≡ λ(d, dip). Newpkt d dip"
  definition eq_newpkt_def: "eq_newpkt m ≡ case m of Newpkt d dip ⇒ True | _ ⇒ 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 × ip × sqn × k × ip × sqn × ip ⇒ msg"
  where "rreq ≡ λ(hops, dip, dsn, dsk, oip, osn, sip).
    Rreq hops dip dsn dsk oip osn 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 × 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 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 = Map.empty,
  rreqs = {},
  store = Map.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  $\xi$  ≡ case msg  $\xi$  of
    Rrep hops' dip' dsn' oip' sip' ⇒
        {  $\xi$  | hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' } }
    | _ ⇒ {}
end

lemma is_rrep_asm [dest!]:
assumes " $\xi'$  ∈ is_rrep  $\xi$ "
shows "( $\exists$  hops' dip' dsn' oip' sip'.
    msg  $\xi$  = Rrep hops' dip' dsn' oip' sip' ∧
     $\xi' = \xi$  | hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' )"
⟨proof⟩
end

definition is_rerr
where "is_rerr  $\xi$  ≡ case msg  $\xi$  of
    Rerr dests' sip' ⇒ {  $\xi$  | dests := dests', sip := sip' } }
    | _ ⇒ {}
end

lemma is_rerr_asm [dest!]:
assumes " $\xi'$  ∈ is_rerr  $\xi$ "
shows "( $\exists$  dests' sip'.
    msg  $\xi$  = Rerr dests' sip' ∧
     $\xi' = \xi$  | dests := dests', sip := sip' )"
⟨proof⟩
end

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]:
" $\xi'$  ∈ is_rerr  $\xi$  ⇒ ip  $\xi'$  = ip  $\xi$ "
" $\xi'$  ∈ is_rrep  $\xi$  ⇒ ip  $\xi'$  = ip  $\xi$ "
" $\xi'$  ∈ is_rreq  $\xi$  ⇒ ip  $\xi'$  = ip  $\xi$ "
" $\xi'$  ∈ is_pkt  $\xi$  ⇒ ip  $\xi'$  = ip  $\xi$ "
" $\xi'$  ∈ is_newpkt  $\xi$  ⇒ ip  $\xi'$  = ip  $\xi$ "
⟨proof⟩
end

lemma is_msg_inv_sn [simp]:
" $\xi'$  ∈ is_rerr  $\xi$  ⇒ sn  $\xi'$  = sn  $\xi$ "
" $\xi'$  ∈ is_rrep  $\xi$  ⇒ sn  $\xi'$  = sn  $\xi$ "
" $\xi'$  ∈ is_rreq  $\xi$  ⇒ sn  $\xi'$  = sn  $\xi$ "
" $\xi'$  ∈ is_pkt  $\xi$  ⇒ sn  $\xi'$  = sn  $\xi$ "
" $\xi'$  ∈ is_newpkt  $\xi$  ⇒ sn  $\xi'$  = sn  $\xi$ "
⟨proof⟩
end

lemma is_msg_inv_rt [simp]:
" $\xi'$  ∈ is_rerr  $\xi$  ⇒ rt  $\xi'$  = rt  $\xi$ "
" $\xi'$  ∈ is_rrep  $\xi$  ⇒ rt  $\xi'$  = rt  $\xi$ "
" $\xi'$  ∈ is_rreq  $\xi$  ⇒ rt  $\xi'$  = rt  $\xi$ "
" $\xi'$  ∈ is_pkt  $\xi$  ⇒ rt  $\xi'$  = rt  $\xi$ "
" $\xi'$  ∈ is_newpkt  $\xi$  ⇒ rt  $\xi'$  = rt  $\xi$ "
⟨proof⟩
end

lemma is_msg_inv_rreqs [simp]:
" $\xi'$  ∈ is_rerr  $\xi$  ⇒ rreqs  $\xi'$  = rreqs  $\xi$ "
" $\xi'$  ∈ is_rrep  $\xi$  ⇒ rreqs  $\xi'$  = rreqs  $\xi$ "
" $\xi'$  ∈ is_rreq  $\xi$  ⇒ rreqs  $\xi'$  = rreqs  $\xi$ "
" $\xi'$  ∈ is_pkt  $\xi$  ⇒ rreqs  $\xi'$  = rreqs  $\xi$ "
" $\xi'$  ∈ is_newpkt  $\xi$  ⇒ rreqs  $\xi'$  = rreqs  $\xi$ "
⟨proof⟩
end

lemma is_msg_inv_store [simp]:
" $\xi'$  ∈ is_rerr  $\xi$  ⇒ store  $\xi'$  = store  $\xi$ "
" $\xi'$  ∈ is_rrep  $\xi$  ⇒ store  $\xi'$  = store  $\xi$ "
" $\xi'$  ∈ is_rreq  $\xi$  ⇒ store  $\xi'$  = store  $\xi$ "

```

```

" $\xi' \in \text{is\_pkt } \xi \implies \text{store } \xi' = \text{store } \xi$ "
" $\xi' \in \text{is\_newpkt } \xi \implies \text{store } \xi' = \text{store } \xi$ "
⟨proof⟩

```

```

lemma is_msg_inv_sip [simp]:
  " $\xi' \in \text{is\_pkt } \xi \implies \text{sip } \xi' = \text{sip } \xi$ "
  " $\xi' \in \text{is\_newpkt } \xi \implies \text{sip } \xi' = \text{sip } \xi$ "
⟨proof⟩

```

### 1.3.3 The protocol process

```

datatype pseqp =
  PAodv
  | PNewPkt
  | PPkt
  | PRreq
  | PRrep
  | PRerr

fun nat_of_seqp :: "pseqp ⇒ nat"
where
  "nat_of_seqp PAodv = 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]: "11 ≤ 12 = (nat_of_seqp 11 ≤ nat_of_seqp 12)"
definition less_seqp [iff]: "11 < 12 = (nat_of_seqp 11 < nat_of_seqp 12)"
instance ⟨proof⟩
end

abbreviation AODV
where
  "AODV ≡ λ_. [clear_locals] call(PAodv)"

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 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, 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()
  )
)

```

```

▷
 $\xi. \xi () \text{dests} := (\lambda rip. \text{if } (rip \in vD(rt \xi) \wedge nhop(rt \xi) rip = nhop(rt \xi) (dip \xi))$ 
 $\quad \text{then Some (inc (sqn(rt \xi) rip)) else None) })]$ 
 $\xi. \xi () rt := invalidate(rt \xi) (\text{dests } \xi) )]$ 
 $\xi. \xi () store := setRRF(store \xi) (\text{dests } \xi) )]$ 
 $\xi. \xi () pre := \bigcup \{ \text{the (precs (rt \xi) rip)} \mid rip. rip \in \text{dom}(\text{dests } \xi) \} )]$ 
 $\xi. \xi () \text{dests} := (\lambda rip. \text{if } ((\text{dests } \xi) rip \neq \text{None} \wedge \text{the (precs (rt \xi) rip)} \neq \{\})$ 
 $\quad \text{then } (\text{dests } \xi) rip \text{ else None) })]$ 
 $\text{groupcast}(\lambda \xi. \text{pre } \xi, \lambda \xi. \text{rerr}(\text{dests } \xi, ip \xi)). AODV()$ 
 $\oplus \langle \xi. dip \xi \notin vD(rt \xi) \rangle$ 
 $($ 
 $\quad \langle \xi. dip \xi \in iD(rt \xi) \rangle$ 
 $\quad \text{groupcast}(\lambda \xi. \text{the (precs (rt \xi) (dip \xi))},$ 
 $\quad \quad \lambda \xi. \text{rerr}([dip \xi \mapsto sqn(rt \xi) (dip \xi)], ip \xi)). AODV()$ 
 $\oplus \langle \xi. dip \xi \notin id(rt \xi) \rangle$ 
 $\quad AODV()$ 
 $)$ 
 $)")$ 

| " $\Gamma_{AODV} PRreq = \text{labelled PRreq} ($ 
 $\langle \xi. (oip \xi, osn \xi) \in rreqs \xi \rangle$ 
 $AODV()$ 
 $\oplus \langle \xi. (oip \xi, osn \xi) \notin rreqs \xi \rangle$ 
 $\xi. \xi () rt := update(rt \xi) (oip \xi) (osn \xi, kno, val, hops \xi + 1, sip \xi, \{\}) )]$ 
 $\xi. \xi () rreqs := rreqs \xi \cup \{(oip \xi, osn \xi)\} )]$ 
 $($ 
 $\quad \langle \xi. dip \xi = ip \xi \rangle$ 
 $\quad \xi. \xi () sn := max(sn \xi) (dsn \xi) )]$ 
 $\quad \text{unicast}(\lambda \xi. \text{the (nhop (rt \xi) (oip \xi))}, \lambda \xi. \text{rrep}(0, dip \xi, sn \xi, oip \xi, ip \xi)). AODV()$ 
 $\triangleright$ 
 $\quad \xi. \xi () \text{dests} := (\lambda rip. \text{if } (rip \in vD(rt \xi) \wedge nhop(rt \xi) rip = nhop(rt \xi) (oip \xi))$ 
 $\quad \quad \text{then Some (inc (sqn(rt \xi) rip)) else None) })]$ 
 $\quad \xi. \xi () rt := invalidate(rt \xi) (\text{dests } \xi) )]$ 
 $\quad \xi. \xi () store := setRRF(store \xi) (\text{dests } \xi) )]$ 
 $\quad \xi. \xi () pre := \bigcup \{ \text{the (precs (rt \xi) rip)} \mid rip. rip \in \text{dom}(\text{dests } \xi) \} )]$ 
 $\quad \xi. \xi () \text{dests} := (\lambda rip. \text{if } ((\text{dests } \xi) rip \neq \text{None} \wedge \text{the (precs (rt \xi) rip)} \neq \{\})$ 
 $\quad \quad \text{then } (\text{dests } \xi) rip \text{ else None) })]$ 
 $\quad \text{groupcast}(\lambda \xi. \text{pre } \xi, \lambda \xi. \text{rerr}(\text{dests } \xi, ip \xi)). AODV()$ 
 $\oplus \langle \xi. dip \xi \neq ip \xi \rangle$ 
 $($ 
 $\quad \langle \xi. dip \xi \in vD(rt \xi) \wedge dsn \xi \leq sqn(rt \xi) (dip \xi) \wedge sqnf(rt \xi) (dip \xi) = kno \rangle$ 
 $\quad \xi. \xi () rt := \text{the (addpreRT (rt \xi) (dip \xi) \{sip \xi\})} )]$ 
 $\quad \xi. \xi () rt := \text{the (addpreRT (rt \xi) (oip \xi) \{the (nhop(rt \xi) (dip \xi))\})} )]$ 
 $\quad \text{unicast}(\lambda \xi. \text{the (nhop (rt \xi) (oip \xi))}, \lambda \xi. \text{rrep}(\text{the (dhops (rt \xi) (dip \xi))}, dip \xi,$ 
 $\quad \quad sqn(rt \xi) (dip \xi), oip \xi, ip \xi)).$ 
 $\quad AODV()$ 
 $\triangleright$ 
 $\quad \xi. \xi () \text{dests} := (\lambda rip. \text{if } (rip \in vD(rt \xi) \wedge nhop(rt \xi) rip = nhop(rt \xi) (oip \xi))$ 
 $\quad \quad \text{then Some (inc (sqn(rt \xi) rip)) else None) })]$ 
 $\quad \xi. \xi () rt := invalidate(rt \xi) (\text{dests } \xi) )]$ 
 $\quad \xi. \xi () store := setRRF(store \xi) (\text{dests } \xi) )]$ 
 $\quad \xi. \xi () pre := \bigcup \{ \text{the (precs (rt \xi) rip)} \mid rip. rip \in \text{dom}(\text{dests } \xi) \} )]$ 
 $\quad \xi. \xi () \text{dests} := (\lambda rip. \text{if } ((\text{dests } \xi) rip \neq \text{None} \wedge \text{the (precs (rt \xi) rip)} \neq \{\})$ 
 $\quad \quad \text{then } (\text{dests } \xi) rip \text{ else None) })]$ 
 $\quad \text{groupcast}(\lambda \xi. \text{pre } \xi, \lambda \xi. \text{rerr}(\text{dests } \xi, ip \xi)). AODV()$ 
 $\oplus \langle \xi. dip \xi \notin vD(rt \xi) \vee sqn(rt \xi) (dip \xi) < dsn \xi \vee sqnf(rt \xi) (dip \xi) = unk \rangle$ 
 $\quad \text{broadcast}(\lambda \xi. \text{rreq}(hops \xi + 1, dip \xi, max(sqn(rt \xi) (dip \xi)) (dsn \xi),$ 
 $\quad \quad dsk \xi, oip \xi, osn \xi, ip \xi)).$ 
 $\quad AODV()$ 
 $)$ 
 $)")$ 

| " $\Gamma_{AODV} PRrep = \text{labelled PRrep} ($ 
 $\langle \xi. rt \xi \neq update(rt \xi) (dip \xi) (dsn \xi, kno, val, hops \xi + 1, sip \xi, \{\}) \rangle$ 

```

```

(
   $\xi. \xi \in \text{rt} := \text{update}(\text{rt } \xi, \text{dip } \xi, \text{dsn } \xi, \text{kno}, \text{val}, \text{hops } \xi + 1, \text{sip } \xi, \{\}) \mid$ 
  (
     $\langle\xi. \text{oip } \xi = \text{ip } \xi \rangle$ 
    AODV()
     $\oplus \langle\xi. \text{oip } \xi \neq \text{ip } \xi \rangle$ 
    (
       $\langle\xi. \text{oip } \xi \in \text{vD}(\text{rt } \xi)\rangle$ 
       $\llbracket \xi. \xi \in \text{rt} := \text{the}(\text{addpreRT}(\text{rt } \xi, \text{dip } \xi, \{\text{the}(\text{nhop}(\text{rt } \xi, \text{oip } \xi)\})) \mid \rrbracket$ 
       $\llbracket \xi. \xi \in \text{rt} := \text{the}(\text{addpreRT}(\text{rt } \xi, \{\text{the}(\text{nhop}(\text{rt } \xi, \text{dip } \xi)\}))$ 
       $\quad \{\text{the}(\text{nhop}(\text{rt } \xi, \text{oip } \xi)\}) \mid \rrbracket$ 
      unicast( $\lambda\xi. \text{the}(\text{nhop}(\text{rt } \xi, \text{oip } \xi))$ ,  $\lambda\xi. \text{rrep}(\text{hops } \xi + 1, \text{dip } \xi, \text{dsn } \xi, \text{oip } \xi, \text{ip } \xi)$ ).
      AODV()
    )
     $\triangleright$ 
     $\llbracket \xi. \xi \in \text{dests} := (\lambda\text{rip}. \text{if } (\text{rip} \in \text{vD}(\text{rt } \xi) \wedge \text{nhop}(\text{rt } \xi, \text{rip}) = \text{nhop}(\text{rt } \xi, \text{oip } \xi))$ 
     $\quad \text{then Some}(\text{inc}(\text{sqn}(\text{rt } \xi, \text{rip})) \text{ else None}) \mid \rrbracket$ 
     $\llbracket \xi. \xi \in \text{rt} := \text{invalidate}(\text{rt } \xi, \text{dests } \xi) \mid \rrbracket$ 
     $\llbracket \xi. \xi \in \text{store} := \text{setRRF}(\text{store } \xi, \text{dests } \xi) \mid \rrbracket$ 
     $\llbracket \xi. \xi \in \text{pre} := \bigcup \{\text{the}(\text{precs}(\text{rt } \xi, \text{rip})) \mid \text{rip} \in \text{dom}(\text{dests } \xi)\} \mid \rrbracket$ 
     $\llbracket \xi. \xi \in \text{dests} := (\lambda\text{rip}. \text{if } ((\text{dests } \xi, \text{rip}) \neq \text{None} \wedge \text{the}(\text{precs}(\text{rt } \xi, \text{rip})) \neq \{\}))$ 
     $\quad \text{then}(\text{dests } \xi, \text{rip} \text{ else None}) \mid \rrbracket$ 
    groupcast( $\lambda\xi. \text{pre } \xi$ ,  $\lambda\xi. \text{rerr}(\text{dests } \xi, \text{ip } \xi)$ ). AODV()
     $\oplus \langle\xi. \text{oip } \xi \notin \text{vD}(\text{rt } \xi)\rangle$ 
    AODV()
  )
)
 $\oplus \langle\xi. \text{rt } \xi = \text{update}(\text{rt } \xi, \text{dip } \xi, \text{dsn } \xi, \text{kno}, \text{val}, \text{hops } \xi + 1, \text{sip } \xi, \{\}) \rangle$ 
AODV()
)"

```

```

| " $\Gamma_{AODV} \text{ PRerr} = \text{labelled PRerr}$  (
   $\llbracket \xi. \xi \in \text{dests} := (\lambda\text{rip}. \text{case}(\text{dests } \xi, \text{rip}) \text{ of None} \Rightarrow \text{None}$ 
   $\quad | \text{Some } \text{rsn} \Rightarrow \text{if } \text{rip} \in \text{vD}(\text{rt } \xi) \wedge \text{the}(\text{nhop}(\text{rt } \xi, \text{rip})) = \text{sip } \xi$ 
   $\quad \quad \wedge \text{sqn}(\text{rt } \xi, \text{rip}) < \text{rsn} \text{ then Some } \text{rsn} \text{ else None}) \mid \rrbracket$ 
   $\llbracket \xi. \xi \in \text{rt} := \text{invalidate}(\text{rt } \xi, \text{dests } \xi) \mid \rrbracket$ 
   $\llbracket \xi. \xi \in \text{store} := \text{setRRF}(\text{store } \xi, \text{dests } \xi) \mid \rrbracket$ 
   $\llbracket \xi. \xi \in \text{pre} := \bigcup \{\text{the}(\text{precs}(\text{rt } \xi, \text{rip})) \mid \text{rip} \in \text{dom}(\text{dests } \xi)\} \mid \rrbracket$ 
   $\llbracket \xi. \xi \in \text{dests} := (\lambda\text{rip}. \text{if } ((\text{dests } \xi, \text{rip}) \neq \text{None} \wedge \text{the}(\text{precs}(\text{rt } \xi, \text{rip})) \neq \{\}))$ 
   $\quad \text{then}(\text{dests } \xi, \text{rip} \text{ else None}) \mid \rrbracket$ 
  groupcast( $\lambda\xi. \text{pre } \xi$ ,  $\lambda\xi. \text{rerr}(\text{dests } \xi, \text{ip } \xi)$ ). AODV()"

```

```

declare  $\Gamma_{AODV}.\text{simp}$ s [simp del, code del]
lemmas  $\Gamma_{AODV}\_simp$ s [simp, code] =  $\Gamma_{AODV}.\text{simp}$ s [simplified]

```

```
fun  $\Gamma_{AODV\_skeleton}$ 
```

```
where
```

```

  " $\Gamma_{AODV\_skeleton} \text{ PAodv} = \text{seqp\_skeleton}(\Gamma_{AODV} \text{ PAodv})$ " 
  | " $\Gamma_{AODV\_skeleton} \text{ PNewPkt} = \text{seqp\_skeleton}(\Gamma_{AODV} \text{ PNewPkt})$ " 
  | " $\Gamma_{AODV\_skeleton} \text{ PPkt} = \text{seqp\_skeleton}(\Gamma_{AODV} \text{ PPkt})$ " 
  | " $\Gamma_{AODV\_skeleton} \text{ PRreq} = \text{seqp\_skeleton}(\Gamma_{AODV} \text{ PRreq})$ " 
  | " $\Gamma_{AODV\_skeleton} \text{ PRrep} = \text{seqp\_skeleton}(\Gamma_{AODV} \text{ PRrep})$ " 
  | " $\Gamma_{AODV\_skeleton} \text{ PRerr} = \text{seqp\_skeleton}(\Gamma_{AODV} \text{ PRerr})$ " 

```

```

lemma  $\Gamma_{AODV\_skeleton\_wf}$  [simp]:
  "wellformed  $\Gamma_{AODV\_skeleton}$ "
  ⟨proof⟩

```

```

declare  $\Gamma_{AODV\_skeleton}.\text{simp}$ s [simp del, code del]
lemmas  $\Gamma_{AODV\_skeleton}\_simp$ s [simp, code]
  =  $\Gamma_{AODV\_skeleton}.\text{simp}$ s [simplified  $\Gamma_{AODV}.\text{simp}$ s seqp_skeleton.simp]

```

```
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 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 aodv_simple_labels [simp]: "simple_labels ΓAODV" 
⟨proof⟩

lemma σAODV_labels [simp]: "(ξ, p) ∈ σAODV i ==> labels ΓAODV p = {PAodv-:0}"
⟨proof⟩

lemma aodv_init_kD_empty [simp]:
  "(ξ, p) ∈ σAODV i ==> kD (rt ξ) = {}"
⟨proof⟩

lemma aodv_init_sip_not_ip [simp]: "¬(sip (aodv_init i) = i)" ⟨proof⟩

lemma aodv_init_sip_not_ip' [simp]:
  assumes "(ξ, p) ∈ σAODV i"
  shows "sip ξ ≠ ip ξ"
⟨proof⟩

lemma aodv_init_sip_not_i [simp]:
  assumes "(ξ, p) ∈ σAODV 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

```

 $\Gamma_{AODV\_simp} \ [cterms\_env]$ 
 $aodv\_proc\_cases \ [cterms1\_cases]$ 
 $seq\_invariant\_ctermsI \ [OF \ aodv\_wf \ aodv\_control\_within \ aodv\_simple\_labels \ aodv\_trans,$ 
 $\qquad \qquad \qquad cterms\_intros]$ 
 $seq\_step\_invariant\_ctermsI \ [OF \ aodv\_wf \ aodv\_control\_within \ aodv\_simple\_labels \ aodv\_trans,$ 
 $\qquad \qquad \qquad cterms \ intros]$ 

```

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 ⇒ ip"
where "msg_sender m ≡ case m of Rreq _ _ _ _ _ ipc ⇒ ipc
          | Rrep _ _ _ _ ipc ⇒ ipc
          | Rerr _ ipc ⇒ ipc
          | Pkt _ _ ipc ⇒ ipc"

```

```

lemma msg_sender_simp [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"
  "¬ ∃ 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 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"
  "¬(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_simp [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"

```

```

"\ $\wedge d \text{ dip}.$             $rreq\_rrep\_sn (\text{Newpkt } d \text{ dip}) = \text{True}$ "  

"\ $\wedge d \text{ dip } sip.$          $rreq\_rrep\_sn (\text{Pkt } d \text{ dip } sip) = \text{True}$ "  

⟨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  $\rightarrow$   

                                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  $\rightarrow$   

                                    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.  

  rreq_rrep_fresh crt (Rreq hops dip dsn dsk oip osn sip) =  

    (sip  $\neq$  oip  $\rightarrow$  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  $\rightarrow$  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).  

                                (rip  $\in$  iD(crt)  $\wedge$  the (destsc ripc) = sqn crt ripc))  

                                | _  $\Rightarrow$  True"

lemma rerr_invalid [simp]:  

"\ $\wedge$  hops dip dsn dsk oip osn sip.  

  rerr_invalid crt (Rreq hops 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) = ( $\forall$  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"  

"\ $\wedge$  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 σ = ( $\lambda$ i. case (fst σ) i of None  $\Rightarrow$  aodv_init i | Some s  $\Rightarrow$  s, snd σ)"

lemma not_in_net_ips_fst_init_missing [simp]:  

assumes "i  $\notin$  net_ips σ"  

shows "fst (initmissing (netgmap fst σ)) i = aodv_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
  nsqn_r :: "r ⇒ sqn"
where
  "nsqn_r r ≡ if π₄(r) = val ∨ π₂(r) = 0 then π₂(r) else (π₂(r) - 1)"

lemma nsqnr_def':
  "nsqn_r r = (if π₄(r) = inv then π₂(r) - 1 else π₂(r))"
  ⟨proof⟩

lemma nsqn_r_zero [simp]:
  "¬(nsqn_r (0, dsk, flag, hops, nhip, pre) = 0)"
  ⟨proof⟩

lemma nsqn_r_val [simp]:
  "¬(nsqn_r (dsn, dsk, val, hops, nhip, pre) = dsn)"
  ⟨proof⟩

lemma nsqn_r_inv [simp]:
  "¬(nsqn_r (dsn, dsk, inv, hops, nhip, pre) = dsn - 1)"
  ⟨proof⟩

lemma nsqn_r_lte_dsn [simp]:
  "¬(nsqn_r (dsn, dsk, flag, hops, nhip, pre) > dsn)"
  ⟨proof⟩

```

#### On routes in routing tables

```

definition
  nsqn :: "rt ⇒ ip ⇒ sqn"
where
  "nsqn ≡ λrt dip. case σ_route(rt, dip) of None ⇒ 0 | Some r ⇒ nsqn_r(r)"

lemma nsqn_sqn_def:
  "¬(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 = nsqn_r(the (σ_route(rt, dip)))"
  ⟨proof⟩

lemma nsqn_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 ≤ 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]: " $\bigwedge rt\ ip\ dsn\ dsk\ hops\ nhip.$ 
   $rt \neq update\ rt\ ip\ (dsn, kno, val, hops, nhip, \{\})$ 
   $\implies nsqn\ (update\ rt\ ip\ (dsn, kno, val, hops, nhip, \{\}))\ ip = dsn$ "
  ⟨proof⟩

lemma nsqn_addpreRT_inv [simp]:
  " $\bigwedge rt\ dip\ npre\ dip'.\ dip \in kD(rt) \implies$ 
   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" ( $\langle \_/\sqsubseteq\_ \rangle$  [51, 51] 50)

where

"*fresher* r r' ≡ ((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')))"

lemma *fresherI1* [intro]:

assumes "nsqn<sub>r</sub> r < nsqn<sub>r</sub> r'"  
 shows "r ⊑ r'"

⟨proof⟩

lemma *fresherI2* [intro]:

assumes "nsqn<sub>r</sub> r = nsqn<sub>r</sub> r'"  
 and "π<sub>5</sub>(r) ≥ π<sub>5</sub>(r')"  
 shows "r ⊑ 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' 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

" $rt\_fresher \equiv \lambda dip\ rt\ rt'. (\text{the } (\sigma_{route}(rt, dip))) \sqsubseteq (\text{the } (\sigma_{route}(rt', dip)))$ "

abbreviation

$rt\_fresher\_syn :: "rt \Rightarrow ip \Rightarrow rt \Rightarrow bool" \langle \_/\sqsubseteq\_ \rangle [51, 999, 51] 50$

where

" $rt1 \sqsubseteq_i rt2 \equiv rt\_fresher i rt1 rt2$ "

lemma  $rt\_fresher\_def'$ :

" $(rt1 \sqsubseteq_i rt2) = (nsqn_r (\text{the } (rt1 i)) < nsqn_r (\text{the } (rt2 i)) \vee nsqn_r (\text{the } (rt1 i)) = nsqn_r (\text{the } (rt2 i)) \wedge \pi_5 (\text{the } (rt2 i)) \leq \pi_5 (\text{the } (rt1 i)))$ "  
 $\langle proof \rangle$

lemma  $single\_rt\_fresher$  [intro]:

assumes "the (rt1 ip)  $\sqsubseteq$  the (rt2 ip)"  
shows "rt1  $\sqsubseteq_{ip}$  rt2"  
 $\langle proof \rangle$

lemma  $rt\_fresher\_single$  [intro]:

assumes "rt1  $\sqsubseteq_{ip}$  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  $\sqsubseteq_{dip}$  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  $\sqsubseteq_{dip}$  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  $\sqsubseteq_{dip}$  rt2"  
 $\langle proof \rangle$

lemma  $rt\_fresherE$  [elim]:

assumes "rt1  $\sqsubseteq_{dip}$  rt2"  
and "dip  $\in kD(rt1)"$   
and "dip  $\in kD(rt2)"$   
and " $\llbracket nsqn rt1 dip < nsqn rt2 dip \rrbracket \implies P rt1 rt2 dip$ "  
and " $\llbracket nsqn rt1 dip = nsqn rt2 dip; the (dhops rt1 dip) \geq the (dhops rt2 dip) \rrbracket \implies P rt1 rt2 dip$ "  
shows "P rt1 rt2 dip"  
 $\langle proof \rangle$

lemma  $rt\_fresher\_refl$  [simp]: "rt  $\sqsubseteq_{dip}$  rt"

$\langle proof \rangle$

lemma  $rt\_fresher\_trans$  [elim, trans]:

assumes "rt1  $\sqsubseteq_{dip}$  rt2"  
and "rt2  $\sqsubseteq_{dip}$  rt3"  
shows "rt1  $\sqsubseteq_{dip}$  rt3"  
 $\langle proof \rangle$

```

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" (<_/≈_>) [51, 999, 51] 50
where
  "rt1 ≈_i rt2 ≡ rt_fresh_as i 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 "π_5(the (rt dip)) = π_5(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_nsqrn:
  assumes "dip ∈ kD(rt1)"
    and "dip ∈ kD(rt2)"
    and "rt1 ≈_dip rt2"
  shows "nsqrn_r (the (rt2 dip)) = nsqrn_r (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 nsqrn_r_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]

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 ⊑_dip rt2"  
  shows "the (rt1 dip) ⊑ the (rt2 dip)"  
(proof)
```

```
lemma rt_strictly_fresher_not_fresh_asD [dest]:  
  assumes "rt1 ⊑_dip rt2"  
  shows "¬ rt1 ≈_dip rt2"  
(proof)
```

```
lemma rt_strictly_fresher_trans [elim, trans]:  
  assumes "rt1 ⊑_dip rt2"  
    and "rt2 ⊑_dip rt3"  
  shows "rt1 ⊑_dip rt3"  
(proof)
```

```
lemma rt_strictly_fresher_irrefl [simp]: "¬ (rt ⊑_dip rt)"  
(proof)
```

```
lemma rt_fresher_trans_rt_strictly_fresher [elim, trans]:  
  assumes "rt1 ⊑_dip rt2"  
    and "rt2 ⊑_dip rt3"  
  shows "rt1 ⊑_dip rt3"  
(proof)
```

```
lemma rt_fresher_trans_rt_strictly_fresher' [elim, trans]:  
  assumes "rt1 ⊑_dip rt2"  
    and "rt2 ⊑_dip rt3"  
  shows "rt1 ⊑_dip rt3"  
(proof)
```

```
lemma rt_fresher_imp_nsqn_le:  
  assumes "rt1 ⊑_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!]:
  " $\bigwedge dip ip rt r rt'. \llbracket dip \neq ip; rt \sqsubset_{dip} rt' \rrbracket \implies update rt ip r \sqsubset_{dip} rt'$ " 
  ⟨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 \neq 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) \leq hops"
    and **: "rt \neq 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 " $\forall ip \in \text{dom}(\text{dests}). ip \in vD(rt) \wedge \text{the}(\text{dests } ip) = \text{inc}(\text{sqn } rt \ ip)$ "
  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 \models_A onl \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, 1). 1 \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 ≡ onl ΓAODV (λ(ξ, 1). (1 = PRrep-:1 →
    rt ξ ≠ update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, {})))"
  ⟨proof⟩

lemma addpreRT_partly_welldefined:
  "paodv i ≡
    onl ΓAODV (λ(ξ, 1). (1 ∈ {PRreq-:16..PRreq-:18} ∪ {PRrep-:2..PRrep-:6} → dip ξ ∈ kD (rt ξ))
    ∧ (1 ∈ {PRreq-:3..PRreq-:17} → oip ξ ∈ kD (rt ξ)))"
  ⟨proof⟩

```

Proposition 7.38

```

lemma includes_nhip:
  "paodv i ≡ onl ΓAODV (λ(ξ, 1). ∀ dip ∈ kD(rt ξ). the (nhop (rt ξ) dip) ∈ kD(rt ξ))"
  ⟨proof⟩

```

Proposition 7.22: needed in Proposition 7.4

```

lemma addpreRT_welldefined:
  "paodv i ≡ onl ΓAODV (λ(ξ, 1). (1 ∈ {PRreq-:16..PRreq-:18} → dip ξ ∈ kD (rt ξ)) ∧
    (1 = PRreq-:17 → oip ξ ∈ kD (rt ξ)) ∧
    (1 = PRrep-:5 → dip ξ ∈ kD (rt ξ)) ∧
    (1 = PRrep-:6 → (the (nhop (rt ξ) (dip ξ))) ∈ kD (rt ξ)))"
  (is "_ ≡ onl ΓAODV ?P")
  ⟨proof⟩

```

Proposition 7.4

```

lemma known_destinations_increase:
  "paodv i ≡A onll ΓAODV (λ((ξ, _), _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ'))"
  ⟨proof⟩

```

Proposition 7.5

```

lemma rreqs_increase:
  "paodv i ≡A onll ΓAODV (λ((ξ, _), _, (ξ', _)). rreqs ξ ⊆ rreqs ξ')"
  ⟨proof⟩

```

```

lemma dests_bigger_than_sqn:
  "paodv i ≡ onl ΓAODV (λ(ξ, 1). 1 ∈ {PAodv-:15..PAodv-:19}
    ∪ {PPkt-:7..PPkt-:11}
    ∪ {PRreq-:9..PRreq-:13}
    ∪ {PRreq-:21..PRreq-:25}
    ∪ {PRrep-:10..PRrep-:14}
    ∪ {PRerr-:1..PRerr-:5})
    → (∀ ip ∈ dom(dests ξ). ip ∈ kD(rt ξ) ∧ sqn (rt ξ) ip ≤ the (dests ξ ip)))"
  ⟨proof⟩

```

Proposition 7.6

```

lemma sqns_increase:
  "paodv i ≡A onll ΓAODV (λ((ξ, _), _, (ξ', _)). ∀ ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip)"
  ⟨proof⟩

```

Proposition 7.7

```

lemma ip_constant:
  "paodv i ≡ onl ΓAODV (λ(ξ, _). ip ξ = i)"
  ⟨proof⟩

```

Proposition 7.8

```

lemma sender_ip_valid':
  "paodv i ≡A onll ΓAODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m → msg_sender m = ip ξ) a)"
  ⟨proof⟩

```

```

lemma sender_ip_valid:

```

```

"paodv i ⊨_A onll ΓAODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m → msg_sender m = i) a)"
⟨proof⟩

lemma received_msg_inv:
"paodv i ⊨ (recvmsg P → onl ΓAODV (λ(ξ, l). l ∈ {PAodv-:1} → P (msg ξ)))"
⟨proof⟩

Proposition 7.9

lemma sip_not_ip':
"paodv i ⊨ (recvmsg (λm. not_Pkt m → msg_sender m ≠ i) → onl ΓAODV (λ(ξ, _). sip ξ ≠ ip ξ))"
⟨proof⟩

lemma sip_not_ip:
"paodv i ⊨ (recvmsg (λm. not_Pkt m → msg_sender m ≠ i) → onl ΓAODV (λ(ξ, _). sip ξ ≠ 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 ⊨ onl ΓAODV (λ(ξ, _). ∀ip∈kD (rt ξ). the (dhops (rt ξ) ip) ≥ 1)"
⟨proof⟩

lemma rreq_dip_in_vD_dip_eq_ip:
"paodv i ⊨ onl ΓAODV (λ(ξ, l). (l ∈ {PRreq-:16..PRreq-:18} → dip ξ ∈ vD(rt ξ))
                                         ∧ (l ∈ {PRreq-:5, PRreq-:6} → dip ξ = ip ξ)
                                         ∧ (l ∈ {PRreq-:15..PRreq-:18} → dip ξ ≠ ip ξ))"
⟨proof⟩

Proposition 7.11

lemma anycast_msg_zhops:
"¬rreqid dip dsn dsk oip osn sip.
   paodv i ⊨_A onll ΓAODV (λ(_, a, _). anycast msg_zhops a)"
⟨proof⟩

lemma hop_count_zero_oip_dip_sip:
"paodv i ⊨ (recvmsg msg_zhops → onl ΓAODV (λ(ξ, l).
                                         (l ∈ {PAodv-:4..PAodv-:5} ∪ {PRreq-:n/n. True} →
                                         (hops ξ = 0 → oip ξ = sip ξ))
                                         ∧
                                         ((l ∈ {PAodv-:6..PAodv-:7} ∪ {PRrep-:n/n. True} →
                                         (hops ξ = 0 → dip ξ = sip ξ))))"
⟨proof⟩

lemma osn_rreq:
"paodv i ⊨ (recvmsg rreq_rrep_sn → onl ΓAODV (λ(ξ, l).
                                         l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n/n. True} → 1 ≤ osn ξ))"
⟨proof⟩

lemma osn_rreq':
"paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) → onl ΓAODV (λ(ξ, l).
                                         l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n/n. True} → 1 ≤ osn ξ))"
⟨proof⟩

lemma dsn_rrep:
"paodv i ⊨ (recvmsg rreq_rrep_sn → onl ΓAODV (λ(ξ, l).
                                         l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n/n. True} → 1 ≤ dsn ξ))"
⟨proof⟩

lemma dsn_rrep':
"paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) → onl ΓAODV (λ(ξ, l).
                                         l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n/n. True} → 1 ≤ dsn ξ))"
⟨proof⟩

```

```

lemma hop_count_zero_oip_dip_sip':
"paodv i ≡ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) → onl ΓAODV (λ(ξ, 1).
    (l ∈ {PAodv-:4..PAodv-:5} ∪ {PRreq-:n/n. True} →
        (hops ξ = 0 → oip ξ = sip ξ))
    ∧
    ((l ∈ {PAodv-:6..PAodv-:7} ∪ {PRrep-:n/n. True} →
        (hops ξ = 0 → dip ξ = sip ξ))))"

```

*(proof)*

Proposition 7.12

```

lemma zero_seq_unk_hops_one':
"paodv i ≡ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) → onl ΓAODV (λ(ξ, _).
    ∀ dip ∈ kD(rt ξ). (sqn(rt ξ) dip = 0 → sqnf(rt ξ) dip = unk)
    ∧ (sqnf(rt ξ) dip = unk → the(dhops(rt ξ) dip) = 1)
    ∧ (the(dhops(rt ξ) dip) = 1 → the(nhop(rt ξ) dip) = dip)))"

```

*(proof)*

```

lemma zero_seq_unk_hops_one:
"paodv i ≡ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) → onl ΓAODV (λ(ξ, _).
    ∀ dip ∈ kD(rt ξ). (sqn(rt ξ) dip = 0 → (sqnf(rt ξ) dip = unk
    ∧ the(dhops(rt ξ) dip) = 1
    ∧ the(nhop(rt ξ) dip) = dip)))"

```

*(proof)*

```

lemma kD_unk_or_atleast_one:
"paodv i ≡ (recvmsg rreq_rrep_sn → onl ΓAODV (λ(ξ, 1).
    ∀ dip ∈ kD(rt ξ). π3(the(rt ξ dip)) = unk ∨ 1 ≤ π2(the(rt ξ dip))))"

```

*(proof)*

Proposition 7.13

```

lemma rreq_rrep_sn_any_step_invariant:
"paodv i ≡A (recvmsg rreq_rrep_sn → onll ΓAODV (λ( _, a, _). anycast rreq_rrep_sn a))"

```

*(proof)*

Proposition 7.14

```

lemma rreq_rrep_fresh_any_step_invariant:
"paodv i ≡A onll ΓAODV (λ((ξ, _), a, _). anycast (rreq_rrep_fresh(rt ξ)) a)"

```

*(proof)*

Proposition 7.15

```

lemma rerr_invalid_any_step_invariant:
"paodv i ≡A onll ΓAODV (λ((ξ, _), a, _). anycast (rerr_invalid(rt ξ)) 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:
"paodv i ≡
    onl ΓAODV (λ(ξ, 1). (l ∈ {PAodv-:15, PPkt-:7, PRreq-:9, PRreq-:21, PRrep-:10}
    → ( ∀ ip ∈ dom(dests ξ). ip ∈ vD(rt ξ) ∧ the(dests ξ ip) = inc(sqn(rt ξ) ip)))
    ∧ (l = PRerr-:1
    → ( ∀ ip ∈ dom(dests ξ). ip ∈ vD(rt ξ) ∧ the(dests ξ ip) > sqn(rt ξ) ip))))"

```

*⟨proof⟩*

Proposition 7.27

```
lemma route_tables_fresher:
  "paodv i ⊨_A (recvmsg rreq_rrep_sn → onll ΓAO DV (λ((ξ, _), _, (ξ', _)).  
                                              ∀ dip ∈ kD(rt ξ). rt ξ ⊑_dip rt ξ'))"  

⟨proof⟩  

end
```

## 1.7 The quality increases predicate

```
theory A_Quality_Increases
imports A_Aodv_Predicates A_Fresher
begin

definition quality_increases :: "state ⇒ state ⇒ bool"
where "quality_increases ξ ξ' ≡ (∀ dip ∈ kD(rt ξ). dip ∈ kD(rt ξ') ∧ rt ξ ⊑_dip rt ξ')  
                           ∧ (∀ dip. sqn(rt ξ) dip ≤ sqn(rt ξ') dip)"
```

```
lemma quality_increasesI [intro!]:
  assumes "¬ dip. dip ∈ kD(rt ξ) ⇒ dip ∈ kD(rt ξ')"
  and "¬ dip. [ dip ∈ kD(rt ξ); dip ∈ kD(rt ξ') ] ⇒ rt ξ ⊑_dip rt ξ'"
  and "¬ dip. sqn(rt ξ) dip ≤ sqn(rt ξ') dip"
  shows "quality_increases ξ ξ"
⟨proof⟩
```

```
lemma quality_increasesE [elim]:
  fixes dip
  assumes "quality_increases ξ ξ'"
  and "dip ∈ kD(rt ξ)"
  and "[ dip ∈ kD(rt ξ'); rt ξ ⊑_dip rt ξ'; sqn(rt ξ) dip ≤ sqn(rt ξ') dip ] ⇒ R dip ξ ξ'"
  shows "R dip ξ ξ"
⟨proof⟩
```

```
lemma quality_increases_rt_fresherD [dest]:
  fixes ip
  assumes "quality_increases ξ ξ'"
  and "ip ∈ kD(rt ξ)"
  shows "rt ξ ⊑_ip rt ξ"
⟨proof⟩
```

```
lemma quality_increases_sqnE [elim]:
  fixes dip
  assumes "quality_increases ξ ξ'"
  and "sqn(rt ξ) dip ≤ sqn(rt ξ') dip ⇒ R dip ξ ξ'"
  shows "R dip ξ ξ"
⟨proof⟩
```

```
lemma quality_increases_refl [intro, simp]: "quality_increases ξ ξ"
⟨proof⟩
```

```
lemma strictly_fresher_quality_increases_right [elim]:
  fixes σ σ' dip
  assumes "rt(σ i) ⊑_dip rt(σ nhip)"
  and qinc: "quality_increases(σ nhip) (σ' nhip)"
  and "dip ∈ kD(rt(σ nhip))"
  shows "rt(σ i) ⊑_dip rt(σ' nhip)"
⟨proof⟩
```

```
lemma kD_quality_increases [elim]:
  assumes "i ∈ kD(rt ξ)"
  and "quality_increases ξ ξ'"
  shows "i ∈ kD(rt ξ')"
⟨proof⟩
```

$\langle proof \rangle$

```
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"  
 $\langle proof \rangle$ 
```

```
lemma nsqn_quality_increases [elim]:  
assumes "i ∈ kD(rt ξ)"  
and "quality_increases ξ ξ'"  
shows "nsqn(rt ξ) i ≤ nsqn(rt ξ') i"  
 $\langle proof \rangle$ 
```

```
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"  
 $\langle proof \rangle$ 
```

```
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"  
 $\langle proof \rangle$ 
```

```
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)"  
 $\langle proof \rangle$ 
```

```
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))"  
 $\langle proof \rangle$ 
```

```
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")  
 $\langle proof \rangle$ 
```

```
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
```

```

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]:**

" $\bigwedge \text{hops dip dsn dsk oip osn sip}$ .

```

msg_fresh σ (Rreq hops 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))))

```

" $\bigwedge \text{hops dip dsn oip sip}$ . 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)))

```

" $\bigwedge \text{dests sip}$ .

```

msg_fresh σ (Rerr dests sip) =
    (∀ripc ∈ dom(dests). (ripc ∈ kD(rt (σ sip)))
     ∧ the (dests ripc) - 1 ≤ nsqn (rt (σ sip)) ripc))"

```

" $\bigwedge \text{d dip}$ .

```
msg_fresh σ (Newpkt d dip) = True"
```

" $\bigwedge \text{d dip sip}$ .

```
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 ⇒ recvmsg rreq\_rrep\_sn m"

*(proof)*

**lemma rreq\_nsqn\_is\_fresh [simp]:**

fixes σ msg hops dip dsn dsk oip osn sip

assumes "rreq\_rrep\_fresh (rt (σ sip)) (Rreq hops dip dsn dsk oip osn sip)"

and "rreq\_rrep\_sn (Rreq hops dip dsn dsk oip osn sip)"

shows "msg\_fresh σ (Rreq hops dip dsn dsk oip osn sip)"

(is "msg\_fresh σ ?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_Oadv
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 ⇒ state) × ((state, msg, pseqp, pseqp label) seqp)) set"
where " $\sigma_{AODV}' \equiv \{(\lambda i. aodv\_init i, \Gamma_{AODV} PAodv)\}"$ 
```

```
abbreviation opaodv
  :: "ip ⇒ ((ip ⇒ state) × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
"opaodv i ≡ () 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]: " $(\sigma, p) \in \sigma_{AODV}' \implies \text{labels } \Gamma_{AODV} p = \{PAodv-:0\}$ "
  ⟨proof⟩
```

```
lemma oaodv_init_kD_empty [simp]:
  " $(\sigma, p) \in \sigma_{AODV}' \implies kD (\text{rt } (\sigma i)) = \{\}$ "
  ⟨proof⟩
```

```
lemma oaodv_init_vD_empty [simp]:
  " $(\sigma, p) \in \sigma_{AODV}' \implies vD (\text{rt } (\sigma i)) = \{\}$ "
  ⟨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_Aodv_Predicates
  A_Fresher
  A_Quality_Increases
  AWN.OAWN_Convert
  A_OAodv
begin

lemma other_quality_increases [elim]:
  assumes "other quality_increases I σ σ'"
  shows "∀ j. quality_increases (σ j) (σ' j)"
  ⟨proof⟩

lemma weaken_otherwith [elim]:
  fixes m
  assumes *: "otherwith P I (orecvmsg Q) σ σ' a"
    and weakenP: "¬¬ P σ m ==> P' σ m"
    and weakenQ: "¬¬ Q σ m ==> Q' σ m"
  shows "otherwith P' I (orecvmsg Q') σ σ' a"
  ⟨proof⟩

lemma oreceived_msg_inv:
  assumes other: "¬¬ ∃ σ σ' m. [ P σ m; other Q {i} σ σ' ] ==> P σ' m"
    and local: "¬¬ ∃ σ m. P σ m ==> P (σ(i := σ i | msg := m)) m"
  shows "opaodv i ≡_A (otherwith Q {i} (orecvmsg P), other Q {i} →)
          onl Γ_AODV (λ(σ, 1). 1 ∈ {PAodv-:1} —> P σ (msg (σ i)))"
  ⟨proof⟩

(Equivalent to) Proposition 7.27

lemma local_quality_increases:
  "opaodv i ≡_A (recvmsg rreq_rrep_sn →) onll Γ_AODV (λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ')"
  ⟨proof⟩

lemmas olocal_quality_increases =
  open_seq_stepInvariant [OF local_quality_increases initiali_aodv oaodv_trans aodv_trans,
  simplified seqll_onll_swap]

lemma oquality_increases:
  "opaodv i ≡_A (otherwith quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)),
    other quality_increases {i} →)
    onll Γ_AODV (λ((σ, _), _, (σ', _)). ∀ j. quality_increases (σ j) (σ' j))"
  (is "_ ≡_A (?S, _ →) _")
  ⟨proof⟩

lemma rreq_rrep_nsqn_fresh_any_step_invariant:
  "opaodv i ≡_A (act (recvmsg rreq_rrep_sn), other A {i} →)
    onll Γ_AODV (λ((σ, _), a, _). anycast (msg_fresh σ) a)"
  ⟨proof⟩

lemma oreceived_rreq_rrep_nsqn_fresh_inv:
  "opaodv i ≡_A (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i} →)
    onl Γ_AODV (λ(σ, 1). 1 ∈ {PAodv-:1} —> msg_fresh σ (msg (σ i)))"
  ⟨proof⟩

lemma oquality_increases_nsqn_fresh:
  "opaodv i ≡_A (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i} →)
    onll Γ_AODV (λ((σ, _), _, (σ', _)). ∀ j. quality_increases (σ j) (σ' j))"

```

$\langle proof \rangle$

**lemma oosn\_rreq:**  
 $"opaodv i \models (\text{otherwith quality\_increases } \{i\} (\text{orecvmsg msg\_fresh}),$   
 $\quad \text{other quality\_increases } \{i\} \rightarrow)$   
 $\quad \text{onl } \Gamma_{AODV} (\text{seql } i (\lambda(\xi, l). l \in \{\text{PAodv-:4}, \text{PAodv-:5}\} \cup \{\text{PRreq-:n/n. True}\} \longrightarrow 1 \leq osn \xi))"$   
 $\langle proof \rangle$

**lemma rreq\_sip:**  
 $"opaodv i \models (\text{otherwith quality\_increases } \{i\} (\text{orecvmsg msg\_fresh}),$   
 $\quad \text{other quality\_increases } \{i\} \rightarrow)$   
 $\quad \text{onl } \Gamma_{AODV} (\lambda(\sigma, l).$   
 $\quad (l \in \{\text{PAodv-:4}, \text{PAodv-:5}, \text{PRreq-:0}, \text{PRreq-:2}\} \wedge \text{sip } (\sigma i) \neq \text{oip } (\sigma i))$   
 $\quad \longrightarrow \text{oip } (\sigma i) \in kD(rt (\sigma (\text{sip } (\sigma i))))$   
 $\quad \wedge \text{nsqn } (rt (\sigma (\text{sip } (\sigma i)))) (\text{oip } (\sigma i)) \geq \text{osn } (\sigma i)$   
 $\quad \wedge (\text{nsqn } (rt (\sigma (\text{sip } (\sigma i)))) (\text{oip } (\sigma i)) = \text{osn } (\sigma i))$   
 $\quad \longrightarrow (\text{hops } (\sigma i) \geq \text{the } (\text{dhops } (rt (\sigma (\text{sip } (\sigma i)))) (\text{oip } (\sigma i)))$   
 $\quad \vee \text{the } (\text{flag } (rt (\sigma (\text{sip } (\sigma i)))) (\text{oip } (\sigma i))) = \text{inv}))"$   
 $\quad (\text{is } \_ \models (?S, ?U \rightarrow) \_)$   
 $\langle proof \rangle$

**lemma odsn\_rrep:**  
 $"opaodv i \models (\text{otherwith quality\_increases } \{i\} (\text{orecvmsg msg\_fresh}),$   
 $\quad \text{other quality\_increases } \{i\} \rightarrow)$   
 $\quad \text{onl } \Gamma_{AODV} (\text{seql } i (\lambda(\xi, l). l \in \{\text{PAodv-:6}, \text{PAodv-:7}\} \cup \{\text{PRrep-:n/n. True}\} \longrightarrow 1 \leq dsn \xi))"$   
 $\langle proof \rangle$

**lemma rrep\_sip:**  
 $"opaodv i \models (\text{otherwith quality\_increases } \{i\} (\text{orecvmsg msg\_fresh}),$   
 $\quad \text{other quality\_increases } \{i\} \rightarrow)$   
 $\quad \text{onl } \Gamma_{AODV} (\lambda(\sigma, l).$   
 $\quad (l \in \{\text{PAodv-:6}, \text{PAodv-:7}, \text{PRrep-:0}, \text{PRrep-:1}\} \wedge \text{sip } (\sigma i) \neq \text{dip } (\sigma i))$   
 $\quad \longrightarrow \text{dip } (\sigma i) \in kD(rt (\sigma (\text{sip } (\sigma i))))$   
 $\quad \wedge \text{nsqn } (rt (\sigma (\text{sip } (\sigma i)))) (\text{dip } (\sigma i)) \geq \text{dsn } (\sigma i)$   
 $\quad \wedge (\text{nsqn } (rt (\sigma (\text{sip } (\sigma i)))) (\text{dip } (\sigma i)) = \text{dsn } (\sigma i))$   
 $\quad \longrightarrow (\text{hops } (\sigma i) \geq \text{the } (\text{dhops } (rt (\sigma (\text{sip } (\sigma i)))) (\text{dip } (\sigma i)))$   
 $\quad \vee \text{the } (\text{flag } (rt (\sigma (\text{sip } (\sigma i)))) (\text{dip } (\sigma i))) = \text{inv}))"$   
 $\quad (\text{is } \_ \models (?S, ?U \rightarrow) \_)$   
 $\langle proof \rangle$

**lemma rerr\_sip:**  
 $"opaodv i \models (\text{otherwith quality\_increases } \{i\} (\text{orecvmsg msg\_fresh}),$   
 $\quad \text{other quality\_increases } \{i\} \rightarrow)$   
 $\quad \text{onl } \Gamma_{AODV} (\lambda(\sigma, l).$   
 $\quad (l \in \{\text{PAodv-:8}, \text{PAodv-:9}, \text{PRerr-:0}, \text{PRerr-:1}\}$   
 $\quad \longrightarrow (\forall \text{rip} \in \text{dom}(\text{dests } (\sigma i)). \text{rip} \in kD(rt (\sigma (\text{sip } (\sigma i)))) \wedge$   
 $\quad \quad \text{the } (\text{dests } (\sigma i) \text{ rip}) - 1 \leq \text{nsqn } (rt (\sigma (\text{sip } (\sigma i))) \text{ rip}))"$   
 $\quad (\text{is } \_ \models (?S, ?U \rightarrow) \_)$   
 $\langle proof \rangle$

**lemma prerr\_guard:** "paodv i  $\models$   
 $\quad \text{onl } \Gamma_{AODV} (\lambda(\xi, l). (l = \text{PRerr-:1}$   
 $\quad \longrightarrow (\forall \text{ip} \in \text{dom}(\text{dests } \xi). \text{ip} \in vD(rt \xi)$   
 $\quad \quad \wedge \text{the } (\text{nhop } (rt \xi) \text{ ip}) = \text{sip } \xi$   
 $\quad \quad \wedge \text{sqn } (rt \xi) \text{ ip} < \text{the } (\text{dests } \xi \text{ ip})))"$   
 $\langle proof \rangle$

**lemmas oaddpreRT\_welldefined =**  
 $\quad \text{open\_seq\_invariant [OF addpreRT\_welldefined initiali\_aodv oaodv\_trans aodv\_trans,}$   
 $\quad \quad \text{simplified seql\_onl\_swap,}$   
 $\quad \quad \text{THEN oinvariant\_anyact]}$

**lemmas odests\_vD\_inc\_sqn =**  
 $\quad \text{open\_seq\_invariant [OF dests\_vD\_inc\_sqn initiali\_aodv oaodv\_trans aodv\_trans,}$

```

    simplified seql_onl_swap,
THEN oinvariant_anyact]

lemmas oprerr_guard =
open_seq_invariant [OF prerr_guard initiali_aodv oaodv_trans aodv_trans,
simplified seql_onl_swap,
THEN oinvariant_anyact]

```

Proposition 7.28

```

lemma seq_compare_next_hop':
"opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} →) onl ΓAODV (λ(σ, _).
  ∀ 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)"
(is "_ ⊨ (?S, ?U →) _")
⟨proof⟩

```

Proposition 7.30

```

lemmas okD_unk_or_atleast_one =
open_seq_invariant [OF kD_unk_or_atleast_one initiali_aodv,
simplified seql_onl_swap]

```

```

lemmas ozero_seq_unk_hops_one =
open_seq_invariant [OF zero_seq_unk_hops_one initiali_aodv,
simplified seql_onl_swap]

```

```

lemma oreachable_fresh_okD_unk_or_atleast_one:
fixes dip
assumes "(σ, p) ∈ oreachable (opaodv i)
          (otherwith ((=)) {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 (opaodv i)
          (otherwith ((=)) {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 "opaodv i ⊨ (otherwith ((=)) {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 "opaodv i ⊨ (otherwith ((=)) {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 "opaodv i ⊨ (otherwith ((=)) {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

```

## 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 ⇒ state) ⇒ ip ⇒ ip rel"
```

**where**

```
"rt_graph σ = (λ dip.
  { (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)"

```

$\langle proof \rangle$

```
theorem inv_to_loop_freedom:
assumes "\forall i dip. let nhop = the (nhop (rt (\sigma i)) dip)
           in dip \in vD (rt (\sigma i)) \cap vD (rt (\sigma nhop)) \wedge nhop \neq dip
              \longrightarrow (rt (\sigma i)) \sqsubseteq_{dip} (rt (\sigma nhop))"
shows "\forall dip. irrefl ((rt_graph \sigma 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 \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 ((=)) \{i\}) (orecvmsg (\lambda\sigma m.
msg_fresh \sigma m \wedge msg_zhops m)),
other quality_increases \{i\} \rightarrow)
global (\lambda\sigma. \forall dip. let nhop = the (nhop (rt (\sigma i)) dip)
           in dip \in vD (rt (\sigma i)) \cap vD (rt (\sigma nhop)) \wedge nhop \neq dip
              \longrightarrow (rt (\sigma i)) \sqsubseteq_{dip} (rt (\sigma nhop)))"
⟨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⟩
```

### 1.11.1 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 : opaodv i \langle\langle i qmsg : R \rangle_o \models
          (otherwith ((=)) \{i\})
          (oarrivemsg (\lambda\sigma m. msg_fresh \sigma m \wedge 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 : opaodv 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 : opaodv 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 : opaodv 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 : opaodv i ⟨⟨i qmsg : R_i⟩⟩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. opaodv i ⟨⟨i qmsg⟩⟩ p ⊨
               (otherwith ((=)) (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. opaodv 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 aodv_openproc: openproc paodv opaodv id
  rewrites "aodv_openproc.initmissing = initmissing"

```

*(proof)*

```

interpretation aodv_openproc_par_qmsg: openproc_parq paodv opaodv id qmsg
  rewrites "aodv_openproc_par_qmsg.netglobal = netglobal"
    and "aodv_openproc_par_qmsg.initmissing = initmissing"
  ⟨proof⟩

lemma net_nhop_quality_increases:
  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paodv i ⟨⟨ qmsg ⟩⟩ n) ⊨= netglobal
    (λσ. ∀i dip. let nhop = the (nhop (rt (σ i)) dip)
      in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhop)) ∧ nhop ≠ dip
      → (rt (σ i)) ⊏dip (rt (σ nhop)))"
    (is "_ ⊨= netglobal (λσ. ∀i. ?inv σ i)")
  ⟨proof⟩

```

### 1.11.5 Loop freedom of AODV

```

theorem aodv_loop_freedom:
  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paodv 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 ⇒ sqn"
  where "inc sn ≡ if sn = 0 then sn else sn + 1"

lemma less_than_inc [simp]: "x ≤ inc x"
  ⟨proof⟩

lemma inc_minus_suc_0 [simp]:
  "inc x - Suc 0 = x"
  ⟨proof⟩

lemma inc_never_one' [simp, intro]: "inc x ≠ Suc 0"
  ⟨proof⟩

lemma inc_never_one [simp, intro]: "inc x ≠ 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 × k × f × nat × ip × ip set"
definition proj2 :: "r ⇒ sqn" (⟨π₂⟩)
```

```

where " $\pi_2 \equiv \lambda(dsn, \_, \_, \_, \_, \_, \_). dsn$ "  

definition proj3 :: "r  $\Rightarrow$  k" ( $\langle\pi_3\rangle$ )
  where " $\pi_3 \equiv \lambda(\_, dsk, \_, \_, \_, \_, \_). dsk$ "  

definition proj4 :: "r  $\Rightarrow$  f" ( $\langle\pi_4\rangle$ )
  where " $\pi_4 \equiv \lambda(\_, \_, flag, \_, \_, \_, \_). flag$ "  

definition proj5 :: "r  $\Rightarrow$  nat" ( $\langle\pi_5\rangle$ )
  where " $\pi_5 \equiv \lambda(\_, \_, \_, hops, \_, \_, \_). hops$ "  

definition proj6 :: "r  $\Rightarrow$  ip" ( $\langle\pi_6\rangle$ )
  where " $\pi_6 \equiv \lambda(\_, \_, \_, \_, nhip, \_). nhip$ "  

definition proj7 :: "r  $\Rightarrow$  ip set" ( $\langle\pi_7\rangle$ )
  where " $\pi_7 \equiv \lambda(\_, \_, \_, \_, \_, pre). pre$ "  

lemma projs [simp]:
  " $\pi_2(dsn, dsk, flag, hops, nhip, pre) = dsn$ "
  " $\pi_3(dsn, dsk, flag, hops, nhip, pre) = dsk$ "
  " $\pi_4(dsn, dsk, flag, hops, nhip, pre) = flag$ "
  " $\pi_5(dsn, dsk, flag, hops, nhip, pre) = hops$ "
  " $\pi_6(dsn, dsk, flag, hops, nhip, pre) = nhip$ "
  " $\pi_7(dsn, dsk, flag, hops, nhip, pre) = pre$ "  

   $\langle proof \rangle$   

lemma proj3_pred [intro]: " $\llbracket P \text{ kno}; P \text{ unk} \rrbracket \implies P (\pi_3 x)$ "  

   $\langle proof \rangle$   

lemma proj4_pred [intro]: " $\llbracket P \text{ val}; P \text{ inv} \rrbracket \implies P (\pi_4 x)$ "  

   $\langle proof \rangle$   

lemma proj6_pair_snd [simp]:
  fixes dsn' r
  shows " $\pi_6(dsn', snd(r)) = \pi_6(r)$ "  

   $\langle proof \rangle$ 

```

### 2.1.3 Routing Tables

Routing tables map ip addresses to route entries.

```

type_synonym rt = "ip  $\rightarrow$  r"  

syntax
  " $\_Sigma\_route$ " :: "rt  $\Rightarrow$  ip  $\rightarrow$  r" ( $\langle\sigma_{route}(\_, \_) \rangle$ )  

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  $\pi_6(\sigma_{route}(rt, dip))$ "  

abbreviation precs :: "rt  $\Rightarrow$  ip  $\rightarrow$  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⟩

```

## 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: " $\text{addpre } r \{ \} = r$ "
⟨proof⟩

lemma addpre_r:
  "addpre (dsn, dsk, f1, hops, nhip, pre) npre = (dsn, dsk, f1, 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 ⇒ ip ⇒ ip set → rt"
  where "addpreRT rt dip npre ≡
    map_option (λs. rt (dip ↪ addpre s npre)) (σ_route(rt, dip))"

lemma snd_addpre [simp]:
  " $\bigwedge dsn dsn' v pre. (dsn, \text{snd}(\text{addpre } (dsn', v) \text{ pre})) = \text{addpre } (dsn, v) \text{ pre}$ "
  ⟨proof⟩

lemma proj2_addpreRT [simp]:
  fixes ip rt ip' npre
  assumes "ip ∈ kD rt"
    and "ip' ∈ kD rt"
  shows " $\pi_2(\text{the } (\text{the } (\text{addpreRT } rt \text{ ip'} \text{ npre}) \text{ ip})) = \pi_2(\text{the } (rt \text{ ip}))$ "
  ⟨proof⟩

lemma proj3_addpreRT [simp]:
  fixes ip rt ip' npre
  assumes "ip ∈ kD rt"
    and "ip' ∈ kD rt"
  shows " $\pi_3(\text{the } (\text{the } (\text{addpreRT } rt \text{ ip'} \text{ npre}) \text{ ip})) = \pi_3(\text{the } (rt \text{ ip}))$ "
  ⟨proof⟩

lemma proj5_addpreRT [simp]:
  " $\bigwedge rt dip ip npre. dip ∈ kD(rt) \implies \pi_5(\text{the } (\text{the } (\text{addpreRT } rt \text{ dip npre}) \text{ ip})) = \pi_5(\text{the } (rt \text{ ip}))$ "
  ⟨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]:
  " $\bigwedge_{ip \in kD(rt)} ip \in kD(rt) \Rightarrow sqnf (the (addpreRT (rt \xi) ip npre)) dip = sqnf (rt \xi) dip$ "
  ⟨proof⟩

```

## Updating route entries

```

lemma in_kD_case [simp]:
  fixes dip rt
  assumes "dip ∈ kD(rt)"
  shows "(case rt dip of None ⇒ 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 ⟧"

```

$\implies \text{update } rt \ ip \ r = rt \ (\text{ip} \mapsto ns)$

$\langle proof \rangle$

```

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(\text{the } \sigma_{\text{route}}(rt, ip))))))"
  and c3: "[ $ip \in kD(rt); sqn rt ip = \pi_2(r); \text{the } (dhops rt ip) > \pi_5(r)$ ]
    \implies P (rt (ip \mapsto addpre r (\pi_7(\text{the } \sigma_{\text{route}}(rt, ip))))))"
  and c4: "[ $ip \in kD(rt); sqn rt ip = \pi_2(r); \text{the } (flag rt ip) = inv$ ]
    \implies P (rt (ip \mapsto addpre r (\pi_7(\text{the } \sigma_{\text{route}}(rt, ip))))))"
  and c5: "[ $ip \in kD(rt); \pi_3(r) = unk$ ]
    \implies P (rt (ip \mapsto (\pi_2(\text{the } \sigma_{\text{route}}(rt, ip)), \pi_3(r),
      \pi_4(r), \pi_5(r), \pi_6(r), \pi_7(addpre r (\pi_7(\text{the } \sigma_{\text{route}}(rt, ip)))))))"
  and c6: "[ $ip \in kD(rt); sqn rt ip \geq \pi_2(r); \pi_3(r) = kno;$ 
     $sqn rt ip = \pi_2(r) \implies \text{the } (dhops rt ip) \leq \pi_5(r) \wedge \text{the } (flag rt ip) = val$ ]
    \implies P (rt (ip \mapsto addpre (\text{the } \sigma_{\text{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(\text{the } \sigma_{\text{route}}(rt, ip))))))"
  and c3: "[ $sqn rt ip = \pi_2(r); \text{the } (dhops rt ip) > \pi_5(r)$ ]
    \implies P (rt (ip \mapsto addpre r (\pi_7(\text{the } \sigma_{\text{route}}(rt, ip))))))"
  and c4: "[ $sqn rt ip = \pi_2(r); \text{the } (flag rt ip) = inv$ ]
    \implies P (rt (ip \mapsto addpre r (\pi_7(\text{the } \sigma_{\text{route}}(rt, ip))))))"
  and c5: "[ $\pi_3(r) = unk \implies P (rt (ip \mapsto (\pi_2(\text{the } \sigma_{\text{route}}(rt, ip)), \pi_3(r),
    \pi_4(r), \pi_5(r), \pi_6(r),
    \pi_7(addpre r (\pi_7(\text{the } \sigma_{\text{route}}(rt, ip)))))))$ ]"
  and c6: "[ $sqn rt ip \geq \pi_2(r); \pi_3(r) = kno;$ 
     $sqn rt ip = \pi_2(r) \implies \text{the } (dhops rt ip) \leq \pi_5(r) \wedge \text{the } (flag rt ip) = val$ ]
    \implies P (rt (ip \mapsto addpre (\text{the } \sigma_{\text{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. if x = rip then Some v 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 ≤ 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]: " $\lambda rt\ dip\ dsn\ dsk\ flg\ hops\ sip.$ 
   $rt \neq update\ rt\ dip\ (dsn,\ dsk,\ flg,\ hops,\ sip,\ {})$ 
   $\implies sqnf\ (update\ rt\ dip\ (dsn,\ dsk,\ flg,\ hops,\ sip,\ {}))\ dip = dsk$ "
  ⟨proof⟩

lemma update_kno_dsn_greater_zero:
  " $\lambda rt\ dip\ ip\ dsn\ hops\ npre.$   $1 \leq dsn \implies 1 \leq (sqn\ (update\ rt\ dip\ (dsn,\ kno,\ val,\ hops,\ ip,\ npre))\ dip)$ "
  ⟨proof⟩

lemma proj3_update [simp]: " $\lambda rt\ dip\ dsn\ dsk\ flg\ hops\ sip.$ 
   $rt \neq update\ rt\ dip\ (dsn,\ dsk,\ flg,\ hops,\ sip,\ {})$ 
   $\implies \pi_3(\text{the}\ (\text{update}\ rt\ dip\ (dsn,\ dsk,\ flg,\ hops,\ sip,\ {})\ dip)) = dsk$ "
  ⟨proof⟩

lemma nhop_update_changed_kno_val [simp]: " $\lambda rt\ ip\ dsn\ dsk\ hops\ nhip.$ 
   $rt \neq update\ rt\ ip\ (dsn,\ kno,\ val,\ hops,\ nhip,\ {})$ 
   $\implies \text{the}\ (\text{nhop}\ (\text{update}\ rt\ ip\ (dsn,\ kno,\ val,\ hops,\ nhip,\ {}))\ ip) = nhip$ "
  ⟨proof⟩

lemma flag_update [simp]: " $\lambda rt\ dip\ dsn\ flg\ hops\ sip.$ 
   $rt \neq update\ rt\ dip\ (dsn,\ kno,\ flg,\ hops,\ sip,\ {})$ 
   $\implies \text{the}\ (\text{flag}\ (\text{update}\ rt\ dip\ (dsn,\ kno,\ flg,\ hops,\ sip,\ {}))\ dip) = flg$ "
  ⟨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]: " $\lambda rt\ dip\ dsn\ dsk\ flg\ hops\ sip.$ 
   $rt \neq update\ rt\ dip\ (dsn,\ dsk,\ flg,\ hops,\ sip,\ {})$ 
   $\implies \text{the}\ (\text{nhop}\ (\text{update}\ rt\ dip\ (dsn,\ dsk,\ flg,\ hops,\ sip,\ {}))\ dip) = sip$ "
  ⟨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]:
  " $\lambda dip\ rt\ dip'\ dsn\ dsk\ hops\ nhip\ pre.$ 
   $dip \in vD(\text{update}\ rt\ dip'\ (dsn,\ dsk,\ val,\ hops,\ nhip,\ pre)) \implies (dip \in vD(rt) \vee dip = dip')$ "
  ⟨proof⟩

```

## Invalidating route entries

```

definition invalidate :: "rt ⇒ (ip → sqn) ⇒ rt"
where "invalidate rt dests ≡
  λip. case (rt ip, dests ip) of

```

```

    (None, _) ⇒ None
  / (Some s, None) ⇒ Some s
  / (Some (_, dsk, _, hops, nhip, pre), Some rsn) ⇒
    Some (rsn, dsk, inv, hops, nhip, 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"

```

$\langle proof \rangle$

```
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 ⇒ 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))
    ∧ pre = π7(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⟩
```

## 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 " $\sigma_{p\text{-}flag}(store, dip) \equiv \text{map\_option } \text{fst } (store \ dip)$ "
```

```

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.  $\sigma_{p\text{-}flag}(store, dip)$  is partial.
4. The routing table (*rt*) is modelled as a map ( $ip \Rightarrow r \text{ option}$ ) 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 "aodv_init i ≡ (
  ip = i,
  sn = 1,
  rt = Map.empty,
  rreqs = {},
  store = Map.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]:
  " $\xi' \in is\_rerr \xi \implies rreqs \xi' = rreqs \xi$ "
  " $\xi' \in is\_rrep \xi \implies rreqs \xi' = rreqs \xi$ "
  " $\xi' \in is\_rreq \xi \implies rreqs \xi' = rreqs \xi$ "
  " $\xi' \in is\_pkt \xi \implies rreqs \xi' = rreqs \xi$ "
  " $\xi' \in is\_newpkt \xi \implies rreqs \xi' = rreqs \xi$ "
⟨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⟩
```

### 2.3.3 The protocol process

```
datatype pseqp =
  PAodv
  | PNewPkt
  | PPkt
  | PRreq
  | PRrep
  | PRerr

fun nat_of_seqp :: "pseqp ⇒ nat"
where
  "nat_of_seqp PAodv = 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]: "11 ≤ 12 = (nat_of_seqp 11 ≤ nat_of_seqp 12)"
definition less_seqp [iff]: "11 < 12 = (nat_of_seqp 11 < nat_of_seqp 12)"
instance ⟨proof⟩
end

abbreviation AODV
where
  "AODV ≡ λ_. [clear_locals] call(PAodv)"

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  $\xi$ ) () data := data, dip := dip )]
call(PNewPkt)"
```

abbreviation *RREQ*

where

```
"RREQ args ≡
[ξ. let (hops, rreqid, dip, dsn, dsk, oip, osn, sip) = args  $\xi$  in
(clear_locals  $\xi$ ) () 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  $\xi$  in
(clear_locals  $\xi$ ) () hops := hops, dip := dip, dsn := dsn,
oip := oip, sip := sip )]
call(PRrep)"
```

abbreviation *RERR*

where

```
"RERR args ≡
[ξ. let (dests, sip) = args  $\xi$  in
(clear_locals  $\xi$ ) () dests := dests, sip := sip )]
call(PRerr)"
```

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

where

```
" $\Gamma_{AODV}$  PAodv = labelled PAodv (
receive( $\lambda msg'$   $\xi$ .  $\xi$  () msg := msg').
( ⟨is_newpkt⟩ NEWPKT( $\lambda \xi$ . (data  $\xi$ , ip  $\xi$ ))
⊕ ⟨is_pkt⟩ PKT( $\lambda \xi$ . (data  $\xi$ , dip  $\xi$ , oip  $\xi$ ))
⊕ ⟨is_rreq⟩
[ $\xi$ .  $\xi$  (rt := update (rt  $\xi$ ) (sip  $\xi$ ) (0, unk, val, 1, sip  $\xi$ , {} ) )
RREQ( $\lambda \xi$ . (hops  $\xi$ , rreqid  $\xi$ , dip  $\xi$ , dsn  $\xi$ , dsk  $\xi$ , oip  $\xi$ , osn  $\xi$ , sip  $\xi$ ))
⊕ ⟨is_rrep⟩
[ $\xi$ .  $\xi$  (rt := update (rt  $\xi$ ) (sip  $\xi$ ) (0, unk, val, 1, sip  $\xi$ , {} ) )
RREP( $\lambda \xi$ . (hops  $\xi$ , dip  $\xi$ , dsn  $\xi$ , oip  $\xi$ , sip  $\xi$ ))
⊕ ⟨is_rerr⟩
[ $\xi$ .  $\xi$  (rt := update (rt  $\xi$ ) (sip  $\xi$ ) (0, unk, val, 1, sip  $\xi$ , {} ) )
RERR( $\lambda \xi$ . (dests  $\xi$ , sip  $\xi$ ))
)
⊕ ⟨ $\lambda \xi$ . {  $\xi$  | dip := dip } | dip. dip ∈ qD(store  $\xi$ ) ∩ vD(rt  $\xi$  ) }
[ $\xi$ .  $\xi$  () data := hd( $\sigma_{queue}(store \xi, dip \xi)$  )]
unicast( $\lambda \xi$ . the (nhop (rt  $\xi$ ) (dip  $\xi$ )),  $\lambda \xi$ . pkt(data  $\xi$ , dip  $\xi$ , ip  $\xi$ )).
[ $\xi$ .  $\xi$  () store := the (drop (dip  $\xi$ ) (store  $\xi$ )) ]
AODV()
▷ [ $\xi$ .  $\xi$  () dests := ( $\lambda rip$ . if (rip ∈ vD(rt  $\xi$ ) ∧ nhop(rt  $\xi$ ) rip = nhop(rt  $\xi$ ) (dip  $\xi$ ))
then Some (inc (sqn(rt  $\xi$ ) rip)) else None) ]
[ $\xi$ .  $\xi$  () rt := invalidate(rt  $\xi$ ) (dests  $\xi$  ) ]
[ $\xi$ .  $\xi$  () store := setRRF(store  $\xi$ ) (dests  $\xi$  ) ]
[ $\xi$ .  $\xi$  () pre :=  $\bigcup\{$  the (precs(rt  $\xi$ ) rip) | rip. rip ∈ dom(dests  $\xi$ ) } ]
[ $\xi$ .  $\xi$  () dests := ( $\lambda rip$ . if ((dests  $\xi$ ) rip ≠ None ∧ the (precs(rt  $\xi$ ) rip) ≠ {} )
then (dests  $\xi$ ) rip else None) ]
groupcast( $\lambda \xi$ . pre  $\xi$ ,  $\lambda \xi$ . rerr(dests  $\xi$ , ip  $\xi$ )). AODV()
⊕ ⟨ $\lambda \xi$ . {  $\xi$  | dip := dip }
| dip. dip ∈ qD(store  $\xi$ ) - vD(rt  $\xi$ ) ∧ the ( $\sigma_{p\text{-flag}}(store \xi, dip)$ ) = req }
[ $\xi$ .  $\xi$  () store := unsetRRF(store  $\xi$ ) (dip  $\xi$  ) ]
[ $\xi$ .  $\xi$  () sn := inc(sn  $\xi$  ) ]
[ $\xi$ .  $\xi$  () rreqid := nrreqid(rreqs  $\xi$ ) (ip  $\xi$  ) ]
[ $\xi$ .  $\xi$  () rreqs := rreqs  $\xi$  ∪ {(ip  $\xi$ , rreqid  $\xi$ )} ]
broadcast( $\lambda \xi$ . rreq(0, rreqid  $\xi$ , dip  $\xi$ , sqn(rt  $\xi$ ) (dip  $\xi$ ), sqnf(rt  $\xi$ ) (dip  $\xi$ ),
ip  $\xi$ , sn  $\xi$ , ip  $\xi$ )). AODV())"
```

```

| " $\Gamma_{AODV}$   $PNewPkt$  = labelled  $PNewPkt$  (
   $\langle \xi. dip \xi = ip \xi \rangle$ 
     $deliver(\lambda \xi. data \xi).AODV()$ 
   $\oplus \langle \xi. dip \xi \neq ip \xi \rangle$ 
     $\llbracket \xi. \xi \mid store := add(data \xi) (dip \xi) (store \xi) \rrbracket$ 
     $AODV()$ )
)

| " $\Gamma_{AODV}$   $PPkt$  = labelled  $PPkt$  (
   $\langle \xi. dip \xi = ip \xi \rangle$ 
     $deliver(\lambda \xi. data \xi).AODV()$ 
   $\oplus \langle \xi. dip \xi \neq ip \xi \rangle$ 
  (
     $\langle \xi. dip \xi \in vD(rt \xi) \rangle$ 
       $unicast(\lambda \xi. the(nhop(rt \xi) (dip \xi)), \lambda \xi. pkt(data \xi, dip \xi, oip \xi)).AODV()$ 
     $\triangleright$ 
       $\llbracket \xi. \xi \mid dests := (\lambda rip. if(rip \in vD(rt \xi) \wedge nhop(rt \xi) rip = nhop(rt \xi) (dip \xi))$ 
         $then Some(inc(sqn(rt \xi) rip)) else None) \rrbracket$ 
       $\llbracket \xi. \xi \mid rt := invalidate(rt \xi) (dests \xi) \rrbracket$ 
       $\llbracket \xi. \xi \mid store := setRRF(store \xi) (dests \xi) \rrbracket$ 
       $\llbracket \xi. \xi \mid pre := \bigcup \{the(precs(rt \xi) rip) \mid rip \in \text{dom}(dests \xi)\} \rrbracket$ 
       $\llbracket \xi. \xi \mid dests := (\lambda rip. if((dests \xi) rip \neq None \wedge the(precs(rt \xi) rip) \neq \{\}))$ 
         $then (dests \xi) rip else None) \rrbracket$ 
       $groupcast(\lambda \xi. pre \xi, \lambda \xi. rerr(dests \xi, ip \xi)).AODV()$ 
     $\oplus \langle \xi. dip \xi \notin vD(rt \xi) \rangle$ 
    (
       $\langle \xi. dip \xi \in iD(rt \xi) \rangle$ 
         $groupcast(\lambda \xi. the(precs(rt \xi) (dip \xi)),$ 
           $\lambda \xi. rerr([dip \xi \mapsto sqn(rt \xi) (dip \xi)], ip \xi)).AODV()$ 
       $\oplus \langle \xi. dip \xi \notin iD(rt \xi) \rangle$ 
         $AODV()$ 
    )
  )
))

| " $\Gamma_{AODV}$   $PRreq$  = labelled  $PRreq$  (
   $\langle \xi. (oip \xi, rreqid \xi) \in rreqs \xi \rangle$ 
     $AODV()$ 
   $\oplus \langle \xi. (oip \xi, rreqid \xi) \notin rreqs \xi \rangle$ 
  (
     $\llbracket \xi. \xi \mid rt := update(rt \xi) (oip \xi) (osn \xi, kno, val, hops \xi + 1, sip \xi, \{\}) \rrbracket$ 
     $\llbracket \xi. \xi \mid rreqs := rreqs \xi \cup \{(oip \xi, rreqid \xi)\} \rrbracket$ 
    (
       $\langle \xi. dip \xi = ip \xi \rangle$ 
         $\llbracket \xi. \xi \mid sn := max(sn \xi) (dsn \xi) \rrbracket$ 
         $unicast(\lambda \xi. the(nhop(rt \xi) (oip \xi)), \lambda \xi. rrep(0, dip \xi, sn \xi, oip \xi, ip \xi)).AODV()$ 
       $\triangleright$ 
         $\llbracket \xi. \xi \mid dests := (\lambda rip. if(rip \in vD(rt \xi) \wedge nhop(rt \xi) rip = nhop(rt \xi) (oip \xi))$ 
           $then Some(inc(sqn(rt \xi) rip)) else None) \rrbracket$ 
         $\llbracket \xi. \xi \mid rt := invalidate(rt \xi) (dests \xi) \rrbracket$ 
         $\llbracket \xi. \xi \mid store := setRRF(store \xi) (dests \xi) \rrbracket$ 
         $\llbracket \xi. \xi \mid pre := \bigcup \{the(precs(rt \xi) rip) \mid rip \in \text{dom}(dests \xi)\} \rrbracket$ 
         $\llbracket \xi. \xi \mid dests := (\lambda rip. if((dests \xi) rip \neq None \wedge the(precs(rt \xi) rip) \neq \{\}))$ 
           $then (dests \xi) rip else None) \rrbracket$ 
         $groupcast(\lambda \xi. pre \xi, \lambda \xi. rerr(dests \xi, ip \xi)).AODV()$ 
       $\oplus \langle \xi. dip \xi \neq ip \xi \rangle$ 
      (
         $\langle \xi. dip \xi \in vD(rt \xi) \wedge dsn \xi \leq sqn(rt \xi) (dip \xi) \wedge sqnf(rt \xi) (dip \xi) = kno \rangle$ 
         $\llbracket \xi. \xi \mid rt := the(addpreRT(rt \xi) (dip \xi) \{sip \xi\}) \rrbracket$ 
         $\llbracket \xi. \xi \mid rt := the(addpreRT(rt \xi) (oip \xi) \{the(nhop(rt \xi) (dip \xi))\}) \rrbracket$ 
         $unicast(\lambda \xi. the(nhop(rt \xi) (oip \xi)), \lambda \xi. rrep(the(dhops(rt \xi) (dip \xi)), dip \xi,$ 
           $sqn(rt \xi) (dip \xi), oip \xi, ip \xi)).$ 
         $AODV()$ 
      )
     $\triangleright$ 
       $\llbracket \xi. \xi \mid dests := (\lambda rip. if(rip \in vD(rt \xi) \wedge nhop(rt \xi) rip = nhop(rt \xi) (oip \xi))$ 
         $then Some(inc(sqn(rt \xi) rip)) else None) \rrbracket$ 
    )
  )
)

```

```

 $\xi. \xi () rt := invalidate(rt \xi) (dests \xi) []$ 
 $\xi. \xi () store := setRRF(store \xi) (dests \xi) []$ 
 $\xi. \xi () pre := \bigcup \{ the(precs(rt \xi) rip) \mid rip. rip \in \text{dom}(dests \xi) \} []$ 
 $\xi. \xi () dests := (\lambda rip. \text{if } ((\text{dests } \xi) rip \neq \text{None} \wedge \text{the}(\text{prec}(rt \xi) rip) \neq \{\})$ 
 $\quad \text{then } (\text{dests } \xi) rip \text{ else } \text{None}) [])$ 
 $\quad \text{groupcast}(\lambda \xi. \text{pre } \xi, \lambda \xi. \text{rerr}(\text{dests } \xi, ip \xi)). AODV()$ 
 $\oplus \langle \xi. dip \xi \notin vD(rt \xi) \vee sqn(rt \xi) (dip \xi) < dsn \xi \vee sqnf(rt \xi) (dip \xi) = unk \rangle$ 
 $\quad \text{broadcast}(\lambda \xi. \text{rreq}(hops \xi + 1, rreqid \xi, dip \xi, max(sqn(rt \xi) (dip \xi)) (dsn \xi),$ 
 $\quad \quad \quad dsk \xi, oip \xi, osn \xi, ip \xi)).$ 
 $AODV()$ 
 $)$ 
 $)")$ 

| " $\Gamma_{AODV} PRrep = \text{labelled } PRrep$  ("
 $\xi. \xi () rt := update(rt \xi) (dip \xi) (dsn \xi, kno, val, hops \xi + 1, sip \xi, \{\}) []$ 
 $($ 
 $\quad \langle \xi. oip \xi = ip \xi \rangle$ 
 $\quad AODV()$ 
 $\oplus \langle \xi. oip \xi \neq ip \xi \rangle$ 
 $($ 
 $\quad \langle \xi. oip \xi \in vD(rt \xi) \wedge dip \xi \in vD(rt \xi) \rangle$ 
 $\quad \xi. \xi () rt := \text{the}(\text{addpreRT}(rt \xi) (dip \xi)$ 
 $\quad \quad \quad \{ \text{the}(nhop(rt \xi) (oip \xi)) \} []$ 
 $\quad \xi. \xi () rt := \text{the}(\text{addpreRT}(rt \xi) (\text{the}(nhop(rt \xi) (dip \xi))) \{ \text{the}(nhop(rt \xi) (oip \xi)) \}$ 
 $)$ 
 $\quad \text{unicast}(\lambda \xi. \text{the}(nhop(rt \xi) (oip \xi)), \lambda \xi. \text{rrep}(\text{the}(dhops(rt \xi) (dip \xi)), dip \xi,$ 
 $\quad \quad \quad sqn(rt \xi) (dip \xi), oip \xi, ip \xi)).$ 
 $\quad AODV()$ 
 $\triangleright$ 
 $\quad \xi. \xi () dests := (\lambda rip. \text{if } (rip \in vD(rt \xi) \wedge nhop(rt \xi) rip = nhop(rt \xi) (oip \xi))$ 
 $\quad \quad \quad \text{then Some}(\text{inc}(sqn(rt \xi) rip)) \text{ else None}) []$ 
 $\quad \xi. \xi () rt := invalidate(rt \xi) (dests \xi) []$ 
 $\quad \xi. \xi () store := setRRF(store \xi) (dests \xi) []$ 
 $\quad \xi. \xi () pre := \bigcup \{ the(precs(rt \xi) rip) \mid rip. rip \in \text{dom}(dests \xi) \} []$ 
 $\quad \xi. \xi () dests := (\lambda rip. \text{if } ((\text{dests } \xi) rip \neq \text{None} \wedge \text{the}(\text{prec}(rt \xi) rip) \neq \{\})$ 
 $\quad \quad \quad \text{then } (\text{dests } \xi) rip \text{ else } \text{None}) []$ 
 $\quad \quad \quad \text{groupcast}(\lambda \xi. \text{pre } \xi, \lambda \xi. \text{rerr}(\text{dests } \xi, ip \xi)). AODV()$ 
 $\oplus \langle \xi. oip \xi \notin vD(rt \xi) \vee dip \xi \notin vD(rt \xi) \rangle$ 
 $\quad AODV()$ 
 $)$ 
 $)")$ 

| " $\Gamma_{AODV} PRerr = \text{labelled } PRerr$  ("
 $\xi. \xi () dests := (\lambda rip. \text{case } (\text{dests } \xi) rip \text{ of } \text{None} \Rightarrow \text{None}$ 
 $\quad | \text{Some } rsn \Rightarrow \text{if } rip \in vD(rt \xi) \wedge \text{the}(nhop(rt \xi) rip) = sip \xi$ 
 $\quad \quad \quad \wedge sqn(rt \xi) rip < rsn \text{ then Some } rsn \text{ else None}) []$ 
 $\xi. \xi () rt := invalidate(rt \xi) (dests \xi) []$ 
 $\xi. \xi () store := setRRF(store \xi) (dests \xi) []$ 
 $\xi. \xi () pre := \bigcup \{ the(precs(rt \xi) rip) \mid rip. rip \in \text{dom}(dests \xi) \} []$ 
 $\xi. \xi () dests := (\lambda rip. \text{if } ((\text{dests } \xi) rip \neq \text{None} \wedge \text{the}(\text{prec}(rt \xi) rip) \neq \{\})$ 
 $\quad \quad \quad \text{then } (\text{dests } \xi) rip \text{ else } \text{None}) []$ 
 $\quad \quad \quad \text{groupcast}(\lambda \xi. \text{pre } \xi, \lambda \xi. \text{rerr}(\text{dests } \xi, ip \xi)). AODV()")$ 

```

declare  $\Gamma_{AODV}.\text{simp}$  [simp del, code del]

lemmas  $\Gamma_{AODV\_simp}$  [simp, code] =  $\Gamma_{AODV}.\text{simp}$  [simplified]

fun  $\Gamma_{AODV\_skeleton}$

where

```

 $"\Gamma_{AODV\_skeleton} PAodv = seqp_skeleton(\Gamma_{AODV} PAodv)"$ 
| " $\Gamma_{AODV\_skeleton} PNewPkt = seqp_skeleton(\Gamma_{AODV} PNewPkt)"$ 
| " $\Gamma_{AODV\_skeleton} PPkt = seqp_skeleton(\Gamma_{AODV} PPkt)"$ 
| " $\Gamma_{AODV\_skeleton} PRreq = seqp_skeleton(\Gamma_{AODV} PRreq)"$ 
| " $\Gamma_{AODV\_skeleton} PRrep = seqp_skeleton(\Gamma_{AODV} PRrep)"$ 

```

```

/ "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-simpseqp_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 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 aodv_simple_labels [simp]: "simple_labels ΓAODV" 
  ⟨proof⟩

lemma σAODV-labels [simp]: "(ξ, p) ∈ σAODV i ⟹ labels ΓAODV p = {PAodv-:0}"
  ⟨proof⟩

lemma aodv_init_kD_empty [simp]:
  "(ξ, p) ∈ σAODV i ⟹ kD (rt ξ) = {}"
  ⟨proof⟩

lemma aodv_init_sip_not_ip [simp]: "¬(sip (aodv_init i) = i)" ⟨proof⟩

lemma aodv_init_sip_not_ip' [simp]:
  assumes "(ξ, p) ∈ σAODV i"

```

```

shows "sip ξ ≠ ip ξ"
⟨proof⟩

lemma aodv_init_sip_not_i [simp]:
assumes "(ξ, p) ∈ σAODV 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 [cterms_env]
aodv_proc_cases [ctermsl_cases]
seq_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
cterms_intros]
seq_step_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
cterms_intros]

end

```

## 2.4 Invariant assumptions and properties

```

theory B_Aodv_Predicates
imports B_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 ⇒ ip"
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 rreqid dip dsn dsk oip osn sip.
msg_sender (Rreq hops rreqid 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)" ≡
"¬¬(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.
msg_zhops (Rreq hops rreqid 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)" ≡
"¬¬(d dip. msg_zhops (Newpkt d dip) = True)" ≡
⟨proof⟩

```

```

" \A 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_simp [simp]:
" \A hops rreqid dip dsn dsk oip osn sip.
  rreq_rrep_sn (Rreq hops rreqid dip dsn dsk oip osn sip) = (osn ≥ 1)"
" \A hops dip dsn oip sip. rreq_rrep_sn (Rrep hops dip dsn oip sip) = (dsn ≥ 1)"
" \A dests sip. rreq_rrep_sn (Rerr dests sip) = True"
" \A d dip. rreq_rrep_sn (Newpkt d dip) = True"
" \A 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]:
" \A 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)))"
" \A 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)"
" \A dests sip. rreq_rrep_fresh crt (Rerr dests sip) = True"
" \A d dip. rreq_rrep_fresh crt (Newpkt d dip) = True"
" \A 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]:
" \A hops rreqid dip dsn dsk oip osn sip.
  rerr_invalid crt (Rreq hops rreqid dip dsn dsk oip osn sip) = True"
" \A hops dip dsn oip sip. rerr_invalid crt (Rrep hops dip dsn oip sip) = True"
" \A dests sip. rerr_invalid crt (Rerr dests sip) = (∀ rip ∈ dom(dests).
  rip ∈ iD(crt) ∧ the (dests rip) = sqn crt rip)"
" \A d dip. rerr_invalid crt (Newpkt d dip) = True"
" \A 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 ⇒ aodv_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 = aodv_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

```

## 2.5 Quality relations between routes

```

theory B_Fresher
imports B_Aodv_Data
begin

```

### 2.5.1 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 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

$\text{nsqn} :: \text{rt} \Rightarrow \text{ip} \Rightarrow \text{sqn}$

where

$\text{nsqn} \equiv \lambda \text{rt } \text{dip}. \text{ case } \sigma_{\text{route}}(\text{rt}, \text{dip}) \text{ of } \text{None} \Rightarrow 0 \mid \text{Some } r \Rightarrow \text{nsqn}_r(r)$

lemma  $\text{nsqn\_sqn\_def}$ :

$\lambda \text{rt } \text{dip}. \text{ nsqn rt dip} = (\text{if flag rt dip} = \text{Some val} \vee \text{sqn rt dip} = 0 \text{ then sqn rt dip else sqn rt dip - 1})$

$\langle \text{proof} \rangle$

lemma  $\text{not\_in\_kD\_nsqn}$  [simp]:

assumes "dip  $\notin \text{kD}(\text{rt})"$

shows "nsqn rt dip = 0"

$\langle \text{proof} \rangle$

lemma  $\text{kD\_nsqn}$ :

assumes "dip  $\in \text{kD}(\text{rt})"$

shows "nsqn rt dip = nsqn<sub>r</sub>(the ( $\sigma_{\text{route}}(\text{rt}, \text{dip})$ ))"

$\langle \text{proof} \rangle$

lemma  $\text{nsqn}_r\text{-flag\_pred}$  [simp, intro]:

fixes dsn dsk flag hops nhip pre

assumes "P (nsqn<sub>r</sub> (dsn, dsk, val, hops, nhip, pre))"

and "P (nsqn<sub>r</sub> (dsn, dsk, inv, hops, nhip, pre))"

shows "P (nsqn<sub>r</sub> (dsn, dsk, flag, hops, nhip, pre))"

$\langle \text{proof} \rangle$

lemma  $\text{nsqn}_r\text{-addpreRT\_inv}$  [simp]:

$\lambda \text{rt } \text{dip } \text{npre } \text{dip}'. \text{ dip} \in \text{kD}(\text{rt}) \implies$

$\text{nsqn}_r (\text{the} (\text{the} (\text{addpreRT } \text{rt } \text{dip } \text{npre}) \text{ dip}')) = \text{nsqn}_r (\text{the} (\text{rt } \text{dip}'))$

$\langle \text{proof} \rangle$

lemma  $\text{sqn\_nsqn}$ :

$\lambda \text{rt } \text{dip}. \text{ sqn rt dip} - 1 \leq \text{nsqn rt dip}$

$\langle \text{proof} \rangle$

lemma  $\text{nsqn\_sqn}$ : "nsqn rt dip  $\leq \text{sqn rt dip}$ "

$\langle \text{proof} \rangle$

lemma  $\text{val\_nsqn\_sqn}$  [elim, simp]:

assumes "ip  $\in \text{kD}(\text{rt})"$

and "the (flag rt ip) = val"

shows "nsqn rt ip = sqn rt ip"

$\langle \text{proof} \rangle$

lemma  $\text{vD\_nsqn\_sqn}$  [elim, simp]:

assumes "ip  $\in \text{vD}(\text{rt})"$

shows "nsqn rt ip = sqn rt ip"

$\langle \text{proof} \rangle$

lemma  $\text{inv\_nsqn\_sqn}$  [elim, simp]:

assumes "ip  $\in \text{kD}(\text{rt})"$

and "the (flag rt ip) = inv"

shows "nsqn rt ip = sqn rt ip - 1"

$\langle \text{proof} \rangle$

lemma  $\text{id\_nsqn\_sqn}$  [elim, simp]:

assumes "ip  $\in \text{id}(\text{rt})"$

shows "nsqn rt ip = sqn rt ip - 1"

$\langle \text{proof} \rangle$

lemma  $\text{nsqn\_update\_changed\_kno\_val}$  [simp]: " $\lambda \text{rt } \text{ip } \text{dsn } \text{dsk } \text{hops } \text{nhip}.$

$\text{rt} \neq \text{update rt ip} (\text{dsn}, \text{kno}, \text{val}, \text{hops}, \text{nhip}, \{\})$

```

 $\implies \text{nsqn}(\text{update } rt \ ip \ (dsn, kno, val, hops, nhop, \{\})) \ ip = dsn$ 
⟨proof⟩

lemma nsqn_addpreRT_inv [simp]:
  "¬(rt dip npre dip'. dip ∈ kD(rt)) \implies
   \text{nsqn}(\text{the } (\text{addpreRT } rt \ dip \ npre)) \ dip' = \text{nsqn } rt \ dip"
⟨proof⟩

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

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

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

```

## 2.5.2 Comparing routes

definition

*fresher* :: "r ⇒ r ⇒ bool" ( $\langle \_ / \sqsubseteq \_ \rangle$ ) [51, 51] 50

where

"*fresher* r r' ≡ (( $\text{nsqn}_r \ r < \text{nsqn}_{r'} \ r'$ ) ∨ ( $\text{nsqn}_r \ r = \text{nsqn}_{r'} \ r' \wedge \pi_5(r) \geq \pi_5(r')$ ))"

lemma *fresherI1* [intro]:

assumes " $\text{nsqn}_r \ r < \text{nsqn}_{r'} \ r'$ "  
 shows "r  $\sqsubseteq$  r'"  
⟨proof⟩

lemma *fresherI2* [intro]:

assumes " $\text{nsqn}_r \ r = \text{nsqn}_{r'} \ r'$ "  
 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 " $\text{nsqn}_r \ r < \text{nsqn}_{r'} \ r' \implies P \ r \ r'$ "  
 and " $\text{nsqn}_r \ r = \text{nsqn}_{r'} \ r' \wedge \pi_5(r) \geq \pi_5(r') \implies P \ r \ r'$ "  
 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]:

```

"[]  $\neg(x \sqsubseteq y); \neg(z \sqsubseteq x) \Rightarrow \neg(z \sqsubseteq y)$ "  

⟨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 ⇒ rt ⇒ rt ⇒ bool"
where
  "rt_fresher ≡ λ dip rt rt'. (the (σ_route(rt, dip))) \sqsubseteq (the (σ_route(rt', dip)))"

abbreviation
  rt_fresher_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" (<(_/ \sqsubseteq_ _)> [51, 999, 51] 50)
where
  "rt1 \sqsubseteq_i rt2 ≡ rt_fresher i rt1 rt2"

lemma rt_fresher_def':
  "(rt1 \sqsubseteq_i rt2) = (nsqn_r (the (rt1 i)) < nsqn_r (the (rt2 i)) ∨
                           nsqn_r (the (rt1 i)) = nsqn_r (the (rt2 i)) ∧ π_5 (the (rt2 i)) ≤ π_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 ∈ kD(rt1)"
    and "dip ∈ kD(rt2)"
  shows "(rt1 \sqsubseteq_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 ∈ kD(rt1)"
    and "dip ∈ kD(rt2)"
    and "nsqn rt1 dip = nsqn rt2 dip"
    and "the (dhops rt1 dip) ≥ the (dhops rt2 dip)"
  shows "rt1 \sqsubseteq_dip rt2"
⟨proof⟩

lemma rt_fresherE [elim]:
  assumes "rt1 \sqsubseteq_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" (<_/≈_ _>) [51, 999, 51] 50
where
"rt1 ≈i rt2 ≡ rt_fresh_as i 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 ≈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_nsqrn:
  assumes "dip ∈ kD(rt1)"
    and "dip ∈ kD(rt2)"
    and "rt1 ≈dip rt2"
  shows "nsqrn_r (the (rt2 dip)) = nsqrn_r (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 "dests: ∀rip ∈ dom(dests). rip ∈ vD(rt) ∧ sqn rt rip < the (dests rip)"
  shows "rt ⊑_dip invalidate rt dests"
  ⟨proof⟩

lemma nsqn_r_invalidate [simp]:
  assumes "dip ∈ kD(rt)"
    and "dip ∈ dom(dests)"
  shows "nsqn_r (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]

2.5.4 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 ⊑_dip rt2) ∧ ¬(rt2 ⊑_dip 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 =
   (nsqn_r (the (rt1 i)) < nsqn_r (the (rt2 i))
    ∨ (nsqn_r (the (rt1 i)) = nsqn_r (the (rt2 i)) ∧ π5(the (rt1 i)) > π5(the (rt2 i))))"
⟨proof⟩

lemma rt_strictly_fresher_fresherD [dest]:
  assumes "rt1 ⊑_dip rt2"
  shows "the (rt1 dip) ⊑ the (rt2 dip)"
⟨proof⟩

lemma rt_strictly_fresher_not_fresh_asD [dest]:
  assumes "rt1 ⊑_dip rt2"
  shows "¬ rt1 ≈_dip rt2"
⟨proof⟩

lemma rt_strictly_fresher_trans [elim, trans]:
  assumes "rt1 ⊑_dip rt2"
    and "rt2 ⊑_dip rt3"
  shows "rt1 ⊑_dip rt3"
⟨proof⟩

lemma rt_strictly_fresher_irrefl [simp]: "¬ (rt ⊑_dip rt)"
⟨proof⟩

lemma rt_fresher_trans_rt_strictly_fresher [elim, trans]:
  assumes "rt1 ⊑_dip rt2"
    and "rt2 ⊑_dip rt3"
  shows "rt1 ⊑_dip rt3"
⟨proof⟩

lemma rt_fresher_trans_rt_strictly_fresher' [elim, trans]:
  assumes "rt1 ⊑_dip rt2"
    and "rt2 ⊑_dip rt3"
  shows "rt1 ⊑_dip rt3"
⟨proof⟩

lemma rt_fresher_imp_nsqn_le:
  assumes "rt1 ⊑_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

```

## 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 ⊨A 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-:4}
    ∪ {PRreq-:0..PRreq-:3}) → sip ξ ∈ kD(rt ξ))"
  ⟨proof⟩

lemma addpreRT_partly_welldefined:
  "paodv i ⊨
    onl ΓAODV (λ(ξ, 1). (1 ∈ {PRreq-:16..PRreq-:18} ∪ {PRrep-:1..PRrep-:5} → dip ξ ∈ kD(rt ξ))
      ∧ (1 ∈ {PRreq-:3..PRreq-:17} → oip ξ ∈ kD(rt ξ)))"
  ⟨proof⟩

```

Proposition 7.38

```

lemma includes_nhip:
  "paodv i ⊨ onl ΓAODV (λ(ξ, 1). ∀ dip ∈ kD(rt ξ). the (nhop(rt ξ) dip) ∈ kD(rt ξ))"
  ⟨proof⟩

```

Proposition 7.22: needed in Proposition 7.4

```

lemma addpreRT_welldefined:
  "paodv i ⊨ onl ΓAODV (λ(ξ, 1). (1 ∈ {PRreq-:16..PRreq-:18} → dip ξ ∈ kD(rt ξ)) ∧
    (1 = PRreq-:17 → oip ξ ∈ kD(rt ξ)) ∧
    (1 = PRrep-:4 → dip ξ ∈ kD(rt ξ)) ∧
    (1 = PRrep-:5 → (the (nhop(rt ξ) (dip ξ)) ∈ kD(rt ξ)))"
  (is "_ ⊨ onl ΓAODV ?P")
  ⟨proof⟩

```

Proposition 7.4

```

lemma known_destinations_increase:
  "paodv i ⊨_A onll ΓAODV (λ((ξ, _), _, (ξ', _)). kD(rt ξ) ⊆ kD(rt ξ'))"
  ⟨proof⟩

```

Proposition 7.5

```

lemma rreqs_increase:
  "paodv i ⊨_A onll ΓAODV (λ((ξ, _), _, (ξ', _)). rreqs ξ ⊆ rreqs ξ')"
  ⟨proof⟩

```

```

lemma dests_bigger_than_sqn:
  "paodv i ⊨ onl ΓAODV (λ(ξ, 1). 1 ∈ {PAodv-:15..PAodv-:19}
    ∪ {PPkt-:7..PPkt-:11}
    ∪ {PRreq-:9..PRreq-:13}
    ∪ {PRreq-:21..PRreq-:25}
    ∪ {PRrep-:9..PRrep-:13}
    ∪ {PRerr-:1..PRerr-:5}
    → (∀ ip ∈ dom(dests ξ). ip ∈ kD(rt ξ) ∧ sqn(rt ξ) ip ≤ the (dests ξ ip)))"
  ⟨proof⟩

```

Proposition 7.6

```

lemma sqns_increase:
  "paodv i ⊨_A onll ΓAODV (λ((ξ, _), _, (ξ', _)). ∀ ip. sqn(rt ξ) ip ≤ sqn(rt ξ') ip)"
  ⟨proof⟩

```

Proposition 7.7

```

lemma ip_constant:

```

```
"paodv i ⊨ onl ΓAODV (λ(ξ, _). ip ξ = i)"  

⟨proof⟩
```

Proposition 7.8

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

```
"paodv i ⊨A onll ΓAODV (λ(ξ, _, a, _). anycast (λm. not_Pkt m → msg_sender m = ip ξ) a)"  

⟨proof⟩
```

**lemma sender\_ip\_valid:**

```
"paodv i ⊨A onll ΓAODV (λ(ξ, _, a, _). anycast (λm. not_Pkt m → msg_sender m = i) a)"  

⟨proof⟩
```

**lemma received\_msg\_inv:**

```
"paodv i ⊨ (recvmsg P →) onl ΓAODV (λ(ξ, l). l ∈ {PAodv-:1} → P (msg ξ))"  

⟨proof⟩
```

Proposition 7.9

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

```
"paodv i ⊨ (recvmsg (λm. not_Pkt m → msg_sender m ≠ i) →) onl ΓAODV (λ(ξ, _). sip ξ ≠ ip ξ)"  

⟨proof⟩
```

**lemma sip\_not\_ip:**

```
"paodv i ⊨ (recvmsg (λm. not_Pkt m → msg_sender m ≠ i) →) onl ΓAODV (λ(ξ, _). sip ξ ≠ 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 ⊨ onl ΓAODV (λ(ξ, _). ∀ ip ∈ kD (rt ξ). the (dhops (rt ξ) ip) ≥ 1)"  

⟨proof⟩
```

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

```
"paodv i ⊨ onl ΓAODV (λ(ξ, l). (l ∈ {PRreq-:16..PRreq-:18} → dip ξ ∈ vD(rt ξ))  

    ∧ (l ∈ {PRreq-:5, PRreq-:6} → dip ξ = ip ξ)  

    ∧ (l ∈ {PRreq-:15..PRreq-:18} → dip ξ ≠ ip ξ))"  

⟨proof⟩
```

**lemma rrep\_dip\_in\_vD:**

```
"paodv i ⊨ onl ΓAODV (λ(ξ, l). (l ∈ {PRrep-:4..PRrep-:6} → dip ξ ∈ vD(rt ξ)))"  

⟨proof⟩
```

Proposition 7.11

**lemma anycast\_msg\_zhops:**

```
"¬rreqid dip dsn dsk oip osn sip.  

  paodv i ⊨A onll ΓAODV (λ(ξ, a, _). anycast msg_zhops a)"  

⟨proof⟩
```

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

```
"paodv i ⊨ (recvmsg msg_zhops →) onl ΓAODV (λ(ξ, l).  

  (l ∈ {PAodv-:4..PAodv-:5} ∪ {PRreq-:n/n. True} →  

   (hops ξ = 0 → oip ξ = sip ξ))  

  ∧  

  ((l ∈ {PAodv-:6..PAodv-:7} ∪ {PRrep-:n/n. True} →  

   (hops ξ = 0 → dip ξ = sip ξ))))"  

⟨proof⟩
```

**lemma osn\_rreq:**

```
"paodv i ⊨ (recvmsg rreq_rrep_sn →) onl ΓAODV (λ(ξ, l).  

  l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n/n. True} → 1 ≤ osn ξ)"  

⟨proof⟩
```

**lemma osn\_rreq':**

"paodv i  $\models$  (recvmsg ( $\lambda m.$  rreq\_rrep\_sn  $m \wedge msg_zhops m$ )  $\rightarrow$  onl  $\Gamma_{AODV} (\lambda(\xi, 1).$   
 $1 \in \{PAodv\text{-}:4, PAodv\text{-}:5\} \cup \{PRreq\text{-}:n/n. True\} \longrightarrow 1 \leq osn \xi)$ "

$\langle proof \rangle$

lemma dsn\_rrep:

"paodv i  $\models$  (recvmsg rreq\_rrep\_sn  $\rightarrow$  onl  $\Gamma_{AODV} (\lambda(\xi, 1).$   
 $1 \in \{PAodv\text{-}:6, PAodv\text{-}:7\} \cup \{PRrep\text{-}:n/n. True\} \longrightarrow 1 \leq dsn \xi)$ "  
 $\langle proof \rangle$

lemma dsn\_rrep':

"paodv i  $\models$  (recvmsg ( $\lambda m.$  rreq\_rrep\_sn  $m \wedge msg_zhops m$ )  $\rightarrow$  onl  $\Gamma_{AODV} (\lambda(\xi, 1).$   
 $1 \in \{PAodv\text{-}:6, PAodv\text{-}:7\} \cup \{PRrep\text{-}:n/n. True\} \longrightarrow 1 \leq dsn \xi)$ "  
 $\langle proof \rangle$

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

"paodv i  $\models$  (recvmsg ( $\lambda m.$  rreq\_rrep\_sn  $m \wedge msg_zhops m$ )  $\rightarrow$  onl  $\Gamma_{AODV} (\lambda(\xi, 1).$   
 $(1 \in \{PAodv\text{-}:4..PAodv\text{-}:5\} \cup \{PRreq\text{-}:n/n. True\} \longrightarrow$   
 $(hops \xi = 0 \longrightarrow oip \xi = sip \xi))$

$\wedge$   
 $((1 \in \{PAodv\text{-}:6..PAodv\text{-}:7\} \cup \{PRrep\text{-}:n/n. True\} \longrightarrow$   
 $(hops \xi = 0 \longrightarrow dip \xi = sip \xi)))")$

$\langle proof \rangle$

Proposition 7.12

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

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

$\langle proof \rangle$

lemma zero\_seq\_unk\_hops\_one:

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

$\langle proof \rangle$

lemma kD\_unk\_or\_atleast\_one:

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

Proposition 7.13

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

"paodv 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:

"paodv i  $\models_A$  onll  $\Gamma_{AODV} (\lambda((\xi, _), a, _).$  anycast (rreq\_rrep\_fresh (rt  $\xi)) a)"  
 $\langle proof \rangle$$

Proposition 7.15

lemma rerr\_invalid\_any\_step\_invariant:

"paodv i  $\models_A$  onll  $\Gamma_{AODV} (\lambda((\xi, _), a, _).$  anycast (rerr\_invalid (rt  $\xi)) a)"  
 $\langle proof \rangle$$

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:
  "paodv i ⊨
    onl ΓAODV (λ(ξ, 1). (1 ∈ {PAodv-:15, PPkt-:7, PRreq-:9, PRreq-:21, PRrep-:9})
      → ( ∀ ip ∈ dom(dests ξ). ip ∈ vD(rt ξ) ∧ the (dests ξ ip) = inc (sqn (rt ξ) ip)))
    ∧ (1 = PRerr-:1
      → ( ∀ ip ∈ dom(dests ξ). ip ∈ vD(rt ξ) ∧ the (dests ξ ip) > sqn (rt ξ) ip)))"
  ⟨proof⟩

```

Proposition 7.27

```

lemma route_tables_fresher:
  "paodv i ⊨A (recvmsg rreq_rrep_sn → onll ΓAODV (λ((ξ, _), _, (ξ', _)).
    ∀ dip ∈ kD(rt ξ). rt ξ ⊑dip rt ξ'))"
  ⟨proof⟩

```

end

## 2.7 The quality increases predicate

```

theory B_Quality_Increases
imports B_Aodv_Predicates B_Fresher
begin

definition quality_increases :: "state ⇒ state ⇒ bool"
where "quality_increases ξ ξ' ≡ ( ∀ dip ∈ kD(rt ξ). dip ∈ kD(rt ξ') ∧ rt ξ ⊑dip rt ξ')
  ∧ ( ∀ dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip)"

lemma quality_increasesI [intro!]:
  assumes " ∀ dip. dip ∈ kD(rt ξ) ⇒ dip ∈ kD(rt ξ')"
  and " ∀ dip. [ dip ∈ kD(rt ξ); dip ∈ kD(rt ξ') ] ⇒ rt ξ ⊑dip rt ξ"
  and " ∀ dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip"
  shows "quality_increases ξ ξ"
  ⟨proof⟩

lemma quality_increasesE [elim]:
  fixes dip
  assumes "quality_increases ξ ξ'"
  and "dip ∈ kD(rt ξ)"
  and "[ dip ∈ kD(rt ξ'); rt ξ ⊑dip rt ξ'; sqn (rt ξ) dip ≤ sqn (rt ξ') dip ] ⇒ R dip ξ ξ'"
  shows "R dip ξ ξ"
  ⟨proof⟩

lemma quality_increases_rt_fresherD [dest]:
  fixes ip
  assumes "quality_increases ξ ξ'"
  and "ip ∈ kD(rt ξ)"
  shows "rt ξ ⊑ip rt ξ"
  ⟨proof⟩

lemma quality_increases_sqnE [elim]:
  fixes dip
  assumes "quality_increases ξ ξ'"
  and "sqn (rt ξ) dip ≤ sqn (rt ξ') dip ⇒ R dip ξ ξ'"
  shows "R dip ξ ξ"
  ⟨proof⟩

lemma quality_increases_refl [intro, simp]: "quality_increases ξ ξ"

```

*(proof)*

```
lemma strictly_fresher_quality_increases_right [elim]:  
  fixes  $\sigma \sigma' dip$   
  assumes "rt ( $\sigma i$ ) \sqsubset_{dip} rt ( $\sigma nhip$ )"  
    and qinc: "quality_increases ( $\sigma nhip$ ) ( $\sigma' nhip$ )"  
    and "dip \in kD(rt ( $\sigma nhip$ ))"  
  shows "rt ( $\sigma i$ ) \sqsubset_{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  
      \longrightarrow (the(dhops(rt ( $\sigma sip$ )) ip) \leq hops  
      \vee the(flag(rt ( $\sigma sip$ )) ip) = inv))"  
(proof)
```

```

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))"
      (is "_ ∧ ?nsqnafter")
⟨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 σ m ==> rreq_rrep_sn m"
  ⟨proof⟩

lemma recv_msg_fresh_inc_sn [simp, elim]:
  "orecvmsg (msg_fresh) σ m ==> recvmsg 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

```

## 2.8 The ‘open’ AODV model

```

theory B_Oadv
imports B_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 ⇒ ((ip ⇒ state) × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
  "opaodv i ≡ () init = σAODV', trans = oseqp_sos ΓAODV i ()"

lemma initiali_aodv [intro!, simp]: "initiali i (init (opaodv i)) (init (paodv i))"
  ⟨proof⟩

lemma oaodv_control_within [simp]: "control_within ΓAODV (init (opaodv i))"
  ⟨proof⟩

lemma σAODV'_labels [simp]: "(σ, p) ∈ σAODV' ==> labels ΓAODV p = {PAodv-:0}"
  ⟨proof⟩

lemma oaodv_init_kD_empty [simp]:

```

```

"(σ, p) ∈ σAODV' ⟹ kD(rt(σ i)) = {}"
⟨proof⟩

lemma oaodv_init_vD_empty [simp]:
"(σ, p) ∈ σAODV' ⟹ vD(rt(σ i)) = {}"
⟨proof⟩

lemma oaodv_trans: "trans(oaodv i) = oseqp_sos ΓAODV 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 σ σ'"
shows "∀j. quality_increases (σ j) (σ' j)"
⟨proof⟩

lemma weaken_otherwith [elim]:
fixes m
assumes *: "otherwith P I (orecvmsg Q) σ σ' a"
  and weakenP: "¬σ m. P σ m ⟹ P' σ m"
  and weakenQ: "¬σ m. Q σ m ⟹ Q' σ m"
shows "otherwith P' I (orecvmsg Q') σ σ' a"
⟨proof⟩

lemma oreceived_msg_inv:
assumes other: "¬σ σ' m. [ P σ m; other Q {i} σ σ' ] ⟹ P σ' m"
  and local: "¬σ m. P σ m ⟹ P (σ(i := σ i | msg := m)) m"
shows "opaodv i ≡ (otherwith Q {i} (orecvmsg P), other Q {i} →)
          onl ΓAODV (λ(σ, l). l ∈ {PAodv-:1} → P σ (msg(σ i)))"
⟨proof⟩

(Equivalent to) Proposition 7.27

lemma local_quality_increases:
"paodv i ≡A (orecvmsg rreq_rrep_sn →) onll ΓAODV (λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ')"
⟨proof⟩

lemmas olocal_quality_increases =
open_seq_stepInvariant [OF local_quality_increases initiali_aodv oaodv_trans aodv_trans,
simplified seqll_onll_swap]

lemma oquality_increases:
"opaodv i ≡A (otherwith quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)),
other quality_increases {i} →)
onll ΓAODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j))"
(is "_ ≡A (?S, _ →) _")
⟨proof⟩

lemma rreq_rrep_nsqn_fresh_any_step_invariant:

```

```

"opaodv i ⊨_A (act (recvmsg rreq_rrep_sn), other A {i} →)
    onl ΓAODV (λ((σ, _), a, _). anycast (msg_fresh σ) a)"
⟨proof⟩

lemma oreceived_rreq_rrep_nsqn_fresh_inv:
"opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i} →)
    onl ΓAODV (λ((σ, 1). l ∈ {PAodv-:1} → msg_fresh σ (msg (σ i))))"
⟨proof⟩

lemma oquality_increases_nsqn_fresh:
"opaodv i ⊨_A (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i} →)
    onl ΓAODV (λ((σ, _, (σ', _)). ∀ j. quality_increases (σ j) (σ' j)))"
⟨proof⟩

lemma oosn_rreq:
"opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i} →)
    onl ΓAODV (seql i (λ(ξ, 1). l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n | n. True} → 1 ≤ osn ξ))"
⟨proof⟩

lemma rreq_sip:
"opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i} →)
    onl ΓAODV (λ((σ, 1).
        (l ∈ {PAodv-:4, PAodv-:5, PRreq-:0, PRreq-:2} ∧ sip (σ i) ≠ oip (σ i))
        → oip (σ i) ∈ kD(rt (σ (sip (σ i))))
        ∧ nsqn (rt (σ (sip (σ i)))) (oip (σ i)) ≥ osn (σ i)
        ∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i)) = osn (σ i))
        → (hops (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (oip (σ i)))
            ∨ the (flag (rt (σ (sip (σ i)))) (oip (σ i))) = inv)))"
(is "_ ⊨ (?S, ?U →) _")
⟨proof⟩

lemma odsn_rrep:
"opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i} →)
    onl ΓAODV (seql i (λ(ξ, 1). l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n | n. True} → 1 ≤ dsn ξ))"
⟨proof⟩

lemma rrep_sip:
"opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i} →)
    onl ΓAODV (λ((σ, 1).
        (l ∈ {PAodv-:6, PAodv-:7, PRrep-:0, PRrep-:1} ∧ sip (σ i) ≠ dip (σ i))
        → dip (σ i) ∈ kD(rt (σ (sip (σ 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:
"opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i} →)
    onl ΓAODV (λ((σ, 1).
        l ∈ {PAodv-:8, PAodv-: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: "paodv i ⊨
  onl ΓAODV (λ(ξ, 1). (1 = 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 oaddpreRT_welldefined =
  open_seq_invariant [OF addpreRT_welldefined initiali_aodv oaodv_trans aodv_trans,
  simplified seql_onl_swap,
  THEN oinvariant_anyact]

```

```

lemmas odests_vD_inc_sqn =
  open_seq_invariant [OF dests_vD_inc_sqn initiali_aodv oaodv_trans aodv_trans,
  simplified seql_onl_swap,
  THEN oinvariant_anyact]

```

```

lemmas oprerr_guard =
  open_seq_invariant [OF prerr_guard initiali_aodv oaodv_trans aodv_trans,
  simplified seql_onl_swap,
  THEN oinvariant_anyact]

```

Proposition 7.28

```

lemma seq_compare_next_hop':
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
  other quality_increases {i} →) onl ΓAODV (λ(σ, _).
  ∀ 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)"
(is "_ ⊨ (?S, ?U →) _")

```

*(proof)*

Proposition 7.30

```

lemmas okD_unk_or_atleast_one =
  open_seq_invariant [OF kD_unk_or_atleast_one initiali_aodv,
  simplified seql_onl_swap]

```

```

lemmas ozero_seq_unk_hops_one =
  open_seq_invariant [OF zero_seq_unk_hops_one initiali_aodv,
  simplified seql_onl_swap]

```

```

lemma oreachable_fresh_okD_unk_or_atleast_one:
  fixes dip
  assumes "(σ, p) ∈ oreachable (opaodv i)
  (otherwith ((=)) {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 (opaodv i)
  (otherwith ((=)) {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")

```

$\langle proof \rangle$

```
lemma seq_nhop_quality_increases':
  shows "opaodv i ⊨ (otherwith ((=)) {i}
    (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
    other quality_increases {i} →)
    onl ΓAODV (λ(σ, _). ∀ dip. let nhop = the (nhop (rt (σ i)) dip)
      in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhop))
      ∧ nhop ≠ dip
      → (rt (σ i)) ⊑dip (rt (σ nhop)))"
  (is "_ ⊨ (?S i, _ →) _")
  ⟨proof⟩
```

```
lemma seq_compare_next_hop:
  fixes w
  shows "opaodv i ⊨ (otherwith ((=)) {i} (orecvmsg msg_fresh),
    other quality_increases {i} →)
    global (λσ. ∀ 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)"
  ⟨proof⟩
```

```
lemma seq_nhop_quality_increases:
  shows "opaodv i ⊨ (otherwith ((=)) {i}
    (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),
    other quality_increases {i} →)
    global (λσ. ∀ dip. let nhop = the (nhop (rt (σ i)) dip)
      in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhop)) ∧ nhop ≠ dip
      → (rt (σ i)) ⊑dip (rt (σ nhop)))"
  ⟨proof⟩
```

end

## 2.10 Routing graphs and loop freedom

```
theory B_Loop_Freedom
imports B_Aodv_Predicates B_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  $σ$ , 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]:
```

```

"\ $\wedge ip' \sigma dip. (ip, ip') \in (rt\_graph \sigma dip)^+ \implies dip \in vD(rt(\sigma ip))$ " 
⟨proof⟩

lemma rt_graph_not_dip [dest]:
"\ $\wedge ip' \sigma dip. (ip, ip') \in rt\_graph \sigma dip \implies ip \neq dip$ " 
⟨proof⟩

lemma rt_graph_not_dip_trans [dest]:
"\ $\wedge ip' \sigma dip. (ip, ip') \in (rt\_graph \sigma dip)^+ \implies ip \neq dip$ " 
⟨proof⟩

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

lemma rt_graph_nhop_is_nhop [dest]:
"\ $\wedge ip' \sigma dip. (ip, ip') \in rt\_graph \sigma dip \implies ip' = \text{the}(nhop(rt(\sigma ip)) dip)$ " 
⟨proof⟩

theorem inv_to_loop_freedom:
assumes "¬ $\exists i dip. \text{let nhop} = \text{the}(nhop(rt(\sigma i)) dip)$   

           in  $dip \in vD(rt(\sigma i)) \cap vD(rt(\sigma nhop)) \wedge nhop \neq dip$   

           →  $(rt(\sigma i)) \sqsubseteq_{dip} (rt(\sigma nhop))$ " 
shows "¬ $\forall dip. \text{irrefl}((rt\_graph \sigma dip)^+)$ " 
⟨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 σ s a σ' s'
assumes "((σ, s), a, (σ', s')) ∈ oparp_sos i (oseqp_sos Γ_AODV i) (seqp_sos Γ_QMSG)"  

and "a ≠ τ"
shows "σ' i = σ i" 
⟨proof⟩

lemma par_nhop_quality_increases:
shows "opaodv i ⟨⟨i qmsg |= (otherwith ((=)) {i} (orecvmsg (λσ m.  

msg_fresh σ m ∧ msg_zhops m)),  

other quality_increases {i} →)  

global (λσ. ∀ dip. let nhop = the(nhop(rt(σ i)) dip)  

in dip ∈ vD(rt(σ i)) ∩ vD(rt(σ nhop)) ∧ nhop ≠ dip  

→ (rt(σ i)) ⊑_{dip} (rt(σ nhop)))" 
⟨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⟩

```

## 2.11.2 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 ((=)) {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 : opaodv 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 : opaodv 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 : opaodv 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 : opaodv 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⟩

```

## 2.11.3 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. opaodv i ⟨⟨i qmsg⟩⟩ p ⊨
            (otherwith ((=)) (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⟩

```

## 2.11.4 Lift to closed networks

```

lemma onet_nhop_quality_increases:
  shows "oclosed (opnet (λi. opaodv i ⟨⟨i qmsg⟩⟩ p)

```

```

 $\models (\lambda_{\_ \_ \_ \_}. \text{True}, \text{other\_quality\_increases} (\text{net\_tree\_ips } p) \rightarrow)$ 
 $\text{global } (\lambda\sigma. \forall i \in \text{net\_tree\_ips } p. \forall dip.$ 
 $\quad \text{let } nhip = \text{the} (\text{nhop} (\text{rt} (\sigma i)) dip)$ 
 $\quad \text{in } dip \in vD (\text{rt} (\sigma i)) \cap vD (\text{rt} (\sigma nhip)) \wedge nhip \neq dip$ 
 $\quad \longrightarrow (\text{rt} (\sigma i)) \sqsubset_{dip} (\text{rt} (\sigma nhip)))"$ 
(is "_  $\models (\_, ?U \rightarrow) ?inv")$ 
⟨proof⟩

```

## 2.11.5 Transfer into the standard model

```

interpretation aodv_openproc: openproc paodv opaodv id
  rewrites "aodv_openproc.initmissing = initmissing"
  ⟨proof⟩

```

```

interpretation aodv_openproc_par_qmsg: openproc_parq paodv opaodv id qmsg
  rewrites "aodv_openproc_par_qmsg.netglobal = netglobal"
  and "aodv_openproc_par_qmsg.initmissing = initmissing"
  ⟨proof⟩

```

```

lemma net_nhop_quality_increases:
  assumes "wf_net_tree n"
  shows "closed (pnet (\lambda i. paodv i < qmsg) n) \models netglobal
    (\lambda\sigma. \forall i 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
      \longrightarrow (\text{rt} (\sigma i)) \sqsubset_{dip} (\text{rt} (\sigma nhip)))"
(is "_ \models netglobal (\lambda\sigma. \forall i. ?inv \sigma i)")"
⟨proof⟩

```

## 2.11.6 Loop freedom of AODV

```

theorem aodv_loop_freedom:
  assumes "wf_net_tree n"
  shows "closed (pnet (\lambda i. paodv i < qmsg) n) \models netglobal (\lambda\sigma. \forall dip. irrefl ((rt_graph \sigma 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 ⇒ sqn"
  where "inc sn ≡ if sn = 0 then sn else sn + 1"

lemma less_than_inc [simp]: "x ≤ inc x"
  ⟨proof⟩

lemma inc_minus_suc_0 [simp]:
  "inc x - Suc 0 = x"
  ⟨proof⟩

lemma inc_never_one' [simp, intro]: "inc x ≠ Suc 0"
  ⟨proof⟩

lemma inc_never_one [simp, intro]: "inc x ≠ 1"
  ⟨proof⟩
```

#### 3.1.2 Modelling Routes

A route is a 5-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. In this variant, the set of ‘precursor nodes’ is not modelled.

```
type_synonym r = "sqn × k × f × nat × 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 ≡ {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.
    σ_route(rt, dip) = Some (dsn, dsk, flag, hops, nhip)"
  ⟨proof⟩

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

lemma vD_Some [dest]:
  fixes dip rt
  assumes "dip ∈ vD rt"
  shows "∃ dsn dsk hops nhip.
    σ_route(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.
    σ_route(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: "¬ 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 ∉ 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. 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')"

```

```

" $\llbracket ip \in kD(rt); sqn rt ip \geq \pi_2(r); \pi_3(r) = kno;$ 
 $\quad sqn rt ip = \pi_2(r) \implies \text{the}(dhops rt ip) \leq \pi_5(r) \wedge \text{the}(flag rt ip) = val \rrbracket$ 
 $\qquad \qquad \qquad \implies update rt ip r = rt(ip \mapsto s)$ "
```

$\langle proof \rangle$

lemma update\_cases [elim]:  
assumes " $(\pi_2(r) = 0) = (\pi_3(r) = unk)$ "  
and c1: " $\llbracket ip \notin kD(rt) \rrbracket \implies P(rt(ip \mapsto r))$ "  
and c2: " $\llbracket ip \in kD(rt); sqn rt ip < \pi_2(r) \rrbracket$   
 $\qquad \qquad \qquad \implies P(rt(ip \mapsto r))$ "  
and c3: " $\llbracket ip \in kD(rt); sqn rt ip = \pi_2(r); \text{the}(dhops rt ip) > \pi_5(r) \rrbracket$   
 $\qquad \qquad \qquad \implies P(rt(ip \mapsto r))$ "  
and c4: " $\llbracket ip \in kD(rt); sqn rt ip = \pi_2(r); \text{the}(flag rt ip) = inv \rrbracket$   
 $\qquad \qquad \qquad \implies P(rt(ip \mapsto r))$ "  
and c5: " $\llbracket ip \in kD(rt); \pi_3(r) = unk \rrbracket$   
 $\qquad \qquad \qquad \implies P(rt(ip \mapsto (\pi_2(\text{the}(\sigma_{route}(rt, ip))), \pi_3(r),$   
 $\qquad \qquad \qquad \qquad \pi_4(r), \pi_5(r), \pi_6(r))))$ "  
and c6: " $\llbracket ip \in kD(rt); sqn rt ip \geq \pi_2(r); \pi_3(r) = kno;$   
 $\quad sqn rt ip = \pi_2(r) \implies \text{the}(dhops rt ip) \leq \pi_5(r) \wedge \text{the}(flag rt ip) = val \rrbracket$   
 $\qquad \qquad \qquad \implies P(rt(ip \mapsto \text{the}(\sigma_{route}(rt, ip))))$ "  
shows "(P(update rt ip r))"  
 $\langle proof \rangle$

lemma update\_cases\_kD:  
assumes " $(\pi_2(r) = 0) = (\pi_3(r) = unk)$ "  
and "ip  $\in kD(rt)"$   
and c2: " $sqn rt ip < \pi_2(r) \implies P(rt(ip \mapsto r))$ "  
and c3: " $\llbracket sqn rt ip = \pi_2(r); \text{the}(dhops rt ip) > \pi_5(r) \rrbracket$   
 $\qquad \qquad \qquad \implies P(rt(ip \mapsto r))$ "  
and c4: " $\llbracket sqn rt ip = \pi_2(r); \text{the}(flag rt ip) = inv \rrbracket$   
 $\qquad \qquad \qquad \implies P(rt(ip \mapsto r))$ "  
and c5: " $\pi_3(r) = unk \implies P(rt(ip \mapsto (\pi_2(\text{the}(\sigma_{route}(rt, ip))), \pi_3(r),$   
 $\qquad \qquad \qquad \qquad \pi_4(r), \pi_5(r), \pi_6(r))))$ "  
and c6: " $\llbracket sqn rt ip \geq \pi_2(r); \pi_3(r) = kno;$   
 $\quad sqn rt ip = \pi_2(r) \implies \text{the}(dhops rt ip) \leq \pi_5(r) \wedge \text{the}(flag rt ip) = val \rrbracket$   
 $\qquad \qquad \qquad \implies P(rt(ip \mapsto \text{the}(\sigma_{route}(rt, ip))))$ "  
shows "(P(update rt ip r))"  
 $\langle proof \rangle$

lemma in\_kD\_after\_update [simp]:  
fixes rt nip dsn dsk flag hops nhip  
shows " $kD(\text{update } rt \text{ } nip \text{ } (dsn, dsk, flag, hops, nhip)) = \text{insert } nip \text{ } (kD \text{ } rt)$ "  
 $\langle proof \rangle$

lemma nhop\_of\_update [simp]:  
fixes rt dip dsn dsk flag hops nhip  
assumes " $rt \neq \text{update } rt \text{ } dip \text{ } (dsn, dsk, flag, hops, nhip)$ "  
shows " $\text{the}(nhop}(\text{update } rt \text{ } dip \text{ } (dsn, dsk, flag, hops, nhip)) \text{ } dip = nhip$ "  
 $\langle proof \rangle$

lemma sqn\_if\_updated:  
fixes rip v rt ip  
shows " $\text{sqn}(\lambda x. \text{if } x = rip \text{ then Some } v \text{ else } rt x) \text{ } ip$   
 $\qquad \qquad \qquad = (\text{if } ip = rip \text{ then } \pi_2(v) \text{ else } \text{sqn } rt \text{ } ip)$ "  
 $\langle proof \rangle$

lemma update\_sqn [simp]:  
fixes rt dip rip dsn dsk hops nhip  
assumes " $(dsn = 0) = (dsk = unk)$ "  
shows " $\text{sqn } rt \text{ } dip \leq \text{sqn}(\text{update } rt \text{ } rip \text{ } (dsn, dsk, val, hops, nhip)) \text{ } dip$ "  
 $\langle proof \rangle$

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]: " $\wedge_{rt \in \text{rt}} dip \in \text{dip} \text{ dsn } \text{dsk } \text{flg } \text{hops } \text{sip}.$ 
   $rt \neq \text{update } rt \text{ dip } (\text{dsn}, \text{dsk}, \text{flg}, \text{hops}, \text{sip})$ 
   $\implies \text{sqnf } (\text{update } rt \text{ dip } (\text{dsn}, \text{dsk}, \text{flg}, \text{hops}, \text{sip})) \text{ dip} = \text{dsk}$ "
   $\langle proof \rangle$ 

lemma update_kno_dsn_greater_zero:
  " $\wedge_{rt \in \text{rt}} dip \in \text{dip} \text{ ip } \text{dsn } \text{hops}.$   $1 \leq \text{dsn} \implies 1 \leq (\text{sqn } (\text{update } rt \text{ dip } (\text{dsn}, \text{kno}, \text{val}, \text{hops}, \text{ip})) \text{ dip})$ ""
   $\langle proof \rangle$ 

lemma proj3_update [simp]: " $\wedge_{rt \in \text{rt}} dip \in \text{dip} \text{ dsn } \text{dsk } \text{flg } \text{hops } \text{sip}.$ 
   $rt \neq \text{update } rt \text{ dip } (\text{dsn}, \text{dsk}, \text{flg}, \text{hops}, \text{sip})$ 
   $\implies \pi_3(\text{the } (\text{update } rt \text{ dip } (\text{dsn}, \text{dsk}, \text{flg}, \text{hops}, \text{sip})) \text{ dip}) = \text{dsk}$ "
   $\langle proof \rangle$ 

lemma nhop_update_changed_kno_val [simp]: " $\wedge_{rt \in \text{rt}} ip \in \text{ip} \text{ dsn } \text{dsk } \text{hops } \text{nhip}.$ 
   $rt \neq \text{update } rt \text{ ip } (\text{dsn}, \text{kno}, \text{val}, \text{hops}, \text{nhip})$ 
   $\implies \text{the } (\text{nhop } (\text{update } rt \text{ ip } (\text{dsn}, \text{kno}, \text{val}, \text{hops}, \text{nhip})) \text{ ip}) = \text{nhip}$ ""
   $\langle proof \rangle$ 

lemma flag_update [simp]: " $\wedge_{rt \in \text{rt}} dip \in \text{dip} \text{ dsn } \text{flg } \text{hops } \text{sip}.$ 
   $rt \neq \text{update } rt \text{ dip } (\text{dsn}, \text{kno}, \text{flg}, \text{hops}, \text{sip})$ 
   $\implies \text{the } (\text{flag } (\text{update } rt \text{ dip } (\text{dsn}, \text{kno}, \text{flg}, \text{hops}, \text{sip})) \text{ dip}) = \text{flg}$ ""
   $\langle proof \rangle$ 

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

lemma kD_update_unchanged [dest]:
  fixes rt dip dsn dsk flag hops nhip
  assumes "rt = update rt dip (\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip})"
  shows "dip ∈ kD(rt)"
   $\langle proof \rangle$ 

lemma nhop_update [simp]: " $\wedge_{rt \in \text{rt}} dip \in \text{dip} \text{ dsn } \text{dsk } \text{flg } \text{hops } \text{sip}.$ 
   $rt \neq \text{update } rt \text{ dip } (\text{dsn}, \text{dsk}, \text{flg}, \text{hops}, \text{sip})$ 
   $\implies \text{the } (\text{nhop } (\text{update } rt \text{ dip } (\text{dsn}, \text{dsk}, \text{flg}, \text{hops}, \text{sip})) \text{ dip}) = \text{sip}$ ""
   $\langle proof \rangle$ 

lemma sqn_update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip
  assumes "ip ≠ dip"
  shows "sqn (update rt dip (\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip})) ip = sqn rt ip"
   $\langle proof \rangle$ 

lemma sqnf_update_another [simp]:
  fixes dip ip rt dsn dsk flag hops nhip
  assumes "ip ≠ dip"
  shows "sqnf (update rt dip (\text{dsn}, \text{dsk}, \text{flag}, \text{hops}, \text{nhip})) ip = sqnf rt ip"
   $\langle proof \rangle$ 

lemma vD_update_val [dest]:
  " $\wedge_{dip \in \text{dip}} rt \text{ dip' } \text{dsn } \text{dsk } \text{hops } \text{nhip}.$ 
   $dip \in vD(\text{update } rt \text{ dip' } (\text{dsn}, \text{dsk}, \text{val}, \text{hops}, \text{nhip})) \implies (dip \in vD(rt) \vee dip = dip')$ ""
   $\langle proof \rangle$ 

```

## In invalidating route entries

```

definition invalidate :: "rt ⇒ (ip → sqn) ⇒ rt"
where "invalidate rt dests ≡

```

```

 $\lambda ip. \text{case } (\text{rt } ip, \text{dests } ip) \text{ of}$ 
   $(\text{None}, \_) \Rightarrow \text{None}$ 
   $/ (\text{Some } s, \text{None}) \Rightarrow \text{Some } s$ 
   $/ (\text{Some } (\_, \text{dsk}, \_, \text{hops}, \text{nhip}), \text{Some } rsn) \Rightarrow$ 
     $\text{Some } (rsn, \text{dsk}, \text{inv}, \text{hops}, \text{nhip})"$ 

lemma proj3_invalidate [simp]:
  " $\lambda dip. \pi_3(\text{the } ((\text{invalidate rt dests}) \text{ dip})) = \pi_3(\text{the } (\text{rt dip}))$ "
  ⟨proof⟩

lemma proj5_invalidate [simp]:
  " $\lambda dip. \pi_5(\text{the } ((\text{invalidate rt dests}) \text{ dip})) = \pi_5(\text{the } (\text{rt dip}))$ "
  ⟨proof⟩

lemma proj6_invalidate [simp]:
  " $\lambda dip. \pi_6(\text{the } ((\text{invalidate rt dests}) \text{ dip})) = \pi_6(\text{the } (\text{rt dip}))$ "
  ⟨proof⟩

lemma invalidate_kD_inv [simp]:
  " $\lambda rt \text{ dests}. kD (\text{invalidate rt dests}) = kD rt$ "
  ⟨proof⟩

lemma invalidate_sqn:
  fixes rt dip dests
  assumes " $\forall rsn. \text{dests dip} = \text{Some } rsn \rightarrow sqn rt dip \leq rsn$ "
  shows "sqn rt dip  $\leq$  sqn (invalidate rt dests) dip"
  ⟨proof⟩

lemma sqn_invalidate_in_dests [simp]:
  fixes dests ipa rsn rt
  assumes "dests ipa = Some rsn"
  and "ipa \in kD(rt)"
  shows "sqn (invalidate rt dests) ipa = rsn"
  ⟨proof⟩

lemma dhops_invalidate [simp]:
  " $\lambda dip. \text{the } (dhops (\text{invalidate rt dests}) \text{ dip}) = \text{the } (dhops rt dip)$ "
  ⟨proof⟩

lemma sqnf_invalidate [simp]:
  " $\lambda dip. sqnf (\text{invalidate } (rt \xi) \text{ (dests } \xi)) \text{ dip} = sqnf (rt \xi) \text{ dip}$ "
  ⟨proof⟩

lemma nhop_invalidate [simp]:
  " $\lambda dip. \text{the } (nhop (\text{invalidate } (rt \xi) \text{ (dests } \xi)) \text{ dip}) = \text{the } (nhop (rt \xi) \text{ dip})$ "
  ⟨proof⟩

lemma invalidate_other [simp]:
  fixes rt dests dip
  assumes "dip \notin \text{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 = \text{None}"
  ⟨proof⟩

lemma vD_invalidate_vD_not_dests:
  " $\lambda dip rt \text{ dests}. dip \in vD(\text{invalidate rt dests}) \implies dip \in vD(rt) \wedge \text{dests dip} = \text{None}$ "
  ⟨proof⟩

```

```

lemma sqn_invalidate_not_in_dests [simp]:
  fixes dests dip rt
  assumes "dip ∉ 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 ≡ 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)"

```

### 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.  $\sigma_{p\text{-}flag}(store, dip)$  is partial.
4. The routing table (*rt*) is modelled as a map ( $ip \Rightarrow r\ option$ ) 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

## 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 → sqn" ip
  | Newpkt data ip
  | Pkt data ip ip

instantiation msg :: msg
begin
  definition newpkt_def [simp]: "newpkt ≡ λ(d, dip). Newpkt d dip"
  definition eq_newpkt_def: "eq_newpkt m ≡ case m of Newpkt d dip ⇒ True | _ ⇒ 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

```

### 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 × rreqid) set"
  store   :: "store"

  msg     :: "msg"
  data    :: "data"
  dests   :: "ip → sqn"
  rreqid  :: "rreqid"
  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 = Map.empty,
  rreqs = {}
  dests = {}
  store = {}
  msg = {}
  data = {}
  oip = {}
  hops = {}
  dsn = {}
  dsk = {}
  osn = {}
  sip = {}"

```

```

rreqs = {},
store = Map.empty,

msg    = (SOME x. True),
data   = (SOME x. True),
dests  = (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),
  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 " $\xi' \in is\_rreq \xi$ "
shows " $(\exists \text{hops}' \text{ rreqid}' \text{ dip}' \text{ dsn}' \text{ dsk}' \text{ oip}' \text{ osn}' \text{ sip}'.$ 
       $\text{msg } \xi = Rreq \text{ hops}' \text{ rreqid}' \text{ dip}' \text{ dsn}' \text{ dsk}' \text{ oip}' \text{ osn}' \text{ sip}' \wedge$ 
       $\xi' = \xi(\text{hops} := \text{hops}', \text{rreqid} := \text{rreqid}', \text{dip} := \text{dip}', \text{dsn} := \text{dsn}',$ 
       $\text{dsk} := \text{dsk}', \text{oip} := \text{oip}', \text{osn} := \text{osn}', \text{sip} := \text{sip}')")$ 
⟨proof⟩

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

lemma is_rrep_asm [dest!]:
assumes " $\xi' \in is\_rrep \xi$ "
shows " $(\exists \text{hops}' \text{ dip}' \text{ dsn}' \text{ oip}' \text{ sip}'.$ 
       $\text{msg } \xi = Rrep \text{ hops}' \text{ dip}' \text{ dsn}' \text{ oip}' \text{ sip}' \wedge$ 
       $\xi' = \xi(\text{hops} := \text{hops}', \text{dip} := \text{dip}', \text{dsn} := \text{dsn}', \text{oip} := \text{oip}', \text{sip} := \text{sip}')")$ 
⟨proof⟩

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

lemma is_rerr_asm [dest!]:
assumes " $\xi' \in is\_rerr \xi$ "
shows " $(\exists \text{dests}' \text{ sip}'.$ 
       $\text{msg } \xi = Rerr \text{ dests}' \text{ sip}' \wedge$ 
       $\xi' = \xi(\text{dests} := \text{dests}', \text{sip} := \text{sip}')")$ 
⟨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]:
" $\xi' \in is\_rerr \xi \implies ip \xi' = ip \xi$ "
" $\xi' \in is\_rrep \xi \implies ip \xi' = ip \xi$ "
" $\xi' \in is\_rreq \xi \implies ip \xi' = ip \xi$ "
" $\xi' \in is\_pkt \xi \implies ip \xi' = ip \xi$ "
" $\xi' \in is\_newpkt \xi \implies ip \xi' = ip \xi$ "
⟨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]:
" $\xi' \in is\_rerr \xi \implies rreqs \xi' = rreqs \xi$ "
" $\xi' \in is\_rrep \xi \implies rreqs \xi' = rreqs \xi$ "
" $\xi' \in is\_rreq \xi \implies rreqs \xi' = rreqs \xi$ 
```

```

" $\xi' \in \text{is\_pkt } \xi \implies \text{rreqs } \xi' = \text{rreqs } \xi"$ 
" $\xi' \in \text{is\_newpkt } \xi \implies \text{rreqs } \xi' = \text{rreqs } \xi"$ 
⟨proof⟩

```

```

lemma is_msg_inv_store [simp]:
  " $\xi' \in \text{is\_rerr } \xi \implies \text{store } \xi' = \text{store } \xi"$ 
  " $\xi' \in \text{is\_rrep } \xi \implies \text{store } \xi' = \text{store } \xi"$ 
  " $\xi' \in \text{is\_rreq } \xi \implies \text{store } \xi' = \text{store } \xi"$ 
  " $\xi' \in \text{is\_pkt } \xi \implies \text{store } \xi' = \text{store } \xi"$ 
  " $\xi' \in \text{is\_newpkt } \xi \implies \text{store } \xi' = \text{store } \xi"$ 
⟨proof⟩

```

```

lemma is_msg_inv_sip [simp]:
  " $\xi' \in \text{is\_pkt } \xi \implies \text{sip } \xi' = \text{sip } \xi"$ 
  " $\xi' \in \text{is\_newpkt } \xi \implies \text{sip } \xi' = \text{sip } \xi"$ 
⟨proof⟩

```

### 3.3.3 The protocol process

```

datatype pseqp =
  PAodv
  | PNewPkt
  | PPkt
  | PRreq
  | PRrep
  | PRerr

fun nat_of_seqp :: "pseqp ⇒ nat"
where
  "nat_of_seqp PAodv = 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]: "11 ≤ 12 = (nat_of_seqp 11 ≤ nat_of_seqp 12)"
definition less_seqp [iff]: "11 < 12 = (nat_of_seqp 11 < nat_of_seqp 12)"
instance ⟨proof⟩
end

abbreviation AODV
where
  "AODV ≡ λ_. [[clear_locals]] call(PAodv)"

abbreviation PKT
where
  "PKT args ≡
     $\xi. \text{let } (\text{data}, \text{dip}, \text{oip}) = \text{args } \xi \text{ in}$ 
     $(\text{clear_locals } \xi) () \text{ data := data, dip := dip, oip := oip } ()$ 
    call(PPkt)"

abbreviation NEWPKT
where
  "NEWPKT args ≡
     $\xi. \text{let } (\text{data}, \text{dip}) = \text{args } \xi \text{ in}$ 
     $(\text{clear_locals } \xi) () \text{ data := data, dip := dip } ()$ 
    call(PNewPkt)"

abbreviation RREQ
where
  "RREQ args ≡
     $\xi. \text{let } (\text{data}, \text{dip}) = \text{args } \xi \text{ in}$ 
     $(\text{clear_locals } \xi) () \text{ data := data, dip := dip } ()$ 
    call(PNewPkt)"

```

```

 $\xi. \text{let } (hops, rreqid, dip, dsn, dsk, oip, osn, sip) = \text{args } \xi \text{ in}$ 
 $\quad (\text{clear_locals } \xi) () \quad \text{hop} := \text{hop}, \text{rreqid} := \text{rreqid}, \text{dip} := \text{dip},$ 
 $\quad \text{dsn} := \text{dsn}, \text{dsk} := \text{dsk}, \text{oip} := \text{oip},$ 
 $\quad \text{osn} := \text{osn}, \text{sip} := \text{sip} ()]$ 
 $\text{call}(PRreq)"}$ 

abbreviation RREP
where
"RREP args ≡
 $\xi. \text{let } (hops, dip, dsn, oip, sip) = \text{args } \xi \text{ in}$ 
 $\quad (\text{clear_locals } \xi) () \quad \text{hop} := \text{hop}, \text{dip} := \text{dip}, \text{dsn} := \text{dsn},$ 
 $\quad \text{oip} := \text{oip}, \text{sip} := \text{sip} ()]$ 
 $\text{call}(PRrep)"$ 

abbreviation RERR
where
"RERR args ≡
 $\xi. \text{let } (dests, sip) = \text{args } \xi \text{ in}$ 
 $\quad (\text{clear_locals } \xi) () \quad \text{dests} := \text{dests}, \text{sip} := \text{sip} ()]$ 
 $\text{call}(PRerr)"$ 

fun  $\Gamma_{AODV} :: "(\text{state}, \text{msg}, \text{pseqp}, \text{pseqp label}) \text{ seqp\_env}"$ 
where
" $\Gamma_{AODV} PAodv = \text{labelled } PAodv ($ 
 $\text{receive}(\lambda msg' \xi. \xi () \mid \text{msg} := \text{msg}' ()).$ 
 $\quad (\langle \text{is\_newpkt} \rangle \text{ NEWPKT}(\lambda \xi. (\text{data } \xi, \text{ip } \xi))$ 
 $\quad \oplus \langle \text{is\_pkt} \rangle \text{ PKT}(\lambda \xi. (\text{data } \xi, \text{dip } \xi, \text{oip } \xi))$ 
 $\quad \oplus \langle \text{is\_rreq} \rangle$ 
 $\quad \quad [\xi. \xi () \mid \text{rt} := \text{update}(\text{rt } \xi) (\text{sip } \xi) (0, \text{unk}, \text{val}, 1, \text{sip } \xi)]$ 
 $\quad \quad \text{RREQ}(\lambda \xi. (\text{hop } \xi, \text{rreqid } \xi, \text{dip } \xi, \text{dsn } \xi, \text{dsk } \xi, \text{oip } \xi, \text{osn } \xi, \text{sip } \xi))$ 
 $\quad \oplus \langle \text{is\_rrep} \rangle$ 
 $\quad \quad [\xi. \xi () \mid \text{rt} := \text{update}(\text{rt } \xi) (\text{sip } \xi) (0, \text{unk}, \text{val}, 1, \text{sip } \xi)]$ 
 $\quad \quad \text{RREP}(\lambda \xi. (\text{hop } \xi, \text{dip } \xi, \text{dsn } \xi, \text{oip } \xi, \text{sip } \xi))$ 
 $\quad \oplus \langle \text{is\_rerr} \rangle$ 
 $\quad \quad [\xi. \xi () \mid \text{rt} := \text{update}(\text{rt } \xi) (\text{sip } \xi) (0, \text{unk}, \text{val}, 1, \text{sip } \xi)]$ 
 $\quad \quad \text{RERR}(\lambda \xi. (\text{dests } \xi, \text{sip } \xi))$ 
 $\quad )$ 
 $\quad \oplus \langle \lambda \xi. \{ \xi () \mid \text{dip} := \text{dip} \} \mid \text{dip}. \text{dip} \in \text{qD(store } \xi) \cap \text{vD(rt } \xi) \}$ 
 $\quad \quad [\xi. \xi () \mid \text{data} := \text{hd}(\sigma_{\text{queue}}(\text{store } \xi, \text{dip } \xi))]$ 
 $\quad \quad \text{unicast}(\lambda \xi. \text{the } (\text{nhop } (\text{rt } \xi) (\text{dip } \xi)), \lambda \xi. \text{pkt}(\text{data } \xi, \text{dip } \xi, \text{ip } \xi)).$ 
 $\quad \quad [\xi. \xi () \mid \text{store} := \text{the } (\text{drop } (\text{dip } \xi) (\text{store } \xi))]$ 
 $\quad \quad \text{AODV}()$ 
 $\quad \triangleright [\xi. \xi () \mid \text{dests} := (\lambda \text{rip}. \text{if } (\text{rip} \in \text{vD(rt } \xi) \wedge \text{nhop(rt } \xi) \text{ rip} = \text{nhop(rt } \xi) \text{ (dip } \xi))$ 
 $\quad \quad \quad \text{then Some } (\text{inc } (\text{sqn } (\text{rt } \xi) \text{ rip})) \text{ else None} ()]$ 
 $\quad \quad \quad [\xi. \xi () \mid \text{rt} := \text{invalidate}(\text{rt } \xi) (\text{dests } \xi)]$ 
 $\quad \quad \quad [\xi. \xi () \mid \text{store} := \text{setRRF}(\text{store } \xi) (\text{dests } \xi)]$ 
 $\quad \quad \quad \text{broadcast}(\lambda \xi. \text{rerr}(\text{dests } \xi, \text{ip } \xi)). \text{AODV}()$ 
 $\quad \oplus \langle \lambda \xi. \{ \xi () \mid \text{dip} := \text{dip} \}$ 
 $\quad \quad \mid \text{dip}. \text{dip} \in \text{qD(store } \xi) - \text{vD(rt } \xi) \wedge \text{the } (\sigma_{\text{p-flag}}(\text{store } \xi, \text{dip})) = \text{req} \}$ 
 $\quad \quad [\xi. \xi () \mid \text{store} := \text{unsetRRF}(\text{store } \xi) (\text{dip } \xi)]$ 
 $\quad \quad [\xi. \xi () \mid \text{sn} := \text{inc } (\text{sn } \xi)]$ 
 $\quad \quad [\xi. \xi () \mid \text{rreqid} := \text{nrreqid}(\text{rreqs } \xi) (\text{ip } \xi)]$ 
 $\quad \quad [\xi. \xi () \mid \text{rreqs} := \text{rreqs } \xi \cup \{(\text{ip } \xi, \text{rreqid } \xi)\}]$ 
 $\quad \quad \text{broadcast}(\lambda \xi. \text{rreq}(0, \text{rreqid } \xi, \text{dip } \xi, \text{sqn } (\text{rt } \xi) (\text{dip } \xi), \text{sqnf } (\text{rt } \xi) (\text{dip } \xi), \text{ip } \xi, \text{sn } \xi,$ 
 $\quad \quad \quad \text{ip } \xi)). \text{AODV}()")$ 

| " $\Gamma_{AODV} PNewPkt = \text{labelled } PNewPkt ($ 
 $\quad \langle \xi. \text{dip } \xi = \text{ip } \xi \rangle$ 
 $\quad \text{deliver}(\lambda \xi. \text{data } \xi). \text{AODV}()$ 
 $\quad \oplus \langle \xi. \text{dip } \xi \neq \text{ip } \xi \rangle$ 
 $\quad \quad [\xi. \xi () \mid \text{store} := \text{add } (\text{data } \xi) (\text{dip } \xi) (\text{store } \xi)]$ 
 $\quad \quad \text{AODV}()")$ 

| " $\Gamma_{AODV} PPkt = \text{labelled } PPkt ($ 

```

```

⟨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.simp [simp del, code del]
lemmas ΓAODV_simp [simp, code] = ΓAODV.simp [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.simp 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**  $\sigma_{AODV} :: "ip \Rightarrow (state \times (state, msg, pseqp, pseqp label) seqp) set"$   
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

" $paodv i \equiv (\| init = \sigma_{AODV} i, trans = seqp\_sos \Gamma_{AODV} \|)$ "

**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**  $aodv\_init\_sip\_not\_i [simp]:$   
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**

$\Gamma_{AODV\_simps} [cterms\_env]$

```

aodv_proc_cases [ctermsI_cases]
seq_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
                      cterms_intros]
seq_step_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
                           cterms_intros]

end

```

### 3.4 Invariant assumptions and properties

```

theory C_Aodv_Predicates
imports C_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 ⇒ ip"
where "msg_sender m ≡ case m of Rreq _ _ _ _ _ ipc ⇒ ipc
                           | Rrep _ _ _ _ ipc ⇒ ipc
                           | Rerr _ ipc ⇒ ipc
                           | Pkt _ _ ipc ⇒ ipc"

```

```

lemma msg_sender_simpss [simp]:
"¬(hops rreqid dip dsn dsk oip osn sip).
   msg_sender (Rreq hops rreqid 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"
"¬(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_simpss [simp]:
"¬(hops rreqid dip dsn dsk oip osn sip).
   msg_zhops (Rreq hops rreqid 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"
"¬(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_simpss [simp]:
"¬(hops rreqid dip dsn dsk oip osn sip).
   rreq_rrep_sn (Rreq hops rreqid 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]:  
 " $\wedge_{\text{hops}} \text{rreqid dip dsn dsk oip osn sip}$ .  
 rreq\_rrep\_fresh crt (Rreq hops rreqid dip dsn dsk oip osn sip) =  
 ( $\text{sip} \neq \text{oip} \rightarrow \text{oip} \in kD(\text{crt})$   
 ∧ (sqn crt oip > osn  
 ∨ (sqn crt oip = osn  
 ∧ the (dhops crt oip) ≤ hops  
 ∧ the (flag crt oip) = val)))"

" $\wedge_{\text{hops}} \text{dip dsn oip sip}$ . rreq\_rrep\_fresh crt (Rrep hops dip dsn oip sip) =  
 ( $\text{sip} \neq \text{dip} \rightarrow \text{dip} \in kD(\text{crt})$   
 ∧ sqn crt dip = dsn  
 ∧ the (dhops crt dip) = hops  
 ∧ the (flag crt dip) = val)"

" $\wedge_{\text{dests}} \text{sip}$ . rreq\_rrep\_fresh crt (Rerr dests sip) = True"  
 " $\wedge_{\text{d dip}}$ . rreq\_rrep\_fresh crt (Newpkt d dip) = True"  
 " $\wedge_{\text{d dip sip}}$ . 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).  
 (rip ∈ iD(crt) ∧ the (destsc ripc) = sqn crt ripc))  
 | \_ ⇒ True"

lemma rerr\_invalid [simp]:  
 " $\wedge_{\text{hops}} \text{rreqid dip dsn dsk oip osn sip}$ .  
 rerr\_invalid crt (Rreq hops rreqid dip dsn dsk oip osn sip) = True"  
 " $\wedge_{\text{hops}} \text{dip dsn oip sip}$ . rerr\_invalid crt (Rrep hops dip dsn oip sip) = True"  
 " $\wedge_{\text{dests}} \text{sip}$ . rerr\_invalid crt (Rerr dests sip) = (∀ rip ∈ dom(dests).  
 rip ∈ iD(crt) ∧ the (dests rip) = sqn crt rip)"  
 " $\wedge_{\text{d dip}}$ . rerr\_invalid crt (Newpkt d dip) = True"  
 " $\wedge_{\text{d dip sip}}$ . 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 ⇒ aodv\_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 = aodv\_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

```

## 3.5 Quality relations between routes

```

theory C_Fresher
imports C_Aodv_Data
begin

```

### 3.5.1 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  $nsqn_r\_\text{def}'$ :

" $nsqn_r r = (\text{if } \pi_4(r) = \text{inv} \text{ then } \pi_2(r) - 1 \text{ else } \pi_2(r))$ "  
*(proof)*

lemma  $nsqn_r\_\text{zero}$  [simp]:

" $\bigwedge dsn\ dsk\ flag\ hops\ nhip. nsqn_r (0, dsk, flag, hops, nhip) = 0$ "  
*(proof)*

lemma  $nsqn_r\_\text{val}$  [simp]:

" $\bigwedge dsn\ dsk\ hops\ nhip. nsqn_r (dsn, dsk, val, hops, nhip) = dsn$ "  
*(proof)*

lemma  $nsqn_r\_\text{inv}$  [simp]:

" $\bigwedge dsn\ dsk\ hops\ nhip. nsqn_r (dsn, dsk, inv, hops, nhip) = dsn - 1$ "  
*(proof)*

lemma  $nsqn_r\_\text{lte\_dsn}$  [simp]:

" $\bigwedge dsn\ dsk\ flag\ hops\ nhip. nsqn_r (dsn, dsk, flag, hops, nhip) \leq dsn$ "  
*(proof)*

#### On routes in routing tables

definition

$nsqn ::= rt \Rightarrow ip \Rightarrow sqn$

where

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

lemma  $nsqn\_\text{sqn}\_\text{def}$ :

" $\bigwedge rt\ dip. nsqn\_\text{sqn}\_\text{def} = (\text{if } \text{flag}\ rt\ dip = \text{Some } \text{val} \vee \text{sqn}\ rt\ dip = 0 \text{ then } \text{sqn}\ rt\ dip \text{ else } \text{sqn}\ rt\ dip - 1)$ "  
*(proof)*

lemma  $not\_\text{in}\_kD\_\text{nsqn}$  [simp]:

assumes " $dip \notin kD(rt)$ "  
shows " $nsqn\ rt\ dip = 0$ "  
*(proof)*

lemma  $kD\_\text{nsqn}$ :

assumes " $dip \in kD(rt)$ "  
shows " $nsqn\ rt\ dip = nsqn_r(\text{the } (\sigma_{\text{route}}(rt, dip)))$ "  
*(proof)*

```

lemma nsqnr_r_flag_pred [simp, intro]:
  fixes dsn dsk flag hops nhip
  assumes "P (nsqn_r (dsn, dsk, val, hops, nhip))"
    and "P (nsqn_r (dsn, dsk, inv, hops, nhip))"
  shows "P (nsqn_r (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' ≡ ((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'
  shows "(dsn, dsk, flag, hops, nhip) ⊑ (dsn, dsk', flag, hops, nhip)"
  ⟨proof⟩

```

### 3.5.3 Comparing routing tables

definition

$rt\_fresher :: "ip \Rightarrow rt \Rightarrow rt \Rightarrow bool"$

where

" $rt\_fresher \equiv \lambda dip\ rt\ rt'. (\text{the } (\sigma_{\text{route}}(rt, dip))) \sqsubseteq (\text{the } (\sigma_{\text{route}}(rt', dip)))$ "

abbreviation

$rt\_fresher\_syn :: "rt \Rightarrow ip \Rightarrow rt \Rightarrow bool" (\langle \_/\sqsubseteq\_ \rangle [51, 999, 51] 50)$

where

" $rt1 \sqsubseteq_i rt2 \equiv rt\_fresher i\ rt1\ rt2$ "

lemma  $rt\_fresher\_def'$ :

" $(rt1 \sqsubseteq_i rt2) = (\text{nsqn}_r (\text{the } (rt1\ i)) < \text{nsqn}_r (\text{the } (rt2\ i)) \vee \text{nsqn}_r (\text{the } (rt1\ i)) = \text{nsqn}_r (\text{the } (rt2\ i)) \wedge \pi_5 (\text{the } (rt2\ i)) \leq \pi_5 (\text{the } (rt1\ i)))$ "

⟨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 ⊑_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 ⊑_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" (⟨/_ ≈/_ _⟩) [51, 999, 51] 50
where
"rt1 ≈_i rt2 ≡ rt_fresh_as i rt1 rt2"

lemma rt_fresh_as_refl [simp]: "¬rt dip. rt ≈_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 ⊑_{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 ] \implies 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_nsqrn:
  assumes "dip ∈ kD(rt1)"
    and "dip ∈ kD(rt2)"
    and "rt1 ≈_dip rt2"
  shows "nsqrn_r (the (rt2 dip)) = nsqrn_r (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 nsqrn_r_invalidate [simp]:
  assumes "dip ∈ kD(rt)"
    and "dip ∈ dom(dests)"
  shows "nsqrn_r (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]

```

### 3.5.4 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 ⊑dip rt2"
shows "the (rt1 dip) ⊑ the (rt2 dip)"
⟨proof⟩

lemma rt_strictly_fresher_not_fresh_asD [dest]:
assumes "rt1 ⊑dip rt2"
shows "¬ rt1 ≈dip rt2"
⟨proof⟩

lemma rt_strictly_fresher_trans [elim, trans]:
assumes "rt1 ⊑dip rt2"
and "rt2 ⊑dip rt3"
shows "rt1 ⊑dip rt3"
⟨proof⟩

lemma rt_strictly_fresher irefl [simp]: "¬ (rt ⊑dip rt)"
⟨proof⟩

lemma rt_fresher_trans_rt_strictly_fresher [elim, trans]:
assumes "rt1 ⊑dip rt2"
and "rt2 ⊑dip rt3"
shows "rt1 ⊑dip rt3"
⟨proof⟩

```

```

lemma rt_fresher_trans_rt_strictly_fresher' [elim, trans]:
  assumes "rt1 ⊑_dip rt2"
    and "rt2 ⊑_dip rt3"
  shows "rt1 ⊑_dip rt3"
  ⟨proof⟩

lemma rt_fresher_imp_nsqn_le:
  assumes "rt1 ⊑_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 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

```

### 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 ≡_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 ≡ (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:
  "paodv i ≡ onl ΓAODV (λ(ξ, 1). (1 = PRrep-:1 →
                                             rt ξ ≠ update(rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ)))"
  ⟨proof⟩
```

Proposition 7.38

```
lemma includes_nhip:
  "paodv i ≡ onl ΓAODV (λ(ξ, 1). ∀ dip ∈ kD(rt ξ). the (nhop(rt ξ) dip) ∈ kD(rt ξ))"
  ⟨proof⟩
```

Proposition 7.4

```
lemma known_destinations_increase:
  "paodv i ≡_A onll ΓAODV (λ((ξ, _), _, (ξ', _)). kD(rt ξ) ⊆ kD(rt ξ'))"
  ⟨proof⟩
```

Proposition 7.5

```
lemma rreqs_increase:
  "paodv i ≡_A onll ΓAODV (λ((ξ, _), _, (ξ', _)). rreqs ξ ⊆ rreqs ξ')"
  ⟨proof⟩

lemma dests_bigger_than_sqn:
  "paodv i ≡ onl ΓAODV (λ(ξ, 1). 1 ∈ {PAodv-:15..PAodv-:17}
                                             ∪ {PPkt-:7..PPkt-:9}
                                             ∪ {PRreq-:9..PRreq-:11}
                                             ∪ {PRreq-:17..PRreq-:19}
                                             ∪ {PRrep-:8..PRrep-:10}
                                             ∪ {PRerr-:1..PRerr-:4} ∪ {PRerr-:6})
                                             → (∀ ip ∈ dom(dests ξ). ip ∈ kD(rt ξ) ∧ sqn(rt ξ) ip ≤ the(dests ξ ip)))"
  ⟨proof⟩
```

Proposition 7.6

```
lemma sqns_increase:
```

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

Proposition 7.7

**lemma** *ip\_constant*:

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

Proposition 7.8

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

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

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

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

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

" $\text{paodv } i \models (\text{recvmsg } P \rightarrow) \ \text{onl } \Gamma_{AODV} (\lambda(\xi, l). \ l \in \{\text{PAodv-:1}\} \longrightarrow P(\text{msg } \xi))$ "  
 $\langle \text{proof} \rangle$

Proposition 7.9

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

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

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

" $\text{paodv } i \models (\text{recvmsg } (\lambda m. \ \text{not\_Pkt } m \longrightarrow \text{msg\_sender } m \neq i) \rightarrow) \ \text{onl } \Gamma_{AODV} (\lambda(\xi, \_). \ \text{sip } \xi \neq i)$ "  
 $\langle \text{proof} \rangle$

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

Proposition 7.10

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

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

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

" $\text{paodv } i \models \text{onl } \Gamma_{AODV} (\lambda(\xi, l). \ (l \in \{\text{PRreq-:14}\} \longrightarrow \text{dip } \xi \in vD(\text{rt } \xi)) \wedge (l \in \{\text{PRreq-:5}, \text{PRreq-:6}\} \longrightarrow \text{dip } \xi = ip \ \xi) \wedge (l \in \{\text{PRreq-:13..PRreq-:14}\} \longrightarrow \text{dip } \xi \neq ip \ \xi))$ "  
 $\langle \text{proof} \rangle$

Proposition 7.11

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

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

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

" $\text{paodv } i \models (\text{recvmsg msg_zhops} \rightarrow) \ \text{onl } \Gamma_{AODV} (\lambda(\xi, l).$   
 $(l \in \{\text{PAodv-:4..PAodv-:5}\} \cup \{\text{PRreq-:n/n. True}\} \longrightarrow (\text{hops } \xi = 0 \longrightarrow \text{oip } \xi = \text{sip } \xi)) \wedge$   
 $((l \in \{\text{PAodv-:6..PAodv-:7}\} \cup \{\text{PRrep-:n/n. True}\} \longrightarrow (\text{hops } \xi = 0 \longrightarrow \text{dip } \xi = \text{sip } \xi)))$ "  
 $\langle \text{proof} \rangle$

**lemma** *osn\_rreq*:

" $\text{paodv } i \models (\text{recvmsg rreq_rrep_sn} \rightarrow) \ \text{onl } \Gamma_{AODV} (\lambda(\xi, l).$   
 $l \in \{\text{PAodv-:4, PAodv-:5}\} \cup \{\text{PRreq-:n/n. True}\} \longrightarrow 1 \leq osn \ \xi)$ "  
 $\langle \text{proof} \rangle$

```

lemma osn_rreq':
  "paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) → onl ΓAODV (λ(ξ, 1).
    1 ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n/n. True}) → 1 ≤ osn ξ)"
  ⟨proof⟩

lemma dsn_rrep:
  "paodv i ⊨ (recvmsg rreq_rrep_sn → onl ΓAODV (λ(ξ, 1).
    1 ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n/n. True}) → 1 ≤ dsn ξ)"
  ⟨proof⟩

lemma dsn_rrep':
  "paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) → onl ΓAODV (λ(ξ, 1).
    1 ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n/n. True}) → 1 ≤ dsn ξ)"
  ⟨proof⟩

lemma hop_count_zero_oip_dip_sip':
  "paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) → onl ΓAODV (λ(ξ, 1).
    (1 ∈ {PAodv-:4..PAodv-:5} ∪ {PRreq-:n/n. True}) →
    (hops ξ = 0 → oip ξ = sip ξ))
    ∧
    ((1 ∈ {PAodv-:6..PAodv-:7} ∪ {PRrep-:n/n. True}) →
    (hops ξ = 0 → dip ξ = sip ξ)))"
  ⟨proof⟩

```

Proposition 7.12

```

lemma zero_seq_unk_hops_one':
  "paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) → onl ΓAODV (λ(ξ, _).
    ∀ dip ∈ kD(rt ξ). (sqn(rt ξ) dip = 0 → sqnf(rt ξ) dip = unk)
    ∧ (sqnf(rt ξ) dip = unk → the(dhops(rt ξ) dip) = 1)
    ∧ (the(dhops(rt ξ) dip) = 1 → the(nhop(rt ξ) dip) = dip)))"
  ⟨proof⟩

lemma zero_seq_unk_hops_one:
  "paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) → onl ΓAODV (λ(ξ, _).
    ∀ dip ∈ kD(rt ξ). (sqn(rt ξ) dip = 0 → (sqnf(rt ξ) dip = unk
    ∧ the(dhops(rt ξ) dip) = 1
    ∧ the(nhop(rt ξ) dip) = dip))))"
  ⟨proof⟩

```

lemma kD\_unk\_or\_atleast\_one:

```

  "paodv i ⊨ (recvmsg rreq_rrep_sn → onl ΓAODV (λ(ξ, 1).
    ∀ dip ∈ kD(rt ξ). π3(the(rt ξ dip)) = unk ∨ 1 ≤ π2(the(rt ξ dip)))"
  ⟨proof⟩

```

Proposition 7.13

```

lemma rreq_rrep_sn_any_step_invariant:
  "paodv i ⊨A (recvmsg rreq_rrep_sn → onll ΓAODV (λ(_, a, _). anycast rreq_rrep_sn a))"
  ⟨proof⟩

```

Proposition 7.14

```

lemma rreq_rrep_fresh_any_step_invariant:
  "paodv i ⊨A onll ΓAODV (λ((ξ, _), a, _). anycast (rreq_rrep_fresh(rt ξ)) a)"
  ⟨proof⟩

```

Proposition 7.15

```

lemma rerr_invalid_any_step_invariant:
  "paodv i ⊨A onll ΓAODV (λ((ξ, _), a, _). anycast (rerr_invalid(rt ξ)) 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:
  "paodv i ⊨
    onl ΓAODV (λ(ξ, 1). (1 ∈ {PAodv-:15, PPkt-:7, PRreq-:9, PRreq-:17, PRrep-:8}
      → ( ∀ ip ∈ dom(dests ξ). ip ∈ vD(rt ξ) ∧ the (dests ξ ip) = inc (sqn (rt ξ) ip)))
      ∧ (1 = PRerr-:1
      → ( ∀ ip ∈ dom(dests ξ). ip ∈ vD(rt ξ) ∧ the (dests ξ ip) > sqn (rt ξ) ip)))"
  ⟨proof⟩

```

Proposition 7.27

```

lemma route_tables_fresher:
  "paodv i ⊨A (recvmsg rreq_rrep_sn → onll ΓAODV (λ((ξ, _), _, (ξ', _)).
    ∀ dip ∈ kD(rt ξ). rt ξ ⊑dip rt ξ'))"
  ⟨proof⟩

```

end

### 3.7 The quality increases predicate

```

theory C_Quality_Increases
imports C_Aodv_Predicates C_Fresher
begin

definition quality_increases :: "state ⇒ state ⇒ bool"
where "quality_increases ξ ξ' ≡ ( ∀ dip ∈ kD(rt ξ). dip ∈ kD(rt ξ') ∧ rt ξ ⊑dip rt ξ')
  ∧ ( ∀ dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip)"

lemma quality_increasesI [intro!]:
  assumes " ∀ dip. dip ∈ kD(rt ξ) ⇒ dip ∈ kD(rt ξ')"
  and " ∀ dip. [ dip ∈ kD(rt ξ); dip ∈ kD(rt ξ') ] ⇒ rt ξ ⊑dip rt ξ"
  and " ∀ dip. sqn (rt ξ) dip ≤ sqn (rt ξ') dip"
  shows "quality_increases ξ ξ"
  ⟨proof⟩

lemma quality_increasesE [elim]:
  fixes dip
  assumes "quality_increases ξ ξ'"
  and "dip ∈ kD(rt ξ)"
  and "[ dip ∈ kD(rt ξ'); rt ξ ⊑dip rt ξ'; sqn (rt ξ) dip ≤ sqn (rt ξ') dip ] ⇒ R dip ξ ξ'"
  shows "R dip ξ ξ"
  ⟨proof⟩

lemma quality_increases_rt_fresherD [dest]:
  fixes ip
  assumes "quality_increases ξ ξ'"
  and "ip ∈ kD(rt ξ)"
  shows "rt ξ ⊑ip rt ξ"
  ⟨proof⟩

lemma quality_increases_sqnE [elim]:
  fixes dip
  assumes "quality_increases ξ ξ'"
  and "sqn (rt ξ) dip ≤ sqn (rt ξ') dip ⇒ R dip ξ ξ'"
  shows "R dip ξ ξ"
  ⟨proof⟩

lemma quality_increases_refl [intro, simp]: "quality_increases ξ ξ"

```

*(proof)*

```
lemma strictly_fresher_quality_increases_right [elim]:  
  fixes  $\sigma \sigma' dip$   
  assumes "rt ( $\sigma i$ ) \sqsubset_{dip} rt ( $\sigma nhip$ )"  
    and qinc: "quality_increases ( $\sigma nhip$ ) ( $\sigma' nhip$ )"  
    and "dip \in kD(rt ( $\sigma nhip$ ))"  
  shows "rt ( $\sigma i$ ) \sqsubset_{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  
      \longrightarrow (the(dhops(rt ( $\sigma sip$ )) ip) \leq hops  
      \vee the(flag(rt ( $\sigma sip$ )) ip) = inv))"  
(proof)
```

```

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))"
      (is "_ ∧ ?nsqnafter")
⟨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."
 "¬¬d dip sip." 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 ==> recvmsg 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

```

## 3.8 The ‘open’ AODV model

```

theory C_Oadv
imports C_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 ⇒ ((ip ⇒ state) × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
  "opaodv i ≡ () init = σAODV', trans = oseqp_sos ΓAODV i ()"

lemma initiali_aodv [intro!, simp]: "initiali i (init (opaodv i)) (init (paodv i))"
  ⟨proof⟩

lemma oaodv_control_within [simp]: "control_within ΓAODV (init (opaodv i))"
  ⟨proof⟩

lemma σAODV'_labels [simp]: "(σ, p) ∈ σAODV' ==> labels ΓAODV p = {PAodv-:0}"
  ⟨proof⟩

lemma oaodv_init_kD_empty [simp]:

```

```

"(σ, p) ∈ σAODV' ⟹ kD(rt(σ i)) = {}"
⟨proof⟩

lemma oaodv_init_vD_empty [simp]:
"(σ, p) ∈ σAODV' ⟹ vD(rt(σ i)) = {}"
⟨proof⟩

lemma oaodv_trans: "trans(oaodv i) = oseqp_sos ΓAODV 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 σ σ'"
shows "∀j. quality_increases (σ j) (σ' j)"
⟨proof⟩

lemma weaken_otherwith [elim]:
fixes m
assumes *: "otherwith P I (orecvmsg Q) σ σ' a"
  and weakenP: "¬σ m. P σ m ⟹ P' σ m"
  and weakenQ: "¬σ m. Q σ m ⟹ Q' σ m"
shows "otherwith P' I (orecvmsg Q') σ σ' a"
⟨proof⟩

lemma oreceived_msg_inv:
assumes other: "¬σ σ' m. [ P σ m; other Q {i} σ σ' ] ⟹ P σ' m"
  and local: "¬σ m. P σ m ⟹ P (σ(i := σ i | msg := m)) m"
shows "opaodv i ≡ (otherwith Q {i} (orecvmsg P), other Q {i} →)
          onl ΓAODV (λ(σ, l). l ∈ {PAodv-:1} → P σ (msg (σ i)))"
⟨proof⟩

(Equivalent to) Proposition 7.27

lemma local_quality_increases:
"paodv i ≡A (orecvmsg rreq_rrep_sn →) onll ΓAODV (λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ')"
⟨proof⟩

lemmas olocal_quality_increases =
open_seq_stepInvariant [OF local_quality_increases initiali_aodv oaodv_trans aodv_trans,
simplified seqll_onll_swap]

lemma oquality_increases:
"opaodv i ≡A (otherwith quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)),
other_quality_increases {i} →)
onll ΓAODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j))"
(is "_ ≡A (?S, _ →) _")
⟨proof⟩

lemma rreq_rrep_nsqn_fresh_any_step_invariant:

```

```

"opaodv i ⊨_A (act (recvmsg rreq_rrep_sn), other A {i} →)
    onl ΓAODV (λ((σ, _), a, _). anycast (msg_fresh σ) a)"
⟨proof⟩

lemma oreceived_rreq_rrep_nsqn_fresh_inv:
"opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i} →)
    onl ΓAODV (λ((σ, 1). l ∈ {PAodv-:1} → msg_fresh σ (msg (σ i))))"
⟨proof⟩

lemma oquality_increases_nsqn_fresh:
"opaodv i ⊨_A (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i} →)
    onl ΓAODV (λ((σ, _, (σ', _)). ∀ j. quality_increases (σ j) (σ' j)))"
⟨proof⟩

lemma oosn_rreq:
"opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i} →)
    onl ΓAODV (seql i (λ(ξ, 1). l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n | n. True} → 1 ≤ osn ξ))"
⟨proof⟩

lemma rreq_sip:
"opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i} →)
    onl ΓAODV (λ((σ, 1).
        (l ∈ {PAodv-:4, PAodv-:5, PRreq-:0, PRreq-:2} ∧ sip (σ i) ≠ oip (σ i))
        → oip (σ i) ∈ kD(rt (σ (sip (σ i))))
        ∧ nsqn (rt (σ (sip (σ i)))) (oip (σ i)) ≥ osn (σ i)
        ∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i)) = osn (σ i))
        → (hops (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (oip (σ i)))
            ∨ the (flag (rt (σ (sip (σ i)))) (oip (σ i))) = inv)))"
(is "_ ⊨ (?S, ?U →) _")
⟨proof⟩

lemma odsn_rrep:
"opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i} →)
    onl ΓAODV (seql i (λ(ξ, 1). l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n | n. True} → 1 ≤ dsn ξ))"
⟨proof⟩

lemma rrep_sip:
"opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i} →)
    onl ΓAODV (λ((σ, 1).
        (l ∈ {PAodv-:6, PAodv-:7, PRrep-:0, PRrep-:1} ∧ sip (σ i) ≠ dip (σ i))
        → dip (σ i) ∈ kD(rt (σ (sip (σ 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:
"opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i} →)
    onl ΓAODV (λ((σ, 1).
        l ∈ {PAodv-:8, PAodv-: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: "paodv i ⊨
    onl ΓAODV (λ(ξ, 1). (1 = 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 odests_vD_inc_sqn =
open_seq_invariant [OF dests_vD_inc_sqn initiali_aodv oaodv_trans aodv_trans,
simplified seql_onl_swap,
THEN oinvariant_anyact]

```

```

lemmas oprerr_guard =
open_seq_invariant [OF prerr_guard initiali_aodv oaodv_trans aodv_trans,
simplified seql_onl_swap,
THEN oinvariant_anyact]

```

Proposition 7.28

```

lemma seq_compare_next_hop':
"opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} → onl ΓAODV (λ(σ, _).
    ∀ 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)"
(is "_ ⊨ (?S, ?U →) _")

```

*(proof)*

Proposition 7.30

```

lemmas okD_unk_or_atleast_one =
open_seq_invariant [OF kD_unk_or_atleast_one initiali_aodv,
simplified seql_onl_swap]

```

```

lemmas ozero_seq_unk_hops_one =
open_seq_invariant [OF zero_seq_unk_hops_one initiali_aodv,
simplified seql_onl_swap]

```

```

lemma oreachable_fresh_okD_unk_or_atleast_one:
fixes dip
assumes "(σ, p) ∈ oreachable (opaodv i)
    (otherwith ((=)) {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 (opaodv i)
    (otherwith ((=)) {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 "opaodv i ⊨ (otherwith ((=)) {i}
    (orecvmsg (λσ m. msg_fresh σ m ∧ msg_zhops m)),

```

```

other quality_increases {i} →)
onl  $\Gamma_{AO DV}$  ( $\lambda(\sigma, \_)$ ).  $\forall dip$ . let  $nhip = \text{the } (nhop (rt (\sigma i)) dip)$ 
in  $dip \in vD (rt (\sigma i)) \cap vD (rt (\sigma nhip))$ 
 $\wedge nhip \neq dip$ 
 $\longrightarrow (rt (\sigma i)) \sqsubset_{dip} (rt (\sigma nhip)))$ ""
(is " $\_ \models (?S i, \_ \rightarrow) \_$ ")
⟨proof⟩

lemma seq_compare_next_hop:
fixes w
shows "opaodv i  $\models$  (otherwith ((=)) {i} (orecvmsg msg_fresh),
other quality_increases {i} →)
global ( $\lambda\sigma$ .  $\forall dip$ . let  $nhip = \text{the } (nhop (rt (\sigma i)) dip)$ 
in  $dip \in kD(rt (\sigma i)) \wedge nhip \neq dip \longrightarrow$ 
 $dip \in kD(rt (\sigma nhip))$ 
 $\wedge nsqn (rt (\sigma i)) dip \leq nsqn (rt (\sigma nhip)) dip)$ ""
⟨proof⟩

lemma seq_nhop_quality_increases:
shows "opaodv i  $\models$  (otherwith ((=)) {i}
(orecvmsg ( $\lambda\sigma m$ . msg_fresh  $\sigma m \wedge msg\_zhops m$ )),
other quality_increases {i} →)
global ( $\lambda\sigma$ .  $\forall dip$ . let  $nhip = \text{the } (nhop (rt (\sigma i)) dip)$ 
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⟩
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**

$rt\_graph :: "(ip \Rightarrow state) \Rightarrow ip \Rightarrow ip \text{ rel}"$

**where**

" $rt\_graph \sigma = (\lambda dip.$   
 $\{(ip, ip') \mid ip \neq ip' \wedge rt (\sigma ip) dip = Some (dsn, dsk, val, hops, ip')\})$ "

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
 $\wedge (\exists dsn dsk hops. r dip = Some (dsn, dsk, val, hops, ip')))$ ""
⟨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]:
  " $\bigwedge ip\ ip' \sigma dip. (ip, ip') \in rt\_graph \sigma dip \implies ip' = \text{the}(nhop(rt(\sigma ip)) dip)$ "
(proof)
```

```
theorem inv_to_loop_freedom:
  assumes " $\forall i\ dip. \text{let } nhop = \text{the}(nhop(rt(\sigma i)) dip)$ 
           $\text{in } dip \in vD(rt(\sigma i)) \cap vD(rt(\sigma nhop)) \wedge nhop \neq dip$ 
           $\longrightarrow (rt(\sigma i)) \sqsubset_{dip} (rt(\sigma nhop))$ "
  shows " $\forall dip. \text{irrefl}((rt\_graph \sigma dip)^+)$ "
(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 i qmsg \models (\text{otherwith}((=)) \{i\} (\text{orecvmsg } (\lambda \sigma m.  

           msg\_fresh \sigma m \wedge msg\_zhops m)),  

           other quality_increases \{i\} \rightarrow)  

         global (\lambda \sigma. \forall dip. \text{let } nhop = \text{the}(nhop(rt(\sigma i)) dip)  

           in dip \in vD(rt(\sigma i)) \cap vD(rt(\sigma nhop)) \wedge nhop \neq dip  

           \longrightarrow (rt(\sigma i)) \sqsubset_{dip} (rt(\sigma nhop)))"
(proof)
```

```
lemma par_rreq_rrep_sn_quality_increases:
  "opaodv i \langle i qmsg \models_A (\lambda \sigma_. \text{orecvmsg } (\lambda \_. rreq\_rrep\_sn) \sigma, \text{other } (\lambda \_. \text{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 i qmsg \models_A (\lambda \sigma_. \text{orecvmsg } (\lambda \_. rreq\_rrep\_sn) \sigma,  

           other (\lambda \_. \text{True}) \{i\} \rightarrow)  

           globala (\lambda (\sigma, a, \sigma'). anycast (msg_fresh \sigma) a)"
(proof)
```

```
lemma par_anycast_msg_zhops:
  shows "opaodv i \langle i qmsg \models_A (\lambda \sigma_. \text{orecvmsg } (\lambda \_. rreq\_rrep\_sn) \sigma, \text{other } (\lambda \_. \text{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 " $((\sigma, \text{NodeS } i P R), a, (\sigma', \text{NodeS } i' P' R')) \in onode\_sos$ 
```

```
(oparp_sos i (oseqp_sos  $\Gamma_{AODV}$  i) (seqp_sos  $\Gamma_{QMSG}$ ))"
```

and "a ≠ τ"

shows " $\sigma' \cdot i = \sigma \cdot i$ "

$\langle proof \rangle$

lemma node\_nhop\_quality\_increases:

```
shows " $\langle i : opaodv i \langle \langle i \ qmsg : R \rangle_o \models$ 
      (otherwith ((=)) {i})
      (oarrivemsg ( $\lambda \sigma \ m. \ msg\_fresh \ \sigma \ m \wedge msg\_zhops \ m$ )),
      other quality_increases {i}
 $\rightarrow \rangle \ global \ (\lambda \sigma. \ \forall dip. \ let \ nhip \ = \ the \ (nhop \ (rt \ (\sigma \ i)) \ dip)$ 
      in  $dip \in vD \ (rt \ (\sigma \ i)) \cap vD \ (rt \ (\sigma \ nhip)) \wedge nhip \neq dip$ 
       $\longrightarrow \ (rt \ (\sigma \ i)) \sqsubset_{dip} \ (rt \ (\sigma \ nhip)))$ "
```

$\langle proof \rangle$

lemma node\_quality\_increases:

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

$\langle proof \rangle$

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

```
shows " $\langle i : opaodv i \langle \langle i \ qmsg : R \rangle_o \models_A$ 
      ( $\lambda \sigma \_. \ oarrivemsg \ (\lambda \_. \ rreq\_rrep\_sn) \ \sigma$ , other ( $\lambda \_. \ True$ ) {i}  $\rightarrow$ )
 $\globala \ (\lambda (\sigma, \ a, \ \sigma'). \ castmsg \ (msg\_fresh \ \sigma) \ a)$ "
```

$\langle proof \rangle$

lemma node\_anycast\_msg\_zhops:

```
shows " $\langle i : opaodv i \langle \langle i \ qmsg : R \rangle_o \models_A$ 
      ( $\lambda \sigma \_. \ oarrivemsg \ (\lambda \_. \ rreq\_rrep\_sn) \ \sigma$ , other ( $\lambda \_. \ True$ ) {i}  $\rightarrow$ )
 $\globala \ (\lambda (a, \ a, \ a). \ castmsg \ msg\_zhops \ a)$ "
```

$\langle proof \rangle$

lemma node\_silent\_change\_only:

```
shows " $\langle i : opaodv i \langle \langle i \ qmsg : R_i \rangle_o \models_A$ 
      ( $\lambda \sigma \_. \ oarrivemsg \ (\lambda \_. \ True) \ \sigma$ ,
       other ( $\lambda \_. \ True$ ) {i}  $\rightarrow$ )
 $\globala \ (\lambda (\sigma, \ a, \ \sigma'). \ a \neq \tau \longrightarrow \sigma' \cdot i = \sigma \cdot i)$ "
```

$\langle proof \rangle$

### 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"
```

$\langle proof \rangle$

lemma opnet\_nhop\_quality\_increases:

```
shows "opnet ( $\lambda i. \ opaodv i \langle \langle i \ qmsg \rangle p \models$ 
      (otherwith ((=)) (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  $dip \in vD \ (rt \ (\sigma \ i)) \cap vD \ (rt \ (\sigma \ nhip)) \wedge nhip \neq dip$ 
       $\longrightarrow \ (rt \ (\sigma \ i)) \sqsubset_{dip} \ (rt \ (\sigma \ nhip)))$ "
```

$\langle proof \rangle$

### 3.11.4 Lift to closed networks

lemma onet\_nhop\_quality\_increases:

```
shows "oclosed (opnet ( $\lambda i. \ opaodv i \langle \langle i \ qmsg \rangle p$ )
 $\models \ (\lambda \_. \ True, \ other \ quality\_increases \ (net\_tree\_ips \ p) \ \rightarrow$ )
 $\global \ (\lambda \sigma. \ \forall i \in net\_tree\_ips \ p. \ \forall dip.$ 
      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 aodv_openproc: openproc paodv opaodv id
  rewrites "aodv_openproc.initmissing = initmissing"
  ⟨proof⟩

```

```

interpretation aodv_openproc_par_qmsg: openproc_paq paodv opaodv id qmsg
  rewrites "aodv_openproc_par_qmsg.netglobal = netglobal"
  and "aodv_openproc_par_qmsg.initmissing = initmissing"
  ⟨proof⟩

```

```

lemma net_nhop_quality_increases:
  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paodv i ⟨⟨ qmsg ⟩⟩ n) ⊨ netglobal
                (λσ. ∀ i dip. let nhop = the (nhop (rt(σ i)) dip)
                           in dip ∈ vD(rt(σ i)) ∩ vD(rt(σ nhop)) ∧ nhop ≠ dip
                           → (rt(σ i)) ⊓dip (rt(σ nhop)))"
  (is "_ ⊨ netglobal (λσ. ∀ i. ?inv σ i)")
  ⟨proof⟩

```

### 3.11.6 Loop freedom of AODV

```

theorem aodv_loop_freedom:
  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paodv 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 ⇒ sqn"
  where "inc sn ≡ if sn = 0 then sn else sn + 1"

lemma less_than_inc [simp]: "x ≤ inc x"
  ⟨proof⟩

lemma inc_minus_suc_0 [simp]:
  "inc x - Suc 0 = x"
  ⟨proof⟩

lemma inc_never_one' [simp, intro]: "inc x ≠ Suc 0"
  ⟨proof⟩

lemma inc_never_one [simp, intro]: "inc x ≠ 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 × k × f × nat × ip × ip set"
definition proj2 :: "r ⇒ sqn" (⟨π₂⟩)
```

```

where " $\pi_2 \equiv \lambda(dsn, \_, \_, \_, \_, \_, \_). dsn$ "  

definition proj3 :: "r  $\Rightarrow$  k" ( $\langle\pi_3\rangle$ )
  where " $\pi_3 \equiv \lambda(\_, dsk, \_, \_, \_, \_, \_). dsk$ "  

definition proj4 :: "r  $\Rightarrow$  f" ( $\langle\pi_4\rangle$ )
  where " $\pi_4 \equiv \lambda(\_, \_, flag, \_, \_, \_, \_). flag$ "  

definition proj5 :: "r  $\Rightarrow$  nat" ( $\langle\pi_5\rangle$ )
  where " $\pi_5 \equiv \lambda(\_, \_, \_, hops, \_, \_, \_). hops$ "  

definition proj6 :: "r  $\Rightarrow$  ip" ( $\langle\pi_6\rangle$ )
  where " $\pi_6 \equiv \lambda(\_, \_, \_, \_, nhip, \_). nhip$ "  

definition proj7 :: "r  $\Rightarrow$  ip set" ( $\langle\pi_7\rangle$ )
  where " $\pi_7 \equiv \lambda(\_, \_, \_, \_, \_, pre). pre$ "  

lemma projs [simp]:
  " $\pi_2(dsn, dsk, flag, hops, nhip, pre) = dsn$ "
  " $\pi_3(dsn, dsk, flag, hops, nhip, pre) = dsk$ "
  " $\pi_4(dsn, dsk, flag, hops, nhip, pre) = flag$ "
  " $\pi_5(dsn, dsk, flag, hops, nhip, pre) = hops$ "
  " $\pi_6(dsn, dsk, flag, hops, nhip, pre) = nhip$ "
  " $\pi_7(dsn, dsk, flag, hops, nhip, pre) = pre$ "  

   $\langle proof \rangle$   

lemma proj3_pred [intro]: " $\llbracket P \text{ kno}; P \text{ unk} \rrbracket \implies P (\pi_3 x)$ "  

   $\langle proof \rangle$   

lemma proj4_pred [intro]: " $\llbracket P \text{ val}; P \text{ inv} \rrbracket \implies P (\pi_4 x)$ "  

   $\langle proof \rangle$   

lemma proj6_pair_snd [simp]:
  fixes dsn' r
  shows " $\pi_6(dsn', snd(r)) = \pi_6(r)$ "  

   $\langle proof \rangle$ 

```

#### 4.1.3 Routing Tables

Routing tables map ip addresses to route entries.

```

type_synonym rt = "ip  $\rightarrow$  r"  

syntax
  " $\_Sigma\_route$ " :: "rt  $\Rightarrow$  ip  $\rightarrow$  r" ( $\langle\sigma_{route}(\_, \_) \rangle$ )  

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  $\pi_6(\sigma_{route}(rt, dip))$ "  

abbreviation precs :: "rt  $\Rightarrow$  ip  $\rightarrow$  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⟩

```

#### 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: " $\text{addpre } r \{ \} = r$ "
⟨proof⟩

lemma addpre_r:
  " $\text{addpre } (\text{dsn}, \text{dsk}, \text{f1}, \text{hops}, \text{nhip}, \text{pre}) \text{ npre} = (\text{dsn}, \text{dsk}, \text{f1}, \text{hops}, \text{nhip}, \text{pre} \cup \text{npre})$ "
  ⟨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)) (σ_{route}(rt, dip))"

lemma snd_addpre [simp]:
  " $\bigwedge \text{dsn } \text{dsn}' \text{ v } \text{pre}. (\text{dsn}, \text{snd}(\text{addpre } (\text{dsn}', \text{ v }) \text{ pre})) = \text{addpre } (\text{dsn}, \text{ v }) \text{ pre}$ "
  ⟨proof⟩

lemma proj2_addpreRT [simp]:
  fixes ip rt ip' npre
  assumes "ip ∈ kD rt"
    and "ip' ∈ kD rt"
  shows " $\pi_2(\text{the } (\text{the } (\text{addpreRT } rt \text{ ip' npre}) \text{ ip})) = \pi_2(\text{the } (rt \text{ ip}))$ "
  ⟨proof⟩

lemma proj3_addpreRT [simp]:
  fixes ip rt ip' npre
  assumes "ip ∈ kD rt"
    and "ip' ∈ kD rt"
  shows " $\pi_3(\text{the } (\text{the } (\text{addpreRT } rt \text{ ip' npre}) \text{ ip})) = \pi_3(\text{the } (rt \text{ ip}))$ "
  ⟨proof⟩

lemma proj5_addpreRT [simp]:
  " $\bigwedge \text{rt } \text{dip } \text{ip } \text{npre}. \text{dip} \in kD(rt) \implies \pi_5(\text{the } (\text{the } (\text{addpreRT } rt \text{ dip npre}) \text{ ip})) = \pi_5(\text{the } (rt \text{ ip}))$ "
  ⟨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]:
  " $\bigwedge ip\ dip. ip \in kD(rt) \Rightarrow sqnf (the (addpreRT (rt) ip npre)) dip = sqnf (rt) dip$ "
  ⟨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 ⟧"

```

$\implies \text{update } rt \ ip \ r = rt \ (\text{ip} \mapsto ns)$

$\langle proof \rangle$

```

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))"

```

$\langle proof \rangle$

```

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))"

```

$\langle proof \rangle$

```

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)"

```

$\langle proof \rangle$

```

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"

```

$\langle proof \rangle$

```

lemma sqn_if_updated:
  fixes rip v rt ip
  shows "sqn(\lambda x. if x = rip then Some v else rt x) ip
    = (if ip = rip then \pi_2(v) else sqn rt ip)"

```

$\langle proof \rangle$

```

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"

```

$\langle proof \rangle$

```

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]: " $\lambda rt\ dip\ dsn\ dsk\ flg\ hops\ sip.$ 
   $rt \neq update\ rt\ dip\ (dsn,\ dsk,\ flg,\ hops,\ sip,\ {})$ 
   $\implies sqnf\ (update\ rt\ dip\ (dsn,\ dsk,\ flg,\ hops,\ sip,\ {}))\ dip = dsk$ "
  ⟨proof⟩

lemma update_kno_dsn_greater_zero:
  " $\lambda rt\ dip\ ip\ dsn\ hops\ npre.$   $1 \leq dsn \implies 1 \leq (sqn\ (update\ rt\ dip\ (dsn,\ kno,\ val,\ hops,\ ip,\ npre))\ dip)$ "
  ⟨proof⟩

lemma proj3_update [simp]: " $\lambda rt\ dip\ dsn\ dsk\ flg\ hops\ sip.$ 
   $rt \neq update\ rt\ dip\ (dsn,\ dsk,\ flg,\ hops,\ sip,\ {})$ 
   $\implies \pi_3(\text{the}\ (\text{update}\ rt\ dip\ (dsn,\ dsk,\ flg,\ hops,\ sip,\ {})\ dip)) = dsk$ "
  ⟨proof⟩

lemma nhop_update_changed_kno_val [simp]: " $\lambda rt\ ip\ dsn\ dsk\ hops\ nhip.$ 
   $rt \neq update\ rt\ ip\ (dsn,\ kno,\ val,\ hops,\ nhip,\ {})$ 
   $\implies \text{the}\ (\text{nhop}\ (\text{update}\ rt\ ip\ (dsn,\ kno,\ val,\ hops,\ nhip,\ {}))\ ip) = nhip$ "
  ⟨proof⟩

lemma flag_update [simp]: " $\lambda rt\ dip\ dsn\ flg\ hops\ sip.$ 
   $rt \neq update\ rt\ dip\ (dsn,\ kno,\ flg,\ hops,\ sip,\ {})$ 
   $\implies \text{the}\ (\text{flag}\ (\text{update}\ rt\ dip\ (dsn,\ kno,\ flg,\ hops,\ sip,\ {}))\ dip) = flg$ "
  ⟨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]: " $\lambda rt\ dip\ dsn\ dsk\ flg\ hops\ sip.$ 
   $rt \neq update\ rt\ dip\ (dsn,\ dsk,\ flg,\ hops,\ sip,\ {})$ 
   $\implies \text{the}\ (\text{nhop}\ (\text{update}\ rt\ dip\ (dsn,\ dsk,\ flg,\ hops,\ sip,\ {}))\ dip) = sip$ "
  ⟨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]:
  " $\lambda dip\ rt\ dip'\ dsn\ dsk\ hops\ nhip\ pre.$ 
   $dip \in vD(\text{update}\ rt\ dip'\ (dsn,\ dsk,\ val,\ hops,\ nhip,\ pre)) \implies (dip \in vD(rt) \vee dip = dip')$ "
  ⟨proof⟩

```

## Invalidating route entries

```

definition invalidate :: "rt ⇒ (ip → sqn) ⇒ rt"
where "invalidate rt dests ≡
  λip. case (rt ip, dests ip) of

```

```

    (None, _) ⇒ None
  / (Some s, None) ⇒ Some s
  / (Some (_, dsk, _, hops, nhip, pre), Some rsn) ⇒
    Some (rsn, dsk, inv, hops, nhip, 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"

```

$\langle proof \rangle$

```
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 ⇒ 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))
    ∧ pre = π7(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⟩
```

#### 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 " $\sigma_{p\text{-}flag}(store, dip) \equiv \text{map\_option } \text{fst } (store \ dip)$ "
```

```

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.  $\sigma_{p\text{-}flag}(store, dip)$  is partial.
4. The routing table (*rt*) is modelled as a map (*ip  $\Rightarrow$  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

## 4.2 AODV protocol messages

```

theory D_Aodv_Message
imports D_Fwdreqs
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  $\langle proof \rangle$ 
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"
   $\langle proof \rangle$ 
```

```

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 aodv_init :: "ip ⇒ state"
where "aodv_init i ≡ ()"
  ip = i,
  sn = 1,
  rt = Map.empty,
  rreqs = {},
  store = Map.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),
  handled= (SOME x. True)
()

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),
  handled:= (SOME x. True)
()

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⟩

```

#### 4.3.2 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

```

$Pkt\ data', dip', oip' \Rightarrow \{ \xi | data := data', dip := dip', oip := oip' \} \}$   
 $| \_ \Rightarrow \{\}$

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]:
  " $\xi' \in \text{is\_rerr } \xi \implies \text{rt } \xi' = \text{rt } \xi$ "
  " $\xi' \in \text{is\_rrep } \xi \implies \text{rt } \xi' = \text{rt } \xi$ "
  " $\xi' \in \text{is\_rreq } \xi \implies \text{rt } \xi' = \text{rt } \xi$ "
  " $\xi' \in \text{is\_pkt } \xi \implies \text{rt } \xi' = \text{rt } \xi$ "
  " $\xi' \in \text{is\_newpkt } \xi \implies \text{rt } \xi' = \text{rt } \xi$ "
  ⟨proof⟩

lemma is_msg_inv_rreqs [simp]:
  " $\xi' \in \text{is\_rerr } \xi \implies \text{rreqs } \xi' = \text{rreqs } \xi$ "
  " $\xi' \in \text{is\_rrep } \xi \implies \text{rreqs } \xi' = \text{rreqs } \xi$ "
  " $\xi' \in \text{is\_rreq } \xi \implies \text{rreqs } \xi' = \text{rreqs } \xi$ "
  " $\xi' \in \text{is\_pkt } \xi \implies \text{rreqs } \xi' = \text{rreqs } \xi$ "
  " $\xi' \in \text{is\_newpkt } \xi \implies \text{rreqs } \xi' = \text{rreqs } \xi$ "
  ⟨proof⟩

lemma is_msg_inv_store [simp]:
  " $\xi' \in \text{is\_rerr } \xi \implies \text{store } \xi' = \text{store } \xi$ "
  " $\xi' \in \text{is\_rrep } \xi \implies \text{store } \xi' = \text{store } \xi$ "
  " $\xi' \in \text{is\_rreq } \xi \implies \text{store } \xi' = \text{store } \xi$ "
  " $\xi' \in \text{is\_pkt } \xi \implies \text{store } \xi' = \text{store } \xi$ "
  " $\xi' \in \text{is\_newpkt } \xi \implies \text{store } \xi' = \text{store } \xi$ "
  ⟨proof⟩

```

```

lemma is_msg_inv_sip [simp]:
  " $\xi' \in \text{is\_pkt } \xi \implies \text{sip } \xi' = \text{sip } \xi$ "
  " $\xi' \in \text{is\_newpkt } \xi \implies \text{sip } \xi' = \text{sip } \xi$ "
  ⟨proof⟩

```

#### 4.3.3 The protocol process

```

datatype pseqp =
  PAodv
  | PNewPkt
  | PPkt
  | PRreq
  | PRrep
  | PRerr

fun nat_of_seqp :: "pseqp ⇒ nat"
where
  "nat_of_seqp PAodv = 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(PAodv)"

abbreviation PKT
where
  "PKT args ≡
    [ξ. let (data, dip, oip) = args ξ in
      ...]
    ..."

```

```

  (clear_locals  $\xi$ ) () data := data, dip := dip, oip := oip )]
call(PPkt)"
abbreviation NEWPKT
where
"NEWPKT args =
 $\llbracket \xi. \text{let } (\text{data}, \text{dip}) = \text{args } \xi \text{ in}$ 
  (clear_locals  $\xi$ ) () data := data, dip := dip )]
call(PNewPkt)"
```

abbreviation RREQ

where

```
"RREQ args =
 $\llbracket \xi. \text{let } (\text{hops}, \text{rreqid}, \text{dip}, \text{dsn}, \text{dsk}, \text{oip}, \text{osn}, \text{sip}, \text{handled}) = \text{args } \xi \text{ in}$ 
  (clear_locals  $\xi$ ) () 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 =
 $\llbracket \xi. \text{let } (\text{hops}, \text{dip}, \text{dsn}, \text{oip}, \text{sip}) = \text{args } \xi \text{ in}$ 
  (clear_locals  $\xi$ ) () hops := hops, dip := dip, dsn := dsn,
  oip := oip, sip := sip )]
call(PRrep)"
```

abbreviation RERR

where

```
"RERR args =
 $\llbracket \xi. \text{let } (\text{dests}, \text{sip}) = \text{args } \xi \text{ in}$ 
  (clear_locals  $\xi$ ) () dests := dests, sip := sip )]
call(PRerr)"
```

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

where

```
" $\Gamma_{AODV}$  PAodv = labelled PAodv (
  receive( $\lambda \text{msg}'$   $\xi. \xi$  () msg := msg' )).
  ( $\langle \text{is\_newpkt} \rangle$  NEWPKT( $\lambda \xi.$  (data  $\xi$ , ip  $\xi$ ))
   $\oplus \langle \text{is\_pkt} \rangle$  PKT( $\lambda \xi.$  (data  $\xi$ , dip  $\xi$ , oip  $\xi$ ))
   $\oplus \langle \text{is\_rreq} \rangle$ 
     $\llbracket \xi. \xi$  () rt := update(rt  $\xi$ ) (sip  $\xi$ ) (0, unk, val, 1, sip  $\xi$ , {}) )
    RREQ( $\lambda \xi.$  (hops  $\xi$ , rreqid  $\xi$ , dip  $\xi$ , dsn  $\xi$ , dsk  $\xi$ , oip  $\xi$ , osn  $\xi$ , sip  $\xi$ , handled  $\xi$ ))
   $\oplus \langle \text{is\_rrep} \rangle$ 
     $\llbracket \xi. \xi$  () rt := update(rt  $\xi$ ) (sip  $\xi$ ) (0, unk, val, 1, sip  $\xi$ , {})
    RREP( $\lambda \xi.$  (hops  $\xi$ , dip  $\xi$ , dsn  $\xi$ , oip  $\xi$ , sip  $\xi$ ))
   $\oplus \langle \text{is\_rerr} \rangle$ 
     $\llbracket \xi. \xi$  () rt := update(rt  $\xi$ ) (sip  $\xi$ ) (0, unk, val, 1, sip  $\xi$ , {})
    RERR( $\lambda \xi.$  (dests  $\xi$ , sip  $\xi$ ))
)
 $\oplus \langle \lambda \xi. \{ \xi | dip := dip \} | dip. dip \in qD(\text{store } \xi) \cap vD(rt \xi) \}$ 
 $\llbracket \xi. \xi$  () data := hd( $\sigma_{\text{queue}}(\text{store } \xi, \text{dip } \xi)$ )
unicast( $\lambda \xi.$  the(nhop(rt  $\xi$ ) (dip  $\xi$ )),  $\lambda \xi.$  pkt(data  $\xi$ , dip  $\xi$ , ip  $\xi$ )).
 $\llbracket \xi. \xi$  () store := the(drop(dip  $\xi$ ) (store  $\xi$ ))
AODV()
 $\triangleright \llbracket \xi. \xi$  () dests := ( $\lambda rip.$  if (rip  $\in vD(rt \xi) \wedge nhop(rt \xi) rip = nhop(rt \xi) (dip \xi)$ )
then Some(inc(sqn(rt  $\xi$ ) rip)) else None)
 $\llbracket \xi. \xi$  () rt := invalidate(rt  $\xi$ ) (dests  $\xi$ )
 $\llbracket \xi. \xi$  () store := setRRF(store  $\xi$ ) (dests  $\xi$ )
 $\llbracket \xi. \xi$  () pre :=  $\bigcup \{ \text{the}(\text{precs}(rt \xi) rip) | rip. rip \in \text{dom}(\text{dests } \xi) \}$ 
 $\llbracket \xi. \xi$  () dests := ( $\lambda rip.$  if ((dests  $\xi$ ) rip  $\neq$  None  $\wedge$  the(preces(rt  $\xi$ ) rip)  $\neq$  {}) then (dests  $\xi$ ) rip else None)
groupcast( $\lambda \xi.$  pre  $\xi$ ,  $\lambda \xi.$  rerr(dests  $\xi$ , ip  $\xi$ )). AODV()
 $\oplus \langle \lambda \xi. \{ \xi | dip := dip \}$ 
 $| dip. dip \in qD(\text{store } \xi) - vD(rt \xi) \wedge \text{the}(\sigma_{\text{p-flag}}(\text{store } \xi, \text{dip})) = req \}$ 
```

```

 $\llbracket \xi. \xi \mid store := unsetRRF(store \xi) (dip \xi) \rrbracket$ 
 $\llbracket \xi. \xi \mid sn := inc(sn \xi) \rrbracket$ 
 $\llbracket \xi. \xi \mid rreqid := nrreqid(rreqs \xi) (ip \xi) \rrbracket$ 
 $\llbracket \xi. \xi \mid rreqs := rreqs \xi \cup \{(ip \xi, rreqid \xi)\} \rrbracket$ 
 $\text{broadcast}(\lambda \xi. \text{rreq}(0, rreqid \xi, dip \xi, sn (rt \xi) (dip \xi), sqnf(rt \xi) (dip \xi), ip \xi, sn \xi, ip \xi, \text{False})). AODV()"$ 

| " $\Gamma_{AODV} PNewPkt = \text{labelled } PNewPkt$  (
   $\langle \xi. \text{dip } \xi = ip \xi \rangle$ 
     $\text{deliver}(\lambda \xi. \text{data } \xi). AODV()$ 
   $\oplus \langle \xi. \text{dip } \xi \neq ip \xi \rangle$ 
     $\llbracket \xi. \xi \mid store := add(\text{data } \xi) (\text{dip } \xi) (\text{store } \xi) \rrbracket$ 
     $AODV()"$ 
)

| " $\Gamma_{AODV} PPkt = \text{labelled } PPkt$  (
   $\langle \xi. \text{dip } \xi = ip \xi \rangle$ 
     $\text{deliver}(\lambda \xi. \text{data } \xi). AODV()$ 
   $\oplus \langle \xi. \text{dip } \xi \neq ip \xi \rangle$ 
  (
     $\langle \xi. \text{dip } \xi \in vD(rt \xi) \rangle$ 
       $\text{unicast}(\lambda \xi. \text{the}(nhop(rt \xi) (dip \xi)), \lambda \xi. \text{pkt}(\text{data } \xi, \text{dip } \xi, oip \xi)). AODV()$ 
     $\triangleright$ 
     $\llbracket \xi. \xi \mid dests := (\lambda rip. \text{if } (rip \in vD(rt \xi) \wedge nhop(rt \xi) rip = nhop(rt \xi) (dip \xi)) \text{ then Some}(inc(sqn(rt \xi) rip)) \text{ else None}) \rrbracket$ 
     $\llbracket \xi. \xi \mid rt := \text{invalidate}(rt \xi) (dests \xi) \rrbracket$ 
     $\llbracket \xi. \xi \mid store := \text{setRRF}(store \xi) (dests \xi) \rrbracket$ 
     $\llbracket \xi. \xi \mid pre := \bigcup \{ \text{the}(\text{precs}(rt \xi) rip) \mid rip \in \text{dom}(dests \xi) \} \rrbracket$ 
     $\llbracket \xi. \xi \mid dests := (\lambda rip. \text{if } ((dests \xi) rip \neq \text{None} \wedge \text{the}(\text{precs}(rt \xi) rip) \neq \{\})) \text{ then } (dests \xi) rip \text{ else None}) \rrbracket$ 
       $\text{groupcast}(\lambda \xi. \text{pre } \xi, \lambda \xi. \text{rerr}(dests \xi, ip \xi)). AODV()$ 
     $\oplus \langle \xi. \text{dip } \xi \notin vD(rt \xi) \rangle$ 
    (
       $\langle \xi. \text{dip } \xi \in iD(rt \xi) \rangle$ 
         $\text{groupcast}(\lambda \xi. \text{the}(\text{precs}(rt \xi) (dip \xi)), \lambda \xi. \text{rerr}([\text{dip } \xi \mapsto sqn(rt \xi) (dip \xi)], ip \xi)). AODV()$ 
       $\oplus \langle \xi. \text{dip } \xi \notin iD(rt \xi) \rangle$ 
         $AODV()$ 
    )
  )
)

| " $\Gamma_{AODV} PRreq = \text{labelled } PRreq$  (
   $\langle \xi. (oip \xi, rreqid \xi) \in rreqs \xi \rangle$ 
     $AODV()$ 
   $\oplus \langle \xi. (oip \xi, rreqid \xi) \notin rreqs \xi \rangle$ 
     $\llbracket \xi. \xi \mid rt := \text{update}(rt \xi) (oip \xi) (osn \xi, kno, val, hops \xi + 1, sip \xi, \{\}) \rrbracket$ 
     $\llbracket \xi. \xi \mid rreqs := rreqs \xi \cup \{(oip \xi, rreqid \xi)\} \rrbracket$ 
  (
     $\langle \xi. \text{handled } \xi = \text{False} \rangle$ 
    (
       $\langle \xi. \text{dip } \xi = ip \xi \rangle$ 
         $\llbracket \xi. \xi \mid sn := \max(sn \xi) (dsn \xi) \rrbracket$ 
         $\text{unicast}(\lambda \xi. \text{the}(nhop(rt \xi) (oip \xi)), \lambda \xi. \text{rrep}(0, dip \xi, sn \xi, oip \xi, ip \xi)).$ 
         $\text{broadcast}(\lambda \xi. \text{rreq}(hops \xi + 1, rreqid \xi, dip \xi, dsn \xi, dsk \xi, oip \xi, osn \xi, ip \xi, \text{True})).$ 
         $AODV()$ 
       $\triangleright$ 
       $\llbracket \xi. \xi \mid dests := (\lambda rip. \text{if } (rip \in vD(rt \xi) \wedge nhop(rt \xi) rip = nhop(rt \xi) (oip \xi)) \text{ then Some}(inc(sqn(rt \xi) rip)) \text{ else None}) \rrbracket$ 
       $\llbracket \xi. \xi \mid rt := \text{invalidate}(rt \xi) (dests \xi) \rrbracket$ 
       $\llbracket \xi. \xi \mid store := \text{setRRF}(store \xi) (dests \xi) \rrbracket$ 
       $\llbracket \xi. \xi \mid pre := \bigcup \{ \text{the}(\text{precs}(rt \xi) rip) \mid rip \in \text{dom}(dests \xi) \} \rrbracket$ 
       $\llbracket \xi. \xi \mid dests := (\lambda rip. \text{if } ((dests \xi) rip \neq \text{None} \wedge \text{the}(\text{precs}(rt \xi) rip) \neq \{\})) \text{ then } (dests \xi) rip \text{ else None}) \rrbracket$ 
         $\text{groupcast}(\lambda \xi. \text{pre } \xi, \lambda \xi. \text{rerr}(dests \xi, ip \xi)). AODV()$ 
       $\oplus \langle \xi. \text{dip } \xi \neq ip \xi \rangle$ 
    )
  )
)

```

```

( 
  ⟨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) []
[ $\xi$ .  $\xi$  () rt := invalidate (rt  $\xi$ ) (dests  $\xi$ ) []]
[ $\xi$ .  $\xi$  () store := setRRF (store  $\xi$ ) (dests  $\xi$ ) []]
[ $\xi$ .  $\xi$  () pre :=  $\bigcup$  {the (precs (rt  $\xi$ ) rip) | rip. rip  $\in$  dom (dests  $\xi$ )} []]
[ $\xi$ .  $\xi$  () dests := ( $\lambda$  rip. if ((dests  $\xi$ ) rip  $\neq$  None  $\wedge$  the (precs (rt  $\xi$ ) rip)  $\neq$  {}) then (dests  $\xi$ ) rip else None) [])
groupcast( $\lambda$   $\xi$ . pre  $\xi$ ,  $\lambda$   $\xi$ . rerr(dests  $\xi$ , ip  $\xi$ )). AODV())

```

declare  $\Gamma_{AODV}.simp$  [simp del, code del]  
lemmas  $\Gamma_{AODV\_simp}$  [simp, code] =  $\Gamma_{AODV}.simp$  [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  $\Gamma_{AODV\_skeleton}.simp$  [simp del, code del]  
lemmas  $\Gamma_{AODV\_skeleton\_simp}$  [simp, code]  
=  $\Gamma_{AODV\_skeleton}.simp$  [simplified  $\Gamma_{AODV\_simp} seqp\_skeleton.simp$ ]

lemma aodv\_proc\_cases [dest]:  
fixes p pn  
shows "p  $\in$  ctermsl ( $\Gamma_{AODV}$  pn)  $\Rightarrow$   
(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  $\sigma_{AODV} :: "ip \Rightarrow (state \times (state, msg, pseqp, pseqp label) seqp) set"$   
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  $\equiv$  () 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 "∀l∈labels ΓAODV p. P l p"
        and "∃p l. P l p ⇒ Q"
        shows "Q"
⟨proof⟩

lemma aodv_simple_labels [simp]: "simple_labels ΓAODV" 
⟨proof⟩

lemma σAODV_labels [simp]: "(ξ, p) ∈ σAODV i ⇒ labels ΓAODV p = {PAodv-:0}⁹"
⟨proof⟩

lemma aodv_init_kD_empty [simp]:
  "(ξ, p) ∈ σAODV i ⇒ kD (rt ξ) = {}"
⟨proof⟩

lemma aodv_init_sip_not_ip [simp]: "¬(sip (aodv_init i) = i)" ⟨proof⟩

lemma aodv_init_sip_not_ip' [simp]:
  assumes "(ξ, p) ∈ σAODV i"
  shows "sip ξ ≠ ip ξ"
⟨proof⟩

lemma aodv_init_sip_not_i [simp]:
  assumes "(ξ, p) ∈ σAODV 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_simp [cterms_env]
aodv_proc_cases [cterms1_cases]
seq_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
                      cterms_intros]
seq_step_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
                           cterms_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 ⇒ bool"
where "not Pkt m ≡ case m of Pkt      ⇒ False |      ⇒ True"

```

```

definition msg_sender :: "msg ⇒ ip"
where "msg_sender m ≡ case m of Rreq _ _ _ _ _ ipc _ ⇒ ipc
          | Rrep _ _ _ _ ipc ⇒ ipc
          | Rerr _ ipc ⇒ ipc
          | Pkt _ _ ipc ⇒ ipc"

```

```
lemma msg_sender_simp [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 _ s ipc _ ⇒ hopsc = 0 → oipc = s ipc
  | Rrep hopsc dipc _ _ s ipc ⇒ hopsc = 0 → dipc = s ipc
  | _ ⇒ 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 = s ip)"
" \hops dip dsn oip sip. msg_zhops (Rrep hops dip dsn oip sip) = (hops = 0 → dip = s ip)"
" \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 = 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"

```

```

"\ $\wedge d \text{ dip } sip.$  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]:
"\ $\wedge$  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) = (∀rip∈dom(dests).  

                                         rip∈iD(crt) ∧ the (dests rip) = sqn crt rip)"  

"\ $\wedge$  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 ⇒ state option) × 'a ⇒ (nat ⇒ state) × 'a"
where
"initmissing σ = (λi. case (fst σ) i of None ⇒ aodv_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 = aodv_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

## 4.5 Quality relations between routes

```

theory D_Fresher
imports D_Aodv_Data
begin

```

### 4.5.1 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_r_zero [simp]:
  " $\bigwedge dsn\ dsk\ flag\ hops\ nhip\ pre.\ nsqn_r(0, dsk, flag, hops, nhip, pre) = 0$ "
  (proof)

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:
  " $\bigwedge rt\ dip.\ nsqn\ rt\ dip = (\text{if } flag\ rt\ dip = Some\ val \vee sqn\ rt\ dip = 0$ 
    $\text{then } sqn\ rt\ dip \text{ else } sqn\ rt\ dip - 1)$ "
  (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(\text{the } (\sigma_{route}(rt, dip)))"
  (proof)

```

```

lemma nsqn_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]:
  " $\bigwedge rt\ dip\ npre\ dip'. dip \in kD(rt) \implies$ 
    $nsqn_r(\text{the } (\text{the } (\text{addpreRT } rt\ dip\ npre) dip')) = nsqn_r(\text{the } (rt\ dip'))$ "
  (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 ∈ 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 nhop.
  rt ≠ update rt ip (dsn, kno, val, hops, nhop, {})
  ⇒ nsqn (update rt ip (dsn, kno, val, hops, nhop, {})) 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 nhop pre rt ip
  assumes "dip ≠ ip"
  shows "nsqn (update rt ip (dsn, dsk, flag, hops, nhop, 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' ≡ ((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')))"

```

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⟩

4.5.3 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 ⊑_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" (<(_/ ≈ _ _)> [51, 999, 51] 50)
where
  "rt1 ≈_i rt2 ≡ rt_fresh_as i 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 ≈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_nsqr:
assumes "dip ∈ kD(rt1)"
and "dip ∈ kD(rt2)"
and "rt1 ≈dip rt2"
shows "nsqr_r (the (rt2 dip)) = nsqr_r (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 nsqn_r_invalidate [simp]:
  assumes "dip ∈ kD(rt)"
    and "dip ∈ dom(dests)"
  shows "nsqn_r (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

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

abbreviation

*rt\_strictly\_fresher\_syn* :: "rt ⇒ ip ⇒ rt ⇒ bool" (<(\_ / ⊑ \_ \_)> [51, 999, 51] 50)  
 where

$$\text{``}rt1 \sqsubset_i rt2 \equiv rt\_strictly\_fresher i rt1 rt2\text{''}$$

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 =
  (nsqn_r (the (rt1 i)) < nsqn_r (the (rt2 i))
   ∨ (nsqn_r (the (rt1 i)) = nsqn_r (the (rt2 i)) ∧ π_5(the (rt1 i)) > π_5(the (rt2 i))))"
⟨proof⟩

lemma rt_strictly_fresher_fresherD [dest]:
  assumes "rt1 ⊑_{dip} rt2"
  shows "the (rt1 dip) ⊑ the (rt2 dip)"
⟨proof⟩

lemma rt_strictly_fresher_not_fresh_asD [dest]:
  assumes "rt1 ⊑_{dip} rt2"
  shows "¬ rt1 ≈_{dip} rt2"
⟨proof⟩

lemma rt_strictly_fresher_trans [elim, trans]:
  assumes "rt1 ⊑_{dip} rt2"
    and "rt2 ⊑_{dip} rt3"
  shows "rt1 ⊑_{dip} rt3"
⟨proof⟩

lemma rt_strictly_fresher_irrefl [simp]: "¬ (rt ⊑_{dip} rt)"
⟨proof⟩

lemma rt_fresher_trans_rt_strictly_fresher [elim, trans]:
  assumes "rt1 ⊑_{dip} rt2"
    and "rt2 ⊑_{dip} rt3"
  shows "rt1 ⊑_{dip} rt3"
⟨proof⟩

lemma rt_fresher_trans_rt_strictly_fresher' [elim, trans]:
  assumes "rt1 ⊑_{dip} rt2"
    and "rt2 ⊑_{dip} rt3"
  shows "rt1 ⊑_{dip} rt3"

```

$\langle proof \rangle$

```
lemma rt_fresher_imp_nsqn_le:
  assumes "rt1 ⊑_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

## 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 ≡_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 ≡ (λ(ξ, _). 1 ≤ sn ξ) . sn ξ ≤ sn ξ'"
  ⟨proof⟩

lemma sequence_number_one_or_bigger':
  "paodv i ≡ (λ(ξ, _). 1 ≤ sn 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:
  "paodv i ≡ onl ΓAODV (λ(ξ, 1). (1 = PRrep-:1 →
                                             rt ξ ≠ update (rt ξ) (dip ξ) (dsn ξ, kno, val, hops ξ + 1, sip ξ, {})))"
  ⟨proof⟩
```

```
lemma addpreRT_partly_welldefined:
  "paodv i ≡
    onl ΓAODV (λ(ξ, 1). (1 ∈ {PRreq-:18..PRreq-:20} ∪ {PRrep-:2..PRrep-:6} → dip ξ ∈ kD (rt ξ))
                           ∧ (1 ∈ {PRreq-:3..PRreq-:19} → oip ξ ∈ kD (rt ξ)))"
  ⟨proof⟩
```

Proposition 7.38

```
lemma includes_nhip:
  "paodv i ≡ onl ΓAODV (λ(ξ, 1). ∀ dip ∈ kD(rt ξ). the (nhop (rt ξ) dip) ∈ kD(rt ξ))"
  ⟨proof⟩
```

Proposition 7.22: needed in Proposition 7.4

```
lemma addpreRT_welldefined:
  "paodv i ≡ onl ΓAODV (λ(ξ, 1). (1 ∈ {PRreq-:18..PRreq-:20} → dip ξ ∈ kD (rt ξ)) ∧
                                             (1 = PRreq-:19 → oip ξ ∈ kD (rt ξ)) ∧
                                             (1 = PRrep-:5 → dip ξ ∈ kD (rt ξ)) ∧
                                             (1 = PRrep-:6 → (the (nhop (rt ξ) (dip ξ))) ∈ kD (rt ξ)))"
  (is "_ ≡ onl ΓAODV ?P")
  ⟨proof⟩
```

Proposition 7.4

```
lemma known_destinations_increase:
  "paodv i ≡_A onll ΓAODV (λ((ξ, _), _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ'))"
  ⟨proof⟩
```

Proposition 7.5

```
lemma rreqs_increase:
```

```

"paodv i ⊨_A onll ΓAODV (λ((ξ, _), _, (ξ', _)). rreqs ξ ⊆ rreqs ξ')"
⟨proof⟩

lemma dests_bigger_than_sqn:
"paodv i ⊨ onl ΓAODV (λ(ξ, 1). 1 ∈ {PAodv-:15..PAodv-:19}
                                ∪ {PPkt-:7..PPkt-:11}
                                ∪ {PRreq-:11..PRreq-:15}
                                ∪ {PRreq-:24..PRreq-:28}
                                ∪ {PRrep-:10..PRrep-:14}
                                ∪ {PRerr-:1..PRerr-:5})
                                → ( ∀ ip ∈ dom(dests ξ). ip ∈ kD(rt ξ) ∧ sqn(rt ξ) ip ≤ the(dests ξ ip)))"
⟨proof⟩

```

Proposition 7.6

```

lemma sqns_increase:
"paodv i ⊨_A onll ΓAODV (λ((ξ, _), _, (ξ', _)). ∀ ip. sqn(rt ξ) ip ≤ sqn(rt ξ') ip)"
⟨proof⟩

```

Proposition 7.7

```

lemma ip_constant:
"paodv i ⊨ onl ΓAODV (λ(ξ, _). ip ξ = i)"
⟨proof⟩

```

Proposition 7.8

```

lemma sender_ip_valid':
"paodv i ⊨_A onll ΓAODV (λ((ξ, _), a, _). anycast(λm. not_Pkt m → msg_sender m = ip ξ) a)"
⟨proof⟩

```

```

lemma sender_ip_valid:
"paodv i ⊨_A onll ΓAODV (λ((ξ, _), a, _). anycast(λm. not_Pkt m → msg_sender m = i) a)"
⟨proof⟩

```

```

lemma received_msg_inv:
"paodv i ⊨ (recvmsg P → onl ΓAODV (λ(ξ, 1). 1 ∈ {PAodv-:1} → P(msg ξ)))"
⟨proof⟩

```

Proposition 7.9

```

lemma sip_not_ip':
"paodv i ⊨ (recvmsg (λm. not_Pkt m → msg_sender m ≠ i) → onl ΓAODV (λ(ξ, _). sip ξ ≠ ip ξ))"
⟨proof⟩

```

```

lemma sip_not_ip:
"paodv i ⊨ (recvmsg (λm. not_Pkt m → msg_sender m ≠ i) → onl ΓAODV (λ(ξ, _). sip ξ ≠ 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 ⊨ onl ΓAODV (λ(ξ, _). ∀ ip ∈ kD(rt ξ). the(dhops(rt ξ) ip) ≥ 1)"
⟨proof⟩

```

```

lemma rreq_dip_in_vD_dip_eq_ip:
"paodv i ⊨ onl ΓAODV (λ(ξ, 1). (1 ∈ {PRreq-:18..PRreq-:21} → dip ξ ∈ vD(rt ξ))
                                ∧ (1 ∈ {PRreq-:6, PRreq-:7} → dip ξ = ip ξ)
                                ∧ (1 ∈ {PRreq-:17..PRreq-:21} → dip ξ ≠ ip ξ))"
⟨proof⟩

```

Proposition 7.11

```

lemma anycast_msg_zhops:
"¬rreqid dip dsn dsk oip osn sip.
   paodv i ⊨_A onll ΓAODV (λ( _, a, _). anycast msg_zhops a)"
⟨proof⟩

```

```

lemma hop_count_zero_oip_dip_sip:
"paodv i ⊨ (recvmsg msg_zhops → onl ΓAODV (λ(ξ, 1).
          (l ∈ {PAodv-:4..PAodv-:5} ∪ {PRreq-:n/n. True} →
           (hops ξ = 0 → oip ξ = sip ξ)))
          ∧
          ((l ∈ {PAodv-:6..PAodv-:7} ∪ {PRrep-:n/n. True} →
           (hops ξ = 0 → dip ξ = sip ξ))))"
⟨proof⟩

lemma osn_rreq:
"paodv i ⊨ (recvmsg rreq_rrep_sn → onl ΓAODV (λ(ξ, 1).
          l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n/n. True} → 1 ≤ osn ξ))"
⟨proof⟩

lemma osn_rreq':
"paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) → onl ΓAODV (λ(ξ, 1).
          l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n/n. True} → 1 ≤ osn ξ))"
⟨proof⟩

lemma dsn_rrep:
"paodv i ⊨ (recvmsg rreq_rrep_sn → onl ΓAODV (λ(ξ, 1).
          l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n/n. True} → 1 ≤ dsn ξ))"
⟨proof⟩

lemma dsn_rrep':
"paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) → onl ΓAODV (λ(ξ, 1).
          l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n/n. True} → 1 ≤ dsn ξ))"
⟨proof⟩

lemma hop_count_zero_oip_dip_sip':
"paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) → onl ΓAODV (λ(ξ, 1).
          (l ∈ {PAodv-:4..PAodv-:5} ∪ {PRreq-:n/n. True} →
           (hops ξ = 0 → oip ξ = sip ξ)))
          ∧
          ((l ∈ {PAodv-:6..PAodv-:7} ∪ {PRrep-:n/n. True} →
           (hops ξ = 0 → dip ξ = sip ξ))))"
⟨proof⟩

```

Proposition 7.12

```

lemma zero_seq_unk_hops_one':
"paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) → onl ΓAODV (λ(ξ, _).
          ∀ dip ∈ kD(rt ξ). (sqn (rt ξ) dip = 0 → sqnf (rt ξ) dip = unk)
          ∧ (sqnf (rt ξ) dip = unk → the (dhops (rt ξ) dip) = 1)
          ∧ (the (dhops (rt ξ) dip) = 1 → the (nhop (rt ξ) dip) = dip))"
⟨proof⟩

lemma zero_seq_unk_hops_one:
"paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) → onl ΓAODV (λ(ξ, _).
          ∀ dip ∈ kD(rt ξ). (sqn (rt ξ) dip = 0 → (sqnf (rt ξ) dip = unk
          ∧ the (dhops (rt ξ) dip) = 1
          ∧ the (nhop (rt ξ) dip) = dip)))"
⟨proof⟩

```

```

lemma kD_unk_or_atleast_one:
"paodv i ⊨ (recvmsg rreq_rrep_sn → onl ΓAODV (λ(ξ, 1).
          ∀ dip ∈ kD(rt ξ). π3(the (rt ξ dip)) = unk ∨ 1 ≤ π2(the (rt ξ dip))))"
⟨proof⟩

```

Proposition 7.13

```

lemma rreq_rrep_sn_any_step_invariant:
"paodv i ⊨A (recvmsg rreq_rrep_sn → onll ΓAODV (λ( _, a, _). anycast rreq_rrep_sn a))"
⟨proof⟩

```

Proposition 7.14

```
lemma rreq_rrep_fresh_any_step_invariant:
  "paodv 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:
  "paodv 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:
  "paodv i \models
    onl \Gamma_{AODV} (\lambda(\xi, 1). (1 \in \{PAodv-:15, PPkt-:7, PRreq-:11, PRreq-:24, PRrep-:10\}
      \longrightarrow (\forall ip \in \text{dom}(dests } \xi). ip \in vD(rt \xi) \wedge \text{the } (dests } \xi ip) = inc (sqn (rt \xi) ip))
    \wedge (1 = PRerr-:1
      \longrightarrow (\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:
  "paodv i \models_A (recvmsg rreq_rrep_sn \rightarrow) onll \Gamma_{AODV} (\lambda((\xi, _), _, (\xi', _)).
    \forall dip \in kD(rt \xi). rt \xi \sqsubseteq_{dip} rt \xi')"
  ⟨proof⟩
```

end

## 4.7 The quality increases predicate

```
theory D_Quality_Increases
imports D_Aodv_Predicates D_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 "\forall dip. dip \in kD(rt \xi) \implies dip \in kD(rt \xi')"
  and "\forall dip. [| dip \in kD(rt \xi); dip \in kD(rt \xi') |] \implies rt \xi \sqsubseteq_{dip} rt \xi'"
  and "\forall 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 "[| dip \in kD(rt \xi'); rt \xi \sqsubseteq_{dip} rt \xi'; sqn (rt \xi) dip \leq sqn (rt \xi') dip |] \implies R dip \xi \xi'"
  shows "R dip \xi \xi"
```

*(proof)*

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

  fixes  $i p$

  assumes "quality\_increases  $\xi \xi'$ "

    and " $i p \in kD(rt \xi)$ "

  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\_{dip} rt (\sigma' nhip)"

    and qinc: "quality\_increases ( $\sigma nhip$ ) ( $\sigma' nhip$ )"

    and " $dip \in kD(rt (\sigma nhip))$ "

  shows "rt ( $\sigma i$ ) \sqsubset\_{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 handled.
   msg_fresh σ (Rreq hops rreqid dip dsn dsk oip osn sip handled) =
   (osn ≥ 1 ∧ (sip ≠ oip → oip ∈ kD(rt(σ sip)))
```

```

 $\wedge \text{nsqn}(\text{rt}(\sigma \text{ sip})) \text{oip} \geq \text{osn}$ 
 $\wedge (\text{nsqn}(\text{rt}(\sigma \text{ sip})) \text{oip} = \text{osn})$ 
 $\rightarrow (\text{hops} \geq \text{the}(\text{dhops}(\text{rt}(\sigma \text{ sip})) \text{oip})$ 
 $\vee \text{the}(\text{flag}(\text{rt}(\sigma \text{ sip})) \text{oip}) = \text{inv}))$ ""
" $\wedge \text{hops dip dsn oip sip. msg\_fresh } \sigma (\text{Rrep hops dip dsn oip sip}) =$ 
 $(\text{dsn} \geq 1 \wedge (\text{sip} \neq \text{dip} \rightarrow \text{dip} \in \text{kD}(\text{rt}(\sigma \text{ sip})))$ 
 $\wedge \text{nsqn}(\text{rt}(\sigma \text{ sip})) \text{dip} \geq \text{dsn}$ 
 $\wedge (\text{nsqn}(\text{rt}(\sigma \text{ sip})) \text{dip} = \text{dsn})$ 
 $\rightarrow (\text{hops} \geq \text{the}(\text{dhops}(\text{rt}(\sigma \text{ sip})) \text{dip}))$ 
 $\vee \text{the}(\text{flag}(\text{rt}(\sigma \text{ sip})) \text{dip}) = \text{inv}))$ ""
" $\wedge \text{dests sip. msg\_fresh } \sigma (\text{Rerr dests sip}) =$ 
 $(\forall \text{ripc} \in \text{dom}(\text{dests}). (\text{ripc} \in \text{kD}(\text{rt}(\sigma \text{ sip})))$ 
 $\wedge \text{the}(\text{dests ripc}) - 1 \leq \text{nsqn}(\text{rt}(\sigma \text{ sip})) \text{ripc}))$ ""
" $\wedge \text{d dip. msg\_fresh } \sigma (\text{Newpkt d dip}) = \text{True}$ ""
" $\wedge \text{d dip sip. msg\_fresh } \sigma (\text{Pkt d dip sip}) = \text{True}$ ""
⟨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 ⟹ recvmsg 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 σ_AODV' :: "((ip ⇒ state) × ((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  $\equiv$  \ 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]: " $(\sigma, p) \in \sigma_{AODV}' \implies \text{labels } \Gamma_{AODV} p = \{PAodv-:0\}$ " 
  ⟨proof⟩

lemma oaodv_init_kD_empty [simp]:
  " $(\sigma, p) \in \sigma_{AODV}' \implies kD (\text{rt } (\sigma i)) = \{\}$ " 
  ⟨proof⟩

lemma oaodv_init_vD_empty [simp]:
  " $(\sigma, p) \in \sigma_{AODV}' \implies vD (\text{rt } (\sigma i)) = \{\}$ " 
  ⟨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

```

## 4.9 Global invariant proofs over sequential processes

```

theory D_Global_Invariants
imports D_Seq_Invariants
  D_Aodv_Predicates
  D_Fresher
  D_Quality_Increases
  AWN.OAWN_Convert
  D_Oaodv
begin

lemma other_quality_increases [elim]:
  assumes "other quality_increases I  $\sigma \sigma'$ "
  shows " $\forall j. \text{quality\_increases } (\sigma j) (\sigma' j)$ " 
  ⟨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: " $\bigwedge \sigma \sigma' m. \llbracket P \sigma m; \text{other } Q \{i\} \sigma \sigma' \rrbracket \implies P \sigma' m$ "
    and local: " $\bigwedge \sigma m. P \sigma m \implies P (\sigma(i := \sigma i \{msg := m\})) m$ "
  shows "opaodv i  $\models$  (\text{otherwith } Q \{i\} (orecvmsg P), \text{other } Q \{i\} \rightarrow
    \text{onl } \Gamma_{AODV} (\lambda(\sigma, l). l \in \{PAodv-:1\} \longrightarrow P \sigma (msg (\sigma i))))"
  ⟨proof⟩

```

(Equivalent to) Proposition 7.27

```

lemma local_quality_increases:
  "opaodv i ⊨_A (recvmsg rreq_rrep_sn → onl ΓAODV (λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ'))"
  ⟨proof⟩

lemmas olocal_quality_increases =
  open_seq_stepInvariant [OF local_quality_increases initiali_aodv oaodv_trans aodv_trans,
  simplified seql_onl_swap]

lemma oquality_increases:
  "opaodv i ⊨_A (otherwith quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)),
  other quality_increases {i} →)
  onl ΓAODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j))"
  (is "_ ⊨_A (?S, _ →) _")
  ⟨proof⟩

lemma rreq_rrep_nsqn_fresh_any_step_invariant:
  "opaodv i ⊨_A (act (recvmsg rreq_rrep_sn), other A {i} →)
  onl ΓAODV (λ((σ, _), a, _). anycast (msg_fresh σ) a)"
  ⟨proof⟩

lemma oreceived_rreq_rrep_nsqn_fresh_inv:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
  other quality_increases {i} →)
  onl ΓAODV (λ(σ, l). l ∈ {PAodv-:1} → msg_fresh σ (msg (σ i)))"
  ⟨proof⟩

lemma oquality_increases_nsqn_fresh:
  "opaodv i ⊨_A (otherwith quality_increases {i} (orecvmsg msg_fresh),
  other quality_increases {i} →)
  onl ΓAODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j))"
  ⟨proof⟩

lemma oosn_rreq:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
  other quality_increases {i} →)
  onl ΓAODV (seql i (λ(ξ, l). l ∈ {PAodv-:4, PAodv-:5} ∪ {PRreq-:n | n. True} → 1 ≤ osn ξ))"
  ⟨proof⟩

lemma rreq_sip:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
  other quality_increases {i} →)
  onl ΓAODV (λ(σ, l).
  (l ∈ {PAodv-:4, PAodv-:5, PRreq-:0, PRreq-:2} ∧ sip (σ i) ≠ oip (σ i))
  → oip (σ i) ∈ kD(rt (σ (sip (σ i))))
  ∧ nsqn (rt (σ (sip (σ i)))) (oip (σ i)) ≥ osn (σ i)
  ∧ (nsqn (rt (σ (sip (σ i)))) (oip (σ i)) = osn (σ i))
  → (hops (σ i) ≥ the (dhops (rt (σ (sip (σ i)))) (oip (σ i)))
  ∨ the (flag (rt (σ (sip (σ i)))) (oip (σ i))) = inv)))"
  (is "_ ⊨_A (?S, ?U →) _")
  ⟨proof⟩

lemma odsn_rrep:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
  other quality_increases {i} →)
  onl ΓAODV (seql i (λ(ξ, l). l ∈ {PAodv-:6, PAodv-:7} ∪ {PRrep-:n | n. True} → 1 ≤ dsn ξ))"
  ⟨proof⟩

lemma rrep_sip:
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
  other quality_increases {i} →)
  onl ΓAODV (λ(σ, l).
  (l ∈ {PAodv-:6, PAodv-:7, PRrep-:0, PRrep-:1} ∧ sip (σ i) ≠ dip (σ i))
  → dip (σ i) ∈ kD(rt (σ (sip (σ i)))))"

```

```


$$\begin{aligned}
& \wedge \text{nsqn}(\text{rt}(\sigma(\text{sip}(\sigma i)))) (\text{dip}(\sigma i)) \geq \text{dsn}(\sigma i) \\
& \wedge (\text{nsqn}(\text{rt}(\sigma(\text{sip}(\sigma i)))) (\text{dip}(\sigma i)) = \text{dsn}(\sigma i) \\
& \quad \rightarrow (\text{hop}(\sigma i) \geq \text{the}(\text{dhops}(\text{rt}(\sigma(\text{sip}(\sigma i)))) (\text{dip}(\sigma i))) \\
& \quad \quad \vee \text{the}(\text{flag}(\text{rt}(\sigma(\text{sip}(\sigma i)))) (\text{dip}(\sigma i))) = \text{inv}))" \\
\langle \text{is } \_ \models (?S, ?U \rightarrow \_) \rangle \\
\langle \text{proof} \rangle
\end{aligned}$$


```

```

lemma rerr_sip:
"opaodv i \models (\text{otherwith quality\_increases } \{i\} (\text{orecvmsg msg\_fresh}), \\
other quality\_increases \{i\} \rightarrow) \\
onl \Gamma_{AODV} (\lambda(\sigma, 1). \\
1 \in \{\text{PAodv-}:8, \text{PAodv-}:9, \text{PRerr-}:0, \text{PRerr-}:1\} \\
\rightarrow (\forall \text{rip} \in \text{dom}(\text{dests } (\sigma i)). \text{rip} \in kD(\text{rt}(\sigma(\text{sip}(\sigma i)))) \wedge \\
\text{the}(\text{dests } (\sigma i) \text{ rip}) - 1 \leq \text{nsqn}(\text{rt}(\sigma(\text{sip}(\sigma i))) \text{ rip}))" \\
\langle \text{is } \_ \models (?S, ?U \rightarrow \_) \rangle \\
\langle \text{proof} \rangle

```

```

lemma prerr_guard: "paodv i \models \\
onl \Gamma_{AODV} (\lambda(\xi, 1). (1 = \text{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) = \text{sip } \xi \\
\wedge \text{sqn}(\text{rt } \xi) ip < \text{the}(\text{dests } \xi ip)))" \\
\langle \text{proof} \rangle

```

```

lemmas oaddpreRT_welldefined =
open_seq_invariant [OF addpreRT_welldefined initiali_aodv oaodv_trans aodv_trans,
simplified seql_onl_swap,
THEN oinvariant_anyact]

```

```

lemmas odests_vD_inc_sqn =
open_seq_invariant [OF dests_vD_inc_sqn initiali_aodv oaodv_trans aodv_trans,
simplified seql_onl_swap,
THEN oinvariant_anyact]

```

```

lemmas oprerr_guard =
open_seq_invariant [OF prerr_guard initiali_aodv oaodv_trans aodv_trans,
simplified seql_onl_swap,
THEN oinvariant_anyact]

```

Proposition 7.28

```

lemma seq_compare_next_hop':
"opaodv i \models (\text{otherwith quality\_increases } \{i\} (\text{orecvmsg msg\_fresh}), \\
other quality\_increases \{i\} \rightarrow) onl \Gamma_{AODV} (\lambda(\sigma, \_). \\
\forall dip. \text{let nh}ip = \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 \text{nsqn}(\text{rt } (\sigma i)) dip \leq \text{nsqn}(\text{rt } (\sigma nhip)) dip)" \\
\langle \text{is } \_ \models (?S, ?U \rightarrow \_) \rangle \\
\langle \text{proof} \rangle

```

Proposition 7.30

```

lemmas okD_unk_or_atleast_one =
open_seq_invariant [OF kD_unk_or_atleast_one initiali_aodv,
simplified seql_onl_swap]

```

```

lemmas ozero_seq_unk_hops_one =
open_seq_invariant [OF zero_seq_unk_hops_one initiali_aodv,
simplified seql_onl_swap]

```

```

lemma oreachable_fresh_okD_unk_or_atleast_one:
fixes dip
assumes "(σ, p) ∈ oreachable (opaodv i)
          (otherwith ((=)) {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 (opaodv i)
            (otherwith ((=)) {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 "opaodv i ⊨ (otherwith ((=)) {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 "opaodv i ⊨ (otherwith ((=)) {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 "opaodv i ⊨ (otherwith ((=)) {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**

$rt\_graph ::= (ip \Rightarrow state) \Rightarrow ip \Rightarrow ip \text{ rel}$

**where**

$"rt\_graph \ σ = (\lambda dip.$

```

{((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 Γ_AODV i) (seqp_sos Γ_QMSG)"
    and "a ≠ τ"
  shows "σ' i = σ i"
  ⟨proof⟩

lemma par_nhop_quality_increases:
  shows "opaodv i ⟨i qmsg |= (otherwith ((=)) {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)"

```

#### 4.11.2 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 ((=)) {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 : opaodv 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 : opaodv 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 : opaodv 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⟩

```

### 4.11.3 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. opaodv i ⟨⟨i qmsg⟩ p) ⊨
    (otherwith ((=)) (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⟩

```

### 4.11.4 Lift to closed networks

```

lemma onet_nhop_quality_increases:
  shows "oclosed (opnet (λi. opaodv 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⟩

```

### 4.11.5 Transfer into the standard model

```

interpretation aodv_openproc: openproc paodv opaodv id
  rewrites "aodv_openproc.initmissing = initmissing"
  ⟨proof⟩

```

```

interpretation aodv_openproc_par_qmsg: openproc_paq paodv opaodv id qmsg
  rewrites "aodv_openproc_par_qmsg.netglobal = netglobal"
  and "aodv_openproc_par_qmsg.initmissing = initmissing"
  ⟨proof⟩

```

```

lemma net_nhop_quality_increases:
  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paodv 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⟩

```

### 4.11.6 Loop freedom of AODV

```

theorem aodv_loop_freedom:
  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paodv i ⟨⟨ qmsg⟩ n) ⊨ netglobal (λσ. ∀dip. irrefl ((rt_graph σ 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 ⇒ sqn"
  where "inc sn ≡ if sn = 0 then sn else sn + 1"
```

```
lemma less_than_inc [simp]: "x ≤ inc x"
  ⟨proof⟩
```

```
lemma inc_minus_suc_0 [simp]:
  "inc x - Suc 0 = x"
  ⟨proof⟩
```

```
lemma inc_never_one' [simp, intro]: "inc x ≠ Suc 0"
  ⟨proof⟩
```

```
lemma inc_never_one [simp, intro]: "inc x ≠ 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 × k × f × nat × 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 " $\pi_6 \equiv \lambda(\_, \_, \_, \_, \_, \_, nhip). nhip$ "
```

**lemma** *proj<sub>s</sub>* [*simp*]:  
 " $\pi_2(dsn, dsk, flag, hops, nhip) = dsn$ "  
 " $\pi_3(dsn, dsk, flag, hops, nhip) = dsk$ "  
 " $\pi_4(dsn, dsk, flag, hops, nhip) = flag$ "  
 " $\pi_5(dsn, dsk, flag, hops, nhip) = hops$ "  
 " $\pi_6(dsn, dsk, flag, hops, nhip) = nhip$ "  
*<proof>*

**lemma** *proj3\_pred* [*intro*]: " $\llbracket P \text{ kno}; P \text{ unk} \rrbracket \implies P (\pi_3 x)$ "  
*<proof>*

**lemma** *proj4\_pred* [*intro*]: " $\llbracket P \text{ val}; P \text{ inv} \rrbracket \implies P (\pi_4 x)$ "  
*<proof>*

**lemma** *proj6\_pair\_snd* [*simp*]:  
 fixes *dsn' r*  
 shows " $\pi_6(dsn', snd(r)) = \pi_6(r)$ "  
*<proof>*

### 5.1.3 Routing Tables

Routing tables map ip addresses to route entries.

**type\_synonym** *rt* = "ip → r"

**syntax**  
 "*\_Sigma\_route*" :: "rt ⇒ ip → r" ( $\langle \sigma_{\text{route}}(\_, \_) \rangle$ )

**translations**

" $\sigma_{\text{route}}(rt, dip)$ " => "rt dip"

**definition** *sqn* :: "rt ⇒ ip ⇒ sqn"  
 where "sqn rt dip ≡ case  $\sigma_{\text{route}}(rt, dip)$  of Some *r* ⇒  $\pi_2(r)$  | None ⇒ 0"

**definition** *sqnf* :: "rt ⇒ ip ⇒ k"  
 where "sqnf rt dip ≡ case  $\sigma_{\text{route}}(rt, dip)$  of Some *r* ⇒  $\pi_3(r)$  | None ⇒ unk"

**abbreviation** *flag* :: "rt ⇒ ip → f"  
 where "flag rt dip ≡ map\_option  $\pi_4(\sigma_{\text{route}}(rt, dip))$ "

**abbreviation** *dhops* :: "rt ⇒ ip → nat"  
 where "dhops rt dip ≡ map\_option  $\pi_5(\sigma_{\text{route}}(rt, dip))$ "

**abbreviation** *nhop* :: "rt ⇒ ip → ip"  
 where "nhop rt dip ≡ map\_option  $\pi_6(\sigma_{\text{route}}(rt, dip))$ "

**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*]:  
 " $\bigwedge ip \text{ rt}. ip \in vD \text{ rt} \implies ip \in kD \text{ rt}$ "  
 " $\bigwedge ip \text{ rt}. ip \in iD \text{ rt} \implies ip \in kD \text{ rt}$ "  
*<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: " $\wedge dip \ rt. \ dip \in kD(rt) \implies \pi_5(\text{the } (rt \ dip)) = \text{the } (dhops \ rt \ dip)$ "  

  ⟨proof⟩

lemma proj4_eq_flag: " $\wedge dip \ rt. \ dip \in kD(rt) \implies \pi_4(\text{the } (rt \ dip)) = \text{the } (flag \ rt \ dip)$ "  

  ⟨proof⟩

lemma proj2_eq_sqn: " $\wedge dip \ rt. \ dip \in kD(rt) \implies \pi_2(\text{the } (rt \ dip)) = sqn \ rt \ dip$ "  

  ⟨proof⟩

lemma kD_sqnf_is_proj3 [simp]:  

  " $\wedge ip \ rt. \ ip \in kD(rt) \implies sqnf \ rt \ ip = \pi_3(\text{the } (rt \ ip))$ "  

  ⟨proof⟩

lemma vD_flag_val [simp]:  

  " $\wedge dip \ rt. \ dip \in vD(rt) \implies \text{the } (flag \ rt \ dip) = val$ "  

  ⟨proof⟩

lemma kD_update [simp]:  

  " $\wedge rt \ nip \ v. \ kD(rt(nip \mapsto v)) = \text{insert } nip \ (kD(rt))$ "  

  ⟨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⟩

```

### 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 "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 nhop  

  assumes "rt ip = Some (dsn, dsk, flag, hops, nhop)"  

  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. 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 ≠ 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]: "¬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. 1 ≤ dsn ⇒ 1 ≤ (sqn (update rt dip (dsn, kno, val, hops, ip))) 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]: " $\lambda rt\ ip\ dsn\ dsk\ hops\ nhip.$ 
 $rt \neq update\ rt\ ip\ (dsn,\ kno,\ val,\ hops,\ nhip)$ 
 $\implies the\ (nhop\ (update\ rt\ ip\ (dsn,\ kno,\ val,\ hops,\ nhip)))\ ip = nhip$ "
(proof)
```

```
lemma flag_update [simp]: " $\lambda rt\ dip\ dsn\ flg\ hops\ sip.$ 
 $rt \neq update\ rt\ dip\ (dsn,\ kno,\ flg,\ hops,\ sip)$ 
 $\implies the\ (flag\ (update\ rt\ dip\ (dsn,\ kno,\ flg,\ hops,\ sip)))\ dip = flg$ "
(proof)
```

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

```
lemma nhop_update [simp]: " $\lambda rt\ dip\ dsn\ dsk\ flg\ hops\ sip.$ 
 $rt \neq update\ rt\ dip\ (dsn,\ dsk,\ flg,\ hops,\ sip)$ 
 $\implies the\ (nhop\ (update\ rt\ dip\ (dsn,\ dsk,\ flg,\ hops,\ sip)))\ dip = sip$ "
(proof)
```

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

```
lemma vD_update_val [dest]:
" $\lambda dip\ rt\ dip'\ dsn\ dsk\ hops\ nhip.$ 
 $dip \in vD(update\ rt\ dip'\ (dsn,\ dsk,\ val,\ hops,\ nhip)) \implies (dip \in vD(rt) \vee dip = dip')$ "
(proof)
```

## Invalidating route entries

```
definition invalidate :: "rt ⇒ (ip → sqn) ⇒ rt"
where "invalidate rt dests ≡
```

```
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]:
" $\lambda dip. \pi_3(the ((invalidate rt dests) dip)) = \pi_3(the (rt dip))$ "
(proof)
```

```
lemma proj5_invalidate [simp]:
" $\lambda dip. \pi_5(the ((invalidate rt dests) dip)) = \pi_5(the (rt dip))$ "
(proof)
```

```

lemma proj6_invalidate [simp]:
  " $\bigwedge dip. \pi_6(\text{the } ((\text{invalidate } rt \text{ dests}) \ dip)) = \pi_6(\text{the } (rt \ dip))$ "
  ⟨proof⟩

5.1.5 Route Requests

lemma invalidate_kD_inv [simp]:
  " $\bigwedge rt \text{ dests}. kD (\text{invalidate } rt \text{ dests}) = kD rt$ "
  ⟨proof⟩

lemma invalidate_sqn:
  fixes rt dip dests
  assumes " $\forall rsn. \text{dests dip} = \text{Some rsn} \rightarrow \text{sqn rt dip} \leq rsn$ "
  shows "sqn rt dip  $\leq$  sqn (invalidate rt dests) dip"
  ⟨proof⟩

lemma sqn_invalidate_in_dests [simp]:
  fixes dests ipa rsn rt
  assumes "dests ipa = \text{Some rsn}"
    and "ipa} \in kD(rt)"
  shows "sqn (\text{invalidate } rt \text{ dests}) ipa = rsn"
  ⟨proof⟩

lemma dhops_invalidate [simp]:
  " $\bigwedge dip. \text{the } (\text{dhops } (\text{invalidate } rt \text{ dests}) \ dip) = \text{the } (\text{dhops } rt \ dip)$ "
  ⟨proof⟩

lemma sqnf_invalidate [simp]:
  " $\bigwedge dip. \text{sqnf } (\text{invalidate } (rt \ \xi) \ (dests \ \xi)) \ dip = \text{sqnf } (rt \ \xi) \ dip$ "
  ⟨proof⟩

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

lemma invalidate_other [simp]:
  fixes rt dests dip
  assumes "dip} \notin \text{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 = \text{None}"
  ⟨proof⟩

lemma vD_invalidate_vD_not_dests:
  " $\bigwedge dip \ rt \text{ dests}. dip} \in vD(\text{invalidate } rt \text{ dests}) \implies dip} \in vD(rt) \wedge \text{dests dip} = \text{None}$ "
  ⟨proof⟩

lemma sqn_invalidate_not_in_dests [simp]:
  fixes dests dip rt
  assumes "dip} \notin \text{dom(dests)}"
  shows "sqn (\text{invalidate } rt \text{ dests}) dip = sqn rt dip"
  ⟨proof⟩

lemma invalidate_changes:
  fixes rt dests dip dsn dsk flag hops nhip pre
  assumes "invalidate rt dests dip = \text{Some } (dsn, dsk, flag, hops, nhip)"
  shows "dsn = (\text{case dests dip of None} \Rightarrow \pi_2(\text{the } (rt \ dip)) \mid \text{Some rsn} \Rightarrow rsn)
    \wedge dsk = \pi_3(\text{the } (rt \ dip))
    \wedge flag = (\text{if dests dip = None} \text{ then } \pi_4(\text{the } (rt \ dip)) \text{ else } \text{inv})"
  ⟨proof⟩

```

```

 $\wedge \text{hops} = \pi_5(\text{the } (\text{rt dip}))$ 
 $\wedge \text{nhop} = \pi_6(\text{the } (\text{rt dip}))"$ 
⟨proof⟩

lemma proj3_inv: " $\lambda \text{dip } \text{rt } \text{dests}. \text{ dip} \in kD(\text{rt})$ 
 $\implies \pi_3(\text{the } (\text{invalidate rt dests dip})) = \pi_3(\text{the } (\text{rt dip}))"$ 
⟨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"      ( $\sigma_{\text{queue}}(\_, \_)$ )
  where " $\sigma_{\text{queue}}(\text{store}, \text{dip}) \equiv \text{case store dip of None} \Rightarrow [] \mid \text{Some } (p, q) \Rightarrow q$ ""

definition qD :: "store ⇒ ip set"
  where "qD ≡ \text{dom}"

definition add :: "data ⇒ ip ⇒ store ⇒ store"
  where "add d dip store ≡ \text{case store dip of}
    None ⇒ store (dip ↦ (req, [d]))
    | \text{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 ≡
    \text{map\_option } (\lambda(p, q). \text{ if } \text{tl } q = [] \text{ then store (dip := None)}
    \text{ else store (dip ↦ (p, tl q))}) (\text{store dip})"

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

definition unsetRRF :: "store ⇒ ip ⇒ store"
  where "unsetRRF store dip ≡ \text{case store dip of}
    None ⇒ store
    | \text{Some } (p, q) ⇒ store (dip ↦ (noreq, q))"

definition setRRF :: "store ⇒ (ip → sqn) ⇒ store"
  where "setRRF store dests ≡ \lambda dip. \text{ if } \text{dests dip} = \text{None} \text{ then store dip}
    \text{ else map\_option } (\lambda(\_, q). (\text{req}, q)) (\text{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.  $\text{nhop}$  is partial, thus a ‘*the*’ is needed, similarly for  $\text{dhops}$  and  $\text{addpreRT}$ .
2.  $\text{precs}$  is partial.
3.  $\sigma_{p\text{-flag}}(\text{store}, \text{dip})$  is partial.
4. The routing table ( $\text{rt}$ ) is modelled as a map ( $ip \Rightarrow r \text{ option}$ ) 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 → sqn" ip
  / Newpkt data ip
  / Pkt data ip ip

instantiation msg :: msg
begin
  definition newpkt_def [simp]: "newpkt ≡ λ(d, dip). Newpkt d dip"
  definition eq_newpkt_def: "eq_newpkt m ≡ case m of Newpkt d dip ⇒ True / _ ⇒ 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 × ip × sqn × k × ip × sqn × ip × bool ⇒ msg"
  where "rreq ≡ λ(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 × 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 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)"
```

$\langle proof \rangle$

```
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 = Map.empty,
  rreqs = {},
  store = Map.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 ⇒ state"
where "clear_locals ξ = ξ"
  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 ≠ ip ξ),
  handled := (SOME x. True)
)"


```

```

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⟩

```

### 5.3.2 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' handled' ⇒
    {ξ(| hops := hops', dip := dip', dsn := dsn',
        dsk := dsk', oip := oip', osn := osn', sip := sip',
        handled := handled' ) }
  | _ ⇒ {}"

```

```

lemma is_rreq_asm [dest!]:
  assumes "ξ ∈ is_rreq ξ"
  shows "(∃ hops' dip' dsn' dsk' oip' osn' sip' handled'.
    msg ξ = Rreq hops' dip' dsn' dsk' oip' osn' sip' handled' ∧
    ξ' = ξ(| hops := hops', dip := dip', dsn := dsn',
      dsk := dsk', oip := oip', osn := osn', sip := sip',
      handled := handled' ))"
  ⟨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 " $\xi' \in is\_rrep \xi$ "
shows " $\exists \text{hops}' \text{ dip}' \text{ dsn}' \text{ oip}' \text{ sip}' .$ 
       $\text{msg } \xi = Rrep \text{ hops}' \text{ dip}' \text{ dsn}' \text{ oip}' \text{ sip}' \wedge$ 
       $\xi' = \xi(\text{hops} := \text{hops}', \text{dip} := \text{dip}', \text{dsn} := \text{dsn}', \text{oip} := \text{oip}', \text{sip} := \text{sip}')$ "
⟨proof⟩

```

**definition** *is\_rerr*

```

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

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

```

assumes " $\xi' \in is\_rerr \xi$ "
shows " $(\exists \text{dests}' \text{ sip}' .$ 
       $\text{msg } \xi = Rerr \text{ dests}' \text{ sip}' \wedge$ 
       $\xi' = \xi(\text{dests} := \text{dests}', \text{sip} := \text{sip}')$ )"
```

⟨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]:

```

" $\xi' \in is\_rerr \xi \Rightarrow ip \xi' = ip \xi$ "
" $\xi' \in is\_rrep \xi \Rightarrow ip \xi' = ip \xi$ "
" $\xi' \in is\_rreq \xi \Rightarrow ip \xi' = ip \xi$ "
" $\xi' \in is\_pkt \xi \Rightarrow ip \xi' = ip \xi$ "
" $\xi' \in is\_newpkt \xi \Rightarrow ip \xi' = ip \xi$ "
```

⟨proof⟩

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

```

" $\xi' \in is\_rerr \xi \Rightarrow sn \xi' = sn \xi$ "
" $\xi' \in is\_rrep \xi \Rightarrow sn \xi' = sn \xi$ "
" $\xi' \in is\_rreq \xi \Rightarrow sn \xi' = sn \xi$ "
" $\xi' \in is\_pkt \xi \Rightarrow sn \xi' = sn \xi$ "
" $\xi' \in is\_newpkt \xi \Rightarrow sn \xi' = sn \xi$ "
```

⟨proof⟩

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

```

" $\xi' \in is\_rerr \xi \Rightarrow rt \xi' = rt \xi$ "
" $\xi' \in is\_rrep \xi \Rightarrow rt \xi' = rt \xi$ "
" $\xi' \in is\_rreq \xi \Rightarrow rt \xi' = rt \xi$ "
" $\xi' \in is\_pkt \xi \Rightarrow rt \xi' = rt \xi$ "
" $\xi' \in is\_newpkt \xi \Rightarrow rt \xi' = rt \xi$ "
```

⟨proof⟩

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

```

" $\xi' \in is\_rerr \xi \Rightarrow rreqs \xi' = rreqs \xi$ "
" $\xi' \in is\_rrep \xi \Rightarrow rreqs \xi' = rreqs \xi$ "
" $\xi' \in is\_rreq \xi \Rightarrow rreqs \xi' = rreqs \xi$ "
" $\xi' \in is\_pkt \xi \Rightarrow rreqs \xi' = rreqs \xi$ "
" $\xi' \in is\_newpkt \xi \Rightarrow rreqs \xi' = rreqs \xi$ "
```

⟨proof⟩

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

```

" $\xi' \in is\_rerr \xi \Rightarrow store \xi' = store \xi$ "
" $\xi' \in is\_rrep \xi \Rightarrow store \xi' = store \xi$ "
" $\xi' \in is\_rreq \xi \Rightarrow store \xi' = store \xi$ "
" $\xi' \in is\_pkt \xi \Rightarrow store \xi' = store \xi$ "
" $\xi' \in is\_newpkt \xi \Rightarrow store \xi' = store \xi$ "
```

⟨proof⟩

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

```

" $\xi' \in is\_pkt \xi \Rightarrow sip \xi' = sip \xi$ "
" $\xi' \in is\_newpkt \xi \Rightarrow sip \xi' = sip \xi$ "
```

*⟨proof⟩*

### 5.3.3 The protocol process

```
datatype pseqp =
  PAodv
  | PNewPkt
  | PPkt
  | PRreq
  | PRrep
  | PRerr

fun nat_of_seqp :: "pseqp ⇒ nat"
where
  "nat_of_seqp PAodv = 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]: "11 ≤ 12 = (nat_of_seqp 11 ≤ nat_of_seqp 12)"
definition less_seqp [iff]: "11 < 12 = (nat_of_seqp 11 < nat_of_seqp 12)"
instance ⟨proof⟩
end

abbreviation AODV
where
  "AODV ≡ λ_. [clear_locals] call(PAodv)"

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

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, 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)]()
      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 () 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, False)). AODV()
)
  ⊕ ⟨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 PNewPkt = labelled PNewPkt (
  ⟨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, 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 Γ<sub>AODV</sub>.simp [simp del, code del]  
lemmas Γ<sub>AODV</sub>\_simp [simp, code] = Γ<sub>AODV</sub>.simp [simplified]

fun Γ<sub>AODV</sub>\_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 Γ<sub>AODV</sub>\_skeleton\_wf [simp]:

```

  "wellformed ΓAODV_skeleton"
  ⟨proof⟩

```

declare Γ<sub>AODV</sub>\_skeleton.simps [simp del, code del]

lemmas Γ<sub>AODV</sub>\_skeleton\_simps [simp, code]
= Γ<sub>AODV</sub>\_skeleton.simps [simplified Γ<sub>AODV</sub>.simp 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 σ<sub>AODV</sub> :: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp) set"  
where "σ<sub>AODV</sub> i ≡ {(aodv\_init i, Γ<sub>AODV</sub> 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 Γ<sub>AODV</sub>"

⟨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]: " $\exists l. l \in \text{labels } \Gamma_{AODV} p$ " 
  ⟨proof⟩

lemma aodv_ex_labelE [elim]:
  assumes " $\forall l \in \text{labels } \Gamma_{AODV} p. P l p$ " 
    and " $\exists p. l. P l p \Rightarrow Q$ " 
  shows "Q" 
  ⟨proof⟩

lemma aodv_simple_labels [simp]: "simple_labels  $\Gamma_{AODV}$ " 
  ⟨proof⟩

lemma  $\sigma_{AODV}\text{-labels}$  [simp]: " $(\xi, p) \in \sigma_{AODV} i \Rightarrow \text{labels } \Gamma_{AODV} p = \{\text{PAodv-}:0\}$ " 
  ⟨proof⟩

lemma aodv_init_kD_empty [simp]:
  " $(\xi, p) \in \sigma_{AODV} i \Rightarrow kD(\text{rt } \xi) = \{\}$ " 
  ⟨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
   $\Gamma_{AODV}\text{-simps}$  [cterms_env]
  aodv_proc_cases [ctermssl_cases]
  seq_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
    cterms_intros]
  seq_step_invariant_ctermsI [OF aodv_wf aodv_control_within aodv_simple_labels aodv_trans,
    cterms_intros]

end

```

## 5.4 Invariant assumptions and properties

```

theory E_Aodv_Predicates
imports E_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]:

" $\bigwedge$  hops dip dsn dsk oip osn sip handled.  
 $\quad \text{msg\_sender}(\text{Rreq hops dip dsn dsk oip osn sip handled}) = \text{sip}$ "  
" $\bigwedge$  hops dip dsn oip sip.  $\text{msg\_sender}(\text{Rrep hops dip dsn oip sip}) = \text{sip}$ "  
" $\bigwedge$  dests sip.  $\text{msg\_sender}(\text{Rerr dests sip}) = \text{sip}$ "  
" $\bigwedge$  d dip sip.  $\text{msg\_sender}(\text{Pkt d dip sip}) = \text{sip}$ "  
{proof}

definition *msg\_zhops* :: "msg  $\Rightarrow$  bool"

where "*msg\_zhops* m  $\equiv$  case m of

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

lemma *msg\_zhops.simps* [simp]:

" $\bigwedge$  hops dip dsn dsk oip osn sip handled.  
 $\quad \text{msg\_zhops}(\text{Rreq hops dip dsn dsk oip osn sip handled}) = (\text{hopsc} = 0 \rightarrow \text{oip} = \text{sip})$ "  
" $\bigwedge$  hops dip dsn oip sip.  $\text{msg\_zhops}(\text{Rrep hops dip dsn oip sip}) = (\text{hopsc} = 0 \rightarrow \text{dip} = \text{sip})$ "  
" $\bigwedge$  dests sip.  $\text{msg\_zhops}(\text{Rerr dests sip}) = \text{True}$ "  
" $\bigwedge$  d dip.  $\text{msg\_zhops}(\text{Newpkt d dip}) = \text{True}$ "  
" $\bigwedge$  d dip sip.  $\text{msg\_zhops}(\text{Pkt d dip sip}) = \text{True}$ "  
{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 dip dsn dsk oip osn sip handled.  
 $\quad \text{rreq\_rrep\_sn}(\text{Rreq hops dip dsn dsk oip osn sip handled}) = (\text{osn} \geq 1)$ "  
" $\bigwedge$  hops dip dsn oip sip.  $\text{rreq\_rrep\_sn}(\text{Rrep hops dip dsn oip sip}) = (\text{dsn} \geq 1)$ "  
" $\bigwedge$  dests sip.  $\text{rreq\_rrep\_sn}(\text{Rerr dests sip}) = \text{True}$ "  
" $\bigwedge$  d dip.  $\text{rreq\_rrep\_sn}(\text{Newpkt d dip}) = \text{True}$ "  
" $\bigwedge$  d dip sip.  $\text{rreq\_rrep\_sn}(\text{Pkt d dip sip}) = \text{True}$ "  
{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  $\rightarrow$   
 $\quad \text{oipc} \in kD(\text{crt}) \wedge (\text{sqn crt oipc} > \text{osnc}$   
 $\quad \vee (\text{sqn crt oipc} = \text{osnc}$   
 $\quad \wedge \text{the}(\text{dhops crt oipc}) \leq \text{hopsc}$   
 $\quad \wedge \text{the}(\text{flag crt oipc}) = \text{val}))$   
| Rrep hopsc dipc dsnc \_ ipcc  $\Rightarrow$  (ipcc  $\neq$  dipc  $\rightarrow$   
 $\quad \text{dipc} \in kD(\text{crt})$   
 $\quad \wedge \text{sqn crt dipc} = \text{dsnc}$   
 $\quad \wedge \text{the}(\text{dhops crt dipc}) = \text{hopsc}$   
 $\quad \wedge \text{the}(\text{flag crt dipc}) = \text{val})$   
| \_  $\Rightarrow$  True"

lemma *rreq\_rrep\_fresh* [simp]:

" $\bigwedge$  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) = (∀ 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 ⇒ aodv_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 = aodv_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

```

## 5.5 Quality relations between routes

```

theory E_Fresher
imports E_Aodv_Data

```

begin

### 5.5.1 Net sequence numbers

#### On individual routes

##### definition

$\text{nsqn}_r ::= "r \Rightarrow \text{sqn}"$

##### where

$"\text{nsqn}_r r \equiv \text{if } \pi_4(r) = \text{val} \vee \pi_2(r) = 0 \text{ then } \pi_2(r) \text{ else } (\pi_2(r) - 1)"$

lemma  $\text{nsqn}_r\_\text{def}'$ :

$"\text{nsqn}_r r = (\text{if } \pi_4(r) = \text{inv} \text{ then } \pi_2(r) - 1 \text{ else } \pi_2(r))"$

$\langle \text{proof} \rangle$

lemma  $\text{nsqn}_r\_\text{zero}$  [simp]:

$"\forall dsn \text{ dsk flag hops nhip. } \text{nsqn}_r (0, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}) = 0"$

$\langle \text{proof} \rangle$

lemma  $\text{nsqn}_r\_\text{val}$  [simp]:

$"\forall dsn \text{ dsk hops nhip. } \text{nsqn}_r (dsn, \text{dsk}, \text{val}, \text{hops}, \text{nhip}) = dsn"$

$\langle \text{proof} \rangle$

lemma  $\text{nsqn}_r\_\text{inv}$  [simp]:

$"\forall dsn \text{ dsk hops nhip. } \text{nsqn}_r (dsn, \text{dsk}, \text{inv}, \text{hops}, \text{nhip}) = dsn - 1"$

$\langle \text{proof} \rangle$

lemma  $\text{nsqn}_r\_\text{lte\_dsn}$  [simp]:

$"\forall dsn \text{ dsk flag hops nhip. } \text{nsqn}_r (dsn, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}) \leq dsn"$

$\langle \text{proof} \rangle$

#### On routes in routing tables

##### definition

$\text{nsqn} ::= "rt \Rightarrow ip \Rightarrow \text{sqn}"$

##### where

$"\text{nsqn} \equiv \lambda rt \text{ dip. case } \sigma_{\text{route}}(rt, \text{dip}) \text{ of } \text{None} \Rightarrow 0 \mid \text{Some } r \Rightarrow \text{nsqn}_r(r)"$

lemma  $\text{nsqn\_sqn\_def}$ :

$"\forall rt \text{ dip. } \text{nsqn rt dip} = (\text{if } \text{flag rt dip} = \text{Some val} \vee \text{sqn rt dip} = 0 \text{ then } \text{sqn rt dip} \text{ else } \text{sqn rt dip} - 1)"$

$\langle \text{proof} \rangle$

lemma  $\text{not\_in\_kD\_nsqn}$  [simp]:

assumes  $"\text{dip} \notin kD(rt)"$

shows  $"\text{nsqn rt dip} = 0"$

$\langle \text{proof} \rangle$

lemma  $kD\_nsqn$ :

assumes  $"\text{dip} \in kD(rt)"$

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

$\langle \text{proof} \rangle$

lemma  $\text{nsqn}_r\_\text{flag\_pred}$  [simp, intro]:

fixes  $dsn \text{ dsk flag hops nhip pre}$

assumes  $"P (\text{nsqn}_r (dsn, \text{dsk}, \text{val}, \text{hops}, \text{nhip}))"$

and  $"P (\text{nsqn}_r (dsn, \text{dsk}, \text{inv}, \text{hops}, \text{nhip}))"$

shows  $"P (\text{nsqn}_r (dsn, \text{dsk}, \text{flag}, \text{hops}, \text{nhip}))"$

$\langle \text{proof} \rangle$

lemma  $\text{sqn\_nsqn}$ :

$"\forall rt \text{ dip. } \text{sqn rt dip} - 1 \leq \text{nsqn rt dip}"$

$\langle \text{proof} \rangle$

lemma  $\text{nsqn\_sqn}$ :  $"\text{nsqn rt dip} \leq \text{sqn rt dip}"$

$\langle proof \rangle$

```
lemma val_nsqn_sqn [elim, simp]:  
  assumes "ip ∈ kD(rt)"  
    and "the (flag rt ip) = val"  
  shows "nsqn rt ip = sqn rt ip"  
 $\langle proof \rangle$ 
```

```
lemma vD_nsqn_sqn [elim, simp]:  
  assumes "ip ∈ vD(rt)"  
  shows "nsqn rt ip = sqn rt ip"  
 $\langle proof \rangle$ 
```

```
lemma inv_nsqn_sqn [elim, simp]:  
  assumes "ip ∈ kD(rt)"  
    and "the (flag rt ip) = inv"  
  shows "nsqn rt ip = sqn rt ip - 1"  
 $\langle proof \rangle$ 
```

```
lemma iD_nsqn_sqn [elim, simp]:  
  assumes "ip ∈ 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 update rt ip (dsn, kno, val, hops, nhip)  
  \implies nsqn (update rt ip (dsn, kno, val, hops, nhip)) ip = dsn"  
 $\langle proof \rangle$ 
```

```
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"  
 $\langle proof \rangle$ 
```

```
lemma nsqn_invalidate_eq:  
  assumes "dip ∈ kD(rt)"  
    and "dests dip = Some rsn"  
  shows "nsqn (invalidate rt dests) dip = rsn - 1"  
 $\langle proof \rangle$ 
```

```
lemma nsqn_invalidate_other [simp]:  
  assumes "dip ∈ kD(rt)"  
    and "dip \notin dom dests"  
  shows "nsqn (invalidate rt dests) dip = nsqn rt dip"  
 $\langle proof \rangle$ 
```

## 5.5.2 Comparing routes

definition

```
fresher :: "r ⇒ r ⇒ bool" (<(_ / ⊑ _)> [51, 51] 50)
```

where

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

```
lemma fresherI1 [intro]:  
  assumes "nsqn_r r < nsqn_r r'"  
  shows "r ⊑ r'"  
 $\langle proof \rangle$ 
```

```
lemma fresherI2 [intro]:  
  assumes "nsqn_r r = nsqn_r r'"  
    and "π5(r) ≥ π5(r')"  
  shows "r ⊑ r'"  
 $\langle proof \rangle$ 
```

```

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

lemma fresherE [elim]:
  assumes "r ⊑ r'"
  and "nsqn_r r < nsqn_r r' ⟹ P r r'"
  and "nsqn_r r = nsqn_r 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'
  shows "(dsn, dsk, flag, hops, nhip) ⊑ (dsn, dsk', flag, hops, nhip')"
  ⟨proof⟩

```

### 5.5.3 Comparing routing tables

definition

$rt\_fresher :: "ip \Rightarrow rt \Rightarrow rt \Rightarrow bool"$

where

" $rt\_fresher \equiv \lambda dip\ rt\ rt'. (\text{the } (\sigma_{\text{route}}(rt, dip))) \sqsubseteq (\text{the } (\sigma_{\text{route}}(rt', dip)))$ "

abbreviation

$rt\_fresher\_syn :: "rt \Rightarrow ip \Rightarrow rt \Rightarrow bool" \quad (<(_/\sqsubseteq_/_)> \quad [51, 999, 51] 50)$

where

" $rt1 \sqsubseteq_i rt2 \equiv rt\_fresher i\ rt1\ rt2$ "

lemma  $rt\_fresher\_def'$ :

" $(rt1 \sqsubseteq_i rt2) = (\text{nsqn}_r (\text{the } (rt1\ i)) < \text{nsqn}_r (\text{the } (rt2\ i)) \vee \text{nsqn}_r (\text{the } (rt1\ i)) = \text{nsqn}_r (\text{the } (rt2\ i)) \wedge \pi5(\text{the } (rt2\ i)) \leq \pi5(\text{the } (rt1\ i)))$ "

⟨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 ⊑_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" (<(_/ ≈ _ _)> [51, 999, 51] 50)
where
  "rt1 ≈_i rt2 ≡ rt_fresh_as i 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 ≈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_nsqr:
assumes "dip ∈ kD(rt1)"
and "dip ∈ kD(rt2)"
and "rt1 ≈dip rt2"
shows "nsqr_r (the (rt2 dip)) = nsqr_r (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 nsqn_r_invalidate [simp]:
  assumes "dip ∈ kD(rt)"
    and "dip ∈ dom(dests)"
  shows "nsqn_r (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  $rt\_strictly\_fresher :: "ip \Rightarrow rt \Rightarrow rt \Rightarrow bool"$

where

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

abbreviation

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

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 ⊑\_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 =  
   (nsqn_r (the (rt1 i)) < nsqn_r (the (rt2 i))  
   ∨ (nsqn_r (the (rt1 i)) = nsqn_r (the (rt2 i)) ∧ π_5(the (rt1 i)) > π_5(the (rt2 i))))"  
(proof)
```

```
lemma rt_strictly_fresher_fresherD [dest]:  
  assumes "rt1 ⊑_dip rt2"  
  shows "the (rt1 dip) ⊑ the (rt2 dip)"  
(proof)
```

```
lemma rt_strictly_fresher_not_fresh_asD [dest]:  
  assumes "rt1 ⊑_dip rt2"  
  shows "¬ rt1 ≈_dip rt2"  
(proof)
```

```
lemma rt_strictly_fresher_trans [elim, trans]:  
  assumes "rt1 ⊑_dip rt2"  
    and "rt2 ⊑_dip rt3"  
  shows "rt1 ⊑_dip rt3"  
(proof)
```

```
lemma rt_strictly_fresher_irrefl [simp]: "¬ (rt ⊑_dip rt)"  
(proof)
```

```
lemma rt_fresher_trans_rt_strictly_fresher [elim, trans]:  
  assumes "rt1 ⊑_dip rt2"  
    and "rt2 ⊑_dip rt3"  
  shows "rt1 ⊑_dip rt3"  
(proof)
```

```
lemma rt_fresher_trans_rt_strictly_fresher' [elim, trans]:  
  assumes "rt1 ⊑_dip rt2"  
    and "rt2 ⊑_dip rt3"  
  shows "rt1 ⊑_dip rt3"  
(proof)
```

```
lemma rt_fresher_imp_nsqn_le:  
  assumes "rt1 ⊑_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 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_Aodv E_Aodv_Data E_Aodv_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:
  "paodv i ≡A onl1 ΓAODV (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"

```

$\langle proof \rangle$

```
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, 1). 1 \in (\{PAodv-:7\} \cup \{PAodv-: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:
  "paodv 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:
  "paodv i \models_{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, 1). 1 \in \{PAodv-:15..PAodv-: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 \text{dom}(dests } \xi). ip \in kD(rt \xi) \wedge sqn(rt \xi) ip \leq \text{the}(dests } \xi ip))"
  ⟨proof⟩
```

Proposition 7.6

```
lemma sqns_increase:
  "paodv i \models_{onll} \Gamma_{AODV} (\lambda((\xi, _), _, (\xi', _)). \forall ip. sqn(rt \xi) ip \leq sqn(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, _). anycast (\lambda m. not_Pkt m \longrightarrow msg_sender m = ip \xi) a)"
  ⟨proof⟩
```

```
lemma sender_ip_valid:
  "paodv i \models_A onll \Gamma_{AODV} (\lambda((\xi, _), a, _). anycast (\lambda m. not_Pkt m \longrightarrow msg_sender m = i) a)"
```

$\langle proof \rangle$

**lemma received\_msg\_inv:**  
 $"paodv i \models (\text{recvmsg } P \rightarrow \text{onl } \Gamma_{AODV} (\lambda(\xi, 1). 1 \in \{\text{PAodv-:1}\} \rightarrow P (\text{msg } \xi)))"$   
 $\langle proof \rangle$

Proposition 7.9

**lemma sip\_not\_ip':**  
 $"paodv i \models (\text{recvmsg } (\lambda m. \text{not\_Pkt } m \rightarrow \text{msg\_sender } m \neq i) \rightarrow \text{onl } \Gamma_{AODV} (\lambda(\xi, _). \text{sip } \xi \neq \text{ip } \xi))"$   
 $\langle proof \rangle$

**lemma sip\_not\_ip:**  
 $"paodv i \models (\text{recvmsg } (\lambda m. \text{not\_Pkt } m \rightarrow \text{msg\_sender } m \neq i) \rightarrow \text{onl } \Gamma_{AODV} (\lambda(\xi, _). \text{sip } \xi \neq i))"$   
 $\langle proof \rangle$

Neither  $\text{sip\_not\_ip}'$  nor  $\text{sip\_not\_ip}$  is needed to show loop freedom.

Proposition 7.10

**lemma hop\_count\_positive:**  
 $"paodv i \models \text{onl } \Gamma_{AODV} (\lambda(\xi, _). \forall ip \in kD (\text{rt } \xi). \text{the } (\text{dhops } (\text{rt } \xi) ip) \geq 1)"$   
 $\langle proof \rangle$

**lemma rreq\_dip\_in\_vD\_dip\_eq\_ip:**  
 $"paodv i \models \text{onl } \Gamma_{AODV} (\lambda(\xi, 1). (1 \in \{\text{PRreq-:16..PRreq-:17}\} \rightarrow \text{dip } \xi \in vD(\text{rt } \xi))$   
 $\wedge (1 \in \{\text{PRreq-:6, PRreq-:7}\} \rightarrow \text{dip } \xi = \text{ip } \xi)$   
 $\wedge (1 \in \{\text{PRreq-:15..PRreq-:17}\} \rightarrow \text{dip } \xi \neq \text{ip } \xi))"$   
 $\langle proof \rangle$

**lemma rrep\_dip\_in\_vD:**  
 $"paodv i \models \text{onl } \Gamma_{AODV} (\lambda(\xi, 1). (1 \in \{\text{PRrep-:4}\} \rightarrow \text{dip } \xi \in vD(\text{rt } \xi)))"$   
 $\langle proof \rangle$

Proposition 7.11

**lemma anycast\_msg\_zhops:**  
 $"\bigwedge rreqid dip dsn dsk oip osn \text{sip}.$   
 $\quad paodv i \models_A \text{onll } \Gamma_{AODV} (\lambda(., a, _). \text{anycast msg_zhops } a)"$   
 $\langle proof \rangle$

**lemma hop\_count\_zero\_oip\_dip\_sip:**  
 $"paodv i \models (\text{recvmsg msg_zhops} \rightarrow \text{onl } \Gamma_{AODV} (\lambda(\xi, 1).$   
 $\quad (1 \in \{\text{PAodv-:4..PAodv-:5}\} \cup \{\text{PRreq-:n/n. True}\} \rightarrow$   
 $\quad \quad (\text{hops } \xi = 0 \rightarrow \text{oip } \xi = \text{sip } \xi))$   
 $\quad \wedge$   
 $\quad ((1 \in \{\text{PAodv-:6..PAodv-:7}\} \cup \{\text{PRrep-:n/n. True}\} \rightarrow$   
 $\quad \quad (\text{hops } \xi = 0 \rightarrow \text{dip } \xi = \text{sip } \xi))))"$   
 $\langle proof \rangle$

**lemma osn\_rreq:**  
 $"paodv i \models (\text{recvmsg rreq_rrep_sn} \rightarrow \text{onl } \Gamma_{AODV} (\lambda(\xi, 1).$   
 $\quad 1 \in \{\text{PAodv-:4, PAodv-:5}\} \cup \{\text{PRreq-:n/n. True}\} \rightarrow 1 \leq \text{osn } \xi))"$   
 $\langle proof \rangle$

**lemma osn\_rreq':**  
 $"paodv i \models (\text{recvmsg } (\lambda m. \text{rreq\_rrep\_sn } m \wedge \text{msg\_zhops } m) \rightarrow \text{onl } \Gamma_{AODV} (\lambda(\xi, 1).$   
 $\quad 1 \in \{\text{PAodv-:4, PAodv-:5}\} \cup \{\text{PRreq-:n/n. True}\} \rightarrow 1 \leq \text{osn } \xi))"$   
 $\langle proof \rangle$

**lemma dsn\_rrep:**  
 $"paodv i \models (\text{recvmsg rreq_rrep_sn} \rightarrow \text{onl } \Gamma_{AODV} (\lambda(\xi, 1).$   
 $\quad 1 \in \{\text{PAodv-:6, PAodv-:7}\} \cup \{\text{PRrep-:n/n. True}\} \rightarrow 1 \leq \text{dsn } \xi))"$   
 $\langle proof \rangle$

**lemma dsn\_rrep':**  
 $"paodv i \models (\text{recvmsg } (\lambda m. \text{rreq\_rrep\_sn } m \wedge \text{msg\_zhops } m) \rightarrow \text{onl } \Gamma_{AODV} (\lambda(\xi, 1).$

$l \in \{PAodv\text{-}:6, PAodv\text{-}:7\} \cup \{PRrep\text{-}:n/n. True\} \longrightarrow 1 \leq dsn \xi\}$ "

$\langle proof \rangle$

```
lemma hop_count_zero_oip_dip_sip':
"paodv i \models (recvmsg (\lambda m. rreq_rrep_sn m \wedge msg_zhops m) \rightarrow onl \Gamma_{AODV} (\lambda (\xi, l).  

\quad (l \in \{PAodv\text{-}:4..PAodv\text{-}:5\} \cup \{PRreq\text{-}:n/n. True\} \longrightarrow  

\quad \quad (hops \xi = 0 \longrightarrow oip \xi = sip \xi)) )  

\quad \wedge  

\quad ((l \in \{PAodv\text{-}:6..PAodv\text{-}:7\} \cup \{PRrep\text{-}:n/n. True\} \longrightarrow  

\quad \quad (hops \xi = 0 \longrightarrow dip \xi = sip \xi)))) )"
```

$\langle proof \rangle$

Proposition 7.12

```
lemma zero_seq_unk_hops_one':
"paodv i \models (recvmsg (\lambda m. rreq_rrep_sn m \wedge msg_zhops m) \rightarrow onl \Gamma_{AODV} (\lambda (\xi, _).  

\quad \forall dip \in kD(rt \xi). (sqn(rt \xi) dip = 0 \longrightarrow sqnf(rt \xi) dip = unk)  

\quad \quad \wedge (sqnf(rt \xi) dip = unk \longrightarrow the(dhops(rt \xi) dip) = 1)  

\quad \quad \wedge (the(dhops(rt \xi) dip) = 1 \longrightarrow the(nhop(rt \xi) dip) = dip))) )"
```

$\langle proof \rangle$

```
lemma zero_seq_unk_hops_one:
"paodv i \models (recvmsg (\lambda m. rreq_rrep_sn m \wedge msg_zhops m) \rightarrow onl \Gamma_{AODV} (\lambda (\xi, _).  

\quad \forall dip \in kD(rt \xi). (sqn(rt \xi) dip = 0 \longrightarrow (sqnf(rt \xi) dip = unk  

\quad \quad \wedge the(dhops(rt \xi) dip) = 1  

\quad \quad \wedge the(nhop(rt \xi) dip) = dip))) )"
```

$\langle proof \rangle$

lemma kD\_unk\_or\_atleast\_one:

```
"paodv i \models (recvmsg rreq_rrep_sn \rightarrow onl \Gamma_{AODV} (\lambda (\xi, l).  

\quad \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:
"paodv 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:
"paodv i \models_A onll \Gamma_{AODV} (\lambda ((\xi, _), a, _). anycast (rreq_rrep_fresh(rt \xi)) a)) )"
```

$\langle proof \rangle$

Proposition 7.15

```
lemma rerr_invalid_any_step_invariant:
"paodv i \models_A onll \Gamma_{AODV} (\lambda ((\xi, _), a, _). anycast (rerr_invalid(rt \xi)) a)) )"
```

$\langle proof \rangle$

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:

```
"paodv i \models  

onl \Gamma_{AODV} (\lambda (\xi, l). (l \in \{PAodv\text{-}:15, PPkt\text{-}:7, PRreq\text{-}:11, PRreq\text{-}:20, PRrep\text{-}:7\})
```

```

    → ( ∀ ip ∈ dom(dests ξ). ip ∈ vD(rt ξ) ∧ the(dests ξ ip) = inc(sqn(rt ξ) ip))
    ∧ (l = PRerr:-1
        → ( ∀ ip ∈ dom(dests ξ). ip ∈ vD(rt ξ) ∧ the(dests ξ ip) > sqn(rt ξ) ip))"

```

*(proof)*

Proposition 7.27

lemma route\_tables\_fresher:

```

"paodv i ⊨A (recvmsg rreq_rrep_sn → onll ΓAO DV (λ((ξ, _), _, (ξ', _)).
    ∀ dip ∈ kD(rt ξ). rt ξ ⊑dip rt ξ'))"

```

*(proof)*

end

## 5.7 The quality increases predicate

theory E\_Quality\_Increases

imports E\_Aodv\_Predicates E\_Fresher

begin

definition quality\_increases :: "state ⇒ state ⇒ bool"

```

where "quality_increases ξ ξ' ≡ ( ∀ dip ∈ kD(rt ξ). dip ∈ kD(rt ξ') ∧ rt ξ ⊑dip rt ξ')
    ∧ ( ∀ dip. sqn(rt ξ) dip ≤ sqn(rt ξ') dip))"

```

lemma quality\_increasesI [intro!]:

```

assumes " ∀ dip. dip ∈ kD(rt ξ) ⇒ dip ∈ kD(rt ξ')"
and " ∀ dip. [ dip ∈ kD(rt ξ); dip ∈ kD(rt ξ') ] ⇒ rt ξ ⊑dip rt ξ'"
and " ∀ dip. sqn(rt ξ) dip ≤ sqn(rt ξ') dip"
shows "quality_increases ξ ξ"

```

*(proof)*

lemma quality\_increasesE [elim]:

```

fixes dip
assumes "quality_increases ξ ξ'"
and "dip ∈ kD(rt ξ)"
and "[ dip ∈ kD(rt ξ'); rt ξ ⊑dip rt ξ'; sqn(rt ξ) dip ≤ sqn(rt ξ') dip ] ⇒ R dip ξ ξ'"
shows "R dip ξ ξ"

```

*(proof)*

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

```

fixes ip
assumes "quality_increases ξ ξ'"
and "ip ∈ kD(rt ξ)"
shows "rt ξ ⊑ip rt ξ"

```

*(proof)*

lemma quality\_increases\_sqnE [elim]:

```

fixes dip
assumes "quality_increases ξ ξ'"
and "sqn(rt ξ) dip ≤ sqn(rt ξ') dip ⇒ R dip ξ ξ"
shows "R dip ξ ξ"

```

*(proof)*

lemma quality\_increases\_refl [intro, simp]: "quality\_increases ξ ξ"

*(proof)*

lemma strictly\_fresher\_quality\_increases\_right [elim]:

```

fixes σ σ' dip
assumes "rt(σ i) ⊑dip rt(σ' nhip)"
and qinc: "quality_increases(σ nhip) (σ' nhip)"
and "dip ∈ kD(rt(σ nhip))"
shows "rt(σ i) ⊑dip rt(σ' 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 "rreq_rrep_sn (Rreq hops dip dsn dsk oip osn sip handled)"
shows "msg_fresh σ (Rreq hops 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

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

abbreviation opaodv
:: "ip ⇒ ((ip ⇒ state) × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
"opaodv i ≡ () init = σAODV', trans = oseqp_sos ΓAODV i ()"

lemma initiali_aodv [intro!, simp]: "initiali i (init (opaodv i)) (init (paodv i))"
⟨proof⟩

lemma oaodv_control_within [simp]: "control_within ΓAODV (init (opaodv i))"
⟨proof⟩

lemma σAODV'_labels [simp]: "(σ, p) ∈ σAODV' ⇒ labels ΓAODV p = {PAodv-:0}"
⟨proof⟩

lemma oaodv_init_kD_empty [simp]:
"(σ, p) ∈ σAODV' ⇒ kD (rt (σ i)) = {}"
⟨proof⟩

lemma oaodv_init_vD_empty [simp]:
"(σ, p) ∈ σAODV' ⇒ vD (rt (σ i)) = {}"
⟨proof⟩

lemma oaodv_trans: "trans (opaodv i) = oseqp_sos ΓAODV 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

## 5.9 Global invariant proofs over sequential processes

```
theory E_Global_Invariants
```

```
imports E_Seq_Invariants
```

```
  E_Aodv_Predicates
```

```
  E_Fresher
```

```
  E_Quality_Increases
```

```
  AWN.OAWN_Convert
```

```
  E_OAodv
```

begin

```
lemma other_quality_increases [elim]:
```

```
  assumes "other quality_increases I σ σ'"

```

```
  shows "∀j. quality_increases (σ j) (σ' j)"

```

```
⟨proof⟩
```

```
lemma weaken_otherwith [elim]:
```

```
  fixes m
```

```
  assumes *: "otherwith P I (orecvmsg Q) σ σ' a"

```

```
  and weakenP: "¬¬(σ m. P σ m ⇒ P' σ m)"

```

```
  and weakenQ: "¬¬(σ m. Q σ m ⇒ Q' σ m)"

```

```
  shows "otherwith P' I (orecvmsg Q') σ σ' a"

```

```
⟨proof⟩
```

```
lemma oreceived_msg_inv:
```

```
  assumes other: "¬¬(σ σ' m. [ P σ m; other Q {i} σ σ' ] ⇒ P σ' m)"

```

```
  and local: "¬¬(σ m. P σ m ⇒ P (σ(i := σ i (msg := m))) m)"

```

```
  shows "opaodv i ⊨ (otherwith Q {i} (orecvmsg P), other Q {i} →)

```

```
    onl ΓAODV (λ(σ, 1). 1 ∈ {PAodv-:1} → P σ (msg (σ i)))"

```

```
⟨proof⟩
```

(Equivalent to) Proposition 7.27

```
lemma local_quality_increases:
```

```
"paodv i ⊨A (recvmsg rreq_rrep_sn →) onll ΓAODV (λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ')"

```

```
⟨proof⟩
```

```
lemmas olocal_quality_increases =
```

```
  open_seq_stepInvariant [OF local_quality_increases initiali_aodv oaodv_trans aodv_trans,
  simplified seqll_onll_swap]
```

```
lemma oquality_increases:
```

```
"opaodv i ⊨A (otherwith quality_increases {i} (orecvmsg (λ_. rreq_rrep_sn)),
```

```
  other quality_increases {i} →)

```

```
  onll ΓAODV (λ((σ, _), _, (σ', _)). ∀j. quality_increases (σ j) (σ' j))"
```

```
(is "_ ⊨A (?S, _ →) _")

```

```
⟨proof⟩
```

```
lemma rreq_rrep_nsqn_fresh_any_step_invariant:
```

```
"opaodv i ⊨A (act (recvmsg rreq_rrep_sn), other A {i} →)

```

```
  onll ΓAODV (λ((σ, _), a, _). anycast (msg_fresh σ) a)"

```

```
⟨proof⟩
```

```
lemma oreceived_rreq_rrep_nsqn_fresh_inv:
```

```
"opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),

```

```
  other quality_increases {i} →)

```

```
  onl ΓAODV (λ(σ, 1). 1 ∈ {PAodv-:1} → msg_fresh σ (msg (σ i)))"

```

```
⟨proof⟩
```

```
lemma oquality_increases_nsqn_fresh:
```

```

"opaodv i  $\models_A$  (otherwith quality_increases {i} (orecvmsg msg_fresh),
  other quality_increases {i}  $\rightarrow$ )
  onl  $\Gamma_{AODV}$  ( $\lambda((\sigma, \_), \_, (\sigma', \_))$ ).  $\forall j$ . quality_increases ( $\sigma j$ ) ( $\sigma' j$ ))"
⟨proof⟩

lemma oosn_rreq:
"opaodv i  $\models$  (otherwith quality_increases {i} (orecvmsg msg_fresh),
  other quality_increases {i}  $\rightarrow$ )
  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, 1)$ ).  $1 \in \{PAodv-:4, PAodv-:5\} \cup \{PRreq-:n \mid n. True\} \longrightarrow 1 \leq osn \xi$ )"
⟨proof⟩

lemma rreq_sip:
"opaodv i  $\models$  (otherwith quality_increases {i} (orecvmsg msg_fresh),
  other quality_increases {i}  $\rightarrow$ )
  onl  $\Gamma_{AODV}$  ( $\lambda(\sigma, 1)$ .
    ( $l \in \{PAodv-:4, PAodv-:5, PRreq-:0, PRreq-:2\} \wedge sip(\sigma i) \neq oip(\sigma i)$ )
     $\longrightarrow oip(\sigma i) \in kD(rt(sip(\sigma i)))$ 
     $\wedge nsqn(rt(sip(\sigma i))) (oip(\sigma i)) \geq osn(\sigma i)$ 
     $\wedge (nsqn(rt(sip(\sigma i))) (oip(\sigma i)) = osn(\sigma i))$ 
     $\longrightarrow (hops(\sigma i) \geq the(dhops(rt(sip(\sigma i)))) (oip(\sigma i)))$ 
     $\vee the(flag(rt(sip(\sigma i))) (oip(\sigma i))) = inv))$ 
  )
  (is " $\_ \models (?S, ?U \rightarrow) \_$ ")
⟨proof⟩

lemma odsn_rrep:
"opaodv i  $\models$  (otherwith quality_increases {i} (orecvmsg msg_fresh),
  other quality_increases {i}  $\rightarrow$ )
  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, 1)$ .  $1 \in \{PAodv-:6, PAodv-:7\} \cup \{PRrep-:n \mid n. True\} \longrightarrow 1 \leq dsn \xi$ )"
⟨proof⟩

lemma rrep_sip:
"opaodv i  $\models$  (otherwith quality_increases {i} (orecvmsg msg_fresh),
  other quality_increases {i}  $\rightarrow$ )
  onl  $\Gamma_{AODV}$  ( $\lambda(\sigma, 1)$ .
    ( $l \in \{PAodv-:6, PAodv-:7, PRrep-:0, PRrep-:1\} \wedge sip(\sigma i) \neq dip(\sigma i)$ )
     $\longrightarrow dip(\sigma i) \in kD(rt(sip(\sigma i)))$ 
     $\wedge nsqn(rt(sip(\sigma i))) (dip(\sigma i)) \geq dsn(\sigma i)$ 
     $\wedge (nsqn(rt(sip(\sigma i))) (dip(\sigma i)) = dsn(\sigma i))$ 
     $\longrightarrow (hops(\sigma i) \geq the(dhops(rt(sip(\sigma i)))) (dip(\sigma i)))$ 
     $\vee the(flag(rt(sip(\sigma i))) (dip(\sigma i))) = inv))$ 
  )
  (is " $\_ \models (?S, ?U \rightarrow) \_$ ")
⟨proof⟩

lemma rerr_sip:
"opaodv i  $\models$  (otherwith quality_increases {i} (orecvmsg msg_fresh),
  other quality_increases {i}  $\rightarrow$ )
  onl  $\Gamma_{AODV}$  ( $\lambda(\sigma, 1)$ .
     $l \in \{PAodv-:8, PAodv-:9, PRerr-:0, PRerr-:1\}$ 
     $\longrightarrow (\forall ripc \in \text{dom}(\text{dests } (\sigma i)). ripc \in kD(rt(sip(\sigma i))) \wedge$ 
     $\text{the}(\text{dests } (\sigma i) ripc) - 1 \leq nsqn(rt(sip(\sigma i))) ripc))$ 
  )
  (is " $\_ \models (?S, ?U \rightarrow) \_$ ")
⟨proof⟩

lemma prerr_guard: "paodv i  $\models$ 
  onl  $\Gamma_{AODV}$  ( $\lambda(\xi, 1)$ . ( $l = PRerr-:1$ 
     $\longrightarrow (\forall ip \in \text{dom}(\text{dests } \xi). ip \in vD(rt \xi)$ 
     $\wedge \text{the}(nhop(rt \xi) ip) = sip \xi$ 
     $\wedge sqn(rt \xi) ip < \text{the}(\text{dests } \xi ip)))$ 
  )
  (is " $\_ \models (?S, ?U \rightarrow) \_$ ")
⟨proof⟩

lemmas odests_vD_inc_sqn =
  open_seq_invariant [OF dests_vD_inc_sqn initiali_aodv oaodv_trans aodv_trans,
    simplified seql_onl_swap,
    THEN oinvariant_anyact]

```

```

lemmas oprerr_guard =
  open_seq_invariant [OF prerr_guard initiali_aodv oaodv_trans aodv_trans,
    simplified seql_onl_swap,
    THEN oinvariant_anyact]

```

Proposition 7.28

```

lemma seq_compare_next_hop':
  "opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
    other quality_increases {i} → onl ΓAODV (λ(σ, _).
      ∀ 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)"
  (is "_ ⊨ (?S, ?U →) _")
  ⟨proof⟩

```

Proposition 7.30

```

lemmas okD_unk_or_atleast_one =
  open_seq_invariant [OF kD_unk_or_atleast_one initiali_aodv,
    simplified seql_onl_swap]

```

```

lemmas ozero_seq_unk_hops_one =
  open_seq_invariant [OF zero_seq_unk_hops_one initiali_aodv,
    simplified seql_onl_swap]

```

```

lemma oreachable_fresh_okD_unk_or_atleast_one:
  fixes dip
  assumes "(σ, p) ∈ oreachable (opaodv i)
    (otherwith ((=)) {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 (opaodv i)
    (otherwith ((=)) {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 "opaodv i ⊨ (otherwith ((=)) {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 "opaodv i ⊨ (otherwith ((=)) {i} (orecvmsg msg_fresh),
    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)))"

```

```

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 "opaodv i ⊨ (otherwith ((=)) {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_Aodv_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 = (λ dip.
  {(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 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. 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 "\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))"
  shows "\forall dip. irrefl ((rt_graph \sigma 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 \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 i qmsg \models (otherwith (=) {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)) \sqsubseteq_{dip} (rt (\sigma nhip)))"
  ⟨proof⟩

```

```

lemma par_rreq_rrep_sn_quality_increases:
  "opaodv i \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 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 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⟩

```

### 5.11.2 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 : opaodv i \langle i qmsg : R \rangle_o \models
    (otherwith (=) {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)) \sqsubseteq_{dip} (rt (\sigma nhip)))"

```

```

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 : opaodv 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 : opaodv 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 : opaodv 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 : opaodv i ⟨⟨i qmsg : R_i⟩o ⟩A (λσ _. oarrivemsg (λ_. True) σ,
other (λ_. True) {i} →)
globala (λ(σ, a, σ'). a ≠ τ → σ' i = σ i)"
⟨proof⟩

```

### 5.11.3 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. opaodv i ⟨⟨i qmsg⟩p ⟩
(oarrive msg (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⟩

```

### 5.11.4 Lift to closed networks

```

lemma onet_nhop_quality_increases:
shows "oclosed (opnet (λi. opaodv 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⟩

```

### 5.11.5 Transfer into the standard model

```

interpretation aodv_openproc: openproc paodv opaodv id
rewrites "aodv_openproc.initmissing = initmissing"
⟨proof⟩

```

```

interpretation aodv_openproc_par_qmsg: openproc_paq paodv opaodv id qmsg

```

```

rewrites "aodv_openproc_par_qmsg.netglobal = netglobal"
  and "aodv_openproc_par_qmsg.initmissing = initmissing"
⟨proof⟩

lemma net_nhop_quality_increases:
  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paodv i ⟨⟨ qmsg ⟩⟩ n) ⊨ netglobal
    (λσ. ∀ i dip. let nhop = the (nhop (rt (σ i)) dip)
      in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhop)) ∧ nhop ≠ dip
      → (rt (σ i)) □dip (rt (σ nhop)))"
    (is "_ ⊨ netglobal (λσ. ∀ i. ?inv σ i)")
⟨proof⟩

```

### 5.11.6 Loop freedom of AODV

```

theorem aodv_loop_freedom:
  assumes "wf_net_tree n"
  shows "closed (pnet (λi. paodv 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.