Loop freedom of the (untimed) AODV routing protocol

Timothy Bourke[1] Peter Höfner[2]

March 17, 2025

[1]Inria, École normale supérieure, and NICTA
[2]NICTA and Computer Science and Engineering, UNSW

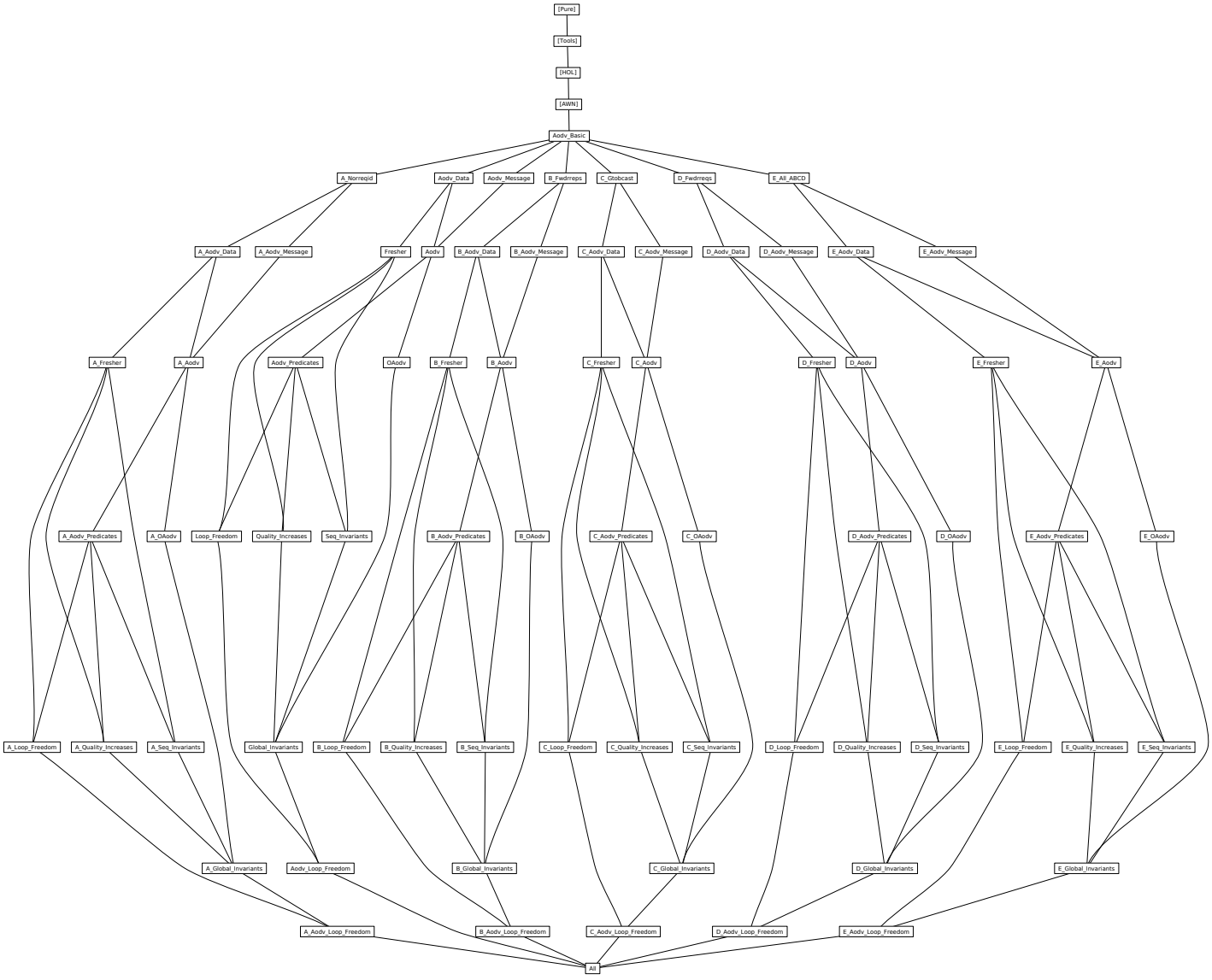
Abstract

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show that Isabelle/HOL can re-establish most proof obligations automatically and identify exactly the steps that are no longer valid.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

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, 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" (‹π_2›)*
  where *"π_2 ≡ λ(dsn, _, _, _, _, _). dsn"*

definition *proj3 :: "r ⇒ k" (‹π_3›)*
  where *"π_3 ≡ λ(_, dsk, _, _, _, _). dsk"*

definition *proj4 :: "r ⇒ f" (‹π_4›)*
  where *"π_4 ≡ λ(_, _, flag, _, _, _). flag"*

definition *proj5 :: "r ⇒ nat" (‹π_5›)*
  where *"π_5 ≡ λ(_, _, _, hops, _, _). hops"*

definition *proj6 :: "r ⇒ ip" (‹π_6›)*
  where *"π_6 ≡ λ(_, _, _, _, nhip, _). nhip"*

definition *proj7 :: "r ⇒ ip set" (‹π_7›)*
  where *"π_7 ≡ λ(_, _, _, _, _, pre). pre"*

lemma *projs [simp]:*
  "π_2(dsn, dsk, flag, hops, nhip, pre) = dsn"
  "π_3(dsn, dsk, flag, hops, nhip, pre) = dsk"
  "π_4(dsn, dsk, flag, hops, nhip, pre) = flag"
  "π_5(dsn, dsk, flag, hops, nhip, pre) = hops"
  "π_6(dsn, dsk, flag, hops, nhip, pre) = nhip"
  "π_7(dsn, dsk, flag, hops, nhip, pre) = pre"
  ⟨*proof*⟩

lemma *proj3_pred [intro]: "⟦ P kno; P unk ⟧ ⟹ P (π_3 x)"*
  ⟨*proof*⟩

lemma *proj4_pred [intro]: "⟦ P val; P inv ⟧ ⟹ P (π_4 x)"*
  ⟨*proof*⟩

lemma *proj6_pair_snd [simp]:*
  fixes *dsn' r*
  shows *"π_6 (dsn', snd (r)) = π_6(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" (‹σ_{route}'(_, _')›)

translations
 "σ_{route}(rt, dip)" => "rt dip"

definition *sqn :: "rt ⇒ ip ⇒ sqn"*
  where *"sqn rt dip ≡ case σ_{route}(rt, dip) of Some r ⇒ π_2(r) | None ⇒ 0"*

definition *sqnf :: "rt ⇒ ip ⇒ k"*
  where *"sqnf rt dip ≡ case σ_{route}(rt, dip) of Some r ⇒ π_3(r) | None ⇒ unk"*

abbreviation *flag :: "rt ⇒ ip ⇀ f"*
  where *"flag rt dip ≡ map_option π₄ (σ_route(rt, dip))"*

abbreviation *dhops :: "rt ⇒ ip ⇀ nat"*
  where *"dhops rt dip ≡ map_option π₅ (σ_route(rt, dip))"*

abbreviation *nhop :: "rt ⇒ ip ⇀ ip"*
  where *"nhop rt dip ≡ map_option π₆ (σ_route(rt, dip))"*

abbreviation *precs :: "rt ⇒ ip ⇀ ip set"*
  where *"precs rt dip ≡ map_option π₇ (σ_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*⟩

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 *"π_2(addpre v pre) = π_2(v)"*
  ⟨*proof*⟩

lemma *proj3_addpre:*
  fixes *v pre*
  shows *"π_3(addpre v pre) = π_3(v)"*
  ⟨*proof*⟩

lemma *proj4_addpre:*
  fixes *v pre*
  shows *"π_4(addpre v pre) = π_4(v)"*
  ⟨*proof*⟩

lemma *proj5_addpre:*
  fixes *v pre*
  shows *"π_5(addpre v pre) = π_5(v)"*
  ⟨*proof*⟩

lemma *proj6_addpre:*
  fixes *dsn dsk flag hops nhip pre npre*
  shows *"π_6(addpre v npre) = π_6(v)"*
  ⟨*proof*⟩

lemma *proj7_addpre:*
  fixes *dsn dsk flag hops nhip pre npre*
  shows *"π_7(addpre v npre) = π_7(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)) (σ_{route}(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 *"π_2(the (the (addpreRT rt ip' npre) ip)) = π_2(the (rt ip))"*

⟨*proof*⟩

lemma *proj3_addpreRT [simp]:*
fixes *ip rt ip' npre*
assumes *"ip∈kD rt"*
and *"ip'∈kD rt"*
shows *"π_3(the (the (addpreRT rt ip' npre) ip)) = π_3(the (rt ip))"*
⟨*proof*⟩

lemma *proj5_addpreRT [simp]:*
"⋀rt dip ip npre. dip∈kD(rt) ⟹ π_5(the (the (addpreRT rt dip npre) ip)) = π_5(the (rt ip))"
⟨*proof*⟩

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

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

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

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

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

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

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

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

Updating route entries

lemma *in_kD_case [simp]:*
fixes *dip rt*
assumes *"dip ∈ kD(rt)"*

shows *"(case rt dip of None ⇒ en | Some r ⇒ es r) = es (the (rt dip))"*
⟨*proof*⟩

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

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

lemma *not_kD_sqn [simp]:*
fixes *dip rt*
assumes *"dip ∉ kD(rt)"*
shows *"sqn rt dip = 0"*
⟨*proof*⟩

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

lemma *update_arg_wf_gives_cases:*
"⋀r. update_arg_wf r ⟹ (π₂(r) = 0) = (π₃(r) = unk)"
⟨*proof*⟩

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

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

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

definition *update :: "rt ⇒ ip ⇒ r ⇒ rt"*
where
"update rt ip r ≡
case σ_route(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 σ_route(rt, ip)"*
and *"nr ≡ addpre r (π₇(s))"*

```
    and "nr' ≡ (π2(s), π3(nr), π4(nr), π5(nr), π6(nr), π7(nr))"
    and "ns ≡ addpre s (π7(r))"
  shows
  "⟦ip ∉ kD(rt)⟧                                  ⟹ update rt ip r = rt (ip ↦ r)"
  "⟦ip ∈ kD(rt); sqn rt ip < π2(r)⟧               ⟹ update rt ip r = rt (ip ↦ nr)"
  "⟦ip ∈ kD(rt); sqn rt ip = π2(r);
                 the (dhops rt ip) > π5(r)⟧ ⟹ update rt ip r = rt (ip ↦ nr)"
  "⟦ip ∈ kD(rt); sqn rt ip = π2(r);
                 flag rt ip = Some inv⟧           ⟹ update rt ip r = rt (ip ↦ nr)"
  "⟦ip ∈ kD(rt); π3(r) = unk; (π2(r) = 0) = (π3(r) = unk)⟧  ⟹ update rt ip r = rt (ip ↦ nr')"
  "⟦ip ∈ kD(rt); sqn rt ip ≥ π2(r); π3(r) = kno;
    sqn rt ip = π2(r) ⟹ the (dhops rt ip) ≤ π5(r) ∧ the (flag rt ip) = val ⟧
                                                   ⟹ update rt ip r = rt (ip ↦ ns)"
  ⟨proof⟩

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

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

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

lemma in_kD_after_update [simp]:
  fixes rt nip dsn dsk flag hops nhip pre
  shows "kD (update rt nip (dsn, dsk, flag, hops, nhip, pre)) = insert nip (kD rt)"
  ⟨proof⟩

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

lemma sqn_if_updated:
  fixes rip v rt ip
```

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

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

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

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

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

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

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

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

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

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

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

lemma *update_rt_split_asm:*
"⋀rt ip dsn dsk flag hops sip.

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

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

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

lemma *update_kno_dsn_greater_zero:*
```
"⋀rt dip ip dsn hops npre. 1 ≤ dsn ⟹ 1 ≤ (sqn (update rt dip (dsn, kno, val, hops, ip, npre)) dip)"
```
⟨*proof*⟩

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

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

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

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

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

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

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

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

⟨*proof*⟩

lemma *vD_update_val [dest]:*
"⋀*dip 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]:*
"⋀*dip.* π_3*(the ((invalidate rt dests) dip)) =* π_3*(the (rt dip))*"
⟨*proof*⟩

lemma *proj5_invalidate [simp]:*
"⋀*dip.* π_5*(the ((invalidate rt dests) dip)) =* π_5*(the (rt dip))*"
⟨*proof*⟩

lemma *proj6_invalidate [simp]:*
"⋀*dip.* π_6*(the ((invalidate rt dests) dip)) =* π_6*(the (rt dip))*"
⟨*proof*⟩

lemma *proj7_invalidate [simp]:*
"⋀*dip.* π_7*(the ((invalidate rt dests) dip)) =* π_7*(the (rt dip))*"
⟨*proof*⟩

lemma *invalidate_kD_inv [simp]:*
"⋀*rt dests. kD (invalidate rt dests) = kD rt*"
⟨*proof*⟩

lemma *invalidate_sqn:*
fixes *rt dip dests*
assumes *"∀rsn. dests dip = Some rsn ⟶ sqn rt dip ≤ rsn"*
shows *"sqn rt dip ≤ sqn (invalidate rt dests) dip"*
⟨*proof*⟩

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

lemma *dhops_invalidate [simp]:*
"⋀*dip. the (dhops (invalidate rt dests) dip) = the (dhops rt dip)*"
⟨*proof*⟩

lemma *sqnf_invalidate [simp]:*
"⋀*dip. sqnf (invalidate (rt ξ) (dests ξ)) dip = sqnf (rt ξ) dip*"
⟨*proof*⟩

lemma *nhop_invalidate [simp]:*
"⋀*dip. the (nhop (invalidate (rt ξ) (dests ξ)) dip) = the (nhop (rt ξ) dip)*"
⟨*proof*⟩

lemma *invalidate_other [simp]:*
fixes *rt dests dip*

assumes *"dip∉dom(dests)"*
shows *"invalidate rt dests dip = rt dip"*
⟨*proof*⟩

lemma *invalidate_none [simp]:*
fixes *rt dests dip*
assumes *"dip∉kD(rt)"*
shows *"invalidate rt dests dip = None"*
⟨*proof*⟩

lemma *vD_invalidate_vD_not_dests:*
"⋀dip rt dests. dip∈vD(invalidate rt dests) ⟹ dip∈vD(rt) ∧ dests dip = None"
⟨*proof*⟩

lemma *sqn_invalidate_not_in_dests [simp]:*
fixes *dests dip rt*
assumes *"dip∉dom(dests)"*
shows *"sqn (invalidate rt dests) dip = sqn rt dip"*
⟨*proof*⟩

lemma *invalidate_changes:*
fixes *rt dests dip dsn dsk flag hops nhip pre*
assumes *"invalidate rt dests dip = Some (dsn, dsk, flag, hops, nhip, pre)"*
shows *" dsn = (case dests dip of None ⇒ 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*⟩

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"* (‹σ_{queue}'(_, _')›)
where *"σ_{queue}(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"* (‹$\sigma_{p\text{-}flag}$'(_, _')›)
where *"$\sigma_{p\text{-}flag}$(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 ⇒ 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 ξ ≡ case msg ξ of*
Newpkt data' dip' ⇒ { ξ(|data := data', dip := dip'|) }
| _ ⇒ {}"

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

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

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

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

lemma *is_rrep_asm [dest!]:*
assumes *"ξ' ∈ is_rrep ξ"*
shows *"(∃hops' dip' dsn' oip' sip'.*
msg ξ = Rrep hops' dip' dsn' oip' sip' ∧
ξ' = ξ(| hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' |))"
⟨*proof*⟩

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

lemma *is_rerr_asm [dest!]:*
assumes *"ξ' ∈ is_rerr ξ"*
shows *"(∃dests' sip'.*
msg ξ = Rerr dests' sip' ∧
ξ' = ξ(| dests := dests', sip := sip' |))"
⟨*proof*⟩

lemmas *is_msg_defs =*
is_rerr_def is_rrep_def is_rreq_def is_pkt_def is_newpkt_def

lemma *is_msg_inv_ip [simp]:*
"ξ' ∈ is_rerr ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_rrep ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_rreq ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_pkt ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_newpkt ξ ⟹ ip ξ' = ip ξ"
⟨*proof*⟩

lemma *is_msg_inv_sn [simp]:*
"ξ' ∈ is_rerr ξ ⟹ sn ξ' = sn ξ"

```
  "ξ' ∈ is_rrep ξ   ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_rreq ξ   ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_pkt ξ    ⟹ sn ξ' = sn ξ"
  "ξ' ∈ is_newpkt ξ ⟹ sn ξ' = sn ξ"
  ⟨proof⟩

lemma is_msg_inv_rt [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_rrep ξ   ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_rreq ξ   ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_pkt ξ    ⟹ rt ξ' = rt ξ"
  "ξ' ∈ is_newpkt ξ ⟹ rt ξ' = rt ξ"
  ⟨proof⟩

lemma is_msg_inv_rreqs [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_rrep ξ   ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_rreq ξ   ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_pkt ξ    ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_newpkt ξ ⟹ rreqs ξ' = rreqs ξ"
  ⟨proof⟩

lemma is_msg_inv_store [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ store ξ' = store ξ"
  "ξ' ∈ is_rrep ξ   ⟹ store ξ' = store ξ"
  "ξ' ∈ is_rreq ξ   ⟹ store ξ' = store ξ"
  "ξ' ∈ is_pkt ξ    ⟹ store ξ' = store ξ"
  "ξ' ∈ is_newpkt ξ ⟹ store ξ' = store ξ"
  ⟨proof⟩

lemma is_msg_inv_sip [simp]:
  "ξ' ∈ is_pkt ξ    ⟹ sip ξ' = sip ξ"
  "ξ' ∈ is_newpkt ξ ⟹ sip ξ' = sip ξ"
  ⟨proof⟩
```

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 ≤ 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 ξ) (| data := data, dip := dip, oip := oip |)⟧
     call(PPkt)"
abbreviation NEWPKT
where
  "NEWPKT args ≡
     ⟦ξ. let (data, dip) = args ξ in
         (clear_locals ξ) (| data := data, dip := dip |)⟧
     call(PNewPkt)"

abbreviation RREQ
where
  "RREQ args ≡
     ⟦ξ. let (hops, rreqid, dip, dsn, dsk, oip, osn, sip) = args ξ in
         (clear_locals ξ) (| hops := hops, rreqid := rreqid, dip := dip,
                             dsn := dsn, dsk := dsk, oip := oip,
                             osn := osn, sip := sip |)⟧
     call(PRreq)"

abbreviation RREP
where
  "RREP args ≡
     ⟦ξ. let (hops, dip, dsn, oip, sip) = args ξ in
         (clear_locals ξ) (| hops := hops, dip := dip, dsn := dsn,
                             oip := oip, sip := sip |)⟧
     call(PRrep)"

abbreviation RERR
where
  "RERR args ≡
     ⟦ξ. let (dests, sip) = args ξ in
         (clear_locals ξ) (| dests := dests, sip := sip |)⟧
     call(PRerr)"

fun Γ_AODV :: "(state, msg, pseqp, pseqp label) seqp_env"
where
  "Γ_AODV PAodv = labelled PAodv (
     receive(λmsg' ξ. ξ (| msg := msg' |)).
     (    ⟨is_newpkt⟩ NEWPKT(λξ. (data ξ, ip ξ))
       ⊕ ⟨is_pkt⟩ PKT(λξ. (data ξ, dip ξ, oip ξ))
       ⊕ ⟨is_rreq⟩
            ⟦ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) |)⟧
            RREQ(λξ. (hops ξ, rreqid ξ, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, sip ξ))
       ⊕ ⟨is_rrep⟩
            ⟦ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) |)⟧
            RREP(λξ. (hops ξ, dip ξ, dsn ξ, oip ξ, sip ξ))
       ⊕ ⟨is_rerr⟩
            ⟦ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) |)⟧
            RERR(λξ. (dests ξ, sip ξ))
     )
     ⊕ ⟨λξ. { ξ(| dip := dip |) | dip. dip ∈ qD(store ξ) ∩ vD(rt ξ) }⟩
          ⟦ξ. ξ (| data := hd(σ_queue(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 (σ_p-flag(store ξ, dip)) = req }⟩
          ⟦ξ. ξ (| store := unsetRRF (store ξ) (dip ξ) |)⟧
          ⟦ξ. ξ (| sn := inc (sn ξ) |)⟧
          ⟦ξ. ξ (| rreqid := nrreqid (rreqs ξ) (ip ξ) |)⟧
          ⟦ξ. ξ (| rreqs := rreqs ξ ∪ {(ip ξ, rreqid ξ)} |)⟧
          broadcast(λξ. rreq(0, rreqid ξ, dip ξ, sqn (rt ξ) (dip ξ), sqnf (rt ξ) (dip ξ),
                            ip ξ, sn ξ, ip ξ)). AODV())"

|  "Γ_AODV 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())"

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

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

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

| "Γ_AODV 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()
    )"

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

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

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

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

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

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

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

definition σAODV :: "ip ⇒ (state × (state, msg, 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* Γ_{AODV}*"*
⟨*proof*⟩

lemma σ_{AODV}*_labels [simp]: "(ξ, p) ∈* σ_{AODV} *i ⟹ labels* Γ_{AODV} *p = {PAodv-:0}"*
⟨*proof*⟩

lemma *aodv_init_kD_empty [simp]:*
"(ξ, p) ∈ σ_{AODV} *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) ∈* σ_{AODV} *i"*
shows *"sip ξ ≠ ip ξ"*
⟨*proof*⟩

lemma *aodv_init_sip_not_i [simp]:*
assumes *"(ξ, p) ∈* σ_{AODV} *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
Γ_{AODV}*_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"

```
  "⋀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"
  "⋀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.
          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]:*
```
  "⋀hops rreqid dip dsn dsk oip osn sip.
          rreq_rrep_fresh crt (Rreq hops rreqid dip dsn dsk oip osn sip) =
                              (sip ≠ oip ⟶ oip∈kD(crt)
                                         ∧ (sqn crt oip > osn
                                            ∨ (sqn crt oip = osn
                                               ∧ the (dhops crt oip) ≤ hops
                                               ∧ the (flag crt oip) = val)))"
  "⋀hops dip dsn oip sip. rreq_rrep_fresh crt (Rrep hops dip dsn oip sip) =
                              (sip ≠ dip ⟶ dip∈kD(crt)
                                         ∧ sqn crt dip = dsn
                                         ∧ the (dhops crt dip) = hops
                                         ∧ the (flag crt dip) = val)"
  "⋀dests sip.            rreq_rrep_fresh crt (Rerr dests sip) = True"
  "⋀d dip.                rreq_rrep_fresh crt (Newpkt d dip)   = True"
  "⋀d dip sip.            rreq_rrep_fresh crt (Pkt d dip sip)  = True"
```
⟨*proof*⟩

definition *rerr_invalid :: "rt ⇒ msg ⇒ bool"*

where *"rerr_invalid crt m ≡ case m of Rerr destsc _ ⇒ (∀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_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

0.6 Quality relations between routes

theory *Fresher*
imports *Aodv_Data*
begin

0.6.1 Net sequence numbers

On individual routes

definition
$nsqn_r$ *:: "r ⇒ sqn"*
where
"$nsqn_r$ *r ≡ if* π_4*(r) = val ∨* π_2*(r) = 0 then* π_2*(r) else (*π_2*(r) - 1)"*

lemma *nsqnr_def':*
"$nsqn_r$ *r = (if* π_4*(r) = inv then* π_2*(r) - 1 else* π_2*(r))"*
⟨*proof*⟩

lemma $nsqn_r$*_zero [simp]:*
"⋀dsn dsk flag hops nhip pre. $nsqn_r$ *(0, dsk, flag, hops, nhip, pre) = 0"*
⟨*proof*⟩

lemma *nsqn$_r$_val [simp]:*
"⋀dsn dsk hops nhip pre. nsqn$_r$ (dsn, dsk, val, hops, nhip, pre) = dsn"
⟨*proof*⟩

lemma *nsqn$_r$_inv [simp]:*
"⋀dsn dsk hops nhip pre. nsqn$_r$ (dsn, dsk, inv, hops, nhip, pre) = dsn - 1"
⟨*proof*⟩

lemma *nsqn$_r$_lte_dsn [simp]:*
"⋀dsn dsk flag hops nhip pre. 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:*
"⋀rt dip. nsqn rt dip = (if flag rt dip = Some val ∨ sqn rt dip = 0
then sqn rt dip else sqn rt dip - 1)"
⟨*proof*⟩

lemma *not_in_kD_nsqn [simp]:*
assumes "dip ∉ kD(rt)"
shows "nsqn rt dip = 0"
⟨*proof*⟩

lemma *kD_nsqn:*
assumes "dip ∈ kD(rt)"
shows "nsqn rt dip = nsqn$_r$(the (σ$_{route}$(rt, dip)))"
⟨*proof*⟩

lemma *nsqnr_r_flag_pred [simp, intro]:*
fixes dsn dsk flag hops nhip pre
assumes "P (nsqn$_r$ (dsn, dsk, val, hops, nhip, pre))"
and "P (nsqn$_r$ (dsn, dsk, inv, hops, nhip, pre))"
shows "P (nsqn$_r$ (dsn, dsk, flag, hops, nhip, pre))"
⟨*proof*⟩

lemma *nsqn$_r$_addpreRT_inv [simp]:*
"⋀rt dip npre dip'. dip ∈ kD(rt) ⟹
nsqn$_r$ (the (the (addpreRT rt dip npre) dip')) = nsqn$_r$ (the (rt dip'))"
⟨*proof*⟩

lemma *sqn_nsqn:*
"⋀rt dip. sqn rt dip - 1 ≤ nsqn rt dip"
⟨*proof*⟩

lemma *nsqn_sqn:* "nsqn rt dip ≤ sqn rt dip"
⟨*proof*⟩

lemma *val_nsqn_sqn [elim, simp]:*
assumes "ip∈kD(rt)"
and "the (flag rt ip) = val"
shows "nsqn rt ip = sqn rt ip"
⟨*proof*⟩

lemma *vD_nsqn_sqn [elim, simp]:*
assumes "ip∈vD(rt)"
shows "nsqn rt ip = sqn rt ip"
⟨*proof*⟩

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

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

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

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

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

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

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

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' ∧ π$_5$(r) ≥ π$_5$(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 *"π$_5$(r) ≥ π$_5$(r')"*
shows *"r ⊑ r'"*
⟨*proof*⟩

lemma *fresherI [intro]:*
assumes *"(nsqn$_r$ r < nsqn$_r$ r') ∨ (nsqn$_r$ r = nsqn$_r$ r' ∧ π$_5$(r) ≥ π$_5$(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' ∧ π$_5$(r) ≥ π$_5$(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 (σ$_{route}$(rt, dip))) ⊑ (the (σ$_{route}$(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':*
"(rt$_1$ ⊑$_i$ rt$_2$) = (nsqn$_r$ (the (rt$_1$ i)) < nsqn$_r$ (the (rt$_2$ i)) ∨
nsqn$_r$ (the (rt$_1$ i)) = nsqn$_r$ (the (rt$_2$ i)) ∧ π$_5$ (the (rt$_2$ i)) ≤ π$_5$ (the (rt$_1$ i)))"
⟨*proof*⟩

lemma *single_rt_fresher [intro]:*
assumes *"the (rt1 ip) ⊑ the (rt2 ip)"*
shows *"rt1 ⊑$_{ip}$ rt2"*
⟨*proof*⟩

lemma *rt_fresher_single [intro]:*
assumes *"rt1 ⊑$_{ip}$ rt2"*
shows *"the (rt1 ip) ⊑ the (rt2 ip)"*
⟨*proof*⟩

lemma *rt_fresher_def2:*
assumes *"dip ∈ kD(rt1)"*
and *"dip ∈ kD(rt2)"*
shows *"(rt1 ⊑$_{dip}$ rt2) = (nsqn rt1 dip < nsqn rt2 dip*
∨ (nsqn rt1 dip = nsqn rt2 dip
∧ the (dhops rt1 dip) ≥ the (dhops rt2 dip)))"
⟨*proof*⟩

lemma *rt_fresherI1 [intro]:*
assumes *"dip ∈ kD(rt1)"*
and *"dip ∈ kD(rt2)"*
and *"nsqn rt1 dip < nsqn rt2 dip"*

shows *"rt1 $\sqsubseteq_{dip}$ rt2"*
⟨*proof*⟩

lemma *rt_fresherI2 [intro]:*
assumes *"dip ∈ 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 $\sqsubseteq_{dip}$ rt"*
⟨*proof*⟩

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

lemma *rt_fresher_if_Some [intro!]:*
assumes *"the (rt dip) ⊑ r"*
shows *"rt $\sqsubseteq_{dip}$ (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 $\sqsubseteq_{dip}$ rt2) ∧ (rt2 $\sqsubseteq_{dip}$ rt1)"

abbreviation
rt_fresh_as_syn :: "rt ⇒ ip ⇒ rt ⇒ bool" (‹(_/ ≈_ _)› [51, 999, 51] 50)
where
"rt1 $\approx_i$ rt2 ≡ rt_fresh_as i rt1 rt2"

lemma *rt_fresh_as_refl [simp]: "⋀rt dip. rt $\approx_{dip}$ rt"*
⟨*proof*⟩

lemma *rt_fresh_as_trans [simp, intro, trans]:*
"⋀rt1 rt2 rt3 dip. ⟦ rt1 $\approx_{dip}$ rt2; rt2 $\approx_{dip}$ rt3 ⟧ ⟹ rt1 $\approx_{dip}$ rt3"
⟨*proof*⟩

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

lemma *rt_fresh_as_fresherI [intro]:*
assumes *"dip∈kD(rt1)"*
and *"dip∈kD(rt2)"*
and *"the (rt1 dip) ⊑ the (rt2 dip)"*
and *"the (rt2 dip) ⊑ the (rt1 dip)"*
shows *"rt1 $\approx_{dip}$ rt2"*
⟨*proof*⟩

lemma *nsqn_rt_fresh_asI:*
assumes *"dip ∈ kD(rt)"*
and *"dip ∈ kD(rt')"*
and *"nsqn rt dip = nsqn rt' dip"*
and *"π_5(the (rt dip)) = π_5(the (rt' dip))"*
shows *"rt $\approx_{dip}$ rt'"*
⟨*proof*⟩

lemma *rt_fresh_asE [elim]:*
assumes *"rt1 $\approx_{dip}$ rt2"*
and *"⟦ rt1 $\sqsubseteq_{dip}$ rt2; rt2 $\sqsubseteq_{dip}$ rt1 ⟧ ⟹ P rt1 rt2 dip"*
shows *"P rt1 rt2 dip"*
⟨*proof*⟩

lemma *rt_fresh_asD1 [dest]:*
assumes *"rt1 $\approx_{dip}$ rt2"*
shows *"rt1 $\sqsubseteq_{dip}$ rt2"*
⟨*proof*⟩

lemma *rt_fresh_asD2 [dest]:*
assumes *"rt1 $\approx_{dip}$ rt2"*
shows *"rt2 $\sqsubseteq_{dip}$ rt1"*
⟨*proof*⟩

lemma *rt_fresh_as_sym:*
assumes *"rt1 $\approx_{dip}$ rt2"*
shows *"rt2 $\approx_{dip}$ rt1"*
⟨*proof*⟩

lemma *not_rt_fresh_asI1 [intro]:*
assumes *"¬ (rt1 $\sqsubseteq_{dip}$ rt2)"*
shows *"¬ (rt1 $\approx_{dip}$ rt2)"*
⟨*proof*⟩

lemma *not_rt_fresh_asI2 [intro]:*
assumes *"¬ (rt2 $\sqsubseteq_{dip}$ rt1)"*
shows *"¬ (rt1 $\approx_{dip}$ rt2)"*
⟨*proof*⟩

lemma *not_single_rt_fresher [elim]:*
assumes *"¬(the (rt1 ip) ⊑ the (rt2 ip))"*
shows *"¬(rt1 $\sqsubseteq_{ip}$ rt2)"*
⟨*proof*⟩

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

lemma *not_rt_fresher_single [elim]:*
assumes *"¬(rt1 $\sqsubseteq_{ip}$ rt2)"*
shows *"¬(the (rt1 ip) ⊑ the (rt2 ip))"*
⟨*proof*⟩

lemma *rt_fresh_as_nsqnr:*
assumes *"dip ∈ kD(rt1)"*
and *"dip ∈ kD(rt2)"*
and *"rt1 $\approx_{dip}$ rt2"*
shows *"nsqn$_r$ (the (rt2 dip)) = nsqn$_r$ (the (rt1 dip))"*
⟨*proof*⟩

lemma *rt_fresher_mapupd [intro!]:*
assumes *"dip∈kD(rt)"*
and *"the (rt dip) ⊑ r"*
shows *"rt $\sqsubseteq_{dip}$ 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 ≡ 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 $\sqsubseteq_i$ rt2"*
and *"$\neg$(rt2 $\sqsubseteq_i$ rt1)"*
shows *"rt1 $\sqsubset_i$ rt2"*
⟨*proof*⟩

lemma *rt_strictly_fresherE' [elim]:*
assumes *"rt1 $\sqsubset_i$ rt2"*
and *"⟦ rt1 $\sqsubseteq_i$ rt2; $\neg$(rt2 $\sqsubseteq_i$ rt1) ⟧ $\Longrightarrow$ P rt1 rt2 i"*
shows *"P rt1 rt2 i"*
⟨*proof*⟩

lemma *rt_strictly_fresherI [intro]:*
assumes *"rt1 $\sqsubseteq_i$ rt2"*
and *"$\neg$(rt1 $\approx_i$ rt2)"*
shows *"rt1 $\sqsubset_i$ rt2"*
⟨*proof*⟩

lemmas *rt_strictly_fresher_singleI [elim] = rt_strictly_fresherI [OF single_rt_fresher]*

lemma *rt_strictly_fresherE [elim]:*
assumes *"rt1 $\sqsubset_i$ rt2"*
and *"⟦ rt1 $\sqsubseteq_i$ rt2; $\neg$(rt1 $\approx_i$ rt2) ⟧ $\Longrightarrow$ P rt1 rt2 i"*
shows *"P rt1 rt2 i"*
⟨*proof*⟩

lemma *rt_strictly_fresher_def':*
"rt1 $\sqsubset_i$ rt2 =
(nsqn$_r$ (the (rt1 i)) < nsqn$_r$ (the (rt2 i))
$\vee$ (nsqn$_r$ (the (rt1 i)) = nsqn$_r$ (the (rt2 i)) $\wedge$ π_5(the (rt1 i)) > π_5(the (rt2 i))))"
⟨*proof*⟩

lemma *rt_strictly_fresher_fresherD [dest]:*
assumes *"rt1 $\sqsubset_{dip}$ rt2"*
shows *"the (rt1 dip) $\sqsubseteq$ the (rt2 dip)"*
⟨*proof*⟩

lemma *rt_strictly_fresher_not_fresh_asD [dest]:*
assumes *"rt1 $\sqsubset_{dip}$ rt2"*
shows *"$\neg$ rt1 $\approx_{dip}$ rt2"*
⟨*proof*⟩

lemma *rt_strictly_fresher_trans [elim, trans]:*
assumes *"rt1 $\sqsubset_{dip}$ rt2"*
and *"rt2 $\sqsubset_{dip}$ rt3"*
shows *"rt1 $\sqsubset_{dip}$ rt3"*
⟨*proof*⟩

lemma *rt_strictly_fresher_irefl [simp]: "$\neg$ (rt $\sqsubset_{dip}$ rt)"*
⟨*proof*⟩

lemma *rt_fresher_trans_rt_strictly_fresher [elim, trans]:*
assumes *"rt1 $\sqsubset_{dip}$ rt2"*
and *"rt2 $\sqsubseteq_{dip}$ rt3"*
shows *"rt1 $\sqsubset_{dip}$ rt3"*
⟨*proof*⟩

lemma *rt_fresher_trans_rt_strictly_fresher' [elim, trans]:*
assumes *"rt1 $\sqsubseteq_{dip}$ rt2"*
and *"rt2 $\sqsubset_{dip}$ rt3"*
shows *"rt1 $\sqsubset_{dip}$ rt3"*
⟨*proof*⟩

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

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

lemma *rt_strictly_fresher_ltI [intro]:*
assumes *"dip ∈ kD(rt1)"*
and *"dip ∈ kD(rt2)"*
and *"nsqn rt1 dip < nsqn rt2 dip"*
shows *"rt1 ⊏$_{dip}$ rt2"*
⟨*proof*⟩

lemma *rt_strictly_fresher_eqI [intro]:*
assumes *"i∈kD(rt1)"*
and *"i∈kD(rt2)"*
and *"nsqn rt1 i = nsqn rt2 i"*
and *"π_5(the (rt2 i)) < π_5(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 ⊨A onll Γ_AODV (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
```
⟨*proof*⟩

lemma *sequence_number_one_or_bigger:*
```
"paodv i ⊨ onl Γ_AODV (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 Γ_AODV (λ(ξ, l). l ∈ ({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 Γ_AODV (λ(ξ, l). (l = 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 Γ_AODV (λ(ξ, l). (l ∈ {PRreq-:16..PRreq-:18} ∪ {PRrep-:2..PRrep-:6} ⟶ dip ξ ∈ kD (rt ξ))
                  ∧ (l ∈ {PRreq-:3..PRreq-:17} ⟶ oip ξ ∈ kD (rt ξ)))"
```
⟨*proof*⟩

Proposition 7.38

lemma *includes_nhip:*
```
"paodv i ⊨ onl Γ_AODV (λ(ξ, l). ∀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 Γ_AODV (λ(ξ, l). (l ∈ {PRreq-:16..PRreq-:18} ⟶ dip ξ ∈ kD (rt ξ)) ∧
                            (l = PRreq-:17 ⟶ oip ξ ∈ kD (rt ξ)) ∧
                            (l = PRrep-:5  ⟶ dip ξ ∈ kD (rt ξ)) ∧
                            (l = PRrep-:6  ⟶ (the (nhop (rt ξ) (dip ξ))) ∈ kD (rt ξ)))"
(is "_ ⊨ onl Γ_AODV ?P")
```
⟨*proof*⟩

Proposition 7.4

lemma *known_destinations_increase:*
```
"paodv i ⊨A onll Γ_AODV (λ((ξ, _), _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ'))"
```
⟨*proof*⟩

Proposition 7.5

lemma *rreqs_increase:*
```
"paodv i ⊨A onll Γ_AODV (λ((ξ, _), _, (ξ', _)). rreqs ξ ⊆ rreqs ξ')"
```
⟨*proof*⟩

lemma *dests_bigger_than_sqn:*

```
"paodv i ⊨ onl Γ_AODV (λ(ξ, l). l ∈ {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 Γ_AODV (λ((ξ, _), _, (ξ', _)). ∀ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip)"
```

⟨*proof*⟩

Proposition 7.7

lemma *ip_constant:*

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

⟨*proof*⟩

Proposition 7.8

lemma *sender_ip_valid':*

```
"paodv i ⊨_A onll Γ_AODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = ip ξ) a)"
```

⟨*proof*⟩

lemma *sender_ip_valid:*

```
"paodv i ⊨_A onll Γ_AODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = i) a)"
```

⟨*proof*⟩

lemma *received_msg_inv:*

```
"paodv i ⊨ (recvmsg P →) onl Γ_AODV (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 Γ_AODV (λ(ξ, _). sip ξ ≠ ip ξ)"
```

⟨*proof*⟩

lemma *sip_not_ip:*

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

⟨*proof*⟩

lemma *rreq_dip_in_vD_dip_eq_ip:*

```
"paodv i ⊨ onl Γ_AODV (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 Γ_AODV (λ(_, a, _). anycast msg_zhops a)"
```

⟨*proof*⟩

lemma *hop_count_zero_oip_dip_sip:*

```
"paodv i ⊫ (recvmsg msg_zhops →) onl Γ_AODV (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 Γ_AODV (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 Γ_AODV (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 Γ_AODV (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 Γ_AODV (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 Γ_AODV (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*⟩

Proposition 7.12

lemma *zero_seq_unk_hops_one':*
```
"paodv i ⊫ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (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 Γ_AODV (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 Γ_AODV (λ(ξ, l).
                          ∀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 Γ_AODV (λ(_, a, _). anycast rreq_rrep_sn a)"
```
⟨*proof*⟩

Proposition 7.14

lemma *rreq_rrep_fresh_any_step_invariant:*

```
"paodv i ⊨_A onll Γ_AODV (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a)"
```
⟨*proof*⟩

Proposition 7.15

lemma *rerr_invalid_any_step_invariant:*
```
"paodv i ⊨_A onll Γ_AODV (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* ⇀ *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 Γ_AODV (λ(ξ, l). (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 Γ_AODV (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 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 *∗: "ip∈kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip*
∧ (nsqn (rt (σ sip)) ip = sn
⟶ (the (dhops (rt (σ sip)) ip) ≤ hops
∨ the (flag (rt (σ sip)) ip) = inv))"
shows *"ip∈kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip*
∧ (nsqn (rt (σ' sip)) ip = sn
⟶ (the (dhops (rt (σ' sip)) ip) ≤ hops
∨ the (flag (rt (σ' sip)) ip) = inv))"
⟨*proof*⟩

lemma *rteq_quality_increases:*
assumes *"∀j. j ≠ i ⟶ quality_increases (σ j) (σ' j)"*
and *"rt (σ' i) = rt (σ i)"*
shows *"∀j. quality_increases (σ j) (σ' j)"*
⟨*proof*⟩

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

lemma *msg_fresh [simp]:*
"⋀hops rreqid dip dsn dsk oip osn sip.
msg_fresh σ (Rreq hops rreqid dip dsn dsk oip osn sip) =
(osn ≥ 1 ∧ (sip ≠ oip ⟶ oip∈kD(rt (σ sip))
∧ nsqn (rt (σ sip)) oip ≥ osn
∧ (nsqn (rt (σ sip)) oip = osn
⟶ (hops ≥ the (dhops (rt (σ sip)) oip)

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

lemma msg_fresh_inc_sn [simp, elim]:
  "msg_fresh 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 σ_AODV' :: "((ip ⇒ state) × ((state, msg, pseqp, pseqp label) seqp)) set"
where "σ_AODV' ≡ {(λi. aodv_init i, Γ_AODV PAodv)}"

abbreviation opaodv
```

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

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

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

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

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

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

lemma oaodv_trans: "trans (opaodv i) = oseqp_sos Γ_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
```

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 Γ_AODV (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 Γ_AODV (λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ')"
```

⟨*proof*⟩

lemmas *olocal_quality_increases =*
open_seq_step_invariant [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, 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 *"_ ⊨ (?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 Γ_AODV (λ(σ, l).
                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 Γ_AODV (λ(ξ, l). (l = PRerr-:1
                     ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ)
                                        ∧ the (nhop (rt ξ) ip) = sip ξ
                                        ∧ sqn (rt ξ) ip < the (dests ξ ip))))"
  ⟨proof⟩

lemmas 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 Γ_AODV (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 Γ$_{AODV}$ (λ(σ, _). ∀dip. let nhip = the (nhop (rt (σ i)) dip)
in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip))
∧ nhip ≠ dip
⟶ (rt (σ i)) ⊏$_{dip}$ (rt (σ nhip)))"
(is *"_ ⊨ (?S i, _ →) _")*
⟨*proof*⟩

lemma *seq_nhop_quality_increases:*
shows *"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

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 ⇒ 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]:*
"⋀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

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 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)"
⟨*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 Γ_{AODV} i) (seqp_sos Γ_{QMSG}))"
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 : 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*⟩

0.12.3 Lift to partial networks

lemma *arrive_rreq_rrep_nsqn_fresh_inc_sn [simp]:*
assumes "oarrivemsg (λσ m. msg_fresh σ m ∧ P σ m) σ m"
shows "oarrivemsg (λ_. rreq_rrep_sn) σ m"
⟨*proof*⟩

lemma *opnet_nhop_quality_increases:*
shows "opnet (λi. 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*⟩

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 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*⟩

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_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 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*⟩

0.12.6 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 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" (‹π_2›)*
  where *"π_2 ≡ λ(dsn, _, _, _, _, _). dsn"*

definition *proj3 :: "r ⇒ k" (‹π_3›)*
  where *"π_3 ≡ λ(_, dsk, _, _, _, _). dsk"*

definition *proj4 :: "r ⇒ f" (‹π_4›)*

where "$\pi_4 \equiv \lambda(\_, \_, flag, \_, \_, \_).\ flag$"

definition *proj5 :: "r ⇒ nat"* (‹π_5›)
where "$\pi_5 \equiv \lambda(\_, \_, \_, hops, \_, \_).\ hops$"

definition *proj6 :: "r ⇒ ip"* (‹π_6›)
where "$\pi_6 \equiv \lambda(\_, \_, \_, \_, nhip, \_).\ nhip$"

definition *proj7 :: "r ⇒ ip set"* (‹π_7›)
where "$\pi_7 \equiv \lambda(\_, \_, \_, \_, \_, pre).\ pre$"

lemma *projs [simp]:*
"π_2(dsn, dsk, flag, hops, nhip, pre) = dsn"
"π_3(dsn, dsk, flag, hops, nhip, pre) = dsk"
"π_4(dsn, dsk, flag, hops, nhip, pre) = flag"
"π_5(dsn, dsk, flag, hops, nhip, pre) = hops"
"π_6(dsn, dsk, flag, hops, nhip, pre) = nhip"
"π_7(dsn, dsk, flag, hops, nhip, pre) = pre"
⟨*proof*⟩

lemma *proj3_pred [intro]:* "⟦ P kno; P unk ⟧ ⟹ P (π_3 x)"
⟨*proof*⟩

lemma *proj4_pred [intro]:* "⟦ P val; P inv ⟧ ⟹ P (π_4 x)"
⟨*proof*⟩

lemma *proj6_pair_snd [simp]:*
fixes *dsn' r*
shows "π_6 (dsn', snd (r)) = π_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" (‹σ_{route}'(_, _')›)

translations
"σ_{route}(rt, dip)" => "rt dip"

definition *sqn :: "rt ⇒ ip ⇒ sqn"*
where "sqn rt dip ≡ case σ_{route}(rt, dip) of Some r ⇒ π_2(r) | None ⇒ 0"

definition *sqnf :: "rt ⇒ ip ⇒ k"*
where "sqnf rt dip ≡ case σ_{route}(rt, dip) of Some r ⇒ π_3(r) | None ⇒ unk"

abbreviation *flag :: "rt ⇒ ip ⇀ f"*
where "flag rt dip ≡ map_option π_4 (σ_{route}(rt, dip))"

abbreviation *dhops :: "rt ⇒ ip ⇀ nat"*
where "dhops rt dip ≡ map_option π_5 (σ_{route}(rt, dip))"

abbreviation *nhop :: "rt ⇒ ip ⇀ ip"*
where "nhop rt dip ≡ map_option π_6 (σ_{route}(rt, dip))"

abbreviation *precs :: "rt ⇒ ip ⇀ ip set"*
where "precs rt dip ≡ map_option π_7 (σ_{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 "π_3(addpre v pre) = π_3(v)"
⟨*proof*⟩

lemma *proj4_addpre:*
fixes v pre
shows "π_4(addpre v pre) = π_4(v)"
⟨*proof*⟩

lemma *proj5_addpre:*
fixes v pre
shows "π_5(addpre v pre) = π_5(v)"
⟨*proof*⟩

lemma *proj6_addpre:*
fixes dsn dsk flag hops nhip pre npre
shows "π_6(addpre v npre) = π_6(v)"
⟨*proof*⟩

lemma *proj7_addpre:*
fixes dsn dsk flag hops nhip pre npre
shows "π_7(addpre v npre) = π_7(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)) (σ_{route}(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 "π_2(the (the (addpreRT rt ip' npre) ip)) = π_2(the (rt ip))"
⟨*proof*⟩

lemma *proj3_addpreRT [simp]:*
fixes ip rt ip' npre
assumes "ip∈kD rt"
and "ip'∈kD rt"
shows "π_3(the (the (addpreRT rt ip' npre) ip)) = π_3(the (rt ip))"
⟨*proof*⟩

lemma *proj5_addpreRT [simp]:*
"⋀rt dip ip npre. dip∈kD(rt) ⟹ π_5(the (the (addpreRT rt dip npre) ip)) = π_5(the (rt ip))"
⟨*proof*⟩

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

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

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

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

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

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

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

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

Updating route entries

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

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

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

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

shows *"sqn rt dip = 0"*
⟨*proof*⟩

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

lemma *update_arg_wf_gives_cases:*
"⋀r. update_arg_wf r ⟹ (π₂(r) = 0) = (π₃(r) = unk)"
⟨*proof*⟩

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

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

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

definition *update :: "rt ⇒ ip ⇒ r ⇒ rt"*
where
"update rt ip r ≡
case σ_route(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 σ_route(rt, ip)"*
and *"nr ≡ addpre r (π₇(s))"*
and *"nr' ≡ (π₂(s), π₃(nr), π₄(nr), π₅(nr), π₆(nr), π₇(nr))"*
and *"ns ≡ addpre s (π₇(r))"*
shows
"⟦ip ∉ kD(rt)⟧ ⟹ update rt ip r = rt (ip ↦ r)"
"⟦ip ∈ kD(rt); sqn rt ip < π₂(r)⟧ ⟹ update rt ip r = rt (ip ↦ nr)"
"⟦ip ∈ kD(rt); sqn rt ip = π₂(r);
the (dhops rt ip) > π₅(r)⟧ ⟹ update rt ip r = rt (ip ↦ nr)"
"⟦ip ∈ kD(rt); sqn rt ip = π₂(r);
flag rt ip = Some inv⟧ ⟹ update rt ip r = rt (ip ↦ nr)"
"⟦ip ∈ kD(rt); π₃(r) = unk; (π₂(r) = 0) = (π₃(r) = unk)⟧ ⟹ update rt ip r = rt (ip ↦ nr')"
"⟦ip ∈ kD(rt); sqn rt ip ≥ π₂(r); π₃(r) = kno;
sqn rt ip = π₂(r) ⟹ the (dhops rt ip) ≤ π₅(r) ∧ the (flag rt ip) = val ⟧
⟹ update rt ip r = rt (ip ↦ ns)"
⟨*proof*⟩

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

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

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

lemma in_kD_after_update [simp]:
  fixes rt nip dsn dsk flag hops nhip pre
  shows "kD (update rt nip (dsn, dsk, flag, hops, nhip, pre)) = insert nip (kD rt)"
  ⟨proof⟩

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

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

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

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

lemma dhops_update [intro]:
    fixes rt dsn dsk flag hops ip rip nhip pre
```

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

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

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

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

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

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

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

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

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

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

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

lemma *update_kno_dsn_greater_zero:*

```
"⋀rt dip ip dsn hops npre. 1 ≤ dsn ⟹ 1 ≤ (sqn (update rt dip (dsn, kno, val, hops, ip, npre)) dip)"
```
⟨*proof*⟩

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

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

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

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

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

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

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

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

lemma *vD_update_val [dest]:*
```
"⋀dip 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]:*

"⋀dip. π_3(the ((invalidate rt dests) dip)) = π_3(the (rt dip))"
⟨proof⟩

lemma *proj5_invalidate [simp]:*
"⋀dip. π_5(the ((invalidate rt dests) dip)) = π_5(the (rt dip))"
⟨proof⟩

lemma *proj6_invalidate [simp]:*
"⋀dip. π_6(the ((invalidate rt dests) dip)) = π_6(the (rt dip))"
⟨proof⟩

lemma *proj7_invalidate [simp]:*
"⋀dip. π_7(the ((invalidate rt dests) dip)) = π_7(the (rt dip))"
⟨proof⟩

1.1.5 Route Requests

lemma *invalidate_kD_inv [simp]:*
"⋀rt dests. kD (invalidate rt dests) = kD rt"
⟨proof⟩

lemma *invalidate_sqn:*
fixes *rt dip dests*
assumes "∀rsn. dests dip = Some rsn ⟶ sqn rt dip ≤ rsn"
shows "sqn rt dip ≤ sqn (invalidate rt dests) dip"
⟨proof⟩

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

lemma *dhops_invalidate [simp]:*
"⋀dip. the (dhops (invalidate rt dests) dip) = the (dhops rt dip)"
⟨proof⟩

lemma *sqnf_invalidate [simp]:*
"⋀dip. sqnf (invalidate (rt ξ) (dests ξ)) dip = sqnf (rt ξ) dip"
⟨proof⟩

lemma *nhop_invalidate [simp]:*
"⋀dip. the (nhop (invalidate (rt ξ) (dests ξ)) dip) = the (nhop (rt ξ) dip)"
⟨proof⟩

lemma *invalidate_other [simp]:*
fixes *rt dests dip*
assumes "dip∉dom(dests)"
shows "invalidate rt dests dip = rt dip"
⟨proof⟩

lemma *invalidate_none [simp]:*
fixes *rt dests dip*
assumes "dip∉kD(rt)"
shows "invalidate rt dests dip = None"
⟨proof⟩

lemma *vD_invalidate_vD_not_dests:*
"⋀dip rt dests. dip∈vD(invalidate rt dests) ⟹ dip∈vD(rt) ∧ dests dip = None"
⟨proof⟩

lemma *sqn_invalidate_not_in_dests [simp]:*
fixes *dests dip rt*

assumes *"dip∉dom(dests)"*
shows *"sqn (invalidate rt dests) dip = sqn rt dip"*
⟨*proof*⟩

lemma *invalidate_changes:*
fixes *rt dests dip dsn dsk flag hops nhip pre*
assumes *"invalidate rt dests dip = Some (dsn, dsk, flag, hops, nhip, pre)"*
shows *" dsn = (case dests dip of None ⇒ 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*⟩

1.1.6 Queued Packets

Functions for sending data packets.

type_synonym *store = "ip ⇀ (p × data list)"*

definition *sigma_queue :: "store ⇒ ip ⇒ data list"* (‹σ_{queue}'(_, _')›)
where *"σ_{queue}(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"* (‹$\sigma_{p\text{-}flag}$'(_, _')›)
where *"$\sigma_{p\text{-}flag}$(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)"

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* ⇒ *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 ξ ≡ case msg ξ of*
Rrep hops' dip' dsn' oip' sip' ⇒
{ ξ(| hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' |) }
| _ ⇒ {}"

lemma *is_rrep_asm [dest!]:*
assumes *"ξ' ∈ is_rrep ξ"*
shows *"(∃hops' dip' dsn' oip' sip'.*
msg ξ = Rrep hops' dip' dsn' oip' sip' ∧
ξ' = ξ(| hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' |))"
⟨*proof*⟩

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

lemma *is_rerr_asm [dest!]:*
assumes *"ξ' ∈ is_rerr ξ"*
shows *"(∃dests' sip'.*
msg ξ = Rerr dests' sip' ∧
ξ' = ξ(| dests := dests', sip := sip' |))"
⟨*proof*⟩

lemmas *is_msg_defs =*
is_rerr_def is_rrep_def is_rreq_def is_pkt_def is_newpkt_def

lemma *is_msg_inv_ip [simp]:*
"ξ' ∈ is_rerr ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_rrep ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_rreq ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_pkt ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_newpkt ξ ⟹ ip ξ' = ip ξ"
⟨*proof*⟩

lemma *is_msg_inv_sn [simp]:*
"ξ' ∈ is_rerr ξ ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_rrep ξ ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_rreq ξ ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_pkt ξ ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_newpkt ξ ⟹ sn ξ' = sn ξ"
⟨*proof*⟩

lemma *is_msg_inv_rt [simp]:*
"ξ' ∈ is_rerr ξ ⟹ rt ξ' = rt ξ"
"ξ' ∈ is_rrep ξ ⟹ rt ξ' = rt ξ"
"ξ' ∈ is_rreq ξ ⟹ rt ξ' = rt ξ"
"ξ' ∈ is_pkt ξ ⟹ rt ξ' = rt ξ"
"ξ' ∈ is_newpkt ξ ⟹ rt ξ' = rt ξ"
⟨*proof*⟩

lemma *is_msg_inv_rreqs [simp]:*
"ξ' ∈ is_rerr ξ ⟹ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_rrep ξ ⟹ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_rreq ξ ⟹ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_pkt ξ ⟹ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_newpkt ξ ⟹ rreqs ξ' = rreqs ξ"
⟨*proof*⟩

lemma *is_msg_inv_store [simp]:*
"ξ' ∈ is_rerr ξ ⟹ store ξ' = store ξ"
"ξ' ∈ is_rrep ξ ⟹ store ξ' = store ξ"
"ξ' ∈ is_rreq ξ ⟹ store ξ' = store ξ"

```
  "ξ' ∈ is_pkt ξ    ⟹ store ξ' = store ξ"
  "ξ' ∈ is_newpkt ξ ⟹ store ξ' = store ξ"
  ⟨proof⟩

lemma is_msg_inv_sip [simp]:
  "ξ' ∈ is_pkt ξ    ⟹ sip ξ' = sip ξ"
  "ξ' ∈ is_newpkt ξ ⟹ sip ξ' = sip ξ"
  ⟨proof⟩
```

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]: "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 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 Γ_AODV :: "(state, msg, pseqp, pseqp label) seqp_env"
where
  "Γ_AODV 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(σ_queue(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 (σ_p-flag(store ξ, dip)) = req }⟩
          ⟦ξ. ξ (| store := unsetRRF (store ξ) (dip ξ) |)⟧
          ⟦ξ. ξ (| sn := inc (sn ξ) |)⟧
          ⟦ξ. ξ (| rreqs := rreqs ξ ∪ {(ip ξ, sn ξ)} |)⟧
          broadcast(λξ. rreq(0, dip ξ, sqn (rt ξ) (dip ξ), sqnf (rt ξ) (dip ξ),
                             ip ξ, sn ξ, ip ξ)). AODV())"

|  "Γ_AODV 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())"

| "Γ_AODV PPkt = labelled PPkt (
     ⟨ξ. dip ξ = ip ξ⟩
        deliver(λξ. data ξ).AODV()
     ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
     (
       ⟨ξ. dip ξ ∈ vD (rt ξ)⟩
         unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, oip ξ)).AODV()
```

```
      ▷
        ⟦ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
                                 then Some (inc (sqn (rt ξ) rip)) else None) |)⟧
        ⟦ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)⟧
        ⟦ξ. ξ (| store := setRRF (store ξ) (dests ξ)|)⟧
        ⟦ξ. ξ (| pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } |)⟧
        ⟦ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                 then (dests ξ) rip else None) |)⟧
        groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
    ⊕ ⟨ξ. dip ξ ∉ vD (rt ξ)⟩
    (
        ⟨ξ. dip ξ ∈ iD (rt ξ)⟩
          groupcast(λξ. the (precs (rt ξ) (dip ξ)),
                    λξ. rerr([dip ξ ↦ sqn (rt ξ) (dip ξ)], ip ξ)). AODV()
        ⊕ ⟨ξ. dip ξ ∉ iD (rt ξ)⟩
           AODV()
    )
  ))"

| "Γ_AODV PRreq = labelled PRreq (
    ⟨ξ. (oip ξ, osn ξ) ∈ rreqs ξ⟩
      AODV()
    ⊕ ⟨ξ. (oip ξ, osn ξ) ∉ rreqs ξ⟩
      ⟦ξ. ξ (| rt := update (rt ξ) (oip ξ) (osn ξ, kno, val, hops ξ + 1, sip ξ, {}) |)⟧
      ⟦ξ. ξ (| rreqs := rreqs ξ ∪ {(oip ξ, osn ξ)} |)⟧
      (
        ⟨ξ. dip ξ = ip ξ⟩
          ⟦ξ. ξ (| sn := max (sn ξ) (dsn ξ) |)⟧
          unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(0, dip ξ, sn ξ, oip ξ, ip ξ)).AODV()
          ▷
            ⟦ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
                                     then Some (inc (sqn (rt ξ) rip)) else None) |)⟧
            ⟦ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)⟧
            ⟦ξ. ξ (| store := setRRF (store ξ) (dests ξ)|)⟧
            ⟦ξ. ξ (| pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } |)⟧
            ⟦ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                     then (dests ξ) rip else None) |)⟧
            groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
        ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
        (
          ⟨ξ. dip ξ ∈ vD (rt ξ) ∧ dsn ξ ≤ sqn (rt ξ) (dip ξ) ∧ sqnf (rt ξ) (dip ξ) = kno⟩
            ⟦ξ. ξ (| rt := the (addpreRT (rt ξ) (dip ξ) {sip ξ}) |)⟧
            ⟦ξ. ξ (| rt := the (addpreRT (rt ξ) (oip ξ) {the (nhop (rt ξ) (dip ξ))}) |)⟧
            unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(the (dhops (rt ξ) (dip ξ)), dip ξ,
                        sqn (rt ξ) (dip ξ), oip ξ, ip ξ)).
            AODV()
          ▷
            ⟦ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
                                     then Some (inc (sqn (rt ξ) rip)) else None) |)⟧
            ⟦ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)⟧
            ⟦ξ. ξ (| store := setRRF (store ξ) (dests ξ)|)⟧
            ⟦ξ. ξ (| pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } |)⟧
            ⟦ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                     then (dests ξ) rip else None) |)⟧
            groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
          ⊕ ⟨ξ. dip ξ ∉ vD (rt ξ) ∨ sqn (rt ξ) (dip ξ) < dsn ξ ∨ sqnf (rt ξ) (dip ξ) = unk⟩
            broadcast(λξ. rreq(hops ξ + 1, dip ξ, max (sqn (rt ξ) (dip ξ)) (dsn ξ),
                               dsk ξ, oip ξ, osn ξ, ip ξ)).
            AODV()
        )
      ))"

| "Γ_AODV 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()
    )"

| "Γ_AODV PRerr = labelled PRerr (
    ⟦ξ. ξ (| dests := (λrip. case (dests ξ) rip of None ⇒ None
                      | Some rsn ⇒ if rip ∈ vD (rt ξ) ∧ the (nhop (rt ξ) rip) = sip ξ
                                     ∧ sqn (rt ξ) rip < rsn then Some rsn else None) |)⟧
    ⟦ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)⟧
    ⟦ξ. ξ (| store := setRRF (store ξ) (dests ξ)|)⟧
    ⟦ξ. ξ (| pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } |)⟧
    ⟦ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                           then (dests ξ) rip else None) |)⟧
    groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)). AODV())"

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

fun Γ_AODV_skeleton
where
    "Γ_AODV_skeleton PAodv   = seqp_skeleton (Γ_AODV PAodv)"
  | "Γ_AODV_skeleton PNewPkt = seqp_skeleton (Γ_AODV PNewPkt)"
  | "Γ_AODV_skeleton PPkt    = seqp_skeleton (Γ_AODV PPkt)"
  | "Γ_AODV_skeleton PRreq   = seqp_skeleton (Γ_AODV PRreq)"
  | "Γ_AODV_skeleton PRrep   = seqp_skeleton (Γ_AODV PRrep)"
  | "Γ_AODV_skeleton PRerr   = seqp_skeleton (Γ_AODV PRerr)"

lemma Γ_AODV_skeleton_wf [simp]:
  "wellformed Γ_AODV_skeleton"
  ⟨proof⟩

declare Γ_AODV_skeleton.simps [simp del, code del]
lemmas Γ_AODV_skeleton_simps [simp, code]
          = Γ_AODV_skeleton.simps [simplified Γ_AODV_simps seqp_skeleton.simps]

lemma aodv_proc_cases [dest]:
  fixes p pn
```

shows "p ∈ ctermsl (Γ_{AODV} pn) ⟹
(p ∈ ctermsl (Γ_{AODV} PAodv) ∨
p ∈ ctermsl (Γ_{AODV} PNewPkt) ∨
p ∈ ctermsl (Γ_{AODV} PPkt) ∨
p ∈ ctermsl (Γ_{AODV} PRreq) ∨
p ∈ ctermsl (Γ_{AODV} PRrep) ∨
p ∈ ctermsl (Γ_{AODV} PRerr))"
⟨*proof*⟩

definition σ_{AODV} :: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp) set"
where "σ_{AODV} i ≡ {(aodv_init i, Γ_{AODV} PAodv)}"

abbreviation paodv
:: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
"paodv i ≡ (| init = σ_{AODV} i, trans = seqp_sos Γ_{AODV} |)"

lemma aodv_trans: "trans (paodv i) = seqp_sos Γ_{AODV}"
⟨*proof*⟩

lemma aodv_control_within [simp]: "control_within Γ_{AODV} (init (paodv i))"
⟨*proof*⟩

lemma aodv_wf [simp]:
"wellformed Γ_{AODV}"
⟨*proof*⟩

lemmas aodv_labels_not_empty [simp] = labels_not_empty [OF aodv_wf]

lemma aodv_ex_label [intro]: "∃l. l∈labels Γ_{AODV} p"
⟨*proof*⟩

lemma aodv_ex_labelE [elim]:
assumes "∀l∈labels Γ_{AODV} p. P l p"
and "∃p l. P l p ⟹ Q"
shows "Q"
⟨*proof*⟩

lemma aodv_simple_labels [simp]: "simple_labels Γ_{AODV}"
⟨*proof*⟩

lemma σ_{AODV}_labels [simp]: "(ξ, p) ∈ σ_{AODV} i ⟹ labels Γ_{AODV} p = {PAodv-:0}"
⟨*proof*⟩

lemma aodv_init_kD_empty [simp]:
"(ξ, p) ∈ σ_{AODV} 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) ∈ σ_{AODV} i"
shows "sip ξ ≠ ip ξ"
⟨*proof*⟩

lemma aodv_init_sip_not_i [simp]:
assumes "(ξ, p) ∈ σ_{AODV} 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
  Γ_{AODV}*_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

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_simps [simp]:*
"⋀hops dip dsn dsk oip osn sip.
msg_sender (Rreq hops dip dsn dsk oip osn sip) = sip"
"⋀hops dip dsn oip sip. msg_sender (Rrep hops dip dsn oip sip) = sip"
"⋀dests sip. msg_sender (Rerr dests sip) = sip"
"⋀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_simps [simp]:*
"⋀hops dip dsn dsk oip osn sip.
rreq_rrep_sn (Rreq hops dip dsn dsk oip osn sip) = (osn ≥ 1)"
"⋀hops dip dsn oip sip. rreq_rrep_sn (Rrep hops dip dsn oip sip) = (dsn ≥ 1)"
"⋀dests sip. rreq_rrep_sn (Rerr dests sip) = True"

```
  "⋀d dip.                rreq_rrep_sn (Newpkt d dip)   = True"
  "⋀d dip sip.            rreq_rrep_sn (Pkt d dip sip)  = True"
```
⟨*proof*⟩

definition *rreq_rrep_fresh :: "rt ⇒ msg ⇒ bool"*
where *"rreq_rrep_fresh crt m ≡ case m of Rreq hopsc _ _ _ oipc osnc ipcc ⇒ (ipcc ≠ oipc ⟶*
oipc∈kD(crt) ∧ (sqn crt oipc > osnc
∨ (sqn crt oipc = osnc
∧ the (dhops crt oipc) ≤ hopsc
∧ the (flag crt oipc) = val)))
| Rrep hopsc dipc dsnc _ ipcc ⇒ (ipcc ≠ dipc ⟶
dipc∈kD(crt)
∧ sqn crt dipc = dsnc
∧ the (dhops crt dipc) = hopsc
∧ the (flag crt dipc) = val)
| _ ⇒ True"

lemma *rreq_rrep_fresh [simp]:*
```
  "⋀hops dip dsn dsk oip osn sip.
          rreq_rrep_fresh crt (Rreq hops dip dsn dsk oip osn sip) =
                              (sip ≠ oip ⟶ oip∈kD(crt)
                                         ∧ (sqn crt oip > osn
                                            ∨ (sqn crt oip = osn
                                               ∧ the (dhops crt oip) ≤ hops
                                               ∧ the (flag crt oip) = val)))"
  "⋀hops dip dsn oip sip. rreq_rrep_fresh crt (Rrep hops dip dsn oip sip) =
                              (sip ≠ dip ⟶ dip∈kD(crt)
                                         ∧ sqn crt dip = dsn
                                         ∧ the (dhops crt dip) = hops
                                         ∧ the (flag crt dip) = val)"
  "⋀dests sip.            rreq_rrep_fresh crt (Rerr dests sip) = True"
  "⋀d dip.                rreq_rrep_fresh crt (Newpkt d dip)   = True"
  "⋀d dip sip.            rreq_rrep_fresh crt (Pkt d dip sip)  = True"
```
⟨*proof*⟩

definition *rerr_invalid :: "rt ⇒ msg ⇒ bool"*
where *"rerr_invalid crt m ≡ case m of Rerr destsc _ ⇒ (∀ripc∈dom(destsc).*
(ripc∈iD(crt) ∧ the (destsc ripc) = sqn crt ripc))
| _ ⇒ True"

lemma *rerr_invalid [simp]:*
```
  "⋀hops dip dsn dsk oip osn sip.
                          rerr_invalid crt (Rreq hops dip dsn dsk oip osn sip) = True"
  "⋀hops dip dsn oip sip. rerr_invalid crt (Rrep hops dip dsn oip sip) = True"
  "⋀dests sip.            rerr_invalid crt (Rerr dests sip) = (∀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

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 π_4(r) = val ∨ π_2(r) = 0 then π_2(r) else (π_2(r) - 1)"

lemma *nsqnr_def':*
"$nsqn_r$ r = (if π_4(r) = inv then π_2(r) - 1 else π_2(r))"
⟨*proof*⟩

lemma *$nsqn_r$_zero [simp]:*
"⋀dsn dsk flag hops nhip pre. $nsqn_r$ (0, dsk, flag, hops, nhip, pre) = 0"
⟨*proof*⟩

lemma *$nsqn_r$_val [simp]:*
"⋀dsn dsk hops nhip pre. $nsqn_r$ (dsn, dsk, val, hops, nhip, pre) = dsn"
⟨*proof*⟩

lemma *$nsqn_r$_inv [simp]:*
"⋀dsn dsk hops nhip pre. $nsqn_r$ (dsn, dsk, inv, hops, nhip, pre) = dsn - 1"
⟨*proof*⟩

lemma *$nsqn_r$_lte_dsn [simp]:*
"⋀dsn dsk flag hops nhip pre. $nsqn_r$ (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:*
"⋀rt dip. nsqn rt dip = (if flag rt dip = Some val ∨ sqn rt dip = 0
then sqn rt dip else sqn rt dip - 1)"
⟨*proof*⟩

lemma *not_in_kD_nsqn [simp]:*
assumes *"dip ∉ kD(rt)"*
shows *"nsqn rt dip = 0"*
⟨*proof*⟩

lemma *kD_nsqn:*
assumes *"dip ∈ kD(rt)"*
shows *"nsqn rt dip = nsqn$_r$(the (σ_{route}(rt, dip)))"*
⟨*proof*⟩

lemma *nsqnr_r_flag_pred [simp, intro]:*
fixes *dsn dsk flag hops nhip pre*
assumes *"P (nsqn$_r$ (dsn, dsk, val, hops, nhip, pre))"*
and *"P (nsqn$_r$ (dsn, dsk, inv, hops, nhip, pre))"*
shows *"P (nsqn$_r$ (dsn, dsk, flag, hops, nhip, pre))"*
⟨*proof*⟩

lemma *nsqn$_r$_addpreRT_inv [simp]:*
"⋀rt dip npre dip'. dip ∈ kD(rt) ⟹
nsqn$_r$ (the (the (addpreRT rt dip npre) dip')) = nsqn$_r$ (the (rt dip'))"
⟨*proof*⟩

lemma *sqn_nsqn:*
"⋀rt dip. sqn rt dip - 1 ≤ nsqn rt dip"
⟨*proof*⟩

lemma *nsqn_sqn: "nsqn rt dip ≤ sqn rt dip"*
⟨*proof*⟩

lemma *val_nsqn_sqn [elim, simp]:*
assumes *"ip∈kD(rt)"*
and *"the (flag rt ip) = val"*
shows *"nsqn rt ip = sqn rt ip"*
⟨*proof*⟩

lemma *vD_nsqn_sqn [elim, simp]:*
assumes *"ip∈vD(rt)"*
shows *"nsqn rt ip = sqn rt ip"*
⟨*proof*⟩

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

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

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

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

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

lemma *nsqn_invalidate_eq:*

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

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

1.5.2 Comparing routes

definition
fresher :: "r ⇒ r ⇒ bool" (‹(_/ ⊑ _)› [51, 51] 50)
where
"fresher r r' ≡ ((nsqn$_r$ r < nsqn$_r$ r') ∨ (nsqn$_r$ r = nsqn$_r$ r' ∧ π$_5$(r) ≥ π$_5$(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 *"π$_5$(r) ≥ π$_5$(r')"*
shows *"r ⊑ r'"*
⟨*proof*⟩

lemma *fresherI [intro]:*
assumes *"(nsqn$_r$ r < nsqn$_r$ r') ∨ (nsqn$_r$ r = nsqn$_r$ r' ∧ π$_5$(r) ≥ π$_5$(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' ∧ π$_5$(r) ≥ π$_5$(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 ≡ λdip rt rt'. (the (σ_route(rt, dip))) ⊑ (the (σ_route(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':*
"(rt_1 ⊑_i rt_2) = (nsqn_r (the (rt_1 i)) < nsqn_r (the (rt_2 i)) ∨
nsqn_r (the (rt_1 i)) = nsqn_r (the (rt_2 i)) ∧ π_5 (the (rt_2 i)) ≤ π_5 (the (rt_1 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 *"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 $\sqsubseteq_{dip}$ rt2)"*
shows *"¬ (rt1 $\approx_{dip}$ rt2)"*
⟨*proof*⟩

lemma *not_rt_fresh_asI2 [intro]:*
assumes *"¬ (rt2 $\sqsubseteq_{dip}$ rt1)"*
shows *"¬ (rt1 $\approx_{dip}$ rt2)"*
⟨*proof*⟩

lemma *not_single_rt_fresher [elim]:*
assumes *"¬(the (rt1 ip) $\sqsubseteq$ the (rt2 ip))"*
shows *"¬(rt1 $\sqsubseteq_{ip}$ rt2)"*
⟨*proof*⟩

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

lemma *not_rt_fresher_single [elim]:*
assumes *"¬(rt1 $\sqsubseteq_{ip}$ rt2)"*
shows *"¬(the (rt1 ip) $\sqsubseteq$ the (rt2 ip))"*
⟨*proof*⟩

lemma *rt_fresh_as_nsqnr:*
assumes *"dip $\in$ kD(rt1)"*
and *"dip $\in$ kD(rt2)"*
and *"rt1 $\approx_{dip}$ rt2"*
shows *"$nsqn_r$ (the (rt2 dip)) = $nsqn_r$ (the (rt1 dip))"*
⟨*proof*⟩

lemma *rt_fresher_mapupd [intro!]:*
assumes *"dip$\in$kD(rt)"*
and *"the (rt dip) $\sqsubseteq$ r"*
shows *"rt $\sqsubseteq_{dip}$ rt(dip $\mapsto$ r)"*
⟨*proof*⟩

lemma *rt_fresher_map_update_other [intro!]:*
assumes *"dip$\in$kD(rt)"*
and *"dip $\neq$ ip"*
shows *"rt $\sqsubseteq_{dip}$ rt(ip $\mapsto$ r)"*
⟨*proof*⟩

lemma *rt_fresher_update_other [simp]:*
assumes *inkD: "dip$\in$kD(rt)"*
and *"dip $\neq$ ip"*
shows *"rt $\sqsubseteq_{dip}$ update rt ip r"*
⟨*proof*⟩

theorem *rt_fresher_update [simp]:*
assumes *"dip$\in$kD(rt)"*
and *"the (dhops rt dip) $\geq$ 1"*
and *"update_arg_wf r"*
shows *"rt $\sqsubseteq_{dip}$ update rt ip r"*
⟨*proof*⟩

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

lemma *$nsqn_r$_invalidate [simp]:*
assumes *"dip$\in$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]

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 =
(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_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 *"π_5(the (rt2 i)) < π_5(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

1.6 Invariant proofs on individual processes

theory *A_Seq_Invariants*
imports *AWN.Invariants A_Aodv A_Aodv_Data A_Aodv_Predicates A_Fresher*

begin

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

Proposition 7.2

lemma *sequence_number_increases:*
"paodv i ⊨$_A$ onll Γ$_{AODV}$ (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
⟨*proof*⟩

lemma *sequence_number_one_or_bigger:*
"paodv i ⊨ onl Γ$_{AODV}$ (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 Γ$_{AODV}$ (λ(ξ, l). l ∈ ({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 Γ_{AODV} *(λ(ξ, l). (l = 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 Γ_{AODV} *(λ(ξ, l). (l ∈ {PRreq-:16..PRreq-:18} ∪ {PRrep-:2..PRrep-:6} ⟶ dip ξ ∈ kD (rt ξ))*
∧ (l ∈ {PRreq-:3..PRreq-:17} ⟶ oip ξ ∈ kD (rt ξ)))"
⟨*proof*⟩

Proposition 7.38

lemma *includes_nhip:*
"paodv i ⊨ onl Γ_{AODV} *(λ(ξ, l). ∀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 Γ_{AODV} *(λ(ξ, l). (l ∈ {PRreq-:16..PRreq-:18} ⟶ dip ξ ∈ kD (rt ξ)) ∧*
(l = PRreq-:17 ⟶ oip ξ ∈ kD (rt ξ)) ∧
(l = PRrep-:5 ⟶ dip ξ ∈ kD (rt ξ)) ∧
(l = PRrep-:6 ⟶ (the (nhop (rt ξ) (dip ξ))) ∈ kD (rt ξ)))"
(is *"_ ⊨ onl* Γ_{AODV} *?P")*
⟨*proof*⟩

Proposition 7.4

lemma *known_destinations_increase:*
"paodv i ⊨$_A$ *onll* Γ_{AODV} *(λ((ξ, _), _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ'))"*
⟨*proof*⟩

Proposition 7.5

lemma *rreqs_increase:*
"paodv i ⊨$_A$ *onll* Γ_{AODV} *(λ((ξ, _), _, (ξ', _)). rreqs ξ ⊆ rreqs ξ')"*
⟨*proof*⟩

lemma *dests_bigger_than_sqn:*
"paodv i ⊨ onl Γ_{AODV} *(λ(ξ, l). l ∈ {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* Γ_{AODV} *(λ((ξ, _), _, (ξ', _)). ∀ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip)"*
⟨*proof*⟩

Proposition 7.7

lemma *ip_constant:*
"paodv i ⊨ onl Γ_{AODV} *(λ(ξ, _). ip ξ = i)"*
⟨*proof*⟩

Proposition 7.8

lemma *sender_ip_valid':*
"paodv i ⊨$_A$ *onll* Γ_{AODV} *(λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = ip ξ) a)"*
⟨*proof*⟩

lemma *sender_ip_valid:*

```
"paodv i ⊨_A onll Γ_AODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = i) a)"
```
⟨*proof*⟩

lemma *received_msg_inv:*
```
"paodv i ⊨ (recvmsg P →) onl Γ_AODV (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 Γ_AODV (λ(ξ, _). sip ξ ≠ ip ξ)"
```
⟨*proof*⟩

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

lemma *rreq_dip_in_vD_dip_eq_ip:*
```
"paodv i ⊨ onl Γ_AODV (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 Γ_AODV (λ(_, a, _). anycast msg_zhops a)"
```
⟨*proof*⟩

lemma *hop_count_zero_oip_dip_sip:*
```
"paodv i ⊨ (recvmsg msg_zhops →) onl Γ_AODV (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 Γ_AODV (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 Γ_AODV (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 Γ_AODV (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 Γ_AODV (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 Γ_AODV (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*⟩

Proposition 7.12

lemma *zero_seq_unk_hops_one':*
```
"paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (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 Γ_AODV (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 Γ_AODV (λ(ξ, l).
                          ∀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 Γ_AODV (λ(_, a, _). anycast rreq_rrep_sn a)"
```
⟨*proof*⟩

Proposition 7.14

lemma *rreq_rrep_fresh_any_step_invariant:*
```
"paodv i ⊨_A onll Γ_AODV (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a)"
```
⟨*proof*⟩

Proposition 7.15

lemma *rerr_invalid_any_step_invariant:*
```
"paodv i ⊨_A onll Γ_AODV (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 ⇀ 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 Γ_AODV (λ(ξ, l). (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 $\models_A$ *(recvmsg rreq_rrep_sn →) onll* Γ_{AODV} *(λ((ξ, _), _, (ξ', _)).*
∀dip∈kD(rt ξ). rt ξ $\sqsubseteq_{dip}$ *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* $\sqsubseteq_{dip}$ *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* $\sqsubseteq_{dip}$ *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* $\sqsubseteq_{dip}$ *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* $\sqsubseteq_{ip}$ *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)* $\sqsubset_{dip}$ *rt (σ nhip)"*
and *qinc: "quality_increases (σ nhip) (σ' nhip)"*
and *"dip∈kD(rt (σ nhip))"*
shows *"rt (σ i)* $\sqsubset_{dip}$ *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 **: "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 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))))"
  "⋀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 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 *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

1.8 The 'open' AODV model

theory *A_OAodv*
imports *A_Aodv AWN.OAWN_SOS_Labels AWN.OAWN_Convert*
begin

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

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

abbreviation *opaodv*
:: "ip ⇒ ((ip ⇒ state) × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
"opaodv i ≡ (| init = σ_{AODV}*', trans = oseqp_sos* Γ_{AODV} *i |)"*

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

lemma *oaodv_control_within [simp]: "control_within* Γ_{AODV} *(init (opaodv i))"*
⟨*proof*⟩

lemma σ_{AODV}*'_labels [simp]: "(σ, p) ∈* σ_{AODV}*' ⟹ labels* Γ_{AODV} *p = {PAodv-:0}"*
⟨*proof*⟩

lemma *oaodv_init_kD_empty [simp]:*
"(σ, p) ∈ σ_{AODV}*' ⟹ kD (rt (σ i)) = {}"*
⟨*proof*⟩

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

lemma *oaodv_trans: "trans (opaodv i) = oseqp_sos* Γ_{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: "⋀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 Γ_{AODV} *(λ(σ, l). l ∈ {PAodv-:1} ⟶ P σ (msg (σ i)))"*
⟨*proof*⟩

(Equivalent to) Proposition 7.27

lemma *local_quality_increases:*
"paodv i $\Vdash_A$ *(recvmsg rreq_rrep_sn →) onll* Γ_{AODV} *(λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ')"*
⟨*proof*⟩

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

lemma *oquality_increases:*
"opaodv i $\models_A$ *(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 *"_* $\models_A$ *(?S, _ →) _")*
⟨*proof*⟩

lemma *rreq_rrep_nsqn_fresh_any_step_invariant:*
"opaodv i $\models_A$ *(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 ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} →)
onl Γ_{AODV} *(λ(σ, l). l ∈ {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} →)
onll Γ_{AODV} *(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 Γ_AODV (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 Γ_AODV (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 *"_ ⊨ (?S, ?U →) _")*
⟨*proof*⟩

lemma *odsn_rrep:*
```
"opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
             other quality_increases {i} →)
    onl Γ_AODV (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 Γ_AODV (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 Γ_AODV (λ(σ, l).
              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 Γ_AODV (λ(ξ, l). (l = PRerr-:1
                ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ)
                                  ∧ the (nhop (rt ξ) ip) = sip ξ
                                  ∧ sqn (rt ξ) ip < the (dests ξ ip))))"
```
⟨*proof*⟩

lemmas *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 Γ_AODV (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 Γ_AODV (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 = (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]:*
"⋀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

1.11 Lift and transfer invariants to show loop freedom

theory *A_Aodv_Loop_Freedom*
imports *AWN.OClosed_Transfer AWN.Qmsg_Lifting A_Global_Invariants A_Loop_Freedom*
begin

lift to parallel processes with queues

lemma *par_step_no_change_on_send_or_receive:*
fixes *σ s a σ' s'*
assumes "*((σ, s), a, (σ', s')) ∈ oparp_sos i (oseqp_sos 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)"
⟨*proof*⟩

1.11.1 Lift to nodes

lemma *node_step_no_change_on_send_or_receive:*
assumes "*((σ, NodeS i P R), a, (σ', NodeS i' P' R')) ∈ onode_sos*
(oparp_sos i (oseqp_sos Γ$_{AODV}$ i) (seqp_sos Γ$_{QMSG}$))"
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*⟩

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 nhip = the (nhop (rt (σ i)) dip)
in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
⟶ (rt (σ i)) ⊏$_{dip}$ (rt (σ nhip)))"
(**is** *"_ ⊨ netglobal (λσ. ∀ i. ?inv σ i)"*)
⟨*proof*⟩

1.11.5 Loop freedom of AODV

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

definition *proj3 :: "r ⇒ k"* (‹π_3›)
where "$\pi_3 \equiv \lambda(\_, dsk, \_, \_, \_, \_).\ dsk$"

definition *proj4 :: "r ⇒ f"* (‹π_4›)
where "$\pi_4 \equiv \lambda(\_, \_, flag, \_, \_, \_).\ flag$"

definition *proj5 :: "r ⇒ nat"* (‹π_5›)
where "$\pi_5 \equiv \lambda(\_, \_, \_, hops, \_, \_).\ hops$"

definition *proj6 :: "r ⇒ ip"* (‹π_6›)
where "$\pi_6 \equiv \lambda(\_, \_, \_, \_, nhip, \_).\ nhip$"

definition *proj7 :: "r ⇒ ip set"* (‹π_7›)
where "$\pi_7 \equiv \lambda(\_, \_, \_, \_, \_, pre).\ pre$"

lemma *projs [simp]:*
"π_2(dsn, dsk, flag, hops, nhip, pre) = dsn"
"π_3(dsn, dsk, flag, hops, nhip, pre) = dsk"
"π_4(dsn, dsk, flag, hops, nhip, pre) = flag"
"π_5(dsn, dsk, flag, hops, nhip, pre) = hops"
"π_6(dsn, dsk, flag, hops, nhip, pre) = nhip"
"π_7(dsn, dsk, flag, hops, nhip, pre) = pre"
⟨*proof*⟩

lemma *proj3_pred [intro]:* "⟦ P kno; P unk ⟧ ⟹ P (π_3 x)"
⟨*proof*⟩

lemma *proj4_pred [intro]:* "⟦ P val; P inv ⟧ ⟹ P (π_4 x)"
⟨*proof*⟩

lemma *proj6_pair_snd [simp]:*
fixes *dsn' r*
shows "π_6 (dsn', snd (r)) = π_6(r)"
⟨*proof*⟩

2.1.3 Routing Tables

Routing tables map ip addresses to route entries.

type_synonym *rt = "ip ⇀ r"*

syntax
"_Sigma_route" :: "rt ⇒ ip ⇀ r" (‹σ_{route}'(_, _')›)

translations
"σ_{route}(rt, dip)" => "rt dip"

definition *sqn :: "rt ⇒ ip ⇒ sqn"*
where "sqn rt dip ≡ case σ_{route}(rt, dip) of Some r ⇒ π_2(r) | None ⇒ 0"

definition *sqnf :: "rt ⇒ ip ⇒ k"*
where "sqnf rt dip ≡ case σ_{route}(rt, dip) of Some r ⇒ π_3(r) | None ⇒ unk"

abbreviation *flag :: "rt ⇒ ip ⇀ f"*
where "flag rt dip ≡ map_option π_4 (σ_{route}(rt, dip))"

abbreviation *dhops :: "rt ⇒ ip ⇀ nat"*
where "dhops rt dip ≡ map_option π_5 (σ_{route}(rt, dip))"

abbreviation *nhop :: "rt ⇒ ip ⇀ ip"*
where "nhop rt dip ≡ map_option π_6 (σ_{route}(rt, dip))"

abbreviation *precs :: "rt ⇒ ip ⇀ ip set"*

where *"precs rt dip ≡ map_option π₇ (σ_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*⟩

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 "π_2(addpre v pre) = π_2(v)"
⟨*proof*⟩

lemma *proj3_addpre:*
fixes *v pre*
shows "π_3(addpre v pre) = π_3(v)"
⟨*proof*⟩

lemma *proj4_addpre:*
fixes *v pre*
shows "π_4(addpre v pre) = π_4(v)"
⟨*proof*⟩

lemma *proj5_addpre:*
fixes *v pre*
shows "π_5(addpre v pre) = π_5(v)"
⟨*proof*⟩

lemma *proj6_addpre:*
fixes *dsn dsk flag hops nhip pre npre*
shows "π_6(addpre v npre) = π_6(v)"
⟨*proof*⟩

lemma *proj7_addpre:*
fixes *dsn dsk flag hops nhip pre npre*
shows "π_7(addpre v npre) = π_7(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)) (σ_{route}(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 "π_2(the (the (addpreRT rt ip' npre) ip)) = π_2(the (rt ip))"
⟨*proof*⟩

lemma *proj3_addpreRT [simp]:*
fixes *ip rt ip' npre*
assumes "ip∈kD rt"
and "ip'∈kD rt"
shows "π_3(the (the (addpreRT rt ip' npre) ip)) = π_3(the (rt ip))"
⟨*proof*⟩

lemma *proj5_addpreRT [simp]:*
"⋀rt dip ip npre. dip∈kD(rt) ⟹ π_5(the (the (addpreRT rt dip npre) ip)) = π_5(the (rt ip))"
⟨*proof*⟩

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

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

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

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

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

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

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

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

Updating route entries

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

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

lemma *rt_Some_sqn [dest]:*
fixes *rt* **and** *ip dsn dsk flag hops nhip pre*
assumes *"rt ip = Some (dsn, dsk, flag, hops, nhip, pre)"*

shows *"sqn rt ip = dsn"*
⟨*proof*⟩

lemma *not_kD_sqn [simp]:*
fixes *dip rt*
assumes *"dip ∉ kD(rt)"*
shows *"sqn rt dip = 0"*
⟨*proof*⟩

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

lemma *update_arg_wf_gives_cases:*
"⋀r. update_arg_wf r ⟹ (π₂(r) = 0) = (π₃(r) = unk)"
⟨*proof*⟩

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

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

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

definition *update :: "rt ⇒ ip ⇒ r ⇒ rt"*
where
"update rt ip r ≡
case σ_route(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 σ_route(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 ⟧

$\Longrightarrow$ update rt ip r = rt (ip $\mapsto$ ns)"
⟨proof⟩

lemma update_cases [elim]:
assumes "(π_2(r) = 0) = (π_3(r) = unk)"
and c1: "⟦ip $\notin$ kD(rt)⟧ $\Longrightarrow$ P (rt (ip $\mapsto$ r))"

and c2: "⟦ip $\in$ kD(rt); sqn rt ip < π_2(r)⟧
$\Longrightarrow$ P (rt (ip $\mapsto$ addpre r (π_7(the σ_{route}(rt, ip)))))"
and c3: "⟦ip $\in$ kD(rt); sqn rt ip = π_2(r); the (dhops rt ip) > π_5(r)⟧
$\Longrightarrow$ P (rt (ip $\mapsto$ addpre r (π_7(the σ_{route}(rt, ip)))))"
and c4: "⟦ip $\in$ kD(rt); sqn rt ip = π_2(r); the (flag rt ip) = inv⟧
$\Longrightarrow$ P (rt (ip $\mapsto$ addpre r (π_7(the σ_{route}(rt, ip)))))"
and c5: "⟦ip $\in$ kD(rt); π_3(r) = unk⟧
$\Longrightarrow$ P (rt (ip $\mapsto$ (π_2(the σ_{route}(rt, ip)), π_3(r),
π_4(r), π_5(r), π_6(r), π_7(addpre r (π_7(the σ_{route}(rt, ip))))))))"
and c6: "⟦ip $\in$ kD(rt); sqn rt ip $\geq$ π_2(r); π_3(r) = kno;
sqn rt ip = π_2(r) $\Longrightarrow$ the (dhops rt ip) $\leq$ π_5(r) $\wedge$ the (flag rt ip) = val⟧
$\Longrightarrow$ P (rt (ip $\mapsto$ addpre (the σ_{route}(rt, ip)) (π_7(r))))"
shows "(P (update rt ip r))"
⟨proof⟩

lemma update_cases_kD:
assumes "(π_2(r) = 0) = (π_3(r) = unk)"
and "ip $\in$ kD(rt)"
and c2: "sqn rt ip < π_2(r) $\Longrightarrow$ P (rt (ip $\mapsto$ addpre r (π_7(the σ_{route}(rt, ip)))))"
and c3: "⟦sqn rt ip = π_2(r); the (dhops rt ip) > π_5(r)⟧
$\Longrightarrow$ P (rt (ip $\mapsto$ addpre r (π_7(the σ_{route}(rt, ip)))))"
and c4: "⟦sqn rt ip = π_2(r); the (flag rt ip) = inv⟧
$\Longrightarrow$ P (rt (ip $\mapsto$ addpre r (π_7(the σ_{route}(rt, ip)))))"
and c5: "π_3(r) = unk $\Longrightarrow$ P (rt (ip $\mapsto$ (π_2(the σ_{route}(rt, ip)), π_3(r),
π_4(r), π_5(r), π_6(r),
π_7(addpre r (π_7(the σ_{route}(rt, ip))))))))"
and c6: "⟦sqn rt ip $\geq$ π_2(r); π_3(r) = kno;
sqn rt ip = π_2(r) $\Longrightarrow$ the (dhops rt ip) $\leq$ π_5(r) $\wedge$ the (flag rt ip) = val⟧
$\Longrightarrow$ P (rt (ip $\mapsto$ addpre (the σ_{route}(rt, ip)) (π_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 (λx. if x = rip then Some v else rt x) ip
= (if ip = rip then π_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]: "⋀rt dip dsn dsk flg hops sip.*
rt ≠ update rt dip (dsn, dsk, flg, hops, sip, {})
⟹ sqnf (update rt dip (dsn, dsk, flg, hops, sip, {})) dip = dsk"
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

lemma *vD_update_val [dest]:*
"⋀dip 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]:*
```
"⋀dip. π3(the ((invalidate rt dests) dip)) = π3(the (rt dip))"
```
⟨*proof*⟩

lemma *proj5_invalidate [simp]:*
```
"⋀dip. π5(the ((invalidate rt dests) dip)) = π5(the (rt dip))"
```
⟨*proof*⟩

lemma *proj6_invalidate [simp]:*
```
"⋀dip. π6(the ((invalidate rt dests) dip)) = π6(the (rt dip))"
```
⟨*proof*⟩

lemma *proj7_invalidate [simp]:*
```
"⋀dip. π7(the ((invalidate rt dests) dip)) = π7(the (rt dip))"
```
⟨*proof*⟩

lemma *invalidate_kD_inv [simp]:*
```
"⋀rt dests. kD (invalidate rt dests) = kD rt"
```
⟨*proof*⟩

lemma *invalidate_sqn:*
fixes *rt dip dests*
assumes *"∀rsn. dests dip = Some rsn ⟶ sqn rt dip ≤ rsn"*
shows *"sqn rt dip ≤ sqn (invalidate rt dests) dip"*
⟨*proof*⟩

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

lemma *dhops_invalidate [simp]:*
```
"⋀dip. the (dhops (invalidate rt dests) dip) = the (dhops rt dip)"
```
⟨*proof*⟩

lemma *sqnf_invalidate [simp]:*
```
"⋀dip. sqnf (invalidate (rt ξ) (dests ξ)) dip = sqnf (rt ξ) dip"
```
⟨*proof*⟩

lemma *nhop_invalidate [simp]:*
```
"⋀dip. the (nhop (invalidate (rt ξ) (dests ξ)) dip) = the (nhop (rt ξ) dip)"
```
⟨*proof*⟩

lemma *invalidate_other [simp]:*
fixes *rt dests dip*
assumes *"dip∉dom(dests)"*
shows *"invalidate rt dests dip = rt dip"*
⟨*proof*⟩

lemma *invalidate_none [simp]:*
fixes *rt dests dip*
assumes *"dip∉kD(rt)"*
shows *"invalidate rt dests dip = None"*
⟨*proof*⟩

lemma *vD_invalidate_vD_not_dests:*
```
"⋀dip rt dests. dip∈vD(invalidate rt dests) ⟹ dip∈vD(rt) ∧ dests dip = None"
```

⟨*proof*⟩

lemma *sqn_invalidate_not_in_dests [simp]:*
fixes *dests dip rt*
assumes *"dip∉dom(dests)"*
shows *"sqn (invalidate rt dests) dip = sqn rt dip"*
⟨*proof*⟩

lemma *invalidate_changes:*
fixes *rt dests dip dsn dsk flag hops nhip pre*
assumes *"invalidate rt dests dip = Some (dsn, dsk, flag, hops, nhip, pre)"*
shows *" dsn = (case dests dip of None ⇒ 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"* (‹σ_{queue}'(_, _')›)
where *"σ_{queue}(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"* (‹$\sigma_{p\text{-}flag}$'(_, _')›)

where "$\sigma_{p\text{-}flag}$(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)"

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* ⇒ *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

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 ⇀ 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

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 ξ ≡ case msg ξ of*
Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip' ⇒
{ ξ(| hops := hops', rreqid := rreqid', dip := dip', dsn := dsn',
dsk := dsk', oip := oip', osn := osn', sip := sip' |) }
| _ ⇒ {}"

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

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

lemma *is_rrep_asm [dest!]:*
assumes *"ξ' ∈ is_rrep ξ"*
shows *"(∃hops' dip' dsn' oip' sip'.*
msg ξ = Rrep hops' dip' dsn' oip' sip' ∧
ξ' = ξ(| hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' |))"
⟨*proof*⟩

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

lemma *is_rerr_asm [dest!]:*
assumes *"ξ' ∈ is_rerr ξ"*
shows *"(∃dests' sip'.*
msg ξ = Rerr dests' sip' ∧
ξ' = ξ(| dests := dests', sip := sip' |))"
⟨*proof*⟩

lemmas *is_msg_defs =*
is_rerr_def is_rrep_def is_rreq_def is_pkt_def is_newpkt_def

lemma *is_msg_inv_ip [simp]:*
"ξ' ∈ is_rerr ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_rrep ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_rreq ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_pkt ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_newpkt ξ ⟹ ip ξ' = ip ξ"
⟨*proof*⟩

lemma *is_msg_inv_sn [simp]:*
"ξ' ∈ is_rerr ξ ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_rrep ξ ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_rreq ξ ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_pkt ξ ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_newpkt ξ ⟹ sn ξ' = sn ξ"
⟨*proof*⟩

lemma *is_msg_inv_rt [simp]:*
"ξ' ∈ is_rerr ξ ⟹ rt ξ' = rt ξ"
"ξ' ∈ is_rrep ξ ⟹ rt ξ' = rt ξ"
"ξ' ∈ is_rreq ξ ⟹ rt ξ' = rt ξ"
"ξ' ∈ is_pkt ξ ⟹ rt ξ' = rt ξ"
"ξ' ∈ is_newpkt ξ ⟹ rt ξ' = rt ξ"

⟨*proof*⟩

lemma *is_msg_inv_rreqs [simp]:*
```
"ξ' ∈ is_rerr ξ   ⟹ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_rrep ξ   ⟹ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_rreq ξ   ⟹ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_pkt ξ    ⟹ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_newpkt ξ ⟹ rreqs ξ' = rreqs ξ"
```
⟨*proof*⟩

lemma *is_msg_inv_store [simp]:*
```
"ξ' ∈ is_rerr ξ   ⟹ store ξ' = store ξ"
"ξ' ∈ is_rrep ξ   ⟹ store ξ' = store ξ"
"ξ' ∈ is_rreq ξ   ⟹ store ξ' = store ξ"
"ξ' ∈ is_pkt ξ    ⟹ store ξ' = store ξ"
"ξ' ∈ is_newpkt ξ ⟹ store ξ' = store ξ"
```
⟨*proof*⟩

lemma *is_msg_inv_sip [simp]:*
```
"ξ' ∈ is_pkt ξ    ⟹ sip ξ' = sip ξ"
"ξ' ∈ is_newpkt ξ ⟹ sip ξ' = sip ξ"
```
⟨*proof*⟩

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]: "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 ξ) (| data := data, dip := dip, oip := oip |)⟧
     call(PPkt)"
```
abbreviation *NEWPKT*
where
```
  "NEWPKT args ≡
     ⟦ξ. let (data, dip) = args ξ in
```

```
        (clear_locals ξ) (| data := data, dip := dip |)⟧
    call(PNewPkt)"

abbreviation RREQ
where
  "RREQ args ≡
     ⟦ξ. let (hops, rreqid, dip, dsn, dsk, oip, osn, sip) = args ξ in
        (clear_locals ξ) (| hops := hops, rreqid := rreqid, dip := dip,
                            dsn := dsn, dsk := dsk, oip := oip,
                            osn := osn, sip := sip |)⟧
     call(PRreq)"

abbreviation RREP
where
  "RREP args ≡
     ⟦ξ. let (hops, dip, dsn, oip, sip) = args ξ in
        (clear_locals ξ) (| hops := hops, dip := dip, dsn := dsn,
                            oip := oip, sip := sip |)⟧
     call(PRrep)"

abbreviation RERR
where
  "RERR args ≡
     ⟦ξ. let (dests, sip) = args ξ in
        (clear_locals ξ) (| dests := dests, sip := sip |)⟧
     call(PRerr)"

fun Γ_AODV :: "(state, msg, pseqp, pseqp label) seqp_env"
where
  "Γ_AODV PAodv = labelled PAodv (
     receive(λmsg' ξ. ξ (| msg := msg' |)).
     (    ⟨is_newpkt⟩ NEWPKT(λξ. (data ξ, ip ξ))
       ⊕ ⟨is_pkt⟩ PKT(λξ. (data ξ, dip ξ, oip ξ))
       ⊕ ⟨is_rreq⟩
            ⟦ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) |)⟧
            RREQ(λξ. (hops ξ, rreqid ξ, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, sip ξ))
       ⊕ ⟨is_rrep⟩
            ⟦ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) |)⟧
            RREP(λξ. (hops ξ, dip ξ, dsn ξ, oip ξ, sip ξ))
       ⊕ ⟨is_rerr⟩
            ⟦ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) |)⟧
            RERR(λξ. (dests ξ, sip ξ))
     )
     ⊕ ⟨λξ. { ξ(| dip := dip |) | dip. dip ∈ qD(store ξ) ∩ vD(rt ξ) }⟩
          ⟦ξ. ξ (| data := hd(σ_queue(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 (σ_p-flag(store ξ, dip)) = req }⟩
          ⟦ξ. ξ (| store := unsetRRF (store ξ) (dip ξ) |)⟧
          ⟦ξ. ξ (| sn := inc (sn ξ) |)⟧
          ⟦ξ. ξ (| rreqid := nrreqid (rreqs ξ) (ip ξ) |)⟧
          ⟦ξ. ξ (| rreqs := rreqs ξ ∪ {(ip ξ, rreqid ξ)} |)⟧
          broadcast(λξ. rreq(0, rreqid ξ, dip ξ, sqn (rt ξ) (dip ξ), sqnf (rt ξ) (dip ξ),
                             ip ξ, sn ξ, ip ξ)). AODV())"
```

```
|  "Γ_AODV 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())"

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

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

```
            ⟦ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)⟧
            ⟦ξ. ξ (| store := setRRF (store ξ) (dests ξ)|)⟧
            ⟦ξ. ξ (| pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } |)⟧
            ⟦ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                then (dests ξ) rip else None) |)⟧
            groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
          ⊕ ⟨ξ. dip ξ ∉ vD (rt ξ) ∨ sqn (rt ξ) (dip ξ) < dsn ξ ∨ sqnf (rt ξ) (dip ξ) = unk⟩
            broadcast(λξ. rreq(hops ξ + 1, rreqid ξ, dip ξ, max (sqn (rt ξ) (dip ξ)) (dsn ξ),
                               dsk ξ, oip ξ, osn ξ, ip ξ)).
            AODV()
        )
      ))"

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

| "Γ_AODV PRerr = labelled PRerr (
    ⟦ξ. ξ (| dests := (λrip. case (dests ξ) rip of None ⇒ None
                      | Some rsn ⇒ if rip ∈ vD (rt ξ) ∧ the (nhop (rt ξ) rip) = sip ξ
                                   ∧ sqn (rt ξ) rip < rsn then Some rsn else None) |)⟧
    ⟦ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)⟧
    ⟦ξ. ξ (| store := setRRF (store ξ) (dests ξ)|)⟧
    ⟦ξ. ξ (| pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } |)⟧
    ⟦ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                           then (dests ξ) rip else None) |)⟧
    groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)). AODV())"

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

fun Γ_AODV_skeleton
where
    "Γ_AODV_skeleton PAodv   = seqp_skeleton (Γ_AODV PAodv)"
  | "Γ_AODV_skeleton PNewPkt = seqp_skeleton (Γ_AODV PNewPkt)"
  | "Γ_AODV_skeleton PPkt    = seqp_skeleton (Γ_AODV PPkt)"
  | "Γ_AODV_skeleton PRreq   = seqp_skeleton (Γ_AODV PRreq)"
  | "Γ_AODV_skeleton PRrep   = seqp_skeleton (Γ_AODV PRrep)"
```

| "Γ_{AODV}_skeleton PRerr = seqp_skeleton (Γ_{AODV} PRerr)"

lemma Γ_{AODV}_skeleton_wf [simp]:
"wellformed Γ_{AODV}_skeleton"
⟨*proof*⟩

declare Γ_{AODV}_skeleton.simps [simp del, code del]
lemmas Γ_{AODV}_skeleton_simps [simp, code]
= Γ_{AODV}_skeleton.simps [simplified Γ_{AODV}_simps seqp_skeleton.simps]

lemma aodv_proc_cases [dest]:
fixes p pn
shows "p ∈ ctermsl (Γ_{AODV} pn) ⟹
(p ∈ ctermsl (Γ_{AODV} PAodv) ∨
p ∈ ctermsl (Γ_{AODV} PNewPkt) ∨
p ∈ ctermsl (Γ_{AODV} PPkt) ∨
p ∈ ctermsl (Γ_{AODV} PRreq) ∨
p ∈ ctermsl (Γ_{AODV} PRrep) ∨
p ∈ ctermsl (Γ_{AODV} PRerr))"
⟨*proof*⟩

definition σ_{AODV} :: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp) set"
where "σ_{AODV} i ≡ {(aodv_init i, Γ_{AODV} PAodv)}"

abbreviation paodv
:: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
"paodv i ≡ (| init = σ_{AODV} i, trans = seqp_sos Γ_{AODV} |)"

lemma aodv_trans: "trans (paodv i) = seqp_sos Γ_{AODV}"
⟨*proof*⟩

lemma aodv_control_within [simp]: "control_within Γ_{AODV} (init (paodv i))"
⟨*proof*⟩

lemma aodv_wf [simp]:
"wellformed Γ_{AODV}"
⟨*proof*⟩

lemmas aodv_labels_not_empty [simp] = labels_not_empty [OF aodv_wf]

lemma aodv_ex_label [intro]: "∃l. l∈labels Γ_{AODV} p"
⟨*proof*⟩

lemma aodv_ex_labelE [elim]:
assumes "∀l∈labels Γ_{AODV} p. P l p"
and "∃p l. P l p ⟹ Q"
shows "Q"
⟨*proof*⟩

lemma aodv_simple_labels [simp]: "simple_labels Γ_{AODV}"
⟨*proof*⟩

lemma σ_{AODV}_labels [simp]: "(ξ, p) ∈ σ_{AODV} i ⟹ labels Γ_{AODV} p = {PAodv-:0}"
⟨*proof*⟩

lemma aodv_init_kD_empty [simp]:
"(ξ, p) ∈ σ_{AODV} 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) ∈ σ_{AODV} i"

shows *"sip ξ ≠ ip ξ"*
⟨*proof*⟩

lemma *aodv_init_sip_not_i [simp]:*
assumes *"(ξ, p) ∈ σ_AODV 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
Γ_AODV_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"

```
  "⋀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.
          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]:*
```
  "⋀hops rreqid dip dsn dsk oip osn sip.
          rreq_rrep_fresh crt (Rreq hops rreqid dip dsn dsk oip osn sip) =
                              (sip ≠ oip ⟶ oip∈kD(crt)
                                         ∧ (sqn crt oip > osn
                                            ∨ (sqn crt oip = osn
                                               ∧ the (dhops crt oip) ≤ hops
                                               ∧ the (flag crt oip) = val)))"
  "⋀hops dip dsn oip sip. rreq_rrep_fresh crt (Rrep hops dip dsn oip sip) =
                              (sip ≠ dip ⟶ dip∈kD(crt)
                                         ∧ sqn crt dip = dsn
                                         ∧ the (dhops crt dip) = hops
                                         ∧ the (flag crt dip) = val)"
  "⋀dests sip.            rreq_rrep_fresh crt (Rerr dests sip) = True"
  "⋀d dip.                rreq_rrep_fresh crt (Newpkt d dip)   = True"
  "⋀d dip sip.            rreq_rrep_fresh crt (Pkt d dip sip)  = True"
```
⟨*proof*⟩

definition *rerr_invalid :: "rt ⇒ msg ⇒ bool"*
where *"rerr_invalid crt m ≡ case m of Rerr destsc _ ⇒ (∀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_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
  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]:
  "⋀dsn dsk flag hops nhip pre. nsqn_r (0, dsk, flag, hops, nhip, pre) = 0"
⟨proof⟩

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

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

lemma nsqn_r_lte_dsn [simp]:
  "⋀dsn dsk flag hops nhip pre. 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:*
"⋀rt dip. nsqn rt dip = (if flag rt dip = Some val ∨ sqn rt dip = 0
then sqn rt dip else sqn rt dip - 1)"
⟨*proof*⟩

lemma *not_in_kD_nsqn [simp]:*
assumes *"dip ∉ kD(rt)"*
shows *"nsqn rt dip = 0"*
⟨*proof*⟩

lemma *kD_nsqn:*
assumes *"dip ∈ kD(rt)"*
shows *"nsqn rt dip = nsqn$_r$(the (σ$_{route}$(rt, dip)))"*
⟨*proof*⟩

lemma *nsqnr_r_flag_pred [simp, intro]:*
fixes *dsn dsk flag hops nhip pre*
assumes *"P (nsqn$_r$ (dsn, dsk, val, hops, nhip, pre))"*
and *"P (nsqn$_r$ (dsn, dsk, inv, hops, nhip, pre))"*
shows *"P (nsqn$_r$ (dsn, dsk, flag, hops, nhip, pre))"*
⟨*proof*⟩

lemma *nsqn$_r$_addpreRT_inv [simp]:*
"⋀rt dip npre dip'. dip ∈ kD(rt) ⟹
nsqn$_r$ (the (the (addpreRT rt dip npre) dip')) = nsqn$_r$ (the (rt dip'))"
⟨*proof*⟩

lemma *sqn_nsqn:*
"⋀rt dip. sqn rt dip - 1 ≤ nsqn rt dip"
⟨*proof*⟩

lemma *nsqn_sqn: "nsqn rt dip ≤ sqn rt dip"*
⟨*proof*⟩

lemma *val_nsqn_sqn [elim, simp]:*
assumes *"ip∈kD(rt)"*
and *"the (flag rt ip) = val"*
shows *"nsqn rt ip = sqn rt ip"*
⟨*proof*⟩

lemma *vD_nsqn_sqn [elim, simp]:*
assumes *"ip∈vD(rt)"*
shows *"nsqn rt ip = sqn rt ip"*
⟨*proof*⟩

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

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

lemma *nsqn_update_changed_kno_val [simp]: "⋀rt ip dsn dsk hops nhip.*
rt ≠ update rt ip (dsn, kno, val, hops, nhip, {})

$\Longrightarrow$ *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*⟩

2.5.2 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' ∧ π$_5$(r) ≥ π$_5$(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 *"π$_5$(r) ≥ π$_5$(r')"*
shows *"r ⊑ r'"*
⟨*proof*⟩

lemma *fresherI [intro]:*
assumes *"(nsqn$_r$ r < nsqn$_r$ r') ∨ (nsqn$_r$ r = nsqn$_r$ r' ∧ π$_5$(r) ≥ π$_5$(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' ∧ π$_5$(r) ≥ π$_5$(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⟩

2.5.3 Comparing routing tables

definition
rt_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
"rt_fresher ≡ λdip rt rt'. (the (σ_{route}(rt, dip))) ⊑ (the (σ_{route}(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':*
"(rt$_1$ ⊑$_i$ rt$_2$) = (nsqn$_r$ (the (rt$_1$ i)) < nsqn$_r$ (the (rt$_2$ i)) ∨
nsqn$_r$ (the (rt$_1$ i)) = nsqn$_r$ (the (rt$_2$ i)) ∧ π_5 (the (rt$_2$ i)) ≤ π_5 (the (rt$_1$ 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 *"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* $\approx_{dip}$ *rt2"*
shows *"rt1* $\sqsubseteq_{dip}$ *rt2"*
⟨*proof*⟩

lemma *rt_fresh_asD2 [dest]:*
assumes *"rt1* $\approx_{dip}$ *rt2"*
shows *"rt2* $\sqsubseteq_{dip}$ *rt1"*
⟨*proof*⟩

lemma *rt_fresh_as_sym:*
assumes *"rt1* $\approx_{dip}$ *rt2"*
shows *"rt2* $\approx_{dip}$ *rt1"*
⟨*proof*⟩

lemma *not_rt_fresh_asI1 [intro]:*
assumes *"¬ (rt1* $\sqsubseteq_{dip}$ *rt2)"*
shows *"¬ (rt1* $\approx_{dip}$ *rt2)"*
⟨*proof*⟩

lemma *not_rt_fresh_asI2 [intro]:*
assumes *"¬ (rt2* $\sqsubseteq_{dip}$ *rt1)"*
shows *"¬ (rt1* $\approx_{dip}$ *rt2)"*
⟨*proof*⟩

lemma *not_single_rt_fresher [elim]:*
assumes *"¬(the (rt1 ip)* $\sqsubseteq$ *the (rt2 ip))"*
shows *"¬(rt1* $\sqsubseteq_{ip}$ *rt2)"*
⟨*proof*⟩

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

lemma *not_rt_fresher_single [elim]:*
assumes *"¬(rt1* $\sqsubseteq_{ip}$ *rt2)"*
shows *"¬(the (rt1 ip)* $\sqsubseteq$ *the (rt2 ip))"*
⟨*proof*⟩

lemma *rt_fresh_as_nsqnr:*
assumes *"dip* $\in$ *kD(rt1)"*
and *"dip* $\in$ *kD(rt2)"*
and *"rt1* $\approx_{dip}$ *rt2"*
shows *"*$nsqn_r$ *(the (rt2 dip)) =* $nsqn_r$ *(the (rt1 dip))"*
⟨*proof*⟩

lemma *rt_fresher_mapupd [intro!]:*
assumes *"dip*$\in$*kD(rt)"*
and *"the (rt dip)* $\sqsubseteq$ *r"*
shows *"rt* $\sqsubseteq_{dip}$ *rt(dip* $\mapsto$ *r)"*
⟨*proof*⟩

lemma *rt_fresher_map_update_other [intro!]:*
assumes *"dip*$\in$*kD(rt)"*
and *"dip* $\neq$ *ip"*
shows *"rt* $\sqsubseteq_{dip}$ *rt(ip* $\mapsto$ *r)"*
⟨*proof*⟩

lemma *rt_fresher_update_other [simp]:*
assumes *inkD: "dip*$\in$*kD(rt)"*
and *"dip* $\neq$ *ip"*
shows *"rt* $\sqsubseteq_{dip}$ *update rt ip r"*
⟨*proof*⟩

theorem *rt_fresher_update [simp]:*
assumes *"dip∈kD(rt)"*
and *"the (dhops rt dip) ≥ 1"*
and *"update_arg_wf r"*
shows *"rt ⊑$_{dip}$ 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]*

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 ⊑$_i$ 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_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 *"π_5(the (rt2 i)) < π_5(the (rt1 i))"*
shows *"rt1 $\sqsubset_i$ rt2"*
⟨*proof*⟩

lemma *invalidate_rtsf_left [simp]:*
"⋀dests dip rt rt'. dests dip = None ⟹ (invalidate rt dests $\sqsubset_{dip}$ rt') = (rt $\sqsubset_{dip}$ rt')"
⟨*proof*⟩

lemma *vD_invalidate_rt_strictly_fresher [simp]:*
assumes *"dip ∈ vD(invalidate rt1 dests)"*
shows *"(invalidate rt1 dests $\sqsubset_{dip}$ rt2) = (rt1 $\sqsubset_{dip}$ rt2)"*
⟨*proof*⟩

lemma *rt_strictly_fresher_update_other [elim!]:*
"⋀dip ip rt r rt'. ⟦ dip ≠ ip; rt $\sqsubset_{dip}$ rt' ⟧ ⟹ update rt ip r $\sqsubset_{dip}$ rt'"
⟨*proof*⟩

lemma *addpreRT_strictly_fresher [simp]:*
assumes *"dip ∈ kD(rt)"*
shows *"(the (addpreRT rt dip npre) $\sqsubset_{ip}$ rt2) = (rt $\sqsubset_{ip}$ rt2)"*
⟨*proof*⟩

lemma *lt_sqn_imp_update_strictly_fresher:*
assumes *"dip ∈ 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, {}) $\sqsubset_{dip}$ 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, {}) $\sqsubset_{dip}$ 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 $\models_A$ onll Γ_{AODV} (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"
⟨*proof*⟩

lemma *sequence_number_one_or_bigger:*
"paodv i ⊨ onl Γ_{AODV} (λ(ξ, _). 1 ≤ sn ξ)"
⟨*proof*⟩

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

lemma *sequence_number_increases':*
"paodv i $\Vdash_A$ *(λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"*
⟨*proof*⟩

lemma *sequence_number_one_or_bigger':*
"paodv i $\Vdash$ *(λ(ξ, _). 1 ≤ sn ξ)"*
⟨*proof*⟩

lemma *sip_in_kD:*
"paodv i $\Vdash$ *onl* Γ_{AODV} *(λ(ξ, l). l ∈ ({PAodv-:7} ∪ {PAodv-:5} ∪ {PRrep-:0..PRrep-:4}*
∪ {PRreq-:0..PRreq-:3}) ⟶ sip ξ ∈ kD (rt ξ))"
⟨*proof*⟩

lemma *addpreRT_partly_welldefined:*
"paodv i $\Vdash$
onl Γ_{AODV} *(λ(ξ, l). (l ∈ {PRreq-:16..PRreq-:18} ∪ {PRrep-:1..PRrep-:5} ⟶ dip ξ ∈ kD (rt ξ))*
∧ (l ∈ {PRreq-:3..PRreq-:17} ⟶ oip ξ ∈ kD (rt ξ)))"
⟨*proof*⟩

Proposition 7.38

lemma *includes_nhip:*
"paodv i $\Vdash$ *onl* Γ_{AODV} *(λ(ξ, l). ∀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 $\Vdash$ *onl* Γ_{AODV} *(λ(ξ, l). (l ∈ {PRreq-:16..PRreq-:18} ⟶ dip ξ ∈ kD (rt ξ)) ∧*
(l = PRreq-:17 ⟶ oip ξ ∈ kD (rt ξ)) ∧
(l = PRrep-:4 ⟶ dip ξ ∈ kD (rt ξ)) ∧
(l = PRrep-:5 ⟶ (the (nhop (rt ξ) (dip ξ))) ∈ kD (rt ξ)))"
(is *"_* $\Vdash$ *onl* Γ_{AODV} *?P")*
⟨*proof*⟩

Proposition 7.4

lemma *known_destinations_increase:*
"paodv i $\Vdash_A$ *onll* Γ_{AODV} *(λ((ξ, _), _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ'))"*
⟨*proof*⟩

Proposition 7.5

lemma *rreqs_increase:*
"paodv i $\Vdash_A$ *onll* Γ_{AODV} *(λ((ξ, _), _, (ξ', _)). rreqs ξ ⊆ rreqs ξ')"*
⟨*proof*⟩

lemma *dests_bigger_than_sqn:*
"paodv i $\Vdash$ *onl* Γ_{AODV} *(λ(ξ, l). l ∈ {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 $\Vdash_A$ *onll* Γ_{AODV} *(λ((ξ, _), _, (ξ', _)). ∀ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip)"*
⟨*proof*⟩

Proposition 7.7

lemma *ip_constant:*

```
"paodv i ⊨ onl Γ_AODV (λ(ξ, _). ip ξ = i)"
```
⟨*proof*⟩

Proposition 7.8

lemma *sender_ip_valid':*
```
"paodv i ⊨_A onll Γ_AODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = ip ξ) a)"
```
⟨*proof*⟩

lemma *sender_ip_valid:*
```
"paodv i ⊨_A onll Γ_AODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = i) a)"
```
⟨*proof*⟩

lemma *received_msg_inv:*
```
"paodv i ⊨ (recvmsg P →) onl Γ_AODV (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 Γ_AODV (λ(ξ, _). sip ξ ≠ ip ξ)"
```
⟨*proof*⟩

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

lemma *rreq_dip_in_vD_dip_eq_ip:*
```
"paodv i ⊨ onl Γ_AODV (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 Γ_AODV (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 Γ_AODV (λ(_, a, _). anycast msg_zhops a)"
```
⟨*proof*⟩

lemma *hop_count_zero_oip_dip_sip:*
```
"paodv i ⊨ (recvmsg msg_zhops →) onl Γ_AODV (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 Γ_AODV (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 Γ_AODV (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 Γ_AODV (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 Γ_AODV (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 Γ_AODV (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*⟩

Proposition 7.12

lemma *zero_seq_unk_hops_one':*
```
"paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (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 Γ_AODV (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 Γ_AODV (λ(ξ, l).
                          ∀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 Γ_AODV (λ(_, a, _). anycast rreq_rrep_sn a)"
```
⟨*proof*⟩

Proposition 7.14

lemma *rreq_rrep_fresh_any_step_invariant:*
```
"paodv i ⊨_A onll Γ_AODV (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a)"
```
⟨*proof*⟩

Proposition 7.15

lemma *rerr_invalid_any_step_invariant:*
```
"paodv i ⊨_A onll Γ_AODV (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* ⇀ *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 Γ_AODV (λ(ξ, l). (l ∈ {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)))
         ∧ (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 Γ_AODV (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 *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 **: "ip∈kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip*
∧ (nsqn (rt (σ sip)) ip = sn
⟶ (the (dhops (rt (σ sip)) ip) ≤ hops
∨ the (flag (rt (σ sip)) ip) = inv))"
shows *"ip∈kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip*
∧ (nsqn (rt (σ' sip)) ip = sn
⟶ (the (dhops (rt (σ' sip)) ip) ≤ hops
∨ the (flag (rt (σ' sip)) ip) = inv))"
⟨*proof*⟩

lemma *rteq_quality_increases:*
assumes *"∀j. j ≠ i ⟶ quality_increases (σ j) (σ' j)"*
and *"rt (σ' i) = rt (σ i)"*
shows *"∀j. quality_increases (σ j) (σ' j)"*
⟨*proof*⟩

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

lemma *msg_fresh [simp]:*
"⋀hops rreqid dip dsn dsk oip osn sip.
msg_fresh σ (Rreq hops rreqid dip dsn dsk oip osn sip) =
(osn ≥ 1 ∧ (sip ≠ oip ⟶ oip∈kD(rt (σ sip))
∧ nsqn (rt (σ sip)) oip ≥ osn
∧ (nsqn (rt (σ sip)) oip = osn
⟶ (hops ≥ the (dhops (rt (σ sip)) oip)
∨ the (flag (rt (σ sip)) oip) = inv))))"
"⋀hops dip dsn oip sip. msg_fresh σ (Rrep hops dip dsn oip sip) =
(dsn ≥ 1 ∧ (sip ≠ dip ⟶ dip∈kD(rt (σ sip))
∧ nsqn (rt (σ sip)) dip ≥ dsn
∧ (nsqn (rt (σ sip)) dip = dsn
⟶ (hops ≥ the (dhops (rt (σ sip)) dip))
∨ the (flag (rt (σ sip)) dip) = inv)))"
"⋀dests sip. msg_fresh σ (Rerr dests sip) =
(∀ripc∈dom(dests). (ripc∈kD(rt (σ sip))
∧ the (dests ripc) - 1 ≤ nsqn (rt (σ sip)) ripc))"
"⋀d dip. msg_fresh σ (Newpkt d dip) = True"
"⋀d dip sip. msg_fresh σ (Pkt d dip sip) = True"
⟨*proof*⟩

lemma *msg_fresh_inc_sn [simp, elim]:*
"msg_fresh 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_OAodv*
imports *B_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 *"*σ_{AODV}*' ≡ {(λi. aodv_init i,* Γ_{AODV} *PAodv)}"*

abbreviation *opaodv*
:: "ip ⇒ ((ip ⇒ state) × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
"opaodv i ≡ (| init = σ_{AODV}*', trans = oseqp_sos* Γ_{AODV} *i |)"*

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

lemma *oaodv_control_within [simp]: "control_within* Γ_{AODV} *(init (opaodv i))"*
⟨*proof*⟩

lemma σ_{AODV}*'_labels [simp]: "(σ, p) ∈* σ_{AODV}*' ⟹ labels* Γ_{AODV} *p = {PAodv-:0}"*
⟨*proof*⟩

lemma *oaodv_init_kD_empty [simp]:*

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

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

lemma *oaodv_trans: "trans (opaodv i) = oseqp_sos Γ_AODV i"*
⟨*proof*⟩

declare
```
oseq_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros]
oseq_step_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros]
```

end

2.9 Global invariant proofs over sequential processes

theory *B_Global_Invariants*
imports *B_Seq_Invariants*
B_Aodv_Predicates
B_Fresher
B_Quality_Increases
AWN.OAWN_Convert
B_OAodv
begin

lemma *other_quality_increases [elim]:*
assumes *"other quality_increases I 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 Γ_AODV (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 Γ_AODV (λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ')"
```
⟨*proof*⟩

lemmas *olocal_quality_increases =*
open_seq_step_invariant [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 ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
             other quality_increases {i} →)
            onl Γ_AODV (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 Γ_AODV (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 Γ_AODV (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 Γ_AODV (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 "_ ⊨ (?S, ?U →) _")
```
⟨*proof*⟩

lemma *odsn_rrep:*
```
"opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
             other quality_increases {i} →)
    onl Γ_AODV (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 Γ_AODV (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 Γ_AODV (λ(σ, l).
              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 Γ_{AODV} (λ(ξ, l). (l = PRerr-:1
⟶ (∀ ip∈dom(dests ξ). ip∈vD(rt ξ)
∧ the (nhop (rt ξ) ip) = sip ξ
∧ sqn (rt ξ) ip < the (dests ξ ip))))"
⟨*proof*⟩

lemmas *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 Γ_{AODV} (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 *"π_3(the (rt (σ i) dip)) = unk ∨ 1 ≤ π_2(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 Γ$_{AODV}$ (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

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]:*

"⋀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

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 nhip = the (nhop (rt (σ i)) dip)
in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
⟶ (rt (σ i)) ⊏$_{dip}$ (rt (σ nhip)))"
⟨proof⟩

lemma *par_rreq_rrep_sn_quality_increases:*
"opaodv i ⟨⟨$_i$ qmsg ⊨$_A$ (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
globala (λ(σ, _, σ'). quality_increases (σ i) (σ' i))"
⟨proof⟩

lemma *par_rreq_rrep_nsqn_fresh_any_step:*
shows "opaodv i ⟨⟨$_i$ qmsg ⊨$_A$ (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ,
other (λ_ _. True) {i} →)
globala (λ(σ, a, σ'). anycast (msg_fresh σ) a)"
⟨proof⟩

lemma *par_anycast_msg_zhops:*
shows "opaodv i ⟨⟨$_i$ qmsg ⊨$_A$ (λσ _. orecvmsg (λ_. rreq_rrep_sn) σ, other (λ_ _. True) {i} →)
globala (λ(_, a, _). anycast msg_zhops a)"
⟨proof⟩

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 Γ_{AODV} i) (seqp_sos Γ_{QMSG}))"
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 : 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*⟩

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)

```
          ⊨ (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*⟩

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 (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*⟩

2.11.6 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 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, nhip)* where *dsn* is the 'destination sequence number', *dsk* is the 'destination-sequence-number status', *flag* is the route status, *hops* is the number of hops to the destination, and *nhip* is the next hop toward the destination. In this variant, the set of 'precursor nodes' is not modelled.

type_synonym *r = "sqn × k × f × nat × ip"*

definition *proj2 :: "r ⇒ sqn" (‹π_2›)*
  where *"π_2 ≡ λ(dsn, _, _, _, _). dsn"*

definition *proj3 :: "r ⇒ k" (‹π_3›)*
  where *"π_3 ≡ λ(_, dsk, _, _, _). dsk"*

definition *proj4 :: "r ⇒ f" (‹π_4›)*
  where *"π_4 ≡ λ(_, _, flag, _, _). flag"*

definition *proj5 :: "r ⇒ nat" (‹π_5›)*
  where *"π_5 ≡ λ(_, _, _, hops, _). hops"*

definition *proj6 :: "r ⇒ ip" (‹π_6›)*
  where *"π_6 ≡ λ(_, _, _, _, nhip). nhip"*

lemma *projs [simp]:*
  "π_2(dsn, dsk, flag, hops, nhip) = dsn"
  "π_3(dsn, dsk, flag, hops, nhip) = dsk"
  "π_4(dsn, dsk, flag, hops, nhip) = flag"
  "π_5(dsn, dsk, flag, hops, nhip) = hops"
  "π_6(dsn, dsk, flag, hops, nhip) = nhip"
  ⟨*proof*⟩

lemma *proj3_pred [intro]: "⟦ P kno; P unk ⟧ ⟹ P (π_3 x)"*
  ⟨*proof*⟩

lemma *proj4_pred [intro]: "⟦ P val; P inv ⟧ ⟹ P (π_4 x)"*
  ⟨*proof*⟩

lemma *proj6_pair_snd [simp]:*
  fixes *dsn' r*
  shows *"π_6 (dsn', snd (r)) = π_6(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" (‹σ_{route}'(_, _')›)

translations
 "σ_{route}(rt, dip)" => "rt dip"

definition *sqn :: "rt ⇒ ip ⇒ sqn"*
  where *"sqn rt dip ≡ case σ_{route}(rt, dip) of Some r ⇒ π_2(r) | None ⇒ 0"*

definition *sqnf :: "rt ⇒ ip ⇒ k"*
  where *"sqnf rt dip ≡ case σ_{route}(rt, dip) of Some r ⇒ π_3(r) | None ⇒ unk"*

abbreviation *flag :: "rt ⇒ ip ⇀ f"*
  where *"flag rt dip ≡ map_option π_4 (σ_{route}(rt, dip))"*

abbreviation *dhops :: "rt ⇒ ip ⇀ nat"*
   where *"dhops rt dip ≡ map_option π_5 (σ_{route}(rt, dip))"*

abbreviation *nhop :: "rt ⇒ ip ⇀ ip"*
   where *"nhop rt dip ≡ map_option π_6 (σ_{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.*
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')"

"⟦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*
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]: "⋀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*
assumes *"rt = update rt dip (dsn, dsk, flag, hops, nhip)"*
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*
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]:*
"⋀dip rt dip' dsn dsk hops nhip.
dip ∈ vD(update rt dip' (dsn, dsk, val, hops, nhip)) ⟹ (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), Some rsn) ⇒
                      Some (rsn, dsk, inv, hops, nhip)"
```

lemma *proj3_invalidate [simp]:*
```
  "⋀dip. π3(the ((invalidate rt dests) dip)) = π3(the (rt dip))"
```
⟨*proof*⟩

lemma *proj5_invalidate [simp]:*
```
  "⋀dip. π5(the ((invalidate rt dests) dip)) = π5(the (rt dip))"
```
⟨*proof*⟩

lemma *proj6_invalidate [simp]:*
```
  "⋀dip. π6(the ((invalidate rt dests) dip)) = π6(the (rt dip))"
```
⟨*proof*⟩

lemma *invalidate_kD_inv [simp]:*
```
  "⋀rt dests. kD (invalidate rt dests) = kD rt"
```
⟨*proof*⟩

lemma *invalidate_sqn:*
```
  fixes rt dip dests
  assumes "∀rsn. dests dip = Some rsn ⟶ sqn rt dip ≤ rsn"
  shows "sqn rt dip ≤ sqn (invalidate rt dests) dip"
```
⟨*proof*⟩

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

lemma *dhops_invalidate [simp]:*
```
  "⋀dip. the (dhops (invalidate rt dests) dip) = the (dhops rt dip)"
```
⟨*proof*⟩

lemma *sqnf_invalidate [simp]:*
```
  "⋀dip. sqnf (invalidate (rt ξ) (dests ξ)) dip = sqnf (rt ξ) dip"
```
⟨*proof*⟩

lemma *nhop_invalidate [simp]:*
```
  "⋀dip. the (nhop (invalidate (rt ξ) (dests ξ)) dip) = the (nhop (rt ξ) dip)"
```
⟨*proof*⟩

lemma *invalidate_other [simp]:*
```
    fixes rt dests dip
  assumes "dip∉dom(dests)"
    shows "invalidate rt dests dip = rt dip"
```
⟨*proof*⟩

lemma *invalidate_none [simp]:*
```
    fixes rt dests dip
  assumes "dip∉kD(rt)"
    shows "invalidate rt dests dip = None"
```
⟨*proof*⟩

lemma *vD_invalidate_vD_not_dests:*
```
  "⋀dip rt dests. dip∈vD(invalidate rt dests) ⟹ dip∈vD(rt) ∧ dests dip = None"
```
⟨*proof*⟩

lemma *sqn_invalidate_not_in_dests [simp]:*
fixes *dests dip rt*
assumes *"dip∉dom(dests)"*
shows *"sqn (invalidate rt dests) dip = sqn rt dip"*
⟨*proof*⟩

lemma *invalidate_changes:*
fixes *rt dests dip dsn dsk flag hops nhip*
assumes *"invalidate rt dests dip = Some (dsn, dsk, flag, hops, nhip)"*
shows *" dsn = (case dests dip of None ⇒ π₂(the (rt dip)) | Some rsn ⇒ rsn)*
∧ dsk = π₃(the (rt dip))
∧ flag = (if dests dip = None then π₄(the (rt dip)) else inv)
∧ hops = π₅(the (rt dip))
∧ nhip = π₆(the (rt dip))"
⟨*proof*⟩

lemma *proj3_inv: "⋀dip rt dests. dip∈kD (rt)*
⟹ π₃(the (invalidate rt dests dip)) = π₃(the (rt dip))"
⟨*proof*⟩

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

3.1.5 Route Requests

Generate a fresh route request identifier.

definition *nrreqid :: "(ip × rreqid) set ⇒ ip ⇒ rreqid"*
where *"nrreqid rreqs ip ≡ Max ({n. (ip, n) ∈ rreqs} ∪ {0}) + 1"*

3.1.6 Queued Packets

Functions for sending data packets.

type_synonym *store = "ip ⇀ (p × data list)"*

definition *sigma_queue :: "store ⇒ ip ⇒ data list"* (‹σ_{queue}'(_, _')›)
where *"σ_{queue}(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"* (‹$\sigma_{p\text{-}flag}$'(_, _')›)
where *"$\sigma_{p\text{-}flag}$(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 ⇒ 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 = {},
        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 *"ξ' ∈ is_rreq ξ"*
shows *"(∃hops' rreqid' dip' dsn' dsk' oip' osn' sip'.*
msg ξ = Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip' ∧
ξ' = ξ(| hops := hops', rreqid := rreqid', dip := dip', dsn := dsn',
dsk := dsk', oip := oip', osn := osn', sip := sip' |))"
⟨*proof*⟩

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

lemma *is_rrep_asm [dest!]:*
assumes *"ξ' ∈ is_rrep ξ"*
shows *"(∃hops' dip' dsn' oip' sip'.*
msg ξ = Rrep hops' dip' dsn' oip' sip' ∧
ξ' = ξ(| hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' |))"
⟨*proof*⟩

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

lemma *is_rerr_asm [dest!]:*
assumes *"ξ' ∈ is_rerr ξ"*
shows *"(∃dests' sip'.*
msg ξ = Rerr dests' sip' ∧
ξ' = ξ(| dests := dests', sip := sip' |))"
⟨*proof*⟩

lemmas *is_msg_defs =*
is_rerr_def is_rrep_def is_rreq_def is_pkt_def is_newpkt_def

lemma *is_msg_inv_ip [simp]:*
"ξ' ∈ is_rerr ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_rrep ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_rreq ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_pkt ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_newpkt ξ ⟹ ip ξ' = ip ξ"
⟨*proof*⟩

lemma *is_msg_inv_sn [simp]:*
"ξ' ∈ is_rerr ξ ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_rrep ξ ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_rreq ξ ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_pkt ξ ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_newpkt ξ ⟹ sn ξ' = sn ξ"
⟨*proof*⟩

lemma *is_msg_inv_rt [simp]:*
"ξ' ∈ is_rerr ξ ⟹ rt ξ' = rt ξ"
"ξ' ∈ is_rrep ξ ⟹ rt ξ' = rt ξ"
"ξ' ∈ is_rreq ξ ⟹ rt ξ' = rt ξ"
"ξ' ∈ is_pkt ξ ⟹ rt ξ' = rt ξ"
"ξ' ∈ is_newpkt ξ ⟹ rt ξ' = rt ξ"
⟨*proof*⟩

lemma *is_msg_inv_rreqs [simp]:*
"ξ' ∈ is_rerr ξ ⟹ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_rrep ξ ⟹ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_rreq ξ ⟹ rreqs ξ' = rreqs ξ"

```
  "ξ' ∈ is_pkt ξ    ⟹ rreqs ξ' = rreqs ξ"
  "ξ' ∈ is_newpkt ξ ⟹ rreqs ξ' = rreqs ξ"
  ⟨proof⟩

lemma is_msg_inv_store [simp]:
  "ξ' ∈ is_rerr ξ   ⟹ store ξ' = store ξ"
  "ξ' ∈ is_rrep ξ   ⟹ store ξ' = store ξ"
  "ξ' ∈ is_rreq ξ   ⟹ store ξ' = store ξ"
  "ξ' ∈ is_pkt ξ    ⟹ store ξ' = store ξ"
  "ξ' ∈ is_newpkt ξ ⟹ store ξ' = store ξ"
  ⟨proof⟩

lemma is_msg_inv_sip [simp]:
  "ξ' ∈ is_pkt ξ    ⟹ sip ξ' = sip ξ"
  "ξ' ∈ is_newpkt ξ ⟹ sip ξ' = sip ξ"
  ⟨proof⟩
```

3.3.3 The protocol process

```
datatype pseqp =
    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 ξ) (| data := data, dip := dip, oip := oip |)⟧
     call(PPkt)"
abbreviation NEWPKT
where
  "NEWPKT args ≡
     ⟦ξ. let (data, dip) = args ξ in
         (clear_locals ξ) (| data := data, dip := dip |)⟧
     call(PNewPkt)"

abbreviation RREQ
where
  "RREQ args ≡
```

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

abbreviation RREP
where
  "RREP args ≡
    ⟦ξ. let (hops, dip, dsn, oip, sip) = args ξ in
        (clear_locals ξ) (| hops := hops, dip := dip, dsn := dsn,
                            oip := oip, sip := sip |)⟧
    call(PRrep)"

abbreviation RERR
where
  "RERR args ≡
    ⟦ξ. let (dests, sip) = args ξ in
        (clear_locals ξ) (| dests := dests, sip := sip |)⟧
    call(PRerr)"

fun Γ_AODV :: "(state, msg, pseqp, pseqp label) seqp_env"
where
  "Γ_AODV PAodv = labelled PAodv (
    receive(λmsg' ξ. ξ (| msg := msg' |)).
    (    ⟨is_newpkt⟩ NEWPKT(λξ. (data ξ, ip ξ))
      ⊕ ⟨is_pkt⟩ PKT(λξ. (data ξ, dip ξ, oip ξ))
      ⊕ ⟨is_rreq⟩
          ⟦ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ) |)⟧
          RREQ(λξ. (hops ξ, rreqid ξ, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, sip ξ))
      ⊕ ⟨is_rrep⟩
          ⟦ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ) |)⟧
          RREP(λξ. (hops ξ, dip ξ, dsn ξ, oip ξ, sip ξ))
      ⊕ ⟨is_rerr⟩
          ⟦ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ) |)⟧
          RERR(λξ. (dests ξ, sip ξ))
    )
    ⊕ ⟨λξ. { ξ(| dip := dip |) | dip. dip ∈ qD(store ξ) ∩ vD(rt ξ) }⟩
        ⟦ξ. ξ (| data := hd(σ_queue(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 (σ_p-flag(store ξ, dip)) = req }⟩
        ⟦ξ. ξ (| store := unsetRRF (store ξ) (dip ξ) |)⟧
        ⟦ξ. ξ (| sn := inc (sn ξ) |)⟧
        ⟦ξ. ξ (| rreqid := nrreqid (rreqs ξ) (ip ξ) |)⟧
        ⟦ξ. ξ (| rreqs := rreqs ξ ∪ {(ip ξ, rreqid ξ)} |)⟧
        broadcast(λξ. rreq(0, rreqid ξ, dip ξ, sqn (rt ξ) (dip ξ), sqnf (rt ξ) (dip ξ), ip ξ, sn ξ,
                           ip ξ)). AODV())"

|  "Γ_AODV 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())"

| "Γ_AODV PPkt = labelled PPkt (
```

```
    ⟨ξ. dip ξ = ip ξ⟩
       deliver(λξ. data ξ).AODV()
    ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
    (
      ⟨ξ. dip ξ ∈ vD (rt ξ)⟩
       unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, oip ξ)).AODV()
       ▷
         ⟦ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
                                  then Some (inc (sqn (rt ξ) rip)) else None) |)⟧
         ⟦ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)⟧
         ⟦ξ. ξ (| store := setRRF (store ξ) (dests ξ)|)⟧
         broadcast(λξ. rerr(dests ξ, ip ξ)).AODV()
      ⊕ ⟨ξ. dip ξ ∉ vD (rt ξ)⟩
      (
          ⟨ξ. dip ξ ∈ iD (rt ξ)⟩
            broadcast(λξ. rerr([dip ξ ↦ sqn (rt ξ) (dip ξ)], ip ξ)). AODV()
          ⊕ ⟨ξ. dip ξ ∉ iD (rt ξ)⟩
             AODV()
      )
    ))"

| "Γ_AODV 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()
        )
      ))"

| "Γ_AODV 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()
      )"

| "Γ_AODV 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_{AODV}*.simps [simp del, code del]*
lemmas Γ_{AODV}*_simps [simp, code] =* Γ_{AODV}*.simps [simplified]*

fun Γ_{AODV}*_skeleton*
where

```
    "Γ_AODV_skeleton PAodv   = seqp_skeleton (Γ_AODV PAodv)"
  | "Γ_AODV_skeleton PNewPkt = seqp_skeleton (Γ_AODV PNewPkt)"
  | "Γ_AODV_skeleton PPkt    = seqp_skeleton (Γ_AODV PPkt)"
  | "Γ_AODV_skeleton PRreq   = seqp_skeleton (Γ_AODV PRreq)"
  | "Γ_AODV_skeleton PRrep   = seqp_skeleton (Γ_AODV PRrep)"
  | "Γ_AODV_skeleton PRerr   = seqp_skeleton (Γ_AODV PRerr)"
```

lemma Γ_{AODV}*_skeleton_wf [simp]:*
"wellformed Γ_{AODV}*_skeleton"*
⟨*proof*⟩

declare Γ_{AODV}*_skeleton.simps [simp del, code del]*
lemmas Γ_{AODV}*_skeleton_simps [simp, code]*
= Γ_{AODV}*_skeleton.simps [simplified* Γ_{AODV}*_simps seqp_skeleton.simps]*

lemma *aodv_proc_cases [dest]:*
fixes *p pn*
shows *"p ∈ ctermsl (*Γ_{AODV} *pn) ⟹*

```
                              (p ∈ ctermsl (Γ_AODV PAodv) ∨
                               p ∈ ctermsl (Γ_AODV PNewPkt)  ∨
                               p ∈ ctermsl (Γ_AODV PPkt)  ∨
                               p ∈ ctermsl (Γ_AODV PRreq) ∨
                               p ∈ ctermsl (Γ_AODV PRrep) ∨
                               p ∈ ctermsl (Γ_AODV PRerr))"
```

⟨*proof*⟩

definition σ_{AODV} :: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp) set"
where "σ_{AODV} i ≡ {(aodv_init i, Γ_{AODV} PAodv)}"

abbreviation paodv
:: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
"paodv i ≡ ⦇ init = σ_{AODV} i, trans = seqp_sos Γ_{AODV} ⦈"

lemma aodv_trans: "trans (paodv i) = seqp_sos Γ_{AODV}"
⟨*proof*⟩

lemma aodv_control_within [simp]: "control_within Γ_{AODV} (init (paodv i))"
⟨*proof*⟩

lemma aodv_wf [simp]:
"wellformed Γ_{AODV}"
⟨*proof*⟩

lemmas aodv_labels_not_empty [simp] = labels_not_empty [OF aodv_wf]

lemma aodv_ex_label [intro]: "∃l. l∈labels Γ_{AODV} p"
⟨*proof*⟩

lemma aodv_ex_labelE [elim]:
assumes "∀l∈labels Γ_{AODV} p. P l p"
and "∃p l. P l p ⟹ Q"
shows "Q"
⟨*proof*⟩

lemma aodv_simple_labels [simp]: "simple_labels Γ_{AODV}"
⟨*proof*⟩

lemma σ_{AODV}_labels [simp]: "(ξ, p) ∈ σ_{AODV} i ⟹ labels Γ_{AODV} p = {PAodv-:0}"
⟨*proof*⟩

lemma aodv_init_kD_empty [simp]:
"(ξ, p) ∈ σ_{AODV} 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) ∈ σ_{AODV} i"
shows "sip ξ ≠ ip ξ"
⟨*proof*⟩

lemma aodv_init_sip_not_i [simp]:
assumes "(ξ, p) ∈ σ_{AODV} 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
Γ_{AODV}_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

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_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"
  "⋀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.
          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]:*

```
"⋀hops rreqid dip dsn dsk oip osn sip.
        rreq_rrep_fresh crt (Rreq hops rreqid dip dsn dsk oip osn sip) =
                            (sip ≠ oip ⟶ oip∈kD(crt)
                                       ∧ (sqn crt oip > osn
                                          ∨ (sqn crt oip = osn
                                             ∧ the (dhops crt oip) ≤ hops
                                             ∧ the (flag crt oip) = val)))"
"⋀hops dip dsn oip sip. rreq_rrep_fresh crt (Rrep hops dip dsn oip sip) =
                            (sip ≠ dip ⟶ dip∈kD(crt)
                                       ∧ sqn crt dip = dsn
                                       ∧ the (dhops crt dip) = hops
                                       ∧ the (flag crt dip) = val)"
"⋀dests sip.            rreq_rrep_fresh crt (Rerr dests sip) = True"
"⋀d dip.                rreq_rrep_fresh crt (Newpkt d dip)   = True"
"⋀d dip sip.            rreq_rrep_fresh crt (Pkt d dip sip)  = True"
```

⟨*proof*⟩

definition *rerr_invalid :: "rt ⇒ msg ⇒ bool"*
where *"rerr_invalid crt m ≡ case m of Rerr destsc _ ⇒ (∀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_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 ⇒ sqn"
where
"$nsqn_r$ r ≡ if π_4(r) = val ∨ π_2(r) = 0 then π_2(r) else (π_2(r) - 1)"

lemma *nsqnr_def':*
"$nsqn_r$ r = (if π_4(r) = inv then π_2(r) - 1 else π_2(r))"
⟨*proof*⟩

lemma *$nsqn_r$_zero [simp]:*
"⋀dsn dsk flag hops nhip. $nsqn_r$ (0, dsk, flag, hops, nhip) = 0"
⟨*proof*⟩

lemma *$nsqn_r$_val [simp]:*
"⋀dsn dsk hops nhip. $nsqn_r$ (dsn, dsk, val, hops, nhip) = dsn"
⟨*proof*⟩

lemma *$nsqn_r$_inv [simp]:*
"⋀dsn dsk hops nhip. $nsqn_r$ (dsn, dsk, inv, hops, nhip) = dsn - 1"
⟨*proof*⟩

lemma *$nsqn_r$_lte_dsn [simp]:*
"⋀dsn dsk flag hops nhip. $nsqn_r$ (dsn, dsk, flag, hops, nhip) ≤ 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:*
"⋀rt dip. nsqn rt dip = (if flag rt dip = Some val ∨ sqn rt dip = 0
then sqn rt dip else sqn rt dip - 1)"
⟨*proof*⟩

lemma *not_in_kD_nsqn [simp]:*
assumes *"dip ∉ kD(rt)"*
shows *"nsqn rt dip = 0"*
⟨*proof*⟩

lemma *kD_nsqn:*
assumes *"dip ∈ kD(rt)"*
shows *"nsqn rt dip = $nsqn_r$(the (σ_{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' ≡ ((nsqn$_r$ r < nsqn$_r$ r') ∨ (nsqn$_r$ r = nsqn$_r$ r' ∧ π$_5$(r) ≥ π$_5$(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 "π$_5$(r) ≥ π$_5$(r')"
shows "r ⊑ r'"
⟨*proof*⟩

lemma *fresherI [intro]:*
assumes "(nsqn$_r$ r < nsqn$_r$ r') ∨ (nsqn$_r$ r = nsqn$_r$ r' ∧ π$_5$(r) ≥ π$_5$(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' ∧ π$_5$(r) ≥ π$_5$(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 ⇒ rt ⇒ rt ⇒ bool"
where
"rt_fresher ≡ λdip rt rt'. (the (σ$_{route}$(rt, dip))) ⊑ (the (σ$_{route}$(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':*
"(rt$_1$ ⊑$_i$ rt$_2$) = (nsqn$_r$ (the (rt$_1$ i)) < nsqn$_r$ (the (rt$_2$ i)) ∨
nsqn$_r$ (the (rt$_1$ i)) = nsqn$_r$ (the (rt$_2$ i)) ∧ π$_5$ (the (rt$_2$ i)) ≤ π$_5$ (the (rt$_1$ 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 $\sqsubseteq_{ip}$ rt2"*
shows *"the (rt1 ip) $\sqsubseteq$ the (rt2 ip)"*
⟨*proof*⟩

lemma *rt_fresher_def2:*
assumes *"dip $\in$ kD(rt1)"*
and *"dip $\in$ kD(rt2)"*
shows *"(rt1 $\sqsubseteq_{dip}$ rt2) = (nsqn rt1 dip < nsqn rt2 dip*
$\vee$ (nsqn rt1 dip = nsqn rt2 dip
$\wedge$ the (dhops rt1 dip) $\geq$ the (dhops rt2 dip)))"
⟨*proof*⟩

lemma *rt_fresherI1 [intro]:*
assumes *"dip $\in$ kD(rt1)"*
and *"dip $\in$ kD(rt2)"*
and *"nsqn rt1 dip < nsqn rt2 dip"*
shows *"rt1 $\sqsubseteq_{dip}$ rt2"*
⟨*proof*⟩

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

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

lemma *rt_fresher_refl [simp]: "rt $\sqsubseteq_{dip}$ rt"*
⟨*proof*⟩

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

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

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

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

lemma *rt_fresh_as_refl [simp]: "$\bigwedge$rt dip. rt $\approx_{dip}$ 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 $\sqsubseteq_{ip}$ rt2)"*
shows *"¬(the (rt1 ip) $\sqsubseteq$ the (rt2 ip))"*
⟨*proof*⟩

lemma *rt_fresh_as_nsqnr:*
assumes *"dip ∈ kD(rt1)"*
and *"dip ∈ kD(rt2)"*
and *"rt1 $\approx_{dip}$ rt2"*
shows *"nsqn$_r$ (the (rt2 dip)) = nsqn$_r$ (the (rt1 dip))"*
⟨*proof*⟩

lemma *rt_fresher_mapupd [intro!]:*
assumes *"dip∈kD(rt)"*
and *"the (rt dip) $\sqsubseteq$ r"*
shows *"rt $\sqsubseteq_{dip}$ rt(dip ↦ r)"*
⟨*proof*⟩

lemma *rt_fresher_map_update_other [intro!]:*
assumes *"dip∈kD(rt)"*
and *"dip ≠ ip"*
shows *"rt $\sqsubseteq_{dip}$ rt(ip ↦ r)"*
⟨*proof*⟩

lemma *rt_fresher_update_other [simp]:*
assumes *inkD: "dip∈kD(rt)"*
and *"dip ≠ ip"*
shows *"rt $\sqsubseteq_{dip}$ update rt ip r"*
⟨*proof*⟩

theorem *rt_fresher_update [simp]:*
assumes *"dip∈kD(rt)"*
and *"the (dhops rt dip) ≥ 1"*
and *"update_arg_wf r"*
shows *"rt $\sqsubseteq_{dip}$ 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 $\sqsubseteq_{dip}$ 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 $\approx_{dip}$ invalidate rt dests"*
⟨*proof*⟩

lemmas *rt_fresher_inc_invalidate [simp] = rt_fresh_as_inc_invalidate [THEN rt_fresh_asD1]*

3.5.4 Strictly comparing routing tables

definition *rt_strictly_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"*
where
"rt_strictly_fresher ≡ λdip rt1 rt2. (rt1 $\sqsubseteq_{dip}$ rt2) ∧ ¬(rt1 $\approx_{dip}$ 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)) ∧ 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* $\sqsubseteq_{dip}$ *rt2"*
and *"rt2* $\sqsubset_{dip}$ *rt3"*
shows *"rt1* $\sqsubset_{dip}$ *rt3"*
⟨*proof*⟩

lemma *rt_fresher_imp_nsqn_le:*
assumes *"rt1* $\sqsubseteq_{ip}$ *rt2"*
and *"ip* $\in$ *kD rt1"*
and *"ip* $\in$ *kD rt2"*
shows *"nsqn rt1 ip* $\leq$ *nsqn rt2 ip"*
⟨*proof*⟩

lemma *rt_strictly_fresher_ltI [intro]:*
assumes *"dip* $\in$ *kD(rt1)"*
and *"dip* $\in$ *kD(rt2)"*
and *"nsqn rt1 dip < nsqn rt2 dip"*
shows *"rt1* $\sqsubset_{dip}$ *rt2"*
⟨*proof*⟩

lemma *rt_strictly_fresher_eqI [intro]:*
assumes *"i*$\in$*kD(rt1)"*
and *"i*$\in$*kD(rt2)"*
and *"nsqn rt1 i = nsqn rt2 i"*
and *"*π_5*(the (rt2 i)) <* π_5*(the (rt1 i))"*
shows *"rt1* $\sqsubset_i$ *rt2"*
⟨*proof*⟩

lemma *invalidate_rtsf_left [simp]:*
"$\bigwedge$*dests dip rt rt'. dests dip = None* $\Longrightarrow$ *(invalidate rt dests* $\sqsubset_{dip}$ *rt') = (rt* $\sqsubset_{dip}$ *rt')"*
⟨*proof*⟩

lemma *vD_invalidate_rt_strictly_fresher [simp]:*
assumes *"dip* $\in$ *vD(invalidate rt1 dests)"*
shows *"(invalidate rt1 dests* $\sqsubset_{dip}$ *rt2) = (rt1* $\sqsubset_{dip}$ *rt2)"*
⟨*proof*⟩

lemma *rt_strictly_fresher_update_other [elim!]:*
"$\bigwedge$*dip ip rt r rt'.* $[\![$ *dip* $\neq$ *ip; rt* $\sqsubset_{dip}$ *rt'* $]\!]$ $\Longrightarrow$ *update rt ip r* $\sqsubset_{dip}$ *rt'"*
⟨*proof*⟩

lemma *lt_sqn_imp_update_strictly_fresher:*
assumes *"dip* $\in$ *vD (rt2 nhip)"*
and **: "osn < sqn (rt2 nhip) dip"*
and ***: "rt* $\neq$ *update rt dip (osn, kno, val, hops, nhip)"*
shows *"update rt dip (osn, kno, val, hops, nhip)* $\sqsubset_{dip}$ *rt2 nhip"*
⟨*proof*⟩

lemma *dhops_le_hops_imp_update_strictly_fresher:*
assumes *"dip* $\in$ *vD(rt2 nhip)"*
and *sqn: "sqn (rt2 nhip) dip = osn"*
and *hop: "the (dhops (rt2 nhip) dip)* $\leq$ *hops"*
and ***: "rt* $\neq$ *update rt dip (osn, kno, val, Suc hops, nhip)"*
shows *"update rt dip (osn, kno, val, Suc hops, nhip)* $\sqsubset_{dip}$ *rt2 nhip"*
⟨*proof*⟩

lemma *nsqn_invalidate:*
assumes *"dip* $\in$ *kD(rt)"*
and *"*$\forall$ *ip*$\in$*dom(dests). ip* $\in$ *vD(rt)* $\wedge$ *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 $\models_A$ *onll* Γ_{AODV} *(λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"*
⟨*proof*⟩

lemma *sequence_number_one_or_bigger:*
"paodv i $\models$ *onl* Γ_{AODV} *(λ(ξ, _). 1 ≤ sn ξ)"*
⟨*proof*⟩

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

lemma *sequence_number_increases':*
"paodv i $\models_A$ *(λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"*
⟨*proof*⟩

lemma *sequence_number_one_or_bigger':*
"paodv i $\models$ *(λ(ξ, _). 1 ≤ sn ξ)"*
⟨*proof*⟩

lemma *sip_in_kD:*
"paodv i $\models$ *onl* Γ_{AODV} *(λ(ξ, l). l ∈ ({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 $\models$ *onl* Γ_{AODV} *(λ(ξ, l). (l = 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 $\models$ *onl* Γ_{AODV} *(λ(ξ, l). ∀dip∈kD(rt ξ). the (nhop (rt ξ) dip)∈kD(rt ξ))"*
⟨*proof*⟩

Proposition 7.4

lemma *known_destinations_increase:*
"paodv i $\models_A$ *onll* Γ_{AODV} *(λ((ξ, _), _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ'))"*
⟨*proof*⟩

Proposition 7.5

lemma *rreqs_increase:*
"paodv i $\models_A$ *onll* Γ_{AODV} *(λ((ξ, _), _, (ξ', _)). rreqs ξ ⊆ rreqs ξ')"*
⟨*proof*⟩

lemma *dests_bigger_than_sqn:*
"paodv i $\models$ *onl* Γ_{AODV} *(λ(ξ, l). l ∈ {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:*

```
"paodv i ⊨_A onll Γ_AODV (λ((ξ, _), _, (ξ', _)). ∀ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip)"
```
⟨*proof*⟩

Proposition 7.7

lemma *ip_constant:*
```
"paodv i ⊨ onl Γ_AODV (λ(ξ, _). ip ξ = i)"
```
⟨*proof*⟩

Proposition 7.8

lemma *sender_ip_valid':*
```
"paodv i ⊨_A onll Γ_AODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = ip ξ) a)"
```
⟨*proof*⟩

lemma *sender_ip_valid:*
```
"paodv i ⊨_A onll Γ_AODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = i) a)"
```
⟨*proof*⟩

lemma *received_msg_inv:*
```
"paodv i ⊨ (recvmsg P →) onl Γ_AODV (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 Γ_AODV (λ(ξ, _). sip ξ ≠ ip ξ)"
```
⟨*proof*⟩

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

lemma *rreq_dip_in_vD_dip_eq_ip:*
```
"paodv i ⊨ onl Γ_AODV (λ(ξ, l). (l ∈ {PRreq-:14} ⟶ dip ξ ∈ vD(rt ξ))
                          ∧ (l ∈ {PRreq-:5, PRreq-:6} ⟶ dip ξ = ip ξ)
                          ∧ (l ∈ {PRreq-:13..PRreq-:14} ⟶ dip ξ ≠ ip ξ))"
```
⟨*proof*⟩

Proposition 7.11

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

lemma *hop_count_zero_oip_dip_sip:*
```
"paodv i ⊨ (recvmsg msg_zhops →) onl Γ_AODV (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 Γ_AODV (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 Γ_AODV (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 Γ_AODV (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 Γ_AODV (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 Γ_AODV (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*⟩

Proposition 7.12

lemma *zero_seq_unk_hops_one':*
```
"paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (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 Γ_AODV (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 Γ_AODV (λ(ξ, l).
                           ∀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 Γ_AODV (λ(_, a, _). anycast rreq_rrep_sn a)"
```
⟨*proof*⟩

Proposition 7.14

lemma *rreq_rrep_fresh_any_step_invariant:*
```
"paodv i ⊨_A onll Γ_AODV (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a)"
```
⟨*proof*⟩

Proposition 7.15

lemma *rerr_invalid_any_step_invariant:*
```
"paodv i ⊨_A onll Γ_AODV (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* ⇀ *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 Γ_AODV (λ(ξ, l). (l ∈ {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)))
         ∧ (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 Γ_AODV (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 *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 **: "ip∈kD(rt (σ sip)) ∧ sn ≤ nsqn (rt (σ sip)) ip*
∧ (nsqn (rt (σ sip)) ip = sn
⟶ (the (dhops (rt (σ sip)) ip) ≤ hops
∨ the (flag (rt (σ sip)) ip) = inv))"
shows *"ip∈kD(rt (σ' sip)) ∧ sn ≤ nsqn (rt (σ' sip)) ip*
∧ (nsqn (rt (σ' sip)) ip = sn
⟶ (the (dhops (rt (σ' sip)) ip) ≤ hops
∨ the (flag (rt (σ' sip)) ip) = inv))"
⟨*proof*⟩

lemma *rteq_quality_increases:*
assumes *"∀j. j ≠ i ⟶ quality_increases (σ j) (σ' j)"*
and *"rt (σ' i) = rt (σ i)"*
shows *"∀j. quality_increases (σ j) (σ' j)"*
⟨*proof*⟩

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

lemma *msg_fresh [simp]:*
"⋀hops rreqid dip dsn dsk oip osn sip.
msg_fresh σ (Rreq hops rreqid dip dsn dsk oip osn sip) =
(osn ≥ 1 ∧ (sip ≠ oip ⟶ oip∈kD(rt (σ sip))
∧ nsqn (rt (σ sip)) oip ≥ osn
∧ (nsqn (rt (σ sip)) oip = osn
⟶ (hops ≥ the (dhops (rt (σ sip)) oip)
∨ the (flag (rt (σ sip)) oip) = inv))))"
"⋀hops dip dsn oip sip. msg_fresh σ (Rrep hops dip dsn oip sip) =
(dsn ≥ 1 ∧ (sip ≠ dip ⟶ dip∈kD(rt (σ sip))
∧ nsqn (rt (σ sip)) dip ≥ dsn
∧ (nsqn (rt (σ sip)) dip = dsn
⟶ (hops ≥ the (dhops (rt (σ sip)) dip))
∨ the (flag (rt (σ sip)) dip) = inv)))"
"⋀dests sip. msg_fresh σ (Rerr dests sip) =
(∀ripc∈dom(dests). (ripc∈kD(rt (σ sip))
∧ the (dests ripc) - 1 ≤ nsqn (rt (σ sip)) ripc))"
"⋀d dip. msg_fresh σ (Newpkt d dip) = True"
"⋀d dip sip. msg_fresh σ (Pkt d dip sip) = True"
⟨*proof*⟩

lemma *msg_fresh_inc_sn [simp, elim]:*
"msg_fresh 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_OAodv*
imports *C_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 *"*σ_{AODV}*' ≡ {(λi. aodv_init i,* Γ_{AODV} *PAodv)}"*

abbreviation *opaodv*
:: "ip ⇒ ((ip ⇒ state) × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
"opaodv i ≡ (| init = σ_{AODV}*', trans = oseqp_sos* Γ_{AODV} *i |)"*

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

lemma *oaodv_control_within [simp]: "control_within* Γ_{AODV} *(init (opaodv i))"*
⟨*proof*⟩

lemma σ_{AODV}*'_labels [simp]: "(σ, p) ∈* σ_{AODV}*' ⟹ labels* Γ_{AODV} *p = {PAodv-:0}"*
⟨*proof*⟩

lemma *oaodv_init_kD_empty [simp]:*

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

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

lemma *oaodv_trans: "trans (opaodv i) = oseqp_sos Γ_AODV i"*
⟨*proof*⟩

declare
```
oseq_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros]
oseq_step_invariant_ctermsI [OF aodv_wf oaodv_control_within aodv_simple_labels oaodv_trans, cterms_intros]
```

end

3.9 Global invariant proofs over sequential processes

theory *C_Global_Invariants*
imports *C_Seq_Invariants*
C_Aodv_Predicates
C_Fresher
C_Quality_Increases
AWN.OAWN_Convert
C_OAodv
begin

lemma *other_quality_increases [elim]:*
assumes *"other quality_increases I 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 Γ_AODV (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 Γ_AODV (λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ')"
```
⟨*proof*⟩

lemmas *olocal_quality_increases =*
open_seq_step_invariant [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 ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
             other quality_increases {i} →)
            onl Γ_AODV (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 Γ_AODV (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 Γ_AODV (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 Γ_AODV (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 "_ ⊨ (?S, ?U →) _")
```
⟨*proof*⟩

lemma *odsn_rrep:*
```
"opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
             other quality_increases {i} →)
    onl Γ_AODV (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 Γ_AODV (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 Γ_AODV (λ(σ, l).
              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 Γ_AODV (λ(ξ, l). (l = PRerr-:1
                    ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ)
                                        ∧ the (nhop (rt ξ) ip) = sip ξ
                                        ∧ sqn (rt ξ) ip < the (dests ξ ip))))"
```
⟨*proof*⟩

```
lemmas 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 Γ_AODV (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 Γ_AODV (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
```

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

Given the state of a network 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. r dip = Some (dsn, dsk, val, hops, ip')))"
  ⟨proof⟩

lemma rt_graph_vD [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⟹ dip ∈ vD(rt (σ ip))"
  ⟨proof⟩

lemma rt_graph_vD_trans [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ (rt_graph σ dip)⁺ ⟹ dip ∈ vD(rt (σ ip))"
  ⟨proof⟩

lemma rt_graph_not_dip [dest]:
  "⋀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⟹ ip ≠ dip"
```

⟨*proof*⟩

lemma *rt_graph_not_dip_trans [dest]:*
"⋀*ip ip' σ dip. (ip, ip') ∈ (rt_graph σ dip)*$^+$ ⟹ *ip ≠ dip*"
⟨*proof*⟩

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

lemma *rt_graph_nhip_is_nhop [dest]:*
"⋀*ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⟹ ip' = the (nhop (rt (σ ip)) dip)*"
⟨*proof*⟩

theorem *inv_to_loop_freedom:*
assumes "∀ *i dip. let nhip = the (nhop (rt (σ i)) dip)*
in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
⟶ (rt (σ i)) ⊏$_{dip}$ *(rt (σ nhip))*"
shows "∀*dip. irrefl ((rt_graph σ dip)*$^+$*)*"
⟨*proof*⟩

end

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 *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)"
⟨*proof*⟩

3.11.2 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 Γ_AODV i) (seqp_sos Γ_QMSG))"
```
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 : 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*⟩

3.11.3 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*⟩

3.11.4 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*⟩

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_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 nhip = the (nhop (rt (σ i)) dip)
                                in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
                                   ⟶ (rt (σ i)) ⊏dip (rt (σ nhip)))"
      (is "_ ⊨ netglobal (λσ. ∀ i. ?inv σ i)")
⟨*proof*⟩

3.11.6 Loop freedom of AODV

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

definition *proj3* :: "r ⇒ k" (‹π_3›)
where "$\pi_3 \equiv \lambda(\_, dsk, \_, \_, \_, \_).\ dsk$"

definition *proj4* :: "r ⇒ f" (‹π_4›)
where "$\pi_4 \equiv \lambda(\_, \_, flag, \_, \_, \_).\ flag$"

definition *proj5* :: "r ⇒ nat" (‹π_5›)
where "$\pi_5 \equiv \lambda(\_, \_, \_, hops, \_, \_).\ hops$"

definition *proj6* :: "r ⇒ ip" (‹π_6›)
where "$\pi_6 \equiv \lambda(\_, \_, \_, \_, nhip, \_).\ nhip$"

definition *proj7* :: "r ⇒ ip set" (‹π_7›)
where "$\pi_7 \equiv \lambda(\_, \_, \_, \_, \_, pre).\ pre$"

lemma *projs [simp]:*
"π_2(dsn, dsk, flag, hops, nhip, pre) = dsn"
"π_3(dsn, dsk, flag, hops, nhip, pre) = dsk"
"π_4(dsn, dsk, flag, hops, nhip, pre) = flag"
"π_5(dsn, dsk, flag, hops, nhip, pre) = hops"
"π_6(dsn, dsk, flag, hops, nhip, pre) = nhip"
"π_7(dsn, dsk, flag, hops, nhip, pre) = pre"
⟨*proof*⟩

lemma *proj3_pred [intro]:* "⟦ P kno; P unk ⟧ ⟹ P (π_3 x)"
⟨*proof*⟩

lemma *proj4_pred [intro]:* "⟦ P val; P inv ⟧ ⟹ P (π_4 x)"
⟨*proof*⟩

lemma *proj6_pair_snd [simp]:*
fixes *dsn' r*
shows "π_6 (dsn', snd (r)) = π_6(r)"
⟨*proof*⟩

4.1.3 Routing Tables

Routing tables map ip addresses to route entries.

type_synonym *rt* = "ip ⇀ r"

syntax
"_Sigma_route" :: "rt ⇒ ip ⇀ r" (‹σ_{route}'(_, _')›)

translations
"σ_{route}(rt, dip)" => "rt dip"

definition *sqn* :: "rt ⇒ ip ⇒ sqn"
where "sqn rt dip ≡ case σ_{route}(rt, dip) of Some r ⇒ π_2(r) | None ⇒ 0"

definition *sqnf* :: "rt ⇒ ip ⇒ k"
where "sqnf rt dip ≡ case σ_{route}(rt, dip) of Some r ⇒ π_3(r) | None ⇒ unk"

abbreviation *flag* :: "rt ⇒ ip ⇀ f"
where "flag rt dip ≡ map_option π_4 (σ_{route}(rt, dip))"

abbreviation *dhops* :: "rt ⇒ ip ⇀ nat"
where "dhops rt dip ≡ map_option π_5 (σ_{route}(rt, dip))"

abbreviation *nhop* :: "rt ⇒ ip ⇀ ip"
where "nhop rt dip ≡ map_option π_6 (σ_{route}(rt, dip))"

abbreviation *precs* :: "rt ⇒ ip ⇀ ip set"

where *"precs rt dip ≡ map_option π₇ (σ_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*⟩

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 "π_2(addpre v pre) = π_2(v)"
⟨*proof*⟩

lemma *proj3_addpre:*
fixes *v pre*
shows "π_3(addpre v pre) = π_3(v)"
⟨*proof*⟩

lemma *proj4_addpre:*
fixes *v pre*
shows "π_4(addpre v pre) = π_4(v)"
⟨*proof*⟩

lemma *proj5_addpre:*
fixes *v pre*
shows "π_5(addpre v pre) = π_5(v)"
⟨*proof*⟩

lemma *proj6_addpre:*
fixes *dsn dsk flag hops nhip pre npre*
shows "π_6(addpre v npre) = π_6(v)"
⟨*proof*⟩

lemma *proj7_addpre:*
fixes *dsn dsk flag hops nhip pre npre*
shows "π_7(addpre v npre) = π_7(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)) (σ_{route}(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 "π_2(the (the (addpreRT rt ip' npre) ip)) = π_2(the (rt ip))"
⟨*proof*⟩

lemma *proj3_addpreRT [simp]:*
fixes *ip rt ip' npre*
assumes "ip∈kD rt"
and "ip'∈kD rt"
shows "π_3(the (the (addpreRT rt ip' npre) ip)) = π_3(the (rt ip))"
⟨*proof*⟩

lemma *proj5_addpreRT [simp]:*
"⋀rt dip ip npre. dip∈kD(rt) ⟹ π_5(the (the (addpreRT rt dip npre) ip)) = π_5(the (rt ip))"
⟨*proof*⟩

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

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

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

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

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

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

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

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

Updating route entries

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

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

lemma *rt_Some_sqn [dest]:*
fixes *rt* **and** *ip dsn dsk flag hops nhip pre*
assumes *"rt ip = Some (dsn, dsk, flag, hops, nhip, pre)"*

shows *"sqn rt ip = dsn"*
⟨*proof*⟩

lemma *not_kD_sqn [simp]:*
fixes *dip rt*
assumes *"dip ∉ kD(rt)"*
shows *"sqn rt dip = 0"*
⟨*proof*⟩

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

lemma *update_arg_wf_gives_cases:*
"⋀r. update_arg_wf r ⟹ (π2(r) = 0) = (π3(r) = unk)"
⟨*proof*⟩

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

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

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

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

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

$\Longrightarrow$ update rt ip r = rt (ip $\mapsto$ ns)"
⟨proof⟩

lemma update_cases [elim]:
assumes "(π_2(r) = 0) = (π_3(r) = unk)"
and c1: "⟦ip $\notin$ kD(rt)⟧ $\Longrightarrow$ P (rt (ip $\mapsto$ r))"

and c2: "⟦ip $\in$ kD(rt); sqn rt ip < π_2(r)⟧
$\Longrightarrow$ P (rt (ip $\mapsto$ addpre r (π_7(the σ_{route}(rt, ip)))))"
and c3: "⟦ip $\in$ kD(rt); sqn rt ip = π_2(r); the (dhops rt ip) > π_5(r)⟧
$\Longrightarrow$ P (rt (ip $\mapsto$ addpre r (π_7(the σ_{route}(rt, ip)))))"
and c4: "⟦ip $\in$ kD(rt); sqn rt ip = π_2(r); the (flag rt ip) = inv⟧
$\Longrightarrow$ P (rt (ip $\mapsto$ addpre r (π_7(the σ_{route}(rt, ip)))))"
and c5: "⟦ip $\in$ kD(rt); π_3(r) = unk⟧
$\Longrightarrow$ P (rt (ip $\mapsto$ (π_2(the σ_{route}(rt, ip)), π_3(r),
π_4(r), π_5(r), π_6(r), π_7(addpre r (π_7(the σ_{route}(rt, ip))))))))"
and c6: "⟦ip $\in$ kD(rt); sqn rt ip $\geq$ π_2(r); π_3(r) = kno;
sqn rt ip = π_2(r) $\Longrightarrow$ the (dhops rt ip) $\leq$ π_5(r) $\wedge$ the (flag rt ip) = val⟧
$\Longrightarrow$ P (rt (ip $\mapsto$ addpre (the σ_{route}(rt, ip)) (π_7(r))))"
shows "(P (update rt ip r))"
⟨proof⟩

lemma update_cases_kD:
assumes "(π_2(r) = 0) = (π_3(r) = unk)"
and "ip $\in$ kD(rt)"
and c2: "sqn rt ip < π_2(r) $\Longrightarrow$ P (rt (ip $\mapsto$ addpre r (π_7(the σ_{route}(rt, ip)))))"
and c3: "⟦sqn rt ip = π_2(r); the (dhops rt ip) > π_5(r)⟧
$\Longrightarrow$ P (rt (ip $\mapsto$ addpre r (π_7(the σ_{route}(rt, ip)))))"
and c4: "⟦sqn rt ip = π_2(r); the (flag rt ip) = inv⟧
$\Longrightarrow$ P (rt (ip $\mapsto$ addpre r (π_7(the σ_{route}(rt, ip)))))"
and c5: "π_3(r) = unk $\Longrightarrow$ P (rt (ip $\mapsto$ (π_2(the σ_{route}(rt, ip)), π_3(r),
π_4(r), π_5(r), π_6(r),
π_7(addpre r (π_7(the σ_{route}(rt, ip))))))))"
and c6: "⟦sqn rt ip $\geq$ π_2(r); π_3(r) = kno;
sqn rt ip = π_2(r) $\Longrightarrow$ the (dhops rt ip) $\leq$ π_5(r) $\wedge$ the (flag rt ip) = val⟧
$\Longrightarrow$ P (rt (ip $\mapsto$ addpre (the σ_{route}(rt, ip)) (π_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 (λx. if x = rip then Some v else rt x) ip
= (if ip = rip then π_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]: "⋀rt dip dsn dsk flg hops sip.*
rt ≠ update rt dip (dsn, dsk, flg, hops, sip, {})
⟹ sqnf (update rt dip (dsn, dsk, flg, hops, sip, {})) dip = dsk"
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

lemma *vD_update_val [dest]:*
"⋀dip 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]:*
```
"⋀dip. π3(the ((invalidate rt dests) dip)) = π3(the (rt dip))"
```
⟨*proof*⟩

lemma *proj5_invalidate [simp]:*
```
"⋀dip. π5(the ((invalidate rt dests) dip)) = π5(the (rt dip))"
```
⟨*proof*⟩

lemma *proj6_invalidate [simp]:*
```
"⋀dip. π6(the ((invalidate rt dests) dip)) = π6(the (rt dip))"
```
⟨*proof*⟩

lemma *proj7_invalidate [simp]:*
```
"⋀dip. π7(the ((invalidate rt dests) dip)) = π7(the (rt dip))"
```
⟨*proof*⟩

lemma *invalidate_kD_inv [simp]:*
```
"⋀rt dests. kD (invalidate rt dests) = kD rt"
```
⟨*proof*⟩

lemma *invalidate_sqn:*
fixes *rt dip dests*
assumes *"∀rsn. dests dip = Some rsn ⟶ sqn rt dip ≤ rsn"*
shows *"sqn rt dip ≤ sqn (invalidate rt dests) dip"*
⟨*proof*⟩

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

lemma *dhops_invalidate [simp]:*
```
"⋀dip. the (dhops (invalidate rt dests) dip) = the (dhops rt dip)"
```
⟨*proof*⟩

lemma *sqnf_invalidate [simp]:*
```
"⋀dip. sqnf (invalidate (rt ξ) (dests ξ)) dip = sqnf (rt ξ) dip"
```
⟨*proof*⟩

lemma *nhop_invalidate [simp]:*
```
"⋀dip. the (nhop (invalidate (rt ξ) (dests ξ)) dip) = the (nhop (rt ξ) dip)"
```
⟨*proof*⟩

lemma *invalidate_other [simp]:*
fixes *rt dests dip*
assumes *"dip∉dom(dests)"*
shows *"invalidate rt dests dip = rt dip"*
⟨*proof*⟩

lemma *invalidate_none [simp]:*
fixes *rt dests dip*
assumes *"dip∉kD(rt)"*
shows *"invalidate rt dests dip = None"*
⟨*proof*⟩

lemma *vD_invalidate_vD_not_dests:*
```
"⋀dip rt dests. dip∈vD(invalidate rt dests) ⟹ dip∈vD(rt) ∧ dests dip = None"
```

⟨*proof*⟩

lemma *sqn_invalidate_not_in_dests [simp]:*
fixes *dests dip rt*
assumes *"dip∉dom(dests)"*
shows *"sqn (invalidate rt dests) dip = sqn rt dip"*
⟨*proof*⟩

lemma *invalidate_changes:*
fixes *rt dests dip dsn dsk flag hops nhip pre*
assumes *"invalidate rt dests dip = Some (dsn, dsk, flag, hops, nhip, pre)"*
shows *" dsn = (case dests dip of None ⇒ 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"* (‹σ_{queue}'(_, _')›)
where *"σ_{queue}(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"* (‹$\sigma_{p\text{-}flag}$'(_, _')›)

where "$\sigma_{p\text{-}flag}$(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)"

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 ⇒ 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_Fwdrreqs
begin

datatype msg =
Rreq nat rreqid 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 × rreqid × ip × sqn × k × ip × sqn × ip × bool ⇒ msg"
where "rreq ≡ λ(hops, rreqid, dip, dsn, dsk, oip, osn, sip, handled).
Rreq hops rreqid dip dsn dsk oip osn sip handled"

lemma rreq_simp [simp]:
"rreq(hops, rreqid, dip, dsn, dsk, oip, osn, sip, handled) = Rreq hops rreqid dip dsn dsk oip osn sip handled"
⟨*proof*⟩

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

lemma *rrep_simp [simp]:*
  "rrep(hops, dip, dsn, oip, sip) = Rrep hops dip dsn oip sip"
  ⟨*proof*⟩

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

lemma *rerr_simp [simp]:*
  "rerr(dests, sip) = Rerr dests sip"
  ⟨*proof*⟩

lemma *not_eq_newpkt_rreq [simp]: "¬eq_newpkt (Rreq hops rreqid dip dsn dsk oip osn sip handled)"*
  ⟨*proof*⟩

lemma *not_eq_newpkt_rrep [simp]: "¬eq_newpkt (Rrep hops dip dsn oip sip)"*
  ⟨*proof*⟩

lemma *not_eq_newpkt_rerr [simp]: "¬eq_newpkt (Rerr dests sip)"*
  ⟨*proof*⟩

lemma *not_eq_newpkt_pkt [simp]: "¬eq_newpkt (Pkt d dip sip)"*
  ⟨*proof*⟩

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

lemma *pkt_simp [simp]:*
  "pkt(d, dip, sip) = Pkt d dip sip"
  ⟨*proof*⟩

end

4.3 The AODV protocol

theory *D_Aodv*
imports *D_Aodv_Data D_Aodv_Message*
        AWN.AWN_SOS_Labels AWN.AWN_Invariants
begin

4.3.1 Data state

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

  msg   :: "msg"
  data   :: "data"
  dests :: "ip ⇀ sqn"
  pre   :: "ip set"
  rreqid :: "rreqid"
  dip   :: "ip"
  oip   :: "ip"
  hops   :: "nat"
  dsn   :: "sqn"
  dsk   :: "k"
  osn   :: "sqn"
  sip   :: "ip"
  handled:: "bool"

abbreviation *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' ⇒ { 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' handled' ⇒
                     { ξ(| hops := hops', rreqid := rreqid', 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' rreqid' dip' dsn' dsk' oip' osn' sip' handled'.*
```
            msg ξ = Rreq hops' rreqid' dip' dsn' dsk' oip' osn' sip' handled' ∧
            ξ' = ξ(| 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 ξ ≡ case msg ξ of*
```
                   Rrep hops' dip' dsn' oip' sip' ⇒
                     { ξ(| hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' |) }
                 | _ ⇒ {}"
```

lemma *is_rrep_asm [dest!]:*
assumes *"ξ' ∈ is_rrep ξ"*
shows *"(∃hops' dip' dsn' oip' sip'.*
```
            msg ξ = Rrep hops' dip' dsn' oip' sip' ∧
            ξ' = ξ(| hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' |))"
```
⟨*proof*⟩

definition *is_rerr*
where *"is_rerr ξ ≡ case msg ξ of*
```
                   Rerr dests' sip' ⇒ { ξ(| dests := dests', sip := sip' |) }
                 | _ ⇒ {}"
```

lemma *is_rerr_asm [dest!]:*
assumes *"ξ' ∈ is_rerr ξ"*
shows *"(∃dests' sip'.*
```
            msg ξ = Rerr dests' sip' ∧
            ξ' = ξ(| dests := dests', sip := sip' |))"
```
⟨*proof*⟩

lemmas *is_msg_defs =*
is_rerr_def is_rrep_def is_rreq_def is_pkt_def is_newpkt_def

lemma *is_msg_inv_ip [simp]:*
```
"ξ' ∈ is_rerr ξ   ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_rrep ξ   ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_rreq ξ   ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_pkt ξ    ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_newpkt ξ ⟹ ip ξ' = ip ξ"
```
⟨*proof*⟩

lemma *is_msg_inv_sn [simp]:*
```
"ξ' ∈ is_rerr ξ   ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_rrep ξ   ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_rreq ξ   ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_pkt ξ    ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_newpkt ξ ⟹ sn ξ' = sn ξ"
```
⟨*proof*⟩

lemma *is_msg_inv_rt [simp]:*
```
"ξ' ∈ is_rerr ξ   ⟹ rt ξ' = rt ξ"
"ξ' ∈ is_rrep ξ   ⟹ rt ξ' = rt ξ"
"ξ' ∈ is_rreq ξ   ⟹ rt ξ' = rt ξ"
"ξ' ∈ is_pkt ξ    ⟹ rt ξ' = rt ξ"
"ξ' ∈ is_newpkt ξ ⟹ rt ξ' = rt ξ"
```
⟨*proof*⟩

lemma *is_msg_inv_rreqs [simp]:*
```
"ξ' ∈ is_rerr ξ   ⟹ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_rrep ξ   ⟹ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_rreq ξ   ⟹ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_pkt ξ    ⟹ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_newpkt ξ ⟹ rreqs ξ' = rreqs ξ"
```
⟨*proof*⟩

lemma *is_msg_inv_store [simp]:*
```
"ξ' ∈ is_rerr ξ   ⟹ store ξ' = store ξ"
"ξ' ∈ is_rrep ξ   ⟹ store ξ' = store ξ"
"ξ' ∈ is_rreq ξ   ⟹ store ξ' = store ξ"
"ξ' ∈ is_pkt ξ    ⟹ store ξ' = store ξ"
"ξ' ∈ is_newpkt ξ ⟹ store ξ' = store ξ"
```
⟨*proof*⟩

lemma *is_msg_inv_sip [simp]:*
```
"ξ' ∈ is_pkt ξ    ⟹ sip ξ' = sip ξ"
"ξ' ∈ is_newpkt ξ ⟹ sip ξ' = sip ξ"
```
⟨*proof*⟩

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 ξ) (| data := data, dip := dip, oip := oip |)]
    call(PPkt)"
abbreviation NEWPKT
where
  "NEWPKT args ≡
    [ξ. let (data, dip) = args ξ in
        (clear_locals ξ) (| data := data, dip := dip |)]
    call(PNewPkt)"

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

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

abbreviation RERR
where
  "RERR args ≡
    [ξ. let (dests, sip) = args ξ in
        (clear_locals ξ) (| dests := dests, sip := sip |)]
    call(PRerr)"

fun Γ_AODV :: "(state, msg, pseqp, pseqp label) seqp_env"
where
  "Γ_AODV PAodv = labelled PAodv (
    receive(λmsg' ξ. ξ (| msg := msg' |)).
    (    ⟨is_newpkt⟩ NEWPKT(λξ. (data ξ, ip ξ))
      ⊕ ⟨is_pkt⟩ PKT(λξ. (data ξ, dip ξ, oip ξ))
      ⊕ ⟨is_rreq⟩
          [ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) |)]
          RREQ(λξ. (hops ξ, rreqid ξ, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, sip ξ, handled ξ))
      ⊕ ⟨is_rrep⟩
          [ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) |)]
          RREP(λξ. (hops ξ, dip ξ, dsn ξ, oip ξ, sip ξ))
      ⊕ ⟨is_rerr⟩
          [ξ. ξ (|rt := update (rt ξ) (sip ξ) (0, unk, val, 1, sip ξ, {}) |)]
          RERR(λξ. (dests ξ, sip ξ))
    )
    ⊕ ⟨λξ. { ξ(| dip := dip |) | dip. dip ∈ qD(store ξ) ∩ vD(rt ξ) }⟩
        [ξ. ξ (| data := hd(σ_queue(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 (σ_p-flag(store ξ, dip)) = req }⟩
```

```
        ⟦ξ. ξ (| store := unsetRRF (store ξ) (dip ξ) |)⟧
        ⟦ξ. ξ (| sn := inc (sn ξ) |)⟧
        ⟦ξ. ξ (| rreqid := nrreqid (rreqs ξ) (ip ξ) |)⟧
        ⟦ξ. ξ (| rreqs := rreqs ξ ∪ {(ip ξ, rreqid ξ)} |)⟧
        broadcast(λξ. rreq(0, rreqid ξ, dip ξ, sqn (rt ξ) (dip ξ), sqnf (rt ξ) (dip ξ), ip ξ, sn ξ,
                           ip ξ, False)). AODV())"

|  "Γ_AODV 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())"

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

| "Γ_AODV PRreq = labelled PRreq (
     ⟨ξ. (oip ξ, rreqid ξ) ∈ rreqs ξ⟩
       AODV()
     ⊕ ⟨ξ. (oip ξ, rreqid ξ) ∉ rreqs ξ⟩
       ⟦ξ. ξ (| rt := update (rt ξ) (oip ξ) (osn ξ, kno, val, hops ξ + 1, sip ξ, {}) |)⟧
       ⟦ξ. ξ (| rreqs := rreqs ξ ∪ {(oip ξ, rreqid ξ)} |)⟧
       (
         ⟨ξ. handled ξ = False⟩
         (
           ⟨ξ. dip ξ = ip ξ⟩
             ⟦ξ. ξ (| sn := max (sn ξ) (dsn ξ) |)⟧
             unicast(λξ. the (nhop (rt ξ) (oip ξ)), λξ. rrep(0, dip ξ, sn ξ, oip ξ, ip ξ)).
               broadcast(λξ. rreq(hops ξ + 1, rreqid ξ, dip ξ, dsn ξ, dsk ξ, oip ξ, osn ξ, ip ξ, True)).
               AODV()
             ▷
               ⟦ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (oip ξ))
                                       then Some (inc (sqn (rt ξ) rip)) else None) |)⟧
               ⟦ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)⟧
               ⟦ξ. ξ (| store := setRRF (store ξ) (dests ξ)|)⟧
               ⟦ξ. ξ (| pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } |)⟧
               ⟦ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                                       then (dests ξ) rip else None) |)⟧
               groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)).AODV()
           ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
```

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

| "Γ_AODV 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()
    )"

| "Γ_AODV PRerr = labelled PRerr (
    ⟦ξ. ξ ⦇ dests := (λrip. case (dests ξ) rip of None ⇒ None
```

```
                    | Some rsn ⇒ if rip ∈ vD (rt ξ) ∧ the (nhop (rt ξ) rip) = sip ξ
                                   ∧ sqn (rt ξ) rip < rsn then Some rsn else None) ⦈⟧
    ⟦ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)⟧
    ⟦ξ. ξ (| store := setRRF (store ξ) (dests ξ)|)⟧
    ⟦ξ. ξ (| pre := ⋃{ the (precs (rt ξ) rip) | rip. rip ∈ dom (dests ξ) } |)⟧
    ⟦ξ. ξ (| dests := (λrip. if ((dests ξ) rip ≠ None ∧ the (precs (rt ξ) rip) ≠ {})
                        then (dests ξ) rip else None) |)⟧
    groupcast(λξ. pre ξ, λξ. rerr(dests ξ, ip ξ)). AODV())"
```

declare Γ_{AODV}`.simps [simp del, code del]`
lemmas Γ_{AODV}`_simps [simp, code] =` Γ_{AODV}`.simps [simplified]`

fun Γ_{AODV}`_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_{AODV}`_skeleton_wf [simp]:`
`"wellformed` Γ_{AODV}`_skeleton"`
⟨*proof*⟩

declare Γ_{AODV}`_skeleton.simps [simp del, code del]`
lemmas Γ_{AODV}`_skeleton_simps [simp, code]`
`=` Γ_{AODV}`_skeleton.simps [simplified` Γ_{AODV}`_simps seqp_skeleton.simps]`

lemma `aodv_proc_cases [dest]:`
fixes `p pn`
shows `"p ∈ ctermsl (`Γ_{AODV} `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_{AODV} `:: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp) set"`
where σ_{AODV} `i ≡ {(aodv_init i,` Γ_{AODV} `PAodv)}"`

abbreviation *paodv*
`:: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"`
where
`"paodv i ≡ (| init =` σ_{AODV} `i, trans = seqp_sos` Γ_{AODV} `|)"`

lemma `aodv_trans: "trans (paodv i) = seqp_sos` Γ_{AODV}`"`
⟨*proof*⟩

lemma `aodv_control_within [simp]: "control_within` Γ_{AODV} `(init (paodv i))"`
⟨*proof*⟩

lemma `aodv_wf [simp]:`
`"wellformed` Γ_{AODV}`"`
⟨*proof*⟩

lemmas `aodv_labels_not_empty [simp] = labels_not_empty [OF aodv_wf]`

lemma `aodv_ex_label [intro]: "∃l. l∈labels` Γ_{AODV} `p"`
⟨*proof*⟩

lemma `aodv_ex_labelE [elim]:`

assumes "∀l∈labels Γ_{AODV} p. P l p"
and "∃p l. P l p ⟹ Q"
shows "Q"
⟨*proof*⟩

lemma *aodv_simple_labels [simp]: "simple_labels* Γ_{AODV}*"*
⟨*proof*⟩

lemma σ_{AODV}*_labels [simp]: "(ξ, p) ∈* σ_{AODV} *i ⟹ labels* Γ_{AODV} *p = {PAodv-:0}"*
⟨*proof*⟩

lemma *aodv_init_kD_empty [simp]:*
"(ξ, p) ∈ σ_{AODV} *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) ∈* σ_{AODV} *i"*
shows *"sip ξ ≠ ip ξ"*
⟨*proof*⟩

lemma *aodv_init_sip_not_i [simp]:*
assumes *"(ξ, p) ∈* σ_{AODV} *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
Γ_{AODV}*_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

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_simps [simp]:*

```
  "⋀hops rreqid dip dsn dsk oip osn sip handled.
                          msg_sender (Rreq hops rreqid dip dsn dsk oip osn sip handled) = sip"
  "⋀hops dip dsn oip sip. msg_sender (Rrep hops dip dsn oip sip) = sip"
  "⋀dests sip.            msg_sender (Rerr dests sip) = sip"
  "⋀d dip sip.            msg_sender (Pkt d dip sip) = sip"
  ⟨proof⟩

definition msg_zhops :: "msg ⇒ bool"
where "msg_zhops m ≡ case m of
                               Rreq hopsc _ dipc _ _ oipc _ sipc _ ⇒ hopsc = 0 ⟶ oipc = sipc
                             | Rrep hopsc dipc _ _ sipc ⇒ hopsc = 0 ⟶ dipc = sipc
                             | _ ⇒ True"

lemma msg_zhops_simps [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip handled.
          msg_zhops (Rreq hops rreqid dip dsn dsk oip osn sip handled) = (hops = 0 ⟶ oip = sip)"
  "⋀hops dip dsn oip sip. msg_zhops (Rrep hops dip dsn oip sip) = (hops = 0 ⟶ dip = sip)"
  "⋀dests sip.            msg_zhops (Rerr dests sip)        = True"
  "⋀d dip.                msg_zhops (Newpkt d dip)          = True"
  "⋀d dip sip.            msg_zhops (Pkt d dip sip)         = True"
  ⟨proof⟩

definition rreq_rrep_sn :: "msg ⇒ bool"
where "rreq_rrep_sn m ≡ case m of Rreq _ _ _ _ _ _ osnc _ _ ⇒ osnc ≥ 1
                             | Rrep _ _ dsnc _ _ ⇒ dsnc ≥ 1
                             | _ ⇒ True"

lemma rreq_rrep_sn_simps [simp]:
  "⋀hops rreqid dip dsn dsk oip osn sip handled.
          rreq_rrep_sn (Rreq hops rreqid dip dsn dsk oip osn sip handled) = (osn ≥ 1)"
  "⋀hops dip dsn oip sip. rreq_rrep_sn (Rrep hops dip dsn oip sip) = (dsn ≥ 1)"
  "⋀dests sip.            rreq_rrep_sn (Rerr dests sip) = True"
  "⋀d dip.                rreq_rrep_sn (Newpkt d dip)   = True"
  "⋀d dip sip.            rreq_rrep_sn (Pkt d dip sip)  = True"
  ⟨proof⟩

definition rreq_rrep_fresh :: "rt ⇒ msg ⇒ bool"
where "rreq_rrep_fresh crt m ≡ case m of Rreq hopsc _ _ _ _ oipc osnc ipcc _ ⇒ (ipcc ≠ oipc ⟶
                                          oipc∈kD(crt) ∧ (sqn crt oipc > osnc
                                                        ∨ (sqn crt oipc = osnc
                                                           ∧ the (dhops crt oipc) ≤ hopsc
                                                           ∧ the (flag crt oipc) = val)))
                             | Rrep hopsc dipc dsnc _ ipcc ⇒ (ipcc ≠ dipc ⟶
                                                              dipc∈kD(crt)
                                                            ∧ sqn crt dipc = dsnc
                                                            ∧ the (dhops crt dipc) = hopsc
                                                            ∧ the (flag crt dipc) = val)
                             | _ ⇒ True"

lemma rreq_rrep_fresh [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"
```

```
  "⋀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 rreqid dip dsn dsk oip osn sip handled.
                          rerr_invalid crt (Rreq hops rreqid 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

4.5 Quality relations between routes

theory *D_Fresher*
imports *D_Aodv_Data*
begin

4.5.1 Net sequence numbers

On individual routes

definition
$nsqn_r$ :: "r ⇒ sqn"
where
"$nsqn_r$ r ≡ if π_4(r) = val ∨ π_2(r) = 0 then π_2(r) else (π_2(r) - 1)"

lemma *nsqnr_def':*
"$nsqn_r$ r = (if π_4(r) = inv then π_2(r) - 1 else π_2(r))"
⟨*proof*⟩

lemma *nsqn$_r$_zero [simp]:*
"⋀dsn dsk flag hops nhip pre. nsqn$_r$ (0, dsk, flag, hops, nhip, pre) = 0"
⟨*proof*⟩

lemma *nsqn$_r$_val [simp]:*
"⋀dsn dsk hops nhip pre. nsqn$_r$ (dsn, dsk, val, hops, nhip, pre) = dsn"
⟨*proof*⟩

lemma *nsqn$_r$_inv [simp]:*
"⋀dsn dsk hops nhip pre. nsqn$_r$ (dsn, dsk, inv, hops, nhip, pre) = dsn - 1"
⟨*proof*⟩

lemma *nsqn$_r$_lte_dsn [simp]:*
"⋀dsn dsk flag hops nhip pre. 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:*
"⋀rt dip. nsqn rt dip = (if flag rt dip = Some val ∨ sqn rt dip = 0
then sqn rt dip else sqn rt dip - 1)"
⟨*proof*⟩

lemma *not_in_kD_nsqn [simp]:*
assumes *"dip ∉ kD(rt)"*
shows *"nsqn rt dip = 0"*
⟨*proof*⟩

lemma *kD_nsqn:*
assumes *"dip ∈ kD(rt)"*
shows *"nsqn rt dip = nsqn$_r$(the (σ$_{route}$(rt, dip)))"*
⟨*proof*⟩

lemma *nsqnr_r_flag_pred [simp, intro]:*
fixes *dsn dsk flag hops nhip pre*
assumes *"P (nsqn$_r$ (dsn, dsk, val, hops, nhip, pre))"*
and *"P (nsqn$_r$ (dsn, dsk, inv, hops, nhip, pre))"*
shows *"P (nsqn$_r$ (dsn, dsk, flag, hops, nhip, pre))"*
⟨*proof*⟩

lemma *nsqn$_r$_addpreRT_inv [simp]:*
"⋀rt dip npre dip'. dip ∈ kD(rt) ⟹
nsqn$_r$ (the (the (addpreRT rt dip npre) dip')) = nsqn$_r$ (the (rt dip'))"
⟨*proof*⟩

lemma *sqn_nsqn:*
"⋀rt dip. sqn rt dip - 1 ≤ nsqn rt dip"
⟨*proof*⟩

lemma *nsqn_sqn: "nsqn rt dip ≤ sqn rt dip"*
⟨*proof*⟩

lemma *val_nsqn_sqn [elim, simp]:*
assumes *"ip∈kD(rt)"*
and *"the (flag rt ip) = val"*
shows *"nsqn rt ip = sqn rt ip"*
⟨*proof*⟩

lemma *vD_nsqn_sqn [elim, simp]:*
assumes *"ip∈vD(rt)"*
shows *"nsqn rt ip = sqn rt ip"*
⟨*proof*⟩

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

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

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

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

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

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

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

4.5.2 Comparing routes

definition
fresher :: "r ⇒ r ⇒ bool" (‹(_/ ⊑ _)› [51, 51] 50)
where
"fresher r r' ≡ ((nsqn$_r$ r < nsqn$_r$ r') ∨ (nsqn$_r$ r = nsqn$_r$ r' ∧ π$_5$(r) ≥ π$_5$(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 *"π$_5$(r) ≥ π$_5$(r')"*
shows *"r ⊑ r'"*
⟨*proof*⟩

lemma *fresherI [intro]:*
assumes *"(nsqn$_r$ r < nsqn$_r$ r') ∨ (nsqn$_r$ r = nsqn$_r$ r' ∧ π$_5$(r) ≥ π$_5$(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' ∧ π$_5$(r) ≥ π$_5$(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 (σ$_{route}$(rt, dip))) ⊑ (the (σ$_{route}$(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':*
"(rt$_1$ ⊑$_i$ rt$_2$) = (nsqn$_r$ (the (rt$_1$ i)) < nsqn$_r$ (the (rt$_2$ i)) ∨
nsqn$_r$ (the (rt$_1$ i)) = nsqn$_r$ (the (rt$_2$ i)) ∧ π$_5$ (the (rt$_2$ i)) ≤ π$_5$ (the (rt$_1$ 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_nsqnr:
assumes "dip ∈ kD(rt1)"
and "dip ∈ kD(rt2)"
and "rt1 ≈dip rt2"
shows "nsqn$_r$ (the (rt2 dip)) = nsqn$_r$ (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
  "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 $\sqsubset_i$ rt2 = ((rt1 $\sqsubseteq_i$ rt2) $\wedge$ $\neg$(rt2 $\sqsubseteq_i$ rt1))"
⟨proof⟩

lemma rt_strictly_fresherI' [intro]:
assumes "rt1 $\sqsubseteq_i$ rt2"
and "$\neg$(rt2 $\sqsubseteq_i$ rt1)"
shows "rt1 $\sqsubset_i$ rt2"
⟨proof⟩

lemma rt_strictly_fresherE' [elim]:
assumes "rt1 $\sqsubset_i$ rt2"
and "$[\![$ rt1 $\sqsubseteq_i$ rt2; $\neg$(rt2 $\sqsubseteq_i$ rt1) $]\!] \Longrightarrow$ P rt1 rt2 i"
shows "P rt1 rt2 i"
⟨proof⟩

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

lemmas rt_strictly_fresher_singleI [elim] = rt_strictly_fresherI [OF single_rt_fresher]

lemma rt_strictly_fresherE [elim]:
assumes "rt1 $\sqsubset_i$ rt2"
and "$[\![$ rt1 $\sqsubseteq_i$ rt2; $\neg$(rt1 $\approx_i$ rt2) $]\!] \Longrightarrow$ P rt1 rt2 i"
shows "P rt1 rt2 i"
⟨proof⟩

lemma rt_strictly_fresher_def':
"rt1 $\sqsubset_i$ rt2 =
(nsqn$_r$ (the (rt1 i)) < nsqn$_r$ (the (rt2 i))
$\vee$ (nsqn$_r$ (the (rt1 i)) = nsqn$_r$ (the (rt2 i)) $\wedge$ π_5(the (rt1 i)) > π_5(the (rt2 i))))"
⟨proof⟩

lemma rt_strictly_fresher_fresherD [dest]:
assumes "rt1 $\sqsubset_{dip}$ rt2"
shows "the (rt1 dip) $\sqsubseteq$ the (rt2 dip)"
⟨proof⟩

lemma rt_strictly_fresher_not_fresh_asD [dest]:
assumes "rt1 $\sqsubset_{dip}$ rt2"
shows "$\neg$ rt1 $\approx_{dip}$ rt2"
⟨proof⟩

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

lemma rt_strictly_fresher_irefl [simp]: "$\neg$ (rt $\sqsubset_{dip}$ rt)"
⟨proof⟩

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

lemma rt_fresher_trans_rt_strictly_fresher' [elim, trans]:
assumes "rt1 $\sqsubseteq_{dip}$ rt2"
and "rt2 $\sqsubset_{dip}$ rt3"
shows "rt1 $\sqsubset_{dip}$ 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 *"π_5(the (rt2 i)) < π_5(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 $\models_A$ *onll* Γ_{AODV} *(λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"*
⟨*proof*⟩

lemma *sequence_number_one_or_bigger:*
"paodv i $\models$ *onl* Γ_{AODV} *(λ(ξ, _). 1 ≤ sn ξ)"*
⟨*proof*⟩

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

lemma *sequence_number_increases':*
"paodv i $\models_A$ *(λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"*
⟨*proof*⟩

lemma *sequence_number_one_or_bigger':*
"paodv i $\models$ *(λ(ξ, _). 1 ≤ sn ξ)"*
⟨*proof*⟩

lemma *sip_in_kD:*
"paodv i $\models$ *onl* Γ_{AODV} *(λ(ξ, l). l ∈ ({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 $\models$ *onl* Γ_{AODV} *(λ(ξ, l). (l = 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 $\models$
onl Γ_{AODV} *(λ(ξ, l). (l ∈ {PRreq-:18..PRreq-:20} ∪ {PRrep-:2..PRrep-:6} ⟶ dip ξ ∈ kD (rt ξ))*
∧ (l ∈ {PRreq-:3..PRreq-:19} ⟶ oip ξ ∈ kD (rt ξ)))"
⟨*proof*⟩

Proposition 7.38

lemma *includes_nhip:*
"paodv i $\models$ *onl* Γ_{AODV} *(λ(ξ, l). ∀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 $\models$ *onl* Γ_{AODV} *(λ(ξ, l). (l ∈ {PRreq-:18..PRreq-:20} ⟶ dip ξ ∈ kD (rt ξ)) ∧*
(l = PRreq-:19 ⟶ oip ξ ∈ kD (rt ξ)) ∧
(l = PRrep-:5 ⟶ dip ξ ∈ kD (rt ξ)) ∧
(l = PRrep-:6 ⟶ (the (nhop (rt ξ) (dip ξ))) ∈ kD (rt ξ)))"
(is *"_* $\models$ *onl* Γ_{AODV} *?P")*
⟨*proof*⟩

Proposition 7.4

lemma *known_destinations_increase:*
"paodv i $\models_A$ *onll* Γ_{AODV} *(λ((ξ, _), _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ'))"*
⟨*proof*⟩

Proposition 7.5

lemma *rreqs_increase:*

```
"paodv i ⊨_A onll Γ_AODV (λ((ξ, _), _, (ξ', _)). rreqs ξ ⊆ rreqs ξ')"
```
⟨*proof*⟩

lemma *dests_bigger_than_sqn:*
```
"paodv i ⊨ onl Γ_AODV (λ(ξ, l). l ∈ {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 Γ_AODV (λ((ξ, _), _, (ξ', _)). ∀ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip)"
```
⟨*proof*⟩

Proposition 7.7

lemma *ip_constant:*
```
"paodv i ⊨ onl Γ_AODV (λ(ξ, _). ip ξ = i)"
```
⟨*proof*⟩

Proposition 7.8

lemma *sender_ip_valid':*
```
"paodv i ⊨_A onll Γ_AODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = ip ξ) a)"
```
⟨*proof*⟩

lemma *sender_ip_valid:*
```
"paodv i ⊨_A onll Γ_AODV (λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = i) a)"
```
⟨*proof*⟩

lemma *received_msg_inv:*
```
"paodv i ⊨ (recvmsg P →) onl Γ_AODV (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 Γ_AODV (λ(ξ, _). sip ξ ≠ ip ξ)"
```
⟨*proof*⟩

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

lemma *rreq_dip_in_vD_dip_eq_ip:*
```
"paodv i ⊨ onl Γ_AODV (λ(ξ, l). (l ∈ {PRreq-:18..PRreq-:21} ⟶ dip ξ ∈ vD(rt ξ))
                          ∧ (l ∈ {PRreq-:6, PRreq-:7} ⟶ dip ξ = ip ξ)
                          ∧ (l ∈ {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 Γ_AODV (λ(_, a, _). anycast msg_zhops a)"
```
⟨*proof*⟩

lemma *hop_count_zero_oip_dip_sip:*
```
"paodv i ⊨ (recvmsg msg_zhops →) onl Γ_AODV (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 Γ_AODV (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 Γ_AODV (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 Γ_AODV (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 Γ_AODV (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 Γ_AODV (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*⟩

Proposition 7.12

lemma *zero_seq_unk_hops_one':*
```
"paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (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 Γ_AODV (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 Γ_AODV (λ(ξ, l).
                          ∀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 Γ_AODV (λ(_, a, _). anycast rreq_rrep_sn a)"
```
⟨*proof*⟩

Proposition 7.14

lemma *rreq_rrep_fresh_any_step_invariant:*
"paodv i $\Vdash_A$ *onll* Γ_{AODV} *(λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a)"*
⟨*proof*⟩

Proposition 7.15

lemma *rerr_invalid_any_step_invariant:*
"paodv i $\Vdash_A$ *onll* Γ_{AODV} *(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* ⇀ *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 $\Vdash$
onl Γ_{AODV} *(λ(ξ, l). (l ∈ {PAodv-:15, PPkt-:7, PRreq-:11, PRreq-:24, 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 $\Vdash_A$ *(recvmsg rreq_rrep_sn →) onll* Γ_{AODV} *(λ((ξ, _), _, (ξ', _)).*
∀dip∈kD(rt ξ). rt ξ $\sqsubseteq_{dip}$ *rt ξ')"*
⟨*proof*⟩

end

4.7 The quality increases predicate

theory *D_Quality_Increases*
imports *D_Aodv_Predicates D_Fresher*
begin

definition *quality_increases :: "state ⇒ state ⇒ bool"*
where *"quality_increases ξ ξ' ≡ (∀dip∈kD(rt ξ). dip ∈ kD(rt ξ') ∧ rt ξ* $\sqsubseteq_{dip}$ *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* $\sqsubseteq_{dip}$ *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* $\sqsubseteq_{dip}$ *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 **: "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))

```
                                    ∧ 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 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 "σ_{AODV}' ≡ {(λi. aodv_init i, Γ_{AODV} PAodv)}"

abbreviation *opaodv*
  :: "ip ⇒ ((ip ⇒ state) × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
  "opaodv i ≡ ⟪ init = σ_{AODV}', trans = oseqp_sos Γ_{AODV} i ⟫"

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

lemma *oaodv_control_within [simp]*: "control_within Γ_{AODV} (init (opaodv i))"
  ⟨*proof*⟩

lemma σ_{AODV}'_labels [simp]: "(σ, p) ∈ σ_{AODV}' ⟹ labels Γ_{AODV} p = {PAodv-:0}"
  ⟨*proof*⟩

lemma *oaodv_init_kD_empty [simp]*:
  "(σ, p) ∈ σ_{AODV}' ⟹ kD (rt (σ i)) = {}"
  ⟨*proof*⟩

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

lemma *oaodv_trans*: "trans (opaodv i) = oseqp_sos Γ_{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 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 Γ_{AODV} (λ(σ, l). l ∈ {PAodv-:1} ⟶ P σ (msg (σ i)))"
  ⟨*proof*⟩

(Equivalent to) Proposition 7.27

lemma *local_quality_increases:*
"paodv i $\Vdash_A$ *(recvmsg rreq_rrep_sn →) onll* Γ_{AODV} *(λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ')"*
⟨*proof*⟩

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

lemma *oquality_increases:*
"opaodv i $\models_A$ *(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 *"_* $\models_A$ *(?S, _ →) _")*
⟨*proof*⟩

lemma *rreq_rrep_nsqn_fresh_any_step_invariant:*
"opaodv i $\models_A$ *(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 ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} →)
onl Γ_{AODV} *(λ(σ, l). l ∈ {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} →)
onll Γ_{AODV} *(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 Γ_{AODV} *(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 Γ_{AODV} *(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 *"_ ⊨ (?S, ?U →) _")*
⟨*proof*⟩

lemma *odsn_rrep:*
"opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} →)
onl Γ_{AODV} *(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 Γ_{AODV} *(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 Γ_AODV (λ(σ, l).
                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 Γ_AODV (λ(ξ, l). (l = PRerr-:1
                    ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ)
                                        ∧ the (nhop (rt ξ) ip) = sip ξ
                                        ∧ sqn (rt ξ) ip < the (dests ξ ip))))"
```
⟨*proof*⟩

lemmas *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 Γ_AODV (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 Γ_AODV (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 ⇒ 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]:*
"⋀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)"
```
⟨*proof*⟩

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*⟩

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*⟩

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_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 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, nhip)* where *dsn* is the 'destination sequence number', *dsk* is the 'destination-sequence-number status', *flag* is the route status, *hops* is the number of hops to the destination, and *nhip* is the next hop toward the destination.

type_synonym *r = "sqn × k × f × nat × ip"*

definition *proj2 :: "r ⇒ sqn" (‹π_2›)*
  where *"π_2 ≡ λ(dsn, _, _, _, _). dsn"*

definition *proj3 :: "r ⇒ k" (‹π_3›)*
  where *"π_3 ≡ λ(_, dsk, _, _, _). dsk"*

definition *proj4 :: "r ⇒ f" (‹π_4›)*
  where *"π_4 ≡ λ(_, _, flag, _, _). flag"*

definition *proj5 :: "r ⇒ nat" (‹π_5›)*
  where *"π_5 ≡ λ(_, _, _, hops, _). hops"*

definition *proj6 :: "r ⇒ ip" (‹π_6›)*

where "$\pi_6 \equiv \lambda(\_, \_, \_, \_, nhip).\ nhip$"

lemma *projs [simp]:*
"π_2(dsn, dsk, flag, hops, nhip) = dsn"
"π_3(dsn, dsk, flag, hops, nhip) = dsk"
"π_4(dsn, dsk, flag, hops, nhip) = flag"
"π_5(dsn, dsk, flag, hops, nhip) = hops"
"π_6(dsn, dsk, flag, hops, nhip) = nhip"
⟨*proof*⟩

lemma *proj3_pred [intro]:* "⟦ P kno; P unk ⟧ ⟹ P (π_3 x)"
⟨*proof*⟩

lemma *proj4_pred [intro]:* "⟦ P val; P inv ⟧ ⟹ P (π_4 x)"
⟨*proof*⟩

lemma *proj6_pair_snd [simp]:*
fixes *dsn' r*
shows "π_6 (dsn', snd (r)) = π_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" (‹σ_{route}'(_, _')›)

translations
"σ_{route}(rt, dip)" => "rt dip"

definition *sqn :: "rt ⇒ ip ⇒ sqn"*
where *"sqn rt dip ≡ case σ_{route}(rt, dip) of Some r ⇒ π_2(r) | None ⇒ 0"*

definition *sqnf :: "rt ⇒ ip ⇒ k"*
where *"sqnf rt dip ≡ case σ_{route}(rt, dip) of Some r ⇒ π_3(r) | None ⇒ unk"*

abbreviation *flag :: "rt ⇒ ip ⇀ f"*
where *"flag rt dip ≡ map_option π_4 (σ_{route}(rt, dip))"*

abbreviation *dhops :: "rt ⇒ ip ⇀ nat"*
where *"dhops rt dip ≡ map_option π_5 (σ_{route}(rt, dip))"*

abbreviation *nhop :: "rt ⇒ ip ⇀ ip"*
where *"nhop rt dip ≡ map_option π_6 (σ_{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.*
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) ⟹ π_5(the (rt dip)) = the (dhops rt dip)"
⟨*proof*⟩

lemma *proj4_eq_flag:* "⋀dip rt. dip∈kD(rt) ⟹ π_4(the (rt dip)) = the (flag rt dip)"
⟨*proof*⟩

lemma *proj2_eq_sqn:* "⋀dip rt. dip∈kD(rt) ⟹ π_2(the (rt dip)) = sqn rt dip"
⟨*proof*⟩

lemma *kD_sqnf_is_proj3 [simp]:*
"⋀ip rt. ip∈kD(rt) ⟹ sqnf rt ip = π_3(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*⟩

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 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')"
"⟦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* = π_2*(r); the (flag rt ip) = inv*⟧
⟹ *P (rt (ip* ↦ *r))"*
and *c5:* "⟦*ip* ∈ *kD(rt);* π_3*(r) = unk*⟧
⟹ *P (rt (ip* ↦ *(*π_2*(the* σ_{route}*(rt, ip)),* π_3*(r),*
π_4*(r),* π_5*(r),* π_6*(r))))"*
and *c6:* "⟦*ip* ∈ *kD(rt); sqn rt ip* ≥ π_2*(r);* π_3*(r) = kno;*
sqn rt ip = π_2*(r)* ⟹ *the (dhops rt ip)* ≤ π_5*(r)* ∧ *the (flag rt ip) = val*⟧
⟹ *P (rt (ip* ↦ *the* σ_{route}*(rt, ip)))"*
shows *"(P (update rt ip r))"*
⟨*proof*⟩

lemma *update_cases_kD:*
assumes *"(*π_2*(r) = 0) = (*π_3*(r) = unk)"*
and *"ip* ∈ *kD(rt)"*
and *c2: "sqn rt ip <* π_2*(r)* ⟹ *P (rt (ip* ↦ *r))"*
and *c3:* "⟦*sqn rt ip* = π_2*(r); the (dhops rt ip) >* π_5*(r)*⟧
⟹ *P (rt (ip* ↦ *r))"*
and *c4:* "⟦*sqn rt ip* = π_2*(r); the (flag rt ip) = inv*⟧
⟹ *P (rt (ip* ↦ *r))"*
and *c5: "*π_3*(r) = unk* ⟹ *P (rt (ip* ↦ *(*π_2*(the* σ_{route}*(rt, ip)),* π_3*(r),*
π_4*(r),* π_5*(r),* π_6*(r))))"*
and *c6:* "⟦*sqn rt ip* ≥ π_2*(r);* π_3*(r) = kno;*
sqn rt ip = π_2*(r)* ⟹ *the (dhops rt ip)* ≤ π_5*(r)* ∧ *the (flag rt ip) = val*⟧
⟹ *P (rt (ip* ↦ *the* σ_{route}*(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 (*λ*x. if x = rip then Some v else rt x) ip*
= (if ip = rip then π_2*(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]: "⋀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*
assumes *"rt = update rt dip (dsn, dsk, flag, hops, nhip)"*
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*
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]:*
"⋀dip rt dip' dsn dsk hops nhip.
dip ∈ vD(update rt dip' (dsn, dsk, val, hops, nhip)) ⟹ (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), Some rsn) ⇒
Some (rsn, dsk, inv, hops, nhip)"

lemma *proj3_invalidate [simp]:*
"⋀dip. π₃(the ((invalidate rt dests) dip)) = π₃(the (rt dip))"
⟨*proof*⟩

lemma *proj5_invalidate [simp]:*
"⋀dip. π₅(the ((invalidate rt dests) dip)) = π₅(the (rt dip))"
⟨*proof*⟩

lemma *proj6_invalidate [simp]:*
"⋀dip. π6(the ((invalidate rt dests) dip)) = π6(the (rt dip))"
⟨*proof*⟩

5.1.5 Route Requests

lemma *invalidate_kD_inv [simp]:*
"⋀rt dests. kD (invalidate rt dests) = kD rt"
⟨*proof*⟩

lemma *invalidate_sqn:*
fixes *rt dip dests*
assumes *"∀rsn. dests dip = Some rsn ⟶ sqn rt dip ≤ rsn"*
shows *"sqn rt dip ≤ sqn (invalidate rt dests) dip"*
⟨*proof*⟩

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

lemma *dhops_invalidate [simp]:*
"⋀dip. the (dhops (invalidate rt dests) dip) = the (dhops rt dip)"
⟨*proof*⟩

lemma *sqnf_invalidate [simp]:*
"⋀dip. sqnf (invalidate (rt ξ) (dests ξ)) dip = sqnf (rt ξ) dip"
⟨*proof*⟩

lemma *nhop_invalidate [simp]:*
"⋀dip. the (nhop (invalidate (rt ξ) (dests ξ)) dip) = the (nhop (rt ξ) dip)"
⟨*proof*⟩

lemma *invalidate_other [simp]:*
fixes *rt dests dip*
assumes *"dip∉dom(dests)"*
shows *"invalidate rt dests dip = rt dip"*
⟨*proof*⟩

lemma *invalidate_none [simp]:*
fixes *rt dests dip*
assumes *"dip∉kD(rt)"*
shows *"invalidate rt dests dip = None"*
⟨*proof*⟩

lemma *vD_invalidate_vD_not_dests:*
"⋀dip rt dests. dip∈vD(invalidate rt dests) ⟹ dip∈vD(rt) ∧ dests dip = None"
⟨*proof*⟩

lemma *sqn_invalidate_not_in_dests [simp]:*
fixes *dests dip rt*
assumes *"dip∉dom(dests)"*
shows *"sqn (invalidate rt dests) dip = sqn rt dip"*
⟨*proof*⟩

lemma *invalidate_changes:*
fixes *rt dests dip dsn dsk flag hops nhip pre*
assumes *"invalidate rt dests dip = Some (dsn, dsk, flag, hops, nhip)"*
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*⟩

5.1.6 Queued Packets

Functions for sending data packets.

type_synonym *store* = "ip ⇀ (p × data list)"

definition *sigma_queue* :: "store ⇒ ip ⇒ data list" (‹σ_{queue}'(_, _')›)
where "σ_{queue}(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" (‹$\sigma_{p\text{-}flag}$'(_, _')›)
where "$\sigma_{p\text{-}flag}$(store, dip) ≡ map_option fst (store dip)"

definition *unsetRRF* :: "store ⇒ ip ⇒ store"
where "unsetRRF store dip ≡ case store dip of
None ⇒ store
| Some (p, q) ⇒ store (dip ↦ (noreq, q))"

definition *setRRF* :: "store ⇒ (ip ⇀ sqn) ⇒ store"
where "setRRF store dests ≡ λdip. if dests dip = None then store dip
else map_option (λ(_, q). (req, q)) (store dip)"

5.1.7 Comparison with the original technical report

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

1. *nhop* is partial, thus a '*the*' is needed, similarly for *dhops* and *addpreRT*.
2. *precs* is partial.
3. $\sigma_{p\text{-}flag}$*(store, dip)* is partial.
4. The routing table (*rt*) is modelled as a map (*ip ⇒ 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

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)"*

⟨*proof*⟩

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

lemma *pkt_simp [simp]:*
"pkt(d, dip, sip) = Pkt d dip sip"
⟨*proof*⟩

end

5.3 The AODV protocol

theory *E_Aodv*
imports *E_Aodv_Data E_Aodv_Message*
AWN.AWN_SOS_Labels AWN.AWN_Invariants
begin

5.3.1 Data state

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

msg :: "msg"
data :: "data"
dests :: "ip ⇀ sqn"

dip :: "ip"
oip :: "ip"
hops :: "nat"
dsn :: "sqn"
dsk :: "k"
osn :: "sqn"
sip :: "ip"
handled:: "bool"

abbreviation *aodv_init :: "ip ⇒ state"*
where *"aodv_init i ≡ (|*
ip = i,
sn = 1,
rt = 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 *"ξ' ∈ is_rrep ξ"*
shows *"(∃hops' dip' dsn' oip' sip'.*
msg ξ = Rrep hops' dip' dsn' oip' sip' ∧
ξ' = ξ(| hops := hops', dip := dip', dsn := dsn', oip := oip', sip := sip' |))"
⟨*proof*⟩

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

lemma *is_rerr_asm [dest!]:*
assumes *"ξ' ∈ is_rerr ξ"*
shows *"(∃dests' sip'.*
msg ξ = Rerr dests' sip' ∧
ξ' = ξ(| dests := dests', sip := sip' |))"
⟨*proof*⟩

lemmas *is_msg_defs =*
is_rerr_def is_rrep_def is_rreq_def is_pkt_def is_newpkt_def

lemma *is_msg_inv_ip [simp]:*
"ξ' ∈ is_rerr ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_rrep ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_rreq ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_pkt ξ ⟹ ip ξ' = ip ξ"
"ξ' ∈ is_newpkt ξ ⟹ ip ξ' = ip ξ"
⟨*proof*⟩

lemma *is_msg_inv_sn [simp]:*
"ξ' ∈ is_rerr ξ ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_rrep ξ ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_rreq ξ ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_pkt ξ ⟹ sn ξ' = sn ξ"
"ξ' ∈ is_newpkt ξ ⟹ sn ξ' = sn ξ"
⟨*proof*⟩

lemma *is_msg_inv_rt [simp]:*
"ξ' ∈ is_rerr ξ ⟹ rt ξ' = rt ξ"
"ξ' ∈ is_rrep ξ ⟹ rt ξ' = rt ξ"
"ξ' ∈ is_rreq ξ ⟹ rt ξ' = rt ξ"
"ξ' ∈ is_pkt ξ ⟹ rt ξ' = rt ξ"
"ξ' ∈ is_newpkt ξ ⟹ rt ξ' = rt ξ"
⟨*proof*⟩

lemma *is_msg_inv_rreqs [simp]:*
"ξ' ∈ is_rerr ξ ⟹ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_rrep ξ ⟹ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_rreq ξ ⟹ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_pkt ξ ⟹ rreqs ξ' = rreqs ξ"
"ξ' ∈ is_newpkt ξ ⟹ rreqs ξ' = rreqs ξ"
⟨*proof*⟩

lemma *is_msg_inv_store [simp]:*
"ξ' ∈ is_rerr ξ ⟹ store ξ' = store ξ"
"ξ' ∈ is_rrep ξ ⟹ store ξ' = store ξ"
"ξ' ∈ is_rreq ξ ⟹ store ξ' = store ξ"
"ξ' ∈ is_pkt ξ ⟹ store ξ' = store ξ"
"ξ' ∈ is_newpkt ξ ⟹ store ξ' = store ξ"
⟨*proof*⟩

lemma *is_msg_inv_sip [simp]:*
"ξ' ∈ is_pkt ξ ⟹ sip ξ' = sip ξ"
"ξ' ∈ is_newpkt ξ ⟹ sip ξ' = sip ξ"

⟨*proof*⟩

5.3.3 The protocol process

```
datatype pseqp =
    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 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 Γ_AODV :: "(state, msg, pseqp, pseqp label) seqp_env"
where
  "Γ_AODV 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(σ_queue(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 (σ_p-flag(store ξ, dip)) = req }⟩
         ⟦ξ. ξ (| store := unsetRRF (store ξ) (dip ξ) |)⟧
         ⟦ξ. ξ (| sn := inc (sn ξ) |)⟧
         ⟦ξ. ξ (| rreqs := rreqs ξ ∪ {(ip ξ, sn ξ)} |)⟧
         broadcast(λξ. rreq(0, dip ξ, sqn (rt ξ) (dip ξ), sqnf (rt ξ) (dip ξ), ip ξ, sn ξ,
                            ip ξ, False)). AODV())"

|  "Γ_AODV 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())"

| "Γ_AODV PPkt = labelled PPkt (
     ⟨ξ. dip ξ = ip ξ⟩
       deliver(λξ. data ξ).AODV()
     ⊕ ⟨ξ. dip ξ ≠ ip ξ⟩
     (
       ⟨ξ. dip ξ ∈ vD (rt ξ)⟩
         unicast(λξ. the (nhop (rt ξ) (dip ξ)), λξ. pkt(data ξ, dip ξ, oip ξ)).AODV()
         ▷
           ⟦ξ. ξ (| dests := (λrip. if (rip ∈ vD (rt ξ) ∧ nhop (rt ξ) rip = nhop (rt ξ) (dip ξ))
                                    then Some (inc (sqn (rt ξ) rip)) else None) |)⟧
           ⟦ξ. ξ (| rt := invalidate (rt ξ) (dests ξ) |)⟧
           ⟦ξ. ξ (| store := setRRF (store ξ) (dests ξ)|)⟧
           broadcast(λξ. rerr(dests ξ, ip ξ)).AODV()
       ⊕ ⟨ξ. dip ξ ∉ vD (rt ξ)⟩
       (
           ⟨ξ. dip ξ ∈ iD (rt ξ)⟩
             broadcast(λξ. rerr([dip ξ ↦ sqn (rt ξ) (dip ξ)], ip ξ)). AODV()
```

```
          ⊕ ⟨ξ. dip ξ ∉ iD (rt ξ)⟩
             AODV()
      )
    ))"

| "Γ_AODV 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()
    ))"

| "Γ_AODV 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()
        )
      )
    )"

| "Γ_AODV 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_{AODV}.simps [simp del, code del]
lemmas Γ_{AODV}_simps [simp, code] = Γ_{AODV}.simps [simplified]

fun Γ_{AODV}_skeleton
where

```
    "Γ_AODV_skeleton PAodv   = seqp_skeleton (Γ_AODV PAodv)"
  | "Γ_AODV_skeleton PNewPkt = seqp_skeleton (Γ_AODV PNewPkt)"
  | "Γ_AODV_skeleton PPkt    = seqp_skeleton (Γ_AODV PPkt)"
  | "Γ_AODV_skeleton PRreq   = seqp_skeleton (Γ_AODV PRreq)"
  | "Γ_AODV_skeleton PRrep   = seqp_skeleton (Γ_AODV PRrep)"
  | "Γ_AODV_skeleton PRerr   = seqp_skeleton (Γ_AODV PRerr)"
```

lemma Γ_{AODV}_skeleton_wf [simp]:
  "wellformed Γ_{AODV}_skeleton"
  ⟨*proof*⟩

declare Γ_{AODV}_skeleton.simps [simp del, code del]
lemmas Γ_{AODV}_skeleton_simps [simp, code]
  = Γ_{AODV}_skeleton.simps [simplified Γ_{AODV}_simps seqp_skeleton.simps]

lemma aodv_proc_cases [dest]:
  fixes p pn
  shows "p ∈ ctermsl (Γ_AODV pn) ⟹

```
                                (p ∈ ctermsl (Γ_AODV PAodv) ∨
                                 p ∈ ctermsl (Γ_AODV PNewPkt)  ∨
                                 p ∈ ctermsl (Γ_AODV PPkt)  ∨
                                 p ∈ ctermsl (Γ_AODV PRreq) ∨
                                 p ∈ ctermsl (Γ_AODV PRrep) ∨
                                 p ∈ ctermsl (Γ_AODV PRerr))"
```

  ⟨*proof*⟩

definition σ_{AODV} :: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp) set"
where "σ_{AODV} i ≡ {(aodv_init i, Γ_AODV PAodv)}"

abbreviation paodv
  :: "ip ⇒ (state × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
  "paodv i ≡ (| init = σ_{AODV} i, trans = seqp_sos Γ_AODV |)"

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

lemma *aodv_control_within [simp]: "control_within* Γ_{AODV} *(init (paodv i))"*
⟨*proof*⟩

lemma *aodv_wf [simp]:*
"wellformed Γ_{AODV}*"*
⟨*proof*⟩

lemmas *aodv_labels_not_empty [simp] = labels_not_empty [OF aodv_wf]*

lemma *aodv_ex_label [intro]: "∃l. l∈labels* Γ_{AODV} *p"*
⟨*proof*⟩

lemma *aodv_ex_labelE [elim]:*
assumes *"∀l∈labels* Γ_{AODV} *p. P l p"*
and *"∃p l. P l p ⟹ Q"*
shows *"Q"*
⟨*proof*⟩

lemma *aodv_simple_labels [simp]: "simple_labels* Γ_{AODV}*"*
⟨*proof*⟩

lemma σ_{AODV}*_labels [simp]: "(ξ, p) ∈* σ_{AODV} *i ⟹ labels* Γ_{AODV} *p = {PAodv-:0}"*
⟨*proof*⟩

lemma *aodv_init_kD_empty [simp]:*
"(ξ, p) ∈ σ_{AODV} *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) ∈* σ_{AODV} *i"*
shows *"sip ξ ≠ ip ξ"*
⟨*proof*⟩

lemma *aodv_init_sip_not_i [simp]:*
assumes *"(ξ, p) ∈* σ_{AODV} *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
Γ_{AODV}*_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

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 ⇒ 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 dip dsn dsk oip osn sip handled.
msg_sender (Rreq hops dip dsn dsk oip osn sip handled) = sip"
"⋀hops dip dsn oip sip. msg_sender (Rrep hops dip dsn oip sip) = sip"
"⋀dests sip. msg_sender (Rerr dests sip) = sip"
"⋀d dip sip. msg_sender (Pkt d dip sip) = sip"
⟨*proof*⟩

definition *msg_zhops :: "msg ⇒ bool"*
where *"msg_zhops m ≡ case m of*
Rreq hopsc dipc _ _ oipc _ sipc _ ⇒ hopsc = 0 ⟶ oipc = sipc
| Rrep hopsc dipc _ _ sipc ⇒ hopsc = 0 ⟶ dipc = sipc
| _ ⇒ True"

lemma *msg_zhops_simps [simp]:*
"⋀hops dip dsn dsk oip osn sip handled.
msg_zhops (Rreq hops dip dsn dsk oip osn sip handled) = (hops = 0 ⟶ oip = sip)"
"⋀hops dip dsn oip sip. msg_zhops (Rrep hops dip dsn oip sip) = (hops = 0 ⟶ dip = sip)"
"⋀dests sip. msg_zhops (Rerr dests sip) = True"
"⋀d dip. msg_zhops (Newpkt d dip) = True"
"⋀d dip sip. msg_zhops (Pkt d dip sip) = True"
⟨*proof*⟩

definition *rreq_rrep_sn :: "msg ⇒ bool"*
where *"rreq_rrep_sn m ≡ case m of Rreq _ _ _ _ _ osnc _ _ ⇒ osnc ≥ 1*
| Rrep _ _ dsnc _ _ ⇒ dsnc ≥ 1
| _ ⇒ True"

lemma *rreq_rrep_sn_simps [simp]:*
"⋀hops dip dsn dsk oip osn sip handled.
rreq_rrep_sn (Rreq hops 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 [simp]:*
"⋀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
  nsqn$_r$ :: "r ⇒ sqn"
where
  "nsqn$_r$ r ≡ if π$_4$(r) = val ∨ π$_2$(r) = 0 then π$_2$(r) else (π$_2$(r) - 1)"

lemma *nsqnr_def':*
  "nsqn$_r$ r = (if π$_4$(r) = inv then π$_2$(r) - 1 else π$_2$(r))"
  ⟨*proof*⟩

lemma *nsqn$_r$_zero [simp]:*
  "⋀dsn dsk flag hops nhip. nsqn$_r$ (0, dsk, flag, hops, nhip) = 0"
  ⟨*proof*⟩

lemma *nsqn$_r$_val [simp]:*
  "⋀dsn dsk hops nhip. nsqn$_r$ (dsn, dsk, val, hops, nhip) = dsn"
  ⟨*proof*⟩

lemma *nsqn$_r$_inv [simp]:*
  "⋀dsn dsk hops nhip. nsqn$_r$ (dsn, dsk, inv, hops, nhip) = dsn - 1"
  ⟨*proof*⟩

lemma *nsqn$_r$_lte_dsn [simp]:*
  "⋀dsn dsk flag hops nhip. nsqn$_r$ (dsn, dsk, flag, hops, nhip) ≤ 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:*
  "⋀rt dip. nsqn rt dip = (if flag rt dip = Some val ∨ sqn rt dip = 0
  then sqn rt dip else sqn rt dip - 1)"
  ⟨*proof*⟩

lemma *not_in_kD_nsqn [simp]:*
  assumes *"dip ∉ kD(rt)"*
  shows *"nsqn rt dip = 0"*
  ⟨*proof*⟩

lemma *kD_nsqn:*
  assumes *"dip ∈ kD(rt)"*
    shows *"nsqn rt dip = nsqn$_r$(the (σ$_{route}$(rt, dip)))"*
  ⟨*proof*⟩

lemma *nsqnr_r_flag_pred [simp, intro]:*
    fixes *dsn dsk flag hops nhip pre*
  assumes *"P (nsqn$_r$ (dsn, dsk, val, hops, nhip))"*
      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 pre 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*⟩

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' ∧ π$_5$(r) ≥ π$_5$(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 *"π$_5$(r) ≥ π$_5$(r')"*
shows *"r ⊑ r'"*
⟨*proof*⟩

lemma *fresherI [intro]:*
assumes *"(nsqn$_r$ r < nsqn$_r$ r') ∨ (nsqn$_r$ r = nsqn$_r$ r' ∧ π$_5$(r) ≥ π$_5$(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' ∧ π$_5$(r) ≥ π$_5$(r') ⟹ P r r'"*
shows *"P r r'"*
⟨*proof*⟩

lemma *fresher_refl [simp]: "r ⊑ r"*
⟨*proof*⟩

lemma *fresher_trans [elim, trans]:*
"⟦ x ⊑ y; y ⊑ z ⟧ ⟹ x ⊑ z"
⟨*proof*⟩

lemma *not_fresher_trans [elim, trans]:*
"⟦ ¬(x ⊑ y); ¬(z ⊑ x) ⟧ ⟹ ¬(z ⊑ y)"
⟨*proof*⟩

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

5.5.3 Comparing routing tables

definition
rt_fresher :: "ip ⇒ rt ⇒ rt ⇒ bool"
where
"rt_fresher ≡ λdip rt rt'. (the (σ$_{route}$(rt, dip))) ⊑ (the (σ$_{route}$(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':*
"(rt$_1$ ⊑$_i$ rt$_2$) = (nsqn$_r$ (the (rt$_1$ i)) < nsqn$_r$ (the (rt$_2$ i)) ∨
nsqn$_r$ (the (rt$_1$ i)) = nsqn$_r$ (the (rt$_2$ i)) ∧ π$_5$ (the (rt$_2$ i)) ≤ π$_5$ (the (rt$_1$ 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_nsqnr:
assumes "dip ∈ kD(rt1)"
and "dip ∈ kD(rt2)"
and "rt1 ≈dip rt2"
shows "nsqn$_r$ (the (rt2 dip)) = nsqn$_r$ (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 ⇒ 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 $\sqsubset_i$ rt2"
and "⟦ rt1 $\sqsubseteq_i$ rt2; ¬(rt2 $\sqsubseteq_i$ rt1) ⟧ ⟹ P rt1 rt2 i"
shows "P rt1 rt2 i"
⟨*proof*⟩

lemma *rt_strictly_fresherI [intro]:*
assumes "rt1 $\sqsubseteq_i$ rt2"
and "¬(rt1 $\approx_i$ rt2)"
shows "rt1 $\sqsubset_i$ rt2"
⟨*proof*⟩

lemmas *rt_strictly_fresher_singleI [elim] = rt_strictly_fresherI [OF single_rt_fresher]*

lemma *rt_strictly_fresherE [elim]:*
assumes "rt1 $\sqsubset_i$ rt2"
and "⟦ rt1 $\sqsubseteq_i$ rt2; ¬(rt1 $\approx_i$ rt2) ⟧ ⟹ P rt1 rt2 i"
shows "P rt1 rt2 i"
⟨*proof*⟩

lemma *rt_strictly_fresher_def':*
"rt1 $\sqsubset_i$ 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 $\sqsubset_{dip}$ rt2"
shows "the (rt1 dip) $\sqsubseteq$ the (rt2 dip)"
⟨*proof*⟩

lemma *rt_strictly_fresher_not_fresh_asD [dest]:*
assumes "rt1 $\sqsubset_{dip}$ rt2"
shows "¬ rt1 $\approx_{dip}$ rt2"
⟨*proof*⟩

lemma *rt_strictly_fresher_trans [elim, trans]:*
assumes "rt1 $\sqsubset_{dip}$ rt2"
and "rt2 $\sqsubset_{dip}$ rt3"
shows "rt1 $\sqsubset_{dip}$ rt3"
⟨*proof*⟩

lemma *rt_strictly_fresher_irefl [simp]:* "¬ (rt $\sqsubset_{dip}$ rt)"
⟨*proof*⟩

lemma *rt_fresher_trans_rt_strictly_fresher [elim, trans]:*
assumes "rt1 $\sqsubset_{dip}$ rt2"
and "rt2 $\sqsubseteq_{dip}$ rt3"
shows "rt1 $\sqsubset_{dip}$ rt3"
⟨*proof*⟩

lemma *rt_fresher_trans_rt_strictly_fresher' [elim, trans]:*
assumes "rt1 $\sqsubseteq_{dip}$ rt2"
and "rt2 $\sqsubset_{dip}$ rt3"
shows "rt1 $\sqsubset_{dip}$ rt3"
⟨*proof*⟩

lemma *rt_fresher_imp_nsqn_le:*
assumes "rt1 $\sqsubseteq_{ip}$ rt2"
and "ip ∈ kD rt1"
and "ip ∈ kD rt2"
shows "nsqn rt1 ip ≤ nsqn rt2 ip"

⟨*proof*⟩

lemma *rt_strictly_fresher_ltI [intro]:*
assumes *"dip ∈ kD(rt1)"*
and *"dip ∈ kD(rt2)"*
and *"nsqn rt1 dip < nsqn rt2 dip"*
shows *"rt1 ⊏$_{dip}$ rt2"*
⟨*proof*⟩

lemma *rt_strictly_fresher_eqI [intro]:*
assumes *"i∈kD(rt1)"*
and *"i∈kD(rt2)"*
and *"nsqn rt1 i = nsqn rt2 i"*
and *"π_5(the (rt2 i)) < π_5(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 ⊨$_A$ onll Γ_{AODV} (λ((ξ, _), _, (ξ', _)). sn ξ ≤ sn ξ')"

⟨*proof*⟩

lemma *sequence_number_one_or_bigger:*
"paodv i ⊫ onl Γ_{AODV} *(λ(ξ, _). 1 ≤ sn ξ)"*
⟨*proof*⟩

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

lemma *sequence_number_increases':*
"paodv i $\Vdash_A$ *(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 Γ_{AODV} *(λ(ξ, l). l ∈ ({PAodv-:7} ∪ {PAodv-:5} ∪ {PRrep-:0..PRrep-:4}*
∪ {PRreq-:0..PRreq-:3}) ⟶ sip ξ ∈ kD (rt ξ))"
⟨*proof*⟩

Proposition 7.38

lemma *includes_nhip:*
"paodv i ⊫ onl Γ_{AODV} *(λ(ξ, l). ∀dip∈kD(rt ξ). the (nhop (rt ξ) dip)∈kD(rt ξ))"*
⟨*proof*⟩

Proposition 7.4

lemma *known_destinations_increase:*
"paodv i $\Vdash_A$ *onll* Γ_{AODV} *(λ((ξ, _), _, (ξ', _)). kD (rt ξ) ⊆ kD (rt ξ'))"*
⟨*proof*⟩

Proposition 7.5

lemma *rreqs_increase:*
"paodv i $\Vdash_A$ *onll* Γ_{AODV} *(λ((ξ, _), _, (ξ', _)). rreqs ξ ⊆ rreqs ξ')"*
⟨*proof*⟩

lemma *dests_bigger_than_sqn:*
"paodv i ⊫ onl Γ_{AODV} *(λ(ξ, l). l ∈ {PAodv-:15..PAodv-:17}*
∪ {PPkt-:7..PPkt-:9}
∪ {PRreq-:11..PRreq-:13}
∪ {PRreq-:20..PRreq-:22}
∪ {PRrep-:7..PRrep-:9}
∪ {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:*
"paodv i $\Vdash_A$ *onll* Γ_{AODV} *(λ((ξ, _), _, (ξ', _)). ∀ip. sqn (rt ξ) ip ≤ sqn (rt ξ') ip)"*
⟨*proof*⟩

Proposition 7.7

lemma *ip_constant:*
"paodv i ⊫ onl Γ_{AODV} *(λ(ξ, _). ip ξ = i)"*
⟨*proof*⟩

Proposition 7.8

lemma *sender_ip_valid':*
"paodv i $\Vdash_A$ *onll* Γ_{AODV} *(λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = ip ξ) a)"*
⟨*proof*⟩

lemma *sender_ip_valid:*
"paodv i $\Vdash_A$ *onll* Γ_{AODV} *(λ((ξ, _), a, _). anycast (λm. not_Pkt m ⟶ msg_sender m = i) a)"*

⟨*proof*⟩

lemma *received_msg_inv:*
"paodv i ⊨ (recvmsg P →) onl Γ_{AODV} (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 Γ_{AODV} (λ(ξ, _). sip ξ ≠ ip ξ)"
⟨*proof*⟩

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

lemma *rreq_dip_in_vD_dip_eq_ip:*
"paodv i ⊨ onl Γ_{AODV} (λ(ξ, l). (l ∈ {PRreq-:16..PRreq-:17} ⟶ dip ξ ∈ vD(rt ξ))
∧ (l ∈ {PRreq-:6, PRreq-:7} ⟶ dip ξ = ip ξ)
∧ (l ∈ {PRreq-:15..PRreq-:17} ⟶ dip ξ ≠ ip ξ))"
⟨*proof*⟩

lemma *rrep_dip_in_vD:*
"paodv i ⊨ onl Γ_{AODV} (λ(ξ, l). (l ∈ {PRrep-:4} ⟶ dip ξ ∈ vD(rt ξ)))"
⟨*proof*⟩

Proposition 7.11

lemma *anycast_msg_zhops:*
"⋀rreqid dip dsn dsk oip osn sip.
paodv i ⊨$_A$ onll Γ_{AODV} (λ(_, a, _). anycast msg_zhops a)"
⟨*proof*⟩

lemma *hop_count_zero_oip_dip_sip:*
"paodv i ⊨ (recvmsg msg_zhops →) onl Γ_{AODV} (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 Γ_{AODV} (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 Γ_{AODV} (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 Γ_{AODV} (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 Γ_{AODV} (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 Γ_AODV (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*⟩

Proposition 7.12

lemma *zero_seq_unk_hops_one'*:
```
"paodv i ⊨ (recvmsg (λm. rreq_rrep_sn m ∧ msg_zhops m) →) onl Γ_AODV (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 Γ_AODV (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 Γ_AODV (λ(ξ, l).
                          ∀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 Γ_AODV (λ(_, a, _). anycast rreq_rrep_sn a)"
```
⟨*proof*⟩

Proposition 7.14

lemma *rreq_rrep_fresh_any_step_invariant*:
```
"paodv i ⊨_A onll Γ_AODV (λ((ξ, _), a, _). anycast (rreq_rrep_fresh (rt ξ)) a)"
```
⟨*proof*⟩

Proposition 7.15

lemma *rerr_invalid_any_step_invariant*:
```
"paodv i ⊨_A onll Γ_AODV (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* ⇀ *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 Γ_AODV (λ(ξ, l). (l ∈ {PAodv-:15, PPkt-:7, PRreq-:11, PRreq-:20, PRrep-:7}
```

```
      ⟶ (∀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 Γ_AODV (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 *: "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 dip dsn dsk oip osn sip handled.
          msg_fresh σ (Rreq hops dip dsn dsk oip osn sip handled) =
                        (osn ≥ 1 ∧ (sip ≠ oip ⟶ oip∈kD(rt (σ sip))
                                ∧ nsqn (rt (σ sip)) oip ≥ osn
                                ∧ (nsqn (rt (σ sip)) oip = osn
                                      ⟶ (hops ≥ the (dhops (rt (σ sip)) oip)
                                           ∨ the (flag (rt (σ sip)) oip) = inv))))"
  "⋀hops dip dsn oip sip. msg_fresh σ (Rrep hops dip dsn oip sip) =
                        (dsn ≥ 1 ∧ (sip ≠ dip ⟶ dip∈kD(rt (σ sip))
                                ∧ nsqn (rt (σ sip)) dip ≥ dsn
                                ∧ (nsqn (rt (σ sip)) dip = dsn
                                      ⟶ (hops ≥ the (dhops (rt (σ sip)) dip))
                                           ∨ the (flag (rt (σ sip)) dip) = inv)))"
  "⋀dests sip.            msg_fresh σ (Rerr dests sip) =
                         (∀ripc∈dom(dests). (ripc∈kD(rt (σ sip))
                                ∧ the (dests ripc) - 1 ≤ nsqn (rt (σ sip)) ripc))"
  "⋀d dip.                msg_fresh σ (Newpkt d dip)   = True"
  "⋀d dip sip.            msg_fresh σ (Pkt d dip sip)  = True"
⟨proof⟩

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

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

lemma rreq_nsqn_is_fresh [simp]:
  fixes σ msg hops dip dsn dsk oip osn sip handled
  assumes "rreq_rrep_fresh (rt (σ sip)) (Rreq hops dip dsn dsk oip osn sip handled)"
```

and *"rreq_rrep_sn (Rreq hops dip dsn dsk oip osn sip handled)"*
shows *"msg_fresh 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 σ_{AODV}*' :: "((ip ⇒ state) × ((state, msg, pseqp, pseqp label) seqp)) set"*
where *"*σ_{AODV}*' ≡ {(λi. aodv_init i,* Γ_{AODV} *PAodv)}"*

abbreviation *opaodv*
:: "ip ⇒ ((ip ⇒ state) × (state, msg, pseqp, pseqp label) seqp, msg seq_action) automaton"
where
"opaodv i ≡ (| init = σ_{AODV}*', trans = oseqp_sos* Γ_{AODV} *i |)"*

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

lemma *oaodv_control_within [simp]: "control_within* Γ_{AODV} *(init (opaodv i))"*
⟨*proof*⟩

lemma σ_{AODV}*'_labels [simp]: "(σ, p) ∈* σ_{AODV}*' ⟹ labels* Γ_{AODV} *p = {PAodv-:0}"*
⟨*proof*⟩

lemma *oaodv_init_kD_empty [simp]:*
"(σ, p) ∈ σ_{AODV}*' ⟹ kD (rt (σ i)) = {}"*
⟨*proof*⟩

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

lemma *oaodv_trans: "trans (opaodv i) = oseqp_sos* Γ_{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

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 Γ_AODV (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 Γ_AODV (λ((ξ, _), _, (ξ', _)). quality_increases ξ ξ')"
⟨*proof*⟩

lemmas *olocal_quality_increases =*
open_seq_step_invariant [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 ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
other quality_increases {i} →)
onl Γ_AODV (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 Γ_AODV (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 Γ_AODV (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 Γ_AODV (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 "_ ⊨ (?S, ?U →) _")
```
⟨*proof*⟩

lemma *odsn_rrep:*
```
"opaodv i ⊨ (otherwith quality_increases {i} (orecvmsg msg_fresh),
             other quality_increases {i} →)
    onl Γ_AODV (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 Γ_AODV (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 Γ_AODV (λ(σ, l).
              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 Γ_AODV (λ(ξ, l). (l = PRerr-:1
                   ⟶ (∀ip∈dom(dests ξ). ip∈vD(rt ξ)
                                        ∧ the (nhop (rt ξ) ip) = sip ξ
                                        ∧ sqn (rt ξ) ip < the (dests ξ ip))))"
```
⟨*proof*⟩

lemmas *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 Γ_AODV (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 Γ_AODV (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

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 = (λdip.
     {(ip, ip') | ip ip' dsn dsk hops.
        ip ≠ dip ∧ rt (σ ip) dip = Some (dsn, dsk, val, hops, ip')})"
```

Given the state of a network 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. r dip = Some (dsn, dsk, val, hops, ip')))"
```
⟨*proof*⟩

lemma *rt_graph_vD [dest]:*
```
  "⋀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⟹ dip ∈ vD(rt (σ ip))"
```
⟨*proof*⟩

lemma *rt_graph_vD_trans [dest]:*
```
  "⋀ip ip' σ dip. (ip, ip') ∈ (rt_graph σ dip)⁺ ⟹ dip ∈ vD(rt (σ ip))"
```
⟨*proof*⟩

lemma *rt_graph_not_dip [dest]:*
```
  "⋀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⟹ ip ≠ dip"
```
⟨*proof*⟩

lemma *rt_graph_not_dip_trans [dest]:*
```
  "⋀ip ip' σ dip. (ip, ip') ∈ (rt_graph σ dip)⁺ ⟹ ip ≠ dip"
```
⟨*proof*⟩

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

lemma *rt_graph_nhip_is_nhop [dest]:*
```
  "⋀ip ip' σ dip. (ip, ip') ∈ rt_graph σ dip ⟹ ip' = the (nhop (rt (σ ip)) dip)"
```
⟨*proof*⟩

theorem *inv_to_loop_freedom:*
assumes "∀ *i dip. let nhip = the (nhop (rt (σ i)) dip)*
in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
⟶ (rt (σ i)) ⊏$_{dip}$ (rt (σ nhip))"
shows "∀ *dip. irrefl ((rt_graph σ dip)$^+$)*"
⟨*proof*⟩

end

5.11 Lift and transfer invariants to show loop freedom

theory *E_Aodv_Loop_Freedom*
imports *AWN.OClosed_Transfer AWN.Qmsg_Lifting E_Global_Invariants E_Loop_Freedom*
begin

5.11.1 Lift to parallel processes with queues

lemma *par_step_no_change_on_send_or_receive:*
fixes *σ s a σ' s'*
assumes "*((σ, s), a, (σ', s')) ∈ oparp_sos i (oseqp_sos 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)"
⟨*proof*⟩

5.11.2 Lift to nodes

lemma *node_step_no_change_on_send_or_receive:*
assumes "*((σ, NodeS i P R), a, (σ', NodeS i' P' R')) ∈ onode_sos*
(oparp_sos i (oseqp_sos Γ$_{AODV}$ i) (seqp_sos Γ$_{QMSG}$))"
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*⟩

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 ⊨*
(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*⟩

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_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 nhip = the (nhop (rt (σ i)) dip)
in dip ∈ vD (rt (σ i)) ∩ vD (rt (σ nhip)) ∧ nhip ≠ dip
⟶ (rt (σ i)) ⊏$_{dip}$ (rt (σ nhip)))"
(is *"_ ⊨ netglobal (λσ. ∀ i. ?inv σ i)")*
⟨*proof*⟩

5.11.6 Loop freedom of AODV

theorem *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.