LOFT —Verified Migration of Linux Firewalls to SDN

Julius Michaelis and Cornelius Diekmann

September 13, 2023

Abstract

We present LOFT — *Linux firewall OpenFlow Translator*, a system that transforms the main routing table and *FORWARD* chain of *iptables* of a Linux-based firewall into a set of static OpenFlow rules. Our implementation is verified against a model of a simplified Linux-based router and we can directly show how much of the original functionality is preserved.

Please note that this document is organized in two distinct parts. The first part contains the necessary definitions, helper lemmas and proofs in all their technicality as made in the theory code. The second part reiterates the most important definitions and proofs in a manner that is more suitable for human readers and enriches them with detailed explanations in natural language. Any interested reader should start from there.

Many of the considerations that have led to the definitions made here have been explained in [8].

Contents

I Code

II Documentation

1 Configuration Translation
   1.1 Linux Firewall Model ......................................................... 44
      1.1.1 Routing Table .............................................................. 45
      1.1.2 iptables Firewall ......................................................... 46
   1.2 OpenFlow Switch Model ...................................................... 46
      1.2.1 Matching Flow Table entries ............................................ 47
      1.2.2 Evaluating a Flow Table ............................................... 47
   1.3 Translation Implementation ................................................. 49
      1.3.1 Chaining Firewalls .......................................................... 49
      1.3.2 Translation Implementation ............................................. 50
      1.3.3 Comparison to Exodus .................................................... 52

2 Evaluation
   2.1 Mininet Examples ..................................................................... 52
   2.2 Performance Evaluation ......................................................... 54

3 Conclusion and Future Work ...................................................... 55
Part I
Code

theory OpenFlow-Matches
imports IP-Addresses,Prefix-Match
    Simple-Firewall,Simple-Packet
    HOL-Library.Monad-Syntax
    HOL-Library.List-Leqorder
    HOL-Library.Char-ord
begin

datatype of-match-field =
  IngressPort string
| EtherSrc 48 word
| EtherDst 48 word
| EtherType 16 word
| VlanId 16 word
| VlanPriority 16 word
| IPv4Src 32 prefix-match
| IPv4Dst 32 prefix-match
| IPv4Proto 8 word
| L4Src 16 word 16 word
| L4Dst 16 word 16 word

schematic-goal of-match-field-typeset: (field-match :: of-match-field) ∈ {
  IngressPort (s::string),
  EtherSrc (?as::48 word), EtherDst (?ad::48 word),
  EtherType (?t::16 word),
  VlanId (?i::16 word), VlanPriority (?p::16 word),
  IPv4Src (?pms::32 prefix-match),
  IPv4Dst (?pmd::32 prefix-match),
  IPv4Proto (?ipp::8 word),
  L4Src (?ps::16 word) (?ms::16 word),
  L4Dst (?pd::16 word) (?md::16 word)
}
proof((cases field-match;clarsimp;goal-simp),goal-cases)
  next case (IngressPort s) thus s = (case field-match of IngressPort s ⇒ s) unfolding IngressPort of-match-field.simps by rule
  next case (EtherSrc s) thus s = (case field-match of EtherSrc s ⇒ s) unfolding EtherSrc of-match-field.simps by rule
  next case (EtherDst s) thus s = (case field-match of EtherDst s ⇒ s) unfolding EtherDst of-match-field.simps by rule
  next case (EtherType s) thus s = (case field-match of EtherType s ⇒ s) unfolding EtherType of-match-field.simps by rule
  next case (VlanId s) thus s = (case field-match of VlanId s ⇒ s) unfolding VlanId of-match-field.simps by rule
  next case (VlanPriority s) thus s = (case field-match of VlanPriority s ⇒ s) unfolding VlanPriority of-match-field.simps by rule
next case (IPv4Src s) thus s = (case field-match of IPv4Src s ⇒ s) unfolding IPv4Src of-match-field.simps by rule
next case (IPv4Dst s) thus s = (case field-match of IPv4Dst s ⇒ s) by simp
next case (IPv4Proto s) thus s = (case field-match of IPv4Proto s ⇒ s) by simp
next case (L4Src p l) thus p = (case field-match of L4Src p m ⇒ p) ∧ l = (case field-match of L4Src p m ⇒ m) by simp
next case (L4Dst p l) thus p = (case field-match of L4Dst p m ⇒ p) ∧ l = (case field-match of L4Dst p m ⇒ m) by simp
qed

function prerequisites :: of-match-field ⇒ of-match-field set ⇒ bool where
  prerequisites (IngressPort -) = True |
  prerequisites (EtherDst -) = True |
  prerequisites (EtherSrc -) = True |
  prerequisites (EtherType -) = True |
  prerequisites (VlanId -) = (∃ id. let v = VlanId id in v ∈ m ∧ prerequisites v m) |
  prerequisites (VlanPriority -) m = (∃ id. let v = VlanId id in v ∈ m ∧ prerequisites v m) |
  prerequisites (IPv4Proto -) m = (let v = EtherType 0x0800 in v ∈ m ∧ prerequisites v m) |
  prerequisites (IPv4Src -) m = (let v = EtherType 0x0800 in v ∈ m ∧ prerequisites v m) |
  prerequisites (IPv4Dst -) m = (let v = EtherType 0x0800 in v ∈ m ∧ prerequisites v m) |
  prerequisites (L4Src -) m = (∃ proto ∈ {TCP, UDP, L4-Protocol.SCTP}. let v = IPv4Proto proto in v ∈ m ∧ prerequisites v m) |
  prerequisites (L4Dst -) m = prerequisites (L4Src undefined undefined) m by pat-completeness auto

fun match-sorter :: of-match-field ⇒ nat where
  match-sorter (IngressPort -) = 1 |
  match-sorter (VlanId -) = 2 |
  match-sorter (VlanPriority -) = 3 |
  match-sorter (EtherType -) = 4 |
  match-sorter (EtherSrc -) = 5 |
  match-sorter (EtherDst -) = 6 |
  match-sorter (IPv4Proto -) = 7 |
  match-sorter (IPv4Src -) = 8 |
  match-sorter (IPv4Dst -) = 9 |
  match-sorter (L4Src -) = 10 |
  match-sorter (L4Dst -) = 11

termination prerequisites by(relation measure (match-sorter o fst), simp-all)

definition less-eq-of-match-field1 :: of-match-field ⇒ of-match-field ⇒ bool
where less-eq-of-match-field1 \( (a::\text{of-match-field}) (b::\text{of-match-field}) \leftrightarrow (\text{case} \ (a, b) \ of \)
\(\text{IngressPort} \ a, \text{IngressPort} \ b) \Rightarrow a \leq b \mid \)
\(\text{VlanId} \ a, \text{VlanId} \ b) \Rightarrow a \leq b \mid \)
\(\text{EtherDst} \ a, \text{EtherDst} \ b) \Rightarrow a \leq b \mid \)
\(\text{EtherSrc} \ a, \text{EtherSrc} \ b) \Rightarrow a \leq b \mid \)
\(\text{EtherType} \ a, \text{EtherType} \ b) \Rightarrow a \leq b \mid \)
\(\text{IPv4Proto} \ a, \text{IPv4Proto} \ b) \Rightarrow a \leq b \mid \)
\(\text{IPv4Src} \ a, \text{IPv4Src} \ b) \Rightarrow a \leq b \mid \)
\(\text{IPv4Dst} \ a, \text{IPv4Dst} \ b) \Rightarrow a \leq b \mid \)
\(\text{IPv4Proto} \ a, \text{IPv4Proto} \ b \Rightarrow a \neq b \mid \)
\(\text{IPv4Src} \ a, \text{IPv4Src} \ b \Rightarrow a \neq b \mid \)
\(\text{IPv4Dst} \ a, \text{IPv4Dst} \ b \Rightarrow a \neq b \mid \)
\((L4\text{Src} \ a1 \ a2, \text{L4Src} \ b1 \ b2) \Rightarrow \text{if} \ a2 = b2 \ \text{then} \ a1 \leq b1 \ \text{else} \ a2 \leq b2 \mid \)
\((L4\text{Dst} \ a1 \ a2, \text{L4Dst} \ b1 \ b2) \Rightarrow \text{if} \ a2 = b2 \ \text{then} \ a1 \leq b1 \ \text{else} \ a2 \leq b2 \mid \)
\((a, b) \Rightarrow \text{match-sorter} \ a < \text{match-sorter} \ b)\)

instantiation of-match-field :: linorder
begin

definition less-eq-of-match-field \( (a::\text{of-match-field}) (b::\text{of-match-field}) \leftrightarrow \text{less-eq-of-match-field1} \ a \ b\)

definition less-of-match-field \( (a::\text{of-match-field}) (b::\text{of-match-field}) \leftrightarrow a \neq b \land \text{less-eq-of-match-field1} \ a \ b\)

instance

by standard (auto simp add: less-eq-of-match-field-def less-of-match-field-def less-eq-of-match-field1-def split: prod.splits of-match-field.splits if-splits)

definition match-no-prereq :: of-match-field \( \Rightarrow (32, 'a) \text{simple-packet-ext-scheme} \Rightarrow \text{bool} \) where
match-no-prereq \( \text{IngressPort} \ i \ p = (\text{p-iface} \ p = i) \mid \)
match-no-prereq \( \text{EtherDst} \ i \ p = (\text{p-l2src} \ p = i) \mid \)
match-no-prereq \( \text{EtherSrc} \ i \ p = (\text{p-l2dst} \ p = i) \mid \)
match-no-prereq \( \text{EtherType} \ i \ p = (\text{p-l2type} \ p = i) \mid \)
match-no-prereq \( \text{VlanId} \ i \ p = (\text{p-vlanid} \ p = i) \mid \)
match-no-prereq \( \text{VlanPriority} \ i \ p = (\text{p-vlanprio} \ p = i) \mid \)
match-no-prereq \( \text{IPv4Proto} \ i \ p = (\text{p-proto} \ p = i) \mid \)
match-no-prereq \( \text{IPv4Src} \ i \ p = (\text{p-src} \ p) \mid \)
match-no-prereq \( \text{IPv4Dst} \ i \ p = (\text{p-dst} \ p) \mid \)
match-no-prereq \( \text{L4Src} \ i \ m \ p = (\text{p-src} \ p \ &\& \ m = i) \mid \)
match-no-prereq \( \text{L4Dst} \ i \ m \ p = (\text{p-dst} \ p \ &\& \ m = i) \)

definition match-prereq :: of-match-field \( \Rightarrow of-match-field \ set \Rightarrow (32, 'a) \text{simple-packet-ext-scheme} \Rightarrow \text{bool} \) option where
match-prereq \( i \ s \ p = (\text{if} \ \text{prerequisites} \ i \ s \ \text{then} \ \text{Some} \ \text{match-no-prereq} \ i \ p \ \text{else} \ \text{None})\)

definition set-seq \( s \equiv (\forall x \in s. \ x \neq \text{None}) \ \text{then} \ \text{Some} \ (\text{the} \ \text{'} s \text{\'} \ \text{else} \ \text{None})\)
definition all-true \( s \equiv (\forall x \in s. \ x\) 
definition map-option

definition OF-match-fields :: of-match-field \( \Rightarrow (32, 'a) \text{simple-packet-ext-scheme} \Rightarrow \text{bool} \) option where
OF-match-fields \( m \ p = \text{map-option} \ \text{all-true} \ \text{(set-seq} \ ((\Lambda f. \ \text{match-prereq} \ f \ m) \ ' m))\)
definition OF-match-fields-unsafe :: of-match-field \( \Rightarrow (32, 'a) \text{simple-packet-ext-scheme} \Rightarrow \text{bool} \) where
OF-match-fields-unsafe \( m \ p = (\forall f \in m. \ \text{match-no-prereq} \ f \ p)\)
definition OF-match-fields-safe \( m \equiv \text{the} \ \text{of} \ \text{OF-match-fields} \ m\)
definition all-prerequisites \( m \equiv \forall f \in m. \text{prerequisites} f m \)

lemma
\[
\text{all-prerequisites} p \implies \\
L4\text{Src} x y \in p \implies \\
\text{IP}e3\text{Proto} \{ \text{TCP}, \text{UDP}, \text{L4-Protocol.SCTP} \} \cap p \neq \{ \}
\]

unfolding all-prerequisites-def by auto

lemma of-safe-unsafe-match-eq: all-prerequisites m \implies \text{OF-match-fields} m p = \text{Some} \ (\text{OF-match-fields-unsafe} m p)

unfolding \text{OF-match-fields-def} \text{OF-match-fields-unsafe-def} \text{comp-def} \text{set-seq-def} \text{match-prereq-def} \text{all-prerequisites-def}

proof goal-cases
  case 1
  have 2: \((\lambda f. \text{if} \ \text{prerequisites} f m \text{ then} \text{Some} (\text{match-no-prereq} f p) \text{ else} \text{None}) \ ^\prime \ m = (\lambda f. \text{Some} (\text{match-no-prereq} f p)) \ ^\prime \ m\)
    using 1 by fastforce
  have 3: \forall x \in (\lambda f. \text{Some} (\text{match-no-prereq} f p)) \ ^\prime \ m. \ x \neq \text{None} by blast
  show cease unsub
  unfolding 2 unfolding eqTrueI[OF 3] unfolding if-True unfolding image-comp comp-def unfolding option.sel by(simp add: all-true-def)
  qed

lemma of-match-fields-safe-eq: assumes all-prerequisites m shows \text{OF-match-fields-safe} m p \iff \text{OF-match-fields} m p = \text{Some} True

unfolding \text{OF-match-fields-safe-def}[abs-def] \text{fun-iff} \text{comp-def} \text{unfolding} \text{OF-match-fields-eq}[OF assms] \text{unfolding option.sel} by clarify

lemma \text{OF-match-fields-alt}: \text{OF-match-fields} m p = 
  \ (if \ \exists f \in m. \neg \text{prerequisites} f m \text{ then} \text{None} else \\
  \ \text{if} \ \forall f \in m. \text{match-no-prereq} f p \text{ then} \text{Some} True \text{ else} \text{Some} False)

unfolding \text{OF-match-fields-def} \text{all-true-def}[abs-def] \text{set-seq-def} \text{match-prereq-def}
by(auto simp add: ball-Un)

lemma of-match-fields-safe-eq2: assumes all-prerequisites m shows \text{OF-match-fields-safe} m p \iff \text{OF-match-fields} m p = \text{Some} True

unfolding \text{OF-match-fields-safe-def}[abs-def] \text{fun-iff} \text{comp-def} \text{unfolding} \text{OF-match-fields-eq}[OF assms] \text{unfolding option.sel} by simp

end

datatype of-action = Forward (oiface-sel: string) | ModifyField-l2dst 48 word

fun of-action-semantics where
of-action-semantics p [] = {} | of-action-semantics p (a#as) = (case a of
Forward i ⇒ insert (i,p) (of-action-semantics p as) |  
ModifyField-l2dst a ⇒ of-action-semantics (p[p-l2dst := a]) as)

value of-action-semantics p []
value of-action-semantics p [ModifyField-l2dst 66, Forward "oif"]

end
theory Semantics-OpenFlow
imports List-Group Sort-Descending
  IP-Addresses.IP4
  OpenFlow-Helpers
begin

datatype 'a flowtable-behavior = Action 'a | NoAction | Undefined

definition option-to-ftb b ≡ case b of Some a ⇒ Action a | None ⇒ NoAction

definition ftb-to-option b ≡ case b of Action a ⇒ Some a | NoAction ⇒ None

datatype ('m, 'a) flow-entry-match = OFEntry (ofe-prio: 16 word) (ofe-fields: 'm set) (ofe-action: 'a)

find-cons (('a × 'b) ⇒ 'c) ⇒ 'a ⇒ 'b ⇒ 'c
find-cons ('a ⇒ 'b ⇒ 'c) ⇒ ('a × 'b) ⇒ 'c

definition split3 f p ≡ case p of (a,b,c) ⇒ f a b c
find-cons ('a ⇒ 'b ⇒ 'c ⇒ 'd) ⇒ ('a × 'b × 'c) ⇒ 'd

type-synonym ('m, 'a) flowtable = (('m, 'a) flow-entry-match) list

type-synonym ('m, 'p) field-matcher = ('m set ⇒ 'p ⇒ bool)

definition OFsame-priority-match2 :: ('m, 'p) field-matcher ⇒ ('m, 'a) flowtable ⇒ 'p ⇒ 'a flowtable-behavior where
  OFsame-priority-match2 γ flow-entries packet ≡ let s =
    (of-action f)\f. f ∈ set flow-entries ∧ γ (ofe-fields f) packet ∧
    (∀ fo ∈ set flow-entries, ofe-prio fo > ofe-prio f → γ (ofe-fields fo) packet)) in
  case card s of 0 ⇒ NoAction
  | (Suc 0) ⇒ Action (the-elem s)
  | - ⇒ Undefined
definition check-no-overlap \(\gamma\) \(ft\) = (\(\forall a \in\) set \(ft\). \(\forall b \in\) set \(ft\). \(\forall p \in\) UNIV. (ofe-prio \(a\) = ofe-prio \(b\) \& \(\gamma\) (ofe-fields \(a\)) \(p\) \& \(a\) \(\neq\) \(b\)) \(\rightarrow\) \(\neg\gamma\) (ofe-fields \(b\)) \(p\))

definition check-no-overlap2 \(\gamma\) \(ft\) = (\(\forall a \in\) set \(ft\). \(\forall b \in\) set \(ft\). (\(a\) \(\neq\) \(b\) \& ofe-prio \(a\) = ofe-prio \(b\)) \(\rightarrow\) \(\neg\)(\(\exists p \in\) UNIV. \(\gamma\) (ofe-fields \(a\)) \(p\) \& \(\gamma\) (ofe-fields \(b\)) \(p\)))

lemma check-no-overlap-alt: check-no-overlap \(\gamma\) \(ft\) = check-no-overlap2 \(\gamma\) \(ft\)

unfolding check-no-overlap2-def check-no-overlap-def
by blast

lemma no-overlap-not-undefined: check-no-overlap \(\gamma\) \(ft\) \(\Rightarrow\) OF-same-priority-match2 \(\gamma\) \(ft\) \(p\) \(\neq\) Undefined

proof
  assume goal1: check-no-overlap \(\gamma\) \(ft\) OF-same-priority-match2 \(\gamma\) \(ft\) \(p\) = Undefined
  let ?as = \{\(f. f \in\) set \(ft\). \(\gamma\) (ofe-fields \(f\)) \(p\) \& (\(\forall fo \in\) set \(ft\). ofe-prio \(f\) < ofe-prio \(fo\) \(\rightarrow\) \(\neg\gamma\) (ofe-fields \(fo\)) \(p\)))\}
  have fin: finite ?as by simp
  note goal1[unfolded OF-same-priority-match2-def]
  then have 2 \(\leq\) card (ofe-action \(\gamma\)) unfolding f-img-ex-set
  unfolding Let-def
  by(cases card (ofe-action \(\gamma\)) simp) (rename-tac nat1, case-tac nat1, simp add: image-Collect, presburger)
  then have 2 \(\leq\) card ?as using card-image-le[OF fin, ofe-action] by linarith
  then obtain \(a\) \(\&\) \(b\) where \(a\) \(\neq\) \(b\) \& \(\forall a \in\) ?as \& \(\forall b \in\) ?as using card2-eq by blast
  then have ab2: \(\forall a \in\) set \(ft\) \(\gamma\) (ofe-fields \(a\)) \(p\) (\(\forall f o\in\) set \(ft\). ofe-prio \(a\) < ofe-prio \(fo\) \(\rightarrow\) \(\neg\gamma\) (ofe-fields \(fo\)) \(p\))
  \(\forall b \in\) set \(ft\) \(\gamma\) (ofe-fields \(b\)) \(p\) (\(\forall f o\in\) set \(ft\). ofe-prio \(b\) < ofe-prio \(fo\) \(\rightarrow\) \(\neg\gamma\) (ofe-fields \(fo\)) \(p\)) by simp-all
  then have ofe-prio \(a\) = ofe-prio \(b\) by blast
  note goal1[unfolded check-no-overlap-def] ab2(1) ab2(4) this ab2(2) ab1(1) ab2(5)
  then show False by blast
qed

fun OF-match-linear :: \((\text{'m}, \text{'p})\) field-matcher \(\Rightarrow\) \((\text{'m}, \text{'a})\) flowtable \(\Rightarrow\) \text{'p} \(\Rightarrow\) \text{'a} flowtable-behavior where
OF-match-linear :: \(\text{'a} \rightarrow\) \text{'a}
OF-match-linear \(\gamma\) (\(\text{'a} \#\) \(\text{'a}\)) \(p\) = (if \(\gamma\) (ofe-fields \(\text{'a}\)) \(p\) then Action (ofe-action \(\text{'a}\)) else OF-match-linear \(\gamma\) as \(p\))

lemma OF-match-linear-ne-Undefined: OF-match-linear \(\gamma\) \(ft\) \(p\) \(\neq\) Undefined
by(induction \(ft\)) auto

lemma OF-match-linear-append: OF-match-linear \(\gamma\) (\(\text{case OF-match-linear \(\gamma\) \(a\) \(p\) of NoAction \Rightarrow OF-match-linear \(\gamma\) \(b\) \(p\) | \(x\) \(\Rightarrow\) \(x\)\)\))
by(induction \(a\)) simp-all

lemma OF-match-linear-match-allsameaction: \(\exists gr \in\) set \(oms\). \(\forall gr\) \(p\) = True \(\Rightarrow\)
OF-match-linear \(\gamma\) (map \((\text{'x}. \text{split3 OFEntry (\text{'pri}, \text{'x}, \text{act})})\) \(oms\)) \(p\) = Action act
by(induction \(oms\)) auto simp add: split3-def

lemma OF-lm-noa-none-iff: OF-match-linear \(\gamma\) \(ft\) \(p\) = NoAction \(\iff\) \(\forall e\in\) set \(ft\). \(\neg\gamma\) (ofe-fields \(e\)) \(p\)
by(induction \(ft\)) (simp-all split: if-splits)

lemma set-eq-rule: \((\\forall x. x \in a \Rightarrow x \in b) \Rightarrow (\\forall x. x \in b \Rightarrow x \in a) \Rightarrow a = b\) by(rule antisym[OF subset1 subsetI])

lemma unmatching-insert-agnostic: \(\neg\gamma\) (ofe-fields \(a\)) \(p\) \(\Rightarrow\) OF-same-priority-match2 \(\gamma\) (\(a\) \# \(ft\)) \(p\) = OF-same-priority-match2 \(\gamma\) \(ft\) \(p\)

proof
  let \(\forall as = \{\(f. f \in\) set \(ft\). \(\gamma\) (ofe-fields \(f\)) \(p\) \& (\(\forall fo \in\) set \(ft\). ofe-prio \(f\) < ofe-prio \(fo\) \(\rightarrow\) \(\neg\gamma\) (ofe-fields \(fo\)) \(p\))\}
  let \(\forall as\) = \{\(f. f \in\) set (\(a\) \# \(ft\)) \& \(\gamma\) (ofe-fields \(f\)) \(p\) \& (\(\forall fo\in\) set (\(a\) \# \(ft\)). ofe-prio \(f\) < ofe-prio \(fo\) \(\rightarrow\) \(\neg\gamma\) (ofe-fields \(fo\)) \(p\))\}
  assume \(nm:\) \(\neg\gamma\) (ofe-fields \(a\)) \(p\)
have aa: \( ?aas = ?as \)

proof (rule set-eq-rule)

\[
\text{fix } x
\]

\[
\text{assume } x \in \{ f | f \in \text{set } (a \neq ft) \land \gamma (\text{ofe-fields } f) p \land (\forall \text{fo } \in \text{set } (a \neq ft). \text{ofe-prio } f < \text{ofe-prio } \text{fo} \rightarrow \neg \gamma (\text{ofe-fields } \text{fo}) p) \} \\
\text{hence as: } x \in \text{set } (a \neq ft) \land \gamma (\text{ofe-fields } x) p \land (\forall \text{fo } \in \text{set } (a \neq ft). \text{ofe-prio } x < \text{ofe-prio } \text{fo} \rightarrow \neg \gamma (\text{ofe-fields } \text{fo}) p) \text{ by simp}
\]

\[\text{with } \text{nn have } x \in \text{set } ft \text{ by fastforce}\]

moreover from as have \((\forall \text{fo } \in \text{set } ft. \text{ofe-prio } x < \text{ofe-prio } \text{fo} \rightarrow \neg \gamma (\text{ofe-fields } \text{fo}) p) \text{ by simp}\]

ultimately show \(x \in \{ f | f \in \text{set } ft. \gamma (\text{ofe-fields } f) p \land (\forall \text{fo } \in \text{set } ft. \text{ofe-prio } f < \text{ofe-prio } \text{fo} \rightarrow \neg \gamma (\text{ofe-fields } \text{fo}) p) \} \) using as by force

next

\[
\text{fix } x
\]

\[
\text{assume } x \in \{ f | f \in \text{set } ft. \gamma (\text{ofe-fields } f) p \land (\forall \text{fo } \in \text{set } ft. \text{ofe-prio } f < \text{ofe-prio } \text{fo} \rightarrow \neg \gamma (\text{ofe-fields } \text{fo}) p) \} \\
\text{hence as: } x \in \text{set } ft \gamma (\text{ofe-fields } x) p \land (\forall \text{fo } \in \text{set } ft. \text{ofe-prio } x < \text{ofe-prio } \text{fo} \rightarrow \neg \gamma (\text{ofe-fields } \text{fo}) p) \text{ by simp-all}
\]

from as(1) have \(x \in \text{set } (a \neq ft) \text{ by simp}\)

moreover from as(3) have \((\forall \text{fo } \in \text{set } (a \neq ft). \text{ofe-prio } x < \text{ofe-prio } \text{fo} \rightarrow \neg \gamma (\text{ofe-fields } \text{fo}) p) \text{ using } \text{nn by simp}\]

ultimately show \(x \in \{ f | f \in \text{set } ft. \gamma (\text{ofe-fields } f) p \land (\forall \text{fo } \in \text{set } ft. \text{ofe-prio } f < \text{ofe-prio } \text{fo} \rightarrow \neg \gamma (\text{ofe-fields } \text{fo}) p) \} \) using as(2) by blast

qed

note uf = arg-cong[OF aa, of (\_) ofe-action, unfolded image-Collect]

show \(?\text{thesis unfolding } \text{OF-same-priority-match2-def using } uf \text{ by presburger}\)

qed

lemma \(\text{OF-match-eq: sorted-descending } (\text{map } \text{ofe-prio } ft) \implies \text{check-no-overlap } \gamma ft \implies \text{OF-same-priority-match2 } \gamma ft p = \text{OF-match-linear } \gamma ft p\)

proof (induction ft)

case (Cons a ft)

have 1: \(\text{sorted-descending } (\text{map } \text{ofe-prio } ft) \text{ using } \text{Cons(2) by simp}\)

have 2: \(\text{check-no-overlap } \gamma ft \text{ using } \text{Cons(3) unfolding check-no-overlap-def using set-subset-Cons by fast}\)

note \(\text{mIH} = \text{Cons(1)} [\text{OF } 1 2]\)

show ?case (is ?kees)

proof (cases (\gamma (\text{ofe-fields } a) p))

case False thus ?kees

by (simp only: \text{OF-match-linear.simps if-False mIH[ symmetric] unmatching-insert-agnostic[of} \gamma, \text{OF False]} )

next

note \(\text{sorted-descending-split}[\text{OF } \text{Cons(2)}]\)

then obtain \(m n \text{ where } \text{mn: } a \neq ft = m \oplus n \land \forall e \in \text{set } m. \text{ofe-prio } a = \text{ofe-prio } e \land \forall e \in \text{set } n. \text{ofe-prio } e < \text{ofe-prio } a \)

unfolding list.sel by blast

hence aem: \(a \in \text{set } m\)

by (metis UnE less_imp_neq list.intros(1) set-append)

have mover: \(\text{check-no-overlap } \gamma m \text{ using } \text{Cons(3) unfolding check-no-overlap-def}\)

by (metis Un_iff mn(1) set-append)

let \(\text{fc} = (\lambda s. \{ f. f \in s \land \gamma (\text{ofe-fields } f) p \land (\forall \text{fo } \in \text{set } (a \neq ft). \text{ofe-prio } f < \text{ofe-prio } \text{fo} \rightarrow \neg \gamma (\text{ofe-fields } \text{fo}) p) \})\)

case True

have \(\text{fc}\) (\(m \oplus n\) = ?fc m \cup ?fc n) by auto

moreover have ?fc n = {}

proof (rule set-eq-rule, rule ccontr, goal-cases)

case (1 x)

hence \(g1: x \in \text{set } n \gamma (\text{ofe-fields } x) p\)

\((\forall \text{fo } \in \text{set } m. \text{ofe-prio } x < \text{ofe-prio } \text{fo} \rightarrow \neg \gamma (\text{ofe-fields } \text{fo}) p)\)

\((\forall \text{fo } \in \text{set } n. \text{ofe-prio } x < \text{ofe-prio } \text{fo} \rightarrow \neg \gamma (\text{ofe-fields } \text{fo}) p)\)

qed
unfolding \textit{mn}(1) by (simp-all)

from \textit{g1}(1) \textit{mn}(3) have le: ofe-prio \(x < \text{ofe-prio} \ a\) by simp

note le \textit{g1}(3) aem True

then show False by blast

qed simp

ultimately have \textit{cc}: \(?fc \ (m \oplus n) = ?fc \ m\) by blast

have \textit{cm}: \(?fc \ m = \{a\}\)

proof –

have \(\forall f \in \text{set} \ m. \ (\forall fo \in \text{set} \ (a \# ft). \ \text{ofe-prio} \ f < \text{ofe-prio} \ fo \longrightarrow \neg \gamma (\text{ofe-fields} \ fo)\ p)\)

by (metis UnE less-asym mn set-append)

hence 1: \(?fc \ m = \{f \in \text{set} \ m. \ \gamma (\text{ofe-fields} \ f) \ p\}\) by blast

show \(\{f \in \text{set} \ m. \ \gamma (\text{ofe-fields} \ f) \ p \wedge (\forall fo \in \text{set} \ (a \# ft). \ \text{ofe-prio} \ f < \text{ofe-prio} \ fo \longrightarrow \neg \gamma (\text{ofe-fields} \ fo)\ p)\} = \{a\}\)

unfolding 1

proof (rule set-eq-rule, goal-cases fwd bwd)

case (bwd x)

have \(a \in \{f \in \text{set} \ m. \ \gamma (\text{ofe-fields} \ f) \ p\}\) using True aem by simp

thus ?case using bwd by simp

next

case (fwd x)

show ?case

proof (rule ccontr)

assume \(x \notin \{a\}\) hence ne: \(x \neq a\) by simp

from fwd have 1: \(x \in \text{set} \ m. \ \gamma (\text{ofe-fields} \ x) \ p\) by simp-all

have 2: \(\text{ofe-prio} \ x = \text{ofe-prio} \ a\) using 1(1) \textit{mn}(2) by simp

show False using 1 ne mover aem True 2 unfolding check-no-overlap-def by blast

qed

qed

qed

show ?kees

unfolding \textit{mn}(1)

unfolding \textit{OF}-same-priority-match2-def

unfolding \textit{f-Img-ex-set}

unfolding \textit{cc}[unfolded \textit{mn}(1)]

unfolding \textit{cm}[unfolded \textit{mn}(1)]

unfolding \textit{Let-def}

by (simp only: \textit{mn}(1)[symmetric] \textit{OF-match-linear}. \textit{simps} True if-True, simp)

qed

qed (simp add: \textit{OF}-same-priority-match2-def)

lemma \textit{overlap-sort-invar}[simp]: \textit{check-no-overlap} \(\gamma (\text{sort-descending-key} \ k \ ft) = \text{check-no-overlap} \ \gamma \ ft\)

unfolding \textit{check-no-overlap-def}

unfolding \textit{sort-descending-set-inv}

..)

lemma \textit{OF-match-eq2}:

assumes \textit{check-no-overlap} \(\gamma \ ft\)

shows \(\textit{OF}-same-priority-match2 \ \gamma \ ft \ p = \textit{OF}-match-linear \ \gamma \ (\text{sort-descending-key} \ \text{ofe-prio} \ ft) \ p\)

proof –

have \textit{sorted-descending} \(\text{map} \ \text{ofe-prio} \ (\text{sort-descending-key} \ \text{ofe-prio} \ ft)\) by (simp add: \textit{sorted-descending-sort-descending-key})

note \(\textit{ceq} = \textit{OF-match-eq}[\textit{OF this}, \textit{unfolded overlap-sort-invar}, \textit{OF} \langle \textit{check-no-overlap} \ \gamma \ ft\rangle, \textit{symmetric}]\)

show \(?thesis\)

unfolding \textit{ceq}

unfolding \textit{OF}-same-priority-match2-def

unfolding \textit{sort-descending-set-inv}

..)

qed
lemma prio-match-matcher-alt: \{ f. f \in \text{set flow-entries} \land \gamma (\text{ofe-fields} f) \text{ packet} \land \\
(\forall fo \in \text{set flow-entries}. \text{ofe-prio} fo > \text{ofe-prio} f \rightarrow \neg \gamma (\text{ofe-fields} fo) \text{ packet})\}

= (\\
\text{let matching} = \{ f. f \in \text{set flow-entries} \land \gamma (\text{ofe-fields} f) \text{ packet}\} \\
in \{ f. f \in \text{matching} \land (\forall fo \in \text{matching}. \text{ofe-prio} fo \leq \text{ofe-prio} f)\}\)

by (auto simp add: Let-def)

lemma prio-match-matcher-alt2: (\\
\text{let matching} = \{ f. f \in \text{set flow-entries} \land \gamma (\text{ofe-fields} f) \text{ packet}\} \\
in \{ f. f \in \text{matching} \land (\forall fo \in \text{matching}. \text{ofe-prio} fo \leq \text{ofe-prio} f)\}\)

by (auto simp add: Let-def)

definition OF-priority-match where
OF-priority-match \gamma \text{ flow-entries packet} \equiv \\
\text{let m} = \text{filter} (\lambda f. \gamma (\text{ofe-fields} f) \text{ packet}) \text{ flow-entries}; \\
m' = \text{filter} (\lambda f. \forall fo \in \text{set m}. \text{ofe-prio} fo \leq \text{ofe-prio} f) \text{ m in} \\
case m' of [] \Rightarrow \text{NoAction} \\
| [s] => \text{Action} (\text{ofe-action} s) \\
| - => \text{Undefined}

definition OF-priority-match-ana where
OF-priority-match-ana \gamma \text{ flow-entries packet} \equiv \\
\text{let m} = \text{filter} (\lambda f. \gamma (\text{ofe-fields} f) \text{ packet}) \text{ flow-entries}; \\
m' = \text{filter} (\lambda f. \forall fo \in \text{set m}. \text{ofe-prio} fo \leq \text{ofe-prio} f) \text{ m in} \\
case m' of [] \Rightarrow \text{NoAction} \\
| [s] => \text{Action} s \\
| - => \text{Undefined}

lemma filter-singleton: \[x\leftarrow s. f x]\ = \[y]\ = \Rightarrow f y \land y \in \text{set s by (metis filter-eq-Cons-iff in-set-conv-decomp)}

lemma OF-spm3-get-fe: OF-priority-match \gamma ft p = \text{Action} a \Rightarrow \exists fe. \text{ofe-action} fe = a \land fe \in \text{set ft} \land OF-priority-match-ana \gamma ft p = \text{Action fe}

unfolding OF-priority-match-def OF-priority-match-ana-def 
by (clarsimp split: flowtable-behavior.splits list.splits) (drule filter-singleton; simp)

fun no-overlaps where
no-overlaps - [] = True | 
no-overlaps \gamma (a\#as) = (no-overlaps \gamma as \land ( \\
\forall b \in \text{set as}. \text{ofe-prio} a = \text{ofe-prio} b \rightarrow \neg (\exists p \in \text{UNIV}. \gamma (\text{ofe-fields} a) p \land \gamma (\text{ofe-fields} b) p)))

lemma no-overlap-Cons1: check-no-overlap2 \gamma (x\#xs) \Rightarrow check-no-overlap2 \gamma xs 
unfolding check-no-overlap2-def by simp

lemma no-overlap-Cons2: check-no-overlap \gamma t \Rightarrow distinct t \Rightarrow no-overlaps \gamma t 
unfolding check-no-overlap-alt
proof(induction t)
  case (Cons a t)
from no-overlap-Cons1[OF Cons(2)] Cons(3,1)
have no-overlaps \( \gamma t \) by simp 
thus \(?case using Cons(2,3) unfolding check-no-overlap2-def by auto 
qed (simp add: check-no-overlap2-def)

lemma check-no-overlap1: no-overlaps \( \gamma t \Longrightarrow \) check-no-overlap \( \gamma t \)
proof (induction \( t \))
  case (Cons \( a t \))
  from Cons(1)[OF conjunct1[OF Cons(2)[unfolded no-overlaps.simps]]]
  show \(?case using conjunct2[OF Cons(2)[unfolded no-overlaps.simps]] unfolding check-no-overlap2-def 
  by auto 
  qed (simp add: check-no-overlap2-def)

lemma \( \bigwedge e p. e \in t \Rightarrow \lnot \gamma (ofe-fields e) p \Longrightarrow \) no-overlaps \( \gamma t \)
by(induction \( t \)) simp-all
lemma no-overlaps-append: no-overlaps \( \gamma (x @ y) \Longrightarrow \) no-overlaps \( \gamma y \)
by(induction \( x \)) simp-all
lemma no-overlaps-ne1: no-overlaps \( \gamma (x @ a \# y @ b \# z) \Longrightarrow ((\exists p. \gamma (ofe-fields a) p) \vee (\exists p. \gamma (ofe-fields b) p)) \Longrightarrow a \neq b \)
proof (rule notI, goal-cases contr)
  case contr
  from contr(1) no-overlaps-append have no-overlaps \( \gamma (a \# y @ b \# z) \) by blast
  note this[unfolded no-overlaps.simps]
  with contr(3) have \( \lnot (\exists p \in \text{UNIV.} \gamma (ofe-fields a) p \land \gamma (ofe-fields b) p) \) by simp
  with contr(2) show False unfolding contr(3) by simp 
  qed

lemma no-overlaps-defeq: no-overlaps \( \gamma fe \Longrightarrow OF\text{-}same-priority-match2 \gamma fe p = OF\text{-}priority-match \gamma fe p \)
unfolding OF\text{-}same-priority-match2-def OF\text{-}priority-match-def
unfolding f\text{-}Img-ex-set
unfolding prio-match-matcher-alt
unfolding prio-match-matcher-alt2
proof (goal-cases uf)
  case uf
  let ?m' = let \( m = [f\leftarrow fe . \gamma (ofe-fields f) p] \) in \([f\leftarrow m . \forall fo \in set m. ofe-prio fo \leq ofe-prio f]\)
  let ?s = ofe-action ' set \(?m' \)
  from uf show \(?case proof (cases \(?m' \)
  case Nil
  moreover then have card \(?s = 0 \) by force
  ultimately show \(?thesis by(simp add: Let-def) 
next
  case (Cons \( a \ as \))
  have as = []
  proof(rule contr)
    assume as \( \neq [] \)
    then obtain \( b \ bs \ where \ bbs: as = b \# bs \) by (meson neq-Nil-conv)
    note no = Cons[unfolded Let-def filter-filter]
    have f1: \( a \in set \(?m' \ b \in set \(?m' \ unfolding bbs local.Cons by simp-all 
    hence ofe-prio a = ofe-prio b by (simp add: antisym)
    moreover have ms: \( \gamma (ofe-fields a) p \gamma (ofe-fields b) p \) using no[symmetric] unfolding bbs by(blast dest: Cons-eq-filterD)+
    moreover have abis: \( a \in set fe b \in set fe \) using f1 by auto
moreover have \( a \neq b \) proof (cases \( \exists x y z. \, fe = x \oplus a \# y \oplus b \# z \))

case True
then obtain \( x y z \) where \( xyz \): \( fe = x \oplus a \# y \oplus b \# z \) by blast

show ?thesis by blast
next

then obtain \( x y z \) where \( xyz \): \( fe = x \oplus b \# y \oplus a \# z \)

using no unfolding bbs

by (metis (no-types, lifting) Cons-eq-filterD)

from no-overlaps-ne1 ms(1) uf[unfolded xyz]

show ?thesis by blast

qed ultimately show False using check-no-overlapI[OF uf, unfolded check-no-overlap-def] by blast

then have oe: \( a \# as = [a] \) by simp

show ?thesis using Cons[unfolded oe] by force

qed

lemma distinct fe \( \implies \) check-no-overlap \( \gamma \) \( fe \implies \) OF-same-priority-match2 \( \gamma \) \( fe \, p = \) OF-priority-match \( \gamma \) \( fe \, p \)

by (rule no-overlaps-defeq) (drule (2) no-overlapsI)

theorem OF-eq:

assumes no: no-overlaps \( \gamma \) \( f \)

and so: sorted-descending (map ofe-prio \( f \))

shows OF-match-linear \( \gamma \) \( f \, p = \) OF-priority-match \( \gamma \) \( f \, p \)

unfolding no-overlaps-defeq[symmetric,OF no] OF-match-eq[OF so check-no-overlapI[OF no]]

..

corollary OF-eq-sort:

assumes no: no-overlaps \( \gamma \) \( f \)

shows OF-priority-match \( \gamma \) \( f \, p = \) OF-match-linear \( \gamma \) (sort-descending-key ofe-prio \( f \)) \( p \)

using OF-match-eq2 check-no-overlapI no no-overlaps-defeq by fastforce

lemma OF-lm-noa-none: OF-match-linear \( \gamma \) \( ft \, p = NoAction \implies \forall e \in \text{set} \, ft. \, \neg \gamma (\text{ofe-fields} \, e) \, p \)

by (induction \( ft \)) (simp-all split: if-splits)

lemma OF-spm3-noa-none:

assumes no: no-overlaps \( \gamma \) \( ft \)

shows OF-priority-match \( \gamma \) \( ft \, p = NoAction \implies \forall e \in \text{set} \, ft. \, \neg \gamma (\text{ofe-fields} \, e) \, p \)

unfolding OF-eq-sort[OF no] by (drule OF-lm-noa-none) simp

lemma no-overlaps-not-undefined: no-overlaps \( \gamma \) \( ft \implies \) OF-priority-match \( \gamma \) \( ft \, p \neq \) Undefined

using check-no-overlapI no-overlap-not-undefined no-overlaps-defeq by fastforce

end

theory OpenFlow-Serialize

imports OpenFlow-Matches

OpenFlow-Action

Semantics-OpenFlow

Simple-Firewall.Primitives-toString

12
\textbf{IP-Addresses.Lib-Word-toString}

\begin{verbatim}
begin

\textbf{definition} serialization-test-entry \equiv OFEntry 7 \{ EtherDst 0x1, IPv4Dst (PrefixMatch 0xA000201 32), IngressPort \textquoteleft s1\textendash lan\textquoteright, L4Dst 0x50 0, L4Src 0x400 0x3FF, IPv4Proto 6, EtherType 0x800 \} [ModifyField-l2dst 0xA641F185E862, Forward \textquoteleft s1\textendash wan\textquoteright]

\textbf{value} (map ((\textless\textless) (1::48 word) \circ (+ 8) \circ rev) [0..<6])

\textbf{definition} serialize-mac (m::48 word) \equiv (intersperse (CHR \\\	extquoteleft ;\textquoteright) \circ map (hex-string-of-word 1 \circ (\lambda h. (m \gg 8 \land\land \\\	extquoteleft 0xff\textquoteright)) \circ rev) [0..<6])

\textbf{lemma} serialize-mac \textquoteleft de:ad:be:cf:fe\textquoteright \textbf{by} eval

\textbf{definition} serialize-action pids a \equiv (case a of
\begin{itemize}
\item Forward of if \textbf{⇒} \textquoteleft output:\textquoteright \circ pids of
\item ModifyField-l2dst na \textbf{⇒} \textquoteleft mod-l-dst:\textquoteright \circ serialize-mac na
\end{itemize}

\textbf{definition} serialize-actions anything [\_] \equiv \textquoteleft drop\textquoteright
\textbf{by} (simp add: serialize-actions-def)

\textbf{definition} prefix-to-string pfx \equiv ipv4-cidr-toString (pfxm-prefix pfx, pfxm-length pfx)

\textbf{primrec} serialize-of-match \textbf{where}
\begin{itemize}
\item serialize-of-match pids (IngressPort p) = \textquoteleft in-port=\textquoteright \circ pids p |
\item serialize-of-match - (VlanId i) = \textquoteleft dl-vlan=\textquoteright \circ dec-string-of-word0 i |
\item serialize-of-match - (VlanPriority -) = undefined |
\item serialize-of-match - (EtherType i) = \textquoteleft dl-type=0x\textquoteright \circ hex-string-of-word0 i |
\item serialize-of-match - (EtherSrc m) = \textquoteleft dl-src=\textquoteright \circ serialize-mac m |
\item serialize-of-match - (EtherDst m) = \textquoteleft dl-dst=\textquoteright \circ serialize-mac m |
\item serialize-of-match - (IPv4Proto i) = \textquoteleft nw-proto=\textquoteright \circ dec-string-of-word0 i |
\item serialize-of-match - (IPv4Src p) = \textquoteleft nw-src=\textquoteright \circ prefix-to-string p |
\item serialize-of-match - (IPv4Dst p) = \textquoteleft nw-dst=\textquoteright \circ prefix-to-string p |
\item serialize-of-match - (L4Src i m) = \textquoteleft tp-src=\textquoteright \circ dec-string-of-word0 i \circ (if m = -1 \textbf{then} \_ else \textquoteleft /0x\textquoteright \circ hex-string-of-word3 m) |
\item serialize-of-match - (L4Dst i m) = \textquoteleft tp-dst=\textquoteright \circ dec-string-of-word0 i \circ (if m = -1 \textbf{then} \\
textbf{else} \textquoteleft /0x\textquoteright \circ hex-string-of-word3 m)
\end{itemize}

\textbf{definition} serialize-of-matches :: (string \Rightarrow string) \Rightarrow of-match-field set \Rightarrow string
\textbf{where}
\begin{itemize}
\item serialize-of-matches pids \equiv (\@) \textquoteleft hard-timeout=0, idle-timeout=0,\textquoteright \circ serialize-mac (CHR \\\	extquoteleft ;\textquoteright) \circ map (serialize-of-match pids) \circ sorted-list-of-set
\end{itemize}

\textbf{lemma} serialize-of-matches pids of-matches=
\textbf{(List append \textquoteleft hard-timeout=0, idle-timeout=0,\textquoteright)}
\textbf{by} (simp add: serialize-of-matches-def)\end{verbatim}
lemma serialize-of-entry (λof. "42") (ofe-fields serialization-test-entry) =
"hard-timeout=0, idle-timeout=0, in-port=42, dl-type=0x800, dl-dst=00:00:00:00:00:01, nw-proto=6, nw-dst=10.0.2.1/32, tp-src=1024, lan=0x8000000000000000)
by eval

definition serialize-of-matches pids e ≡ (case e of (OFEntry p f a) ⇒ "priority=" @ dec-string-of-word0 p @ "," @ serialize-of-matches pids f @ "," @ "action=" @ serialize-actions pids a)

lemma serialize-of-entry (the o map-of ["s1-lan","42"],"s1-wan","1337")) serialization-test-entry =
"priority=7, hard-timeout=0, idle-timeout=0, in-port=42, dl-type=0x800, dl-dst=00:00:00:00:00:01, nw-proto=6, nw-dst=10.0.2.1/32, tp-src=1024, lan=0x8000000000000000)
by eval

end
theory Featherweight-OpenFlow-Comparison
imports Semantics-OpenFlow
begin

inductive guha-table-semantics :: (m', p') field-matcher ⇒ (m, a) flowtable ⇒ p ⇒ a option ⇒ bool where
guha-matched: γ (ofe-fields fe) p = True ⇒
∀ fe' ∈ set (ft1 @ ft2), ofe-prio fe' > ofe-prio fe → γ (ofe-fields fe') p = False ⇒
guha-table-semantics γ (ft1 @ fe # ft2) p (Some (ofe-action fe)) |
guha-unmatched: ∀ fe ∈ set ft. γ (ofe-fields fe) p = False ⇒
guha-table-semantics γ ft p None

lemma guha-table-semantics-ex2res:
  assumes to: CARD('a) ≥ 2
  assumes ms: ∃ ff. γ ff p
  shows ∃ ft (a1 :: 'a) (a2 :: 'a). a1 ≠ a2 ∧ guha-table-semantics γ ft p (Some a1) ∧ guha-table-semantics γ ft p (Some a2)
proof
  from ms obtain ff where m: γ ff p ..
  from to obtain a1 a2 :: 'a where as: a1 ≠ a2 using card2-e1 by blast
  let ?fe1 = OFEntry 0 ff a1
  let ?fe2 = OFEntry 0 ff a2
  let ?ft = [?fe1, ?fe2]
  have guha-table-semantics γ ?ft p (Some a1) guha-table-semantics γ ?ft p (Some a2)
  by(rule guha-table-semantics.intros(1)[of γ ?fe1 p [] ?fe2], unfolded append-nil flow-entry-match.sel) |
  rule guha-table-semantics.intros(1)[of γ ?fe2 p [?fe1] []]. unfolded append-nil2 flow-entry-match.sel append.simps] |
  simp add: m)+
  thus ?thesis using as by(intro ez1 conjI)
qed

lemma guha-amstaendlich:
  assumes ae: a = ofe-action fe
  assumes ele: fe ∈ set ft
  assumes rest: γ (ofe-fields fe) p
  ∀ fe' ∈ set ft. ofe-prio fe' > ofe-prio fe → γ (ofe-fields fe') p
  shows guha-table-semantics γ ft p (Some a)
proof
  from ele obtain ft1 ft2 where ftspl: ft = ft1 @ fe # ft2 using split-list by fastforce
  show ?thesis unfolding ae ftspl
apply \( \text{rule guha-table-semantics.intros(1)} \)
using rest(1) apply(simp)
using rest(2)[unfolded ftsp] apply simp
done
qed

lemma guha-matched-rule-inversion:
assumes guha-table-semantics \( \gamma \) ft p (Some a)
shows \( \exists fe \in \text{set ft}. \; a = \text{ofe-action fe} \land \gamma (\text{ofe-fields fe}) \land p \land (\forall fe' \in \text{set ft}. \; \text{ofe-prio fe'} > \text{ofe-prio fe} \rightarrow \lnot \gamma (\text{ofe-fields fe'})) \)
proof –
{  
  fix d
  assume guha-table-semantics \( \gamma \) ft p d
hence Some a = d = \Rightarrow (\exists fe \in \text{set ft}. \; a = \text{ofe-action fe} \land \gamma (\text{ofe-fields fe}) \land p \land (\forall fe' \in \text{set ft}. \; \text{ofe-prio fe'} > \text{ofe-prio fe} \rightarrow \lnot \gamma (\text{ofe-fields fe'}))
by (induction rule: guha-table-semantics.induct) simp-all
}
from this[OF assms refl] show \(?thesis\).
qed

lemma guha-equal-Action:
assumes no: no-overlaps \( \gamma \) ft
assumes spm: OF-priority-match \( \gamma \) ft p = Action a
shows guha-table-semantics \( \gamma \) ft p (Some a)
proof –
  note spm[THEN OF-spm3-get-fe] then obtain fe where a: \text{ofe-action fe} = a and fein: fe \in \text{set ft} and feana: OF-priority-match-ana \( \gamma \) ft p = Action fe by blast
show ?thesis
  apply(rule guha-amstaendlich)
  apply(rule a[symmetric])
  apply(rule fein)
  using feana unfolding OF-priority-match-ana-def
  apply(auto dest!: filter-singleton split: list.splits)
done
qed

lemma guha-equal-NoAction:
assumes no: no-overlaps \( \gamma \) ft
assumes spm: OF-priority-match \( \gamma \) ft p = NoAction
shows guha-table-semantics \( \gamma \) ft p None
using spm unfolding OF-priority-match-def
by (auto simp add: filter-empty-conv OF-spm3-noa-none[OF no spm] intro: guha-table-semantics.intros(2) split: list.splits)

lemma guha-equal-hlp:
assumes no: no-overlaps \( \gamma \) ft
shows guha-table-semantics \( \gamma \) ft p (ftb-to-option (OF-priority-match \( \gamma \) ft p))
unfolding ftb-to-option-def
apply(cases (OF-priority-match \( \gamma \) ft p))
apply(simp add: guha-equal-Action[OF no])
apply(simp add: guha-equal-NoAction[OF no])
apply(subgoal_tac False, simp)
apply(simp add: no no-overlaps-not-undefined)
done

lemma guha-deterministic1: guha-table-semantics γ ft p (Some x1) ⇒¬ guha-table-semantics γ ft p None 
by(auto simp add: guha-table-semantics.simps)

lemma guha-deterministic2: [no-overlaps γ ft; guha-table-semantics γ ft p (Some x1); guha-table-semantics γ ft p (Some a)]
⇒ x1 = a
proof(rule ccontr, goal-cases)
  case 1
  note 1(2−3)[THEN guha-matched-rule-inversion] then obtain fe1 fe2 where fes:
  fe1∈set ft x1 = ofe-action fe1 γ (ofe-fields fe1) p (∀fe′∈set ft. ofe-prio fe1 < ofe-prio fe′ −→ ¬ γ (ofe-fields fe′) p)
  fe2∈set ft a = ofe-action fe2 γ (ofe-fields fe2) p (∀fe′∈set ft. ofe-prio fe2 < ofe-prio fe′ −→ ¬ γ (ofe-fields fe′) p)
  by blast
  from x1 ≠ a have fene: fe1 ≠ fe2 using fes(2,6) by blast
  have pe: ofe-prio fe1 = ofe-prio fe2 using fes(1,3−4,5,7−8) less-linear by blast
  note [no-overlaps γ ft] THEN check-no-overlapI, unfolded check-no-overlap-def
  thus False by blast
qed

lemma guha-equal:
  assumes no: no-overlaps γ ft
  shows OF-priority-match γ ft p = option-to-ftb d ⇔ guha-table-semantics γ ft p d
using guha-equal-hlp[OF no, of p] unfolding ftb-to-option-def option-to-ftb-def
apply(cases OF-priority-match γ ft p; cases d)
apply(simp-all)
using guha-deterministic1 apply fast
using guha-deterministic2[OF no] apply blast
using guha-deterministic1 apply fast
using no-overlaps-not-unefined[OF no] apply fastforce
using no-overlaps-not-unified[OF no] apply fastforce
done

lemma guha-nondeterministicD:
  assumes ¬check-no-overlap γ ft
  shows ∃fe1 fe2 p. fe1 ∈ set ft ∧ fe2 ∈ set ft ∧ guha-table-semantics γ ft p (Some (ofe-action fe1)) ∧ guha-table-semantics γ ft p (Some (ofe-action fe2))
using assms
apply(unfold check-no-overlap-def)
apply(clarsimp)
apply(rename-tac fe1 fe2 p)
apply(rule-tac x = fe1 in ez1)
apply(simp)
apply(rule-tac x = fe2 in ez1)
apply(simp)
apply(rule conjI)
apply(subst guha-table-semantics.simps)
apply(rule disjI)
apply(clarsimp)
apply(rule_tac x = fe1 in ez1)
apply(drule split-list)
apply(clarify)
apply(rename-tac ft1 ft2)
apply(rule-tac x = ft1 in exI)
apply(rule-tac x = ft2 in exI)
apply(simp)
oops

The above lemma does indeed not hold, the reason for this are (possibly partially) shadowed overlaps. This is exemplified below: If there are at least three different possible actions (necessary assumption) and a match expression that matches all packets (convenience assumption), it is possible to construct a flow table that is admonished by check-no-overlap but still will never run into undefined behavior.

lemma
assumes \( \text{CARD('action')} \geq 3 \)
assumes \( \forall p. \gamma x p \)
shows \( \exists ft :: (\text{ft1}, \text{ft2}) \text{. flow-entry-match list. } \text{¬check-no-overlap } \gamma ft \wedge \) \( \neg(\exists fe1 fe2 p. fe1 \in \text{set } ft \wedge fe2 \in \text{set } ft \wedge fe1 \neq fe2 \wedge \text{ofe-prio } fe1 = \text{ofe-prio } fe2 \) \wedge \( \text{guha-table-semantics } \gamma ft p \) (Some (ofe-action fe1)) \wedge \( \text{guha-table-semantics } \gamma ft p \) (Some (ofe-action fe2)))
proof
  obtain adef aa ab :: 'action where anb[simp]: aa \neq ab adef \neq ab adef \neq ab using assms(1) card3-eI by blast
  let ?cex = [OFEntry 1 x adef, OFEntry 0 x aa, OFEntry 0 x ab]
  have ol: \( \text{¬check-no-overlap } \gamma ?cex \)
    unfolding check-no-overlap-def ball-simps
    apply(rule bexI[where x = OFEntry 0 x aa, rotated], (simp;fail))
    apply(rule bexI[where x = OFEntry 0 x ab, rotated], (simp;fail))
    apply(simp add: assms)
  done
  have df: guha-table-semantics \( \gamma ?cex p oc \Rightarrow oc = \text{Some adef for } p oc \)
    unfolding guha-table-semantics.simps
    apply(elim disjE; clarsimp simp: assms)
    subgoal for fe ft1 ft2
      apply(cases ft1 = [])
      apply(fastforce)
      apply(cases ft2 = [])
      apply(fastforce)
    apply(subgoal-tac ft1 = [OFEntry 1 x adef] \wedge fe = OFEntry 0 x aa \wedge ft2 = [OFEntry 0 x ab])
    apply(simp;fail)
    apply(clarsimp simp add: List.neq-Nil-conv)
    apply(rename-tac ya ys yz)
    apply(case-tac ys; clarsimp simp add: List.neq-Nil-conv)
    done
done
  show ?thesis
    apply(intro exI[where x = ?cex], intro conjI, fact ol)
    apply(clarify)
    apply(unfold set-simps)
    apply(elim insertE;clarsimp)
    apply((drule df)+; unfold option.inject; (elim anb[symmetric, THEN notE] | (simp;fail)))+
  done
qed
**imports** IP-Addresses.CIDR-Split
Automatic-Refinement.Misc
Simple-Firewall.Generic-SimpleFw
Semantics-OpenFlow
OpenFlow-Matches
OpenFlow-Action
Routing.Linux-Router
Pure-ex.Guess

**begin**

**hide-const** Misc.uncurry

**hide-fact** Misc.uncurry-def

**definition** route2match r =
\[\langle iiface = ifaceAny, oiface = ifaceAny, src = (0, 0), dst = (pfxm-prefix \text{(routing-match } r), \text{pfxm-length (routing-match } r)), proto = ProtoAny, sports = (0, -1), ports = (0, -1)\]

**definition** toprefixmatch where
toprefixmatch m \equiv (let pm = PrefixMatch (fst m) (snd m) in if pm = PrefixMatch 0 0 then None else Some pm)

**lemma** prefix-match-semantics-simple-match:
assumes some:
toprefixmatch m = Some pm
assumes vld:
valid-prefix pm
shows prefix-match-semantics pm = simple-match-ip m

**using** some
by (cases m)
(clarsimp
  simp add: toprefixmatch-def ipset-from-cidr-def pfxm-mask-def fun-eq-iff
  prefix-match-semantics-ipset-from-netmask[\text{OF vld}] NOT-mask-shifted-lenword[\text{symmetric}]
  split: if-splits)

**definition** simple-match-to-of-match-single ::
\[(32, 'a) \text{ simple-match-scheme} \Rightarrow \text{char list option} \Rightarrow \text{protocol} \Rightarrow (16 \text{ word} \times 16 \text{ word}) \text{ option} \Rightarrow \text{of-match-field set} \]

\[\text{where} \]
simple-match-to-of-match-single m ifs proto sport dport \equiv
uncurry L4Src ' option2set sport \cup uncurry L4Dst ' option2set dport
\cup IPv4Proto ' \text{(case prot of ProtoAny \Rightarrow \{\} \mid Proto p \Rightarrow \{p\})} -- protocol is an 8 word option anyway...
\cup IngressPort ' option2set iif
\cup IPv4Src ' option2set (toprefixmatch (src m)) \cup IPv4Dst ' option2set (toprefixmatch (dst m))
\cup \{EtherType 0x0800\}

**definition** simple-match-to-of-match :: 32 simple-match \Rightarrow \text{string list} \Rightarrow \text{of-match-field set list} \text{ where}
simple-match-to-of-match m ifs \equiv (let
nqm = (\lambda p. \text{fst p} = 0 \land \text{snd p} = -1); sb = (\lambda p. \text{if nqm p then [None] else if \text{fst p} = \text{snd p} \text{ then map (Some o (\lambda pfs. (pfxm-prefix pfs, Bit-Operations.not (pfxm-mask pfs)))) (wordinterval-CIDR-split-pfxmatch (WordInterval (\text{fst p} (\text{snd p}))) else [])) in [simple-match-to-of-match-single m ifs (proto m) sport dport.
if f = (\lambda iface m = ifaceAny \text{then [None] else [Some i \leftarrow ifs, match-iface (iiface m) i]},
sport \leftarrow sb (sports m),
dport \leftarrow sb (dports m)]
)}

18
lemma smtoms-eq-hlp: simple-match-to-of-match-single $r$ $a$ $b$ $c$ $d$ = simple-match-to-of-match-single $r$ $f$ $g$ $h$ $i$ $\iff$ ($a = f \and b = g \and c = h \and d = i$)

proof (rule iffl, goal-cases)
  case 1
  thus ?case proof (intro conjI)
    have $*$: $\forall P \, x. \, \forall x :: \text{of-match-field.} \, P \, x; \, z = \text{Some} \, x \Rightarrow P \, (\text{IngressPort} \, x)$ by simp
    show $a = f$ using 1 by (cases $a$; cases $f$)
      (simp add: option2set-None simple-match-to-of-match-single-def toprefixmatch-def option2set-def; subst(asm) set-eq-iff; drule (1) *; simp split: option.splits uncurry-splits protocol.splits)+
  next
    have $*$: $\forall P \, z \, x. \, \forall x :: \text{of-match-field.} \, P \, x; \, z = \text{Proto} \, x \Rightarrow P \, (\text{IPv4Proto} \, x)$ by simp
    show $b = g$ using 1 by (cases $b$; cases $g$)
      (simp add: option2set-None simple-match-to-of-match-single-def toprefixmatch-def option2set-def; subst(asm) set-eq-iff; drule (1) *; simp split: option.splits uncurry-splits protocol.splits)+
  next
    have $*$: $\forall P \, z \, x. \, \forall x :: \text{of-match-field.} \, P \, x; \, z = \text{Some} \, x \Rightarrow P \, (\text{uncurry L4Src} \, x)$ by simp
    show $c = h$ using 1 by (cases $c$; cases $h$)
      (simp add: option2set-None simple-match-to-of-match-single-def toprefixmatch-def option2set-def; subst(asm) set-eq-iff; drule (1) *; simp split: option.splits uncurry-splits protocol.splits)+
  next
    have $*$: $\forall P \, z \, x. \, \forall x :: \text{of-match-field.} \, P \, x; \, z = \text{Some} \, x \Rightarrow P \, (\text{uncurry L4Dst} \, x)$ by simp
    show $d = i$ using 1 by (cases $d$; cases $i$)
      (simp add: option2set-None simple-match-to-of-match-single-def toprefixmatch-def option2set-def; subst(asm) set-eq-iff; drule (1) *; simp split: option.splits uncurry-splits protocol.splits)+
  qed
qed simp

lemma simple-match-to-of-match-generates-prereqs: simple-match-valid $m$ $\Rightarrow$ $r \in$ set (simple-match-to-of-match $m$ ifs) $\Rightarrow$ all-prerequisites $r$

unfolding simple-match-to-of-match-def Let-def
proof (clarsimp, goal-cases)
  case (1 xiface xsrcp xdstp)
  note $a = this$
  show ?case unfolding simple-match-to-of-match-single-def all-prerequisites-def
    unfolding ball-Ln
    proof (intro conjI; (simp fail) -), goal-cases)
      case 1
      have $e$: (fst (sports $m$) = 0 $\and$ snd (sports $m$) = $-1$ $\or$ proto $m$ = Proto TCP $\or$ proto $m$ = Proto UDP $\or$ proto $m$ = Proto L4-Protocol.SCTP
        using $o$(1)
        unfolding simple-match-valid-alt Let-def
        by (clarsimp split: if-splits)
      show ?case
        using $o$(3) $e$
        by (elim disjE; simp add: option2set-def split: if-splits prod.splits uncurry-splits)
      next
      case 2
      have $e$: (fst (dports $m$) = 0 $\and$ snd (dports $m$) = $-1$ $\or$ proto $m$ = Proto TCP $\or$ proto $m$ = Proto UDP $\or$ proto $m$ = Proto L4-Protocol.SCTP
        using $o$(1)
        unfolding simple-match-valid-alt Let-def

by (clarsimp split: if-splits)
show ?case
using o(4) e
by (elim disjE; simp add: option2set-def split: if-splits prod.splits uncurry-splits)
qed
qd

lemma and-assoc: a ∧ b ∧ c ⟷ (a ∧ b) ∧ c by simp

lemmas custom-simpset = Let-def set-concat set-map map-map map-map comp-def concat-map-maps set-maps UN-iff fun-app-def Set.
image-iff
abbreviation simple-fw-prefix-to-wordinterval ≡ prefix-to-wordinterval ◦ uncurry PrefixMatch

lemma simple-match-port-alt: simple-match-port m p ⟷ p ∈ wordinterval-to-set (uncurry WordInterval m) by (simp split: uncurry-splits)

lemma simple-match-src-alt: simple-match-valid r =⇒ simple-match-ip (src r) p ⟷ prefix-match-semantics (PrefixMatch (fst (src r)) (snd (src r))) p by (cases (src r)); (simp add: prefix-match-semantics-ipset-from-netmask2 prefix-to-wordset-ipset-from-cidr simple-match-valid-def valid-prefix-fw-def)

lemma simple-match-dst-alt: simple-match-valid r =⇒ simple-match-ip (dst r) p ⟷ prefix-match-semantics (PrefixMatch (fst (dst r)) (snd (dst r))) p by (cases (dst r)); (simp add: prefix-match-semantics-ipset-from-netmask2 prefix-to-wordset-ipset-from-cidr simple-match-valid-def valid-prefix-fw-def)

lemma x ∈ set (wordinterval-CIDR-split-prefixmatch w) =⇒ valid-prefix x using wordinterval-CIDR-split-prefixmatch-all-valid-Ball[THEN bspec, THEN conjunct1]

lemma simple-match-to-of-matchI:
assumes mv:: simple-match-valid r
assumes mm:: simple-matches r p
assumes ii:: p-iiface p ∈ set ifs
assumes ippkt:: p-l2type p = 0x800
shows eq:: ∃ gr∈ set (simple-match-to-of-match r ifs)
OF-match-fields gr p = Some True
proof -
let ?npm = λp. fst p = 0 ∧ snd p = −1
let ?sb = λp r. (if ?npm p then None else Some r)
obtain si where si: case si of Some ssi ⇒ p-sport p ∈ prefix-to-wordset ssi | None ⇒ True
    case si of None ⇒ True | Some ssi ⇒ ssi ∈ set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (sports r)))
si = None ⟷ ?npm (sports r)
proof (cases ?npm (sports r), goal-cases)
  case 1
  hence (case None of None ⇒ True | Some ssi ⇒ p-sport p ∈ prefix-to-wordset ssi) ∧
          (case None of None ⇒ True | Some ssi ⇒ ssi ∈ set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (sports r)))) by simp
  with 1 show ?thesis by blast
next
case 2
from mm have p-sport p ∈ wordinterval-to-set (uncurry WordInterval (sports r))
  by (simp only: simple-matches.simps simple-match-port-alt)
then obtain ssi where ssi: ssi ∈ set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (sports r)))
proof

using wordinterval-CIDR-split-existential by fast

hence (case Some ssi of None ⇒ True | Some ssi ⇒ p-sport p ∈ prefix-to-wordset ssi) ∧

(case Some ssi of None ⇒ True

| Some ssi ⇒ ssi ∈ set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (sports r)))) by simp

with 2 show ?thesis by blast

qed

obtain di where di: case di of Some ddi ⇒ p-dport p ∈ prefix-to-wordset ddi | None ⇒ True
case di of None ⇒ True | Some ddi ⇒ ddi ∈ set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (sports r)))
di = None ↔ ?npm (dports r)

proof (cases ?npm (dports r), goal-cases)

case 1

hence (case None of None ⇒ True | Some ssi ⇒ p-dport p ∈ prefix-to-wordset ssi) ∧

(case None of None ⇒ True

| Some ssi ⇒ ssi ∈ set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (sports r)))) by simp

with 1 show ?thesis by blast

next

case 2

from mm have p-dport p ∈ wordinterval-to-set (uncurry WordInterval (dports r))

by (simp only: simple-matches.simps simple-match-port-alt)

then obtain ddi where ddi:

ddi ∈ set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (dports r)))
p-dport p ∈ prefix-to-wordset ddi

using wordinterval-CIDR-split-existential by fast

hence (case Some ddi of None ⇒ True | Some ssi ⇒ p-dport p ∈ prefix-to-wordset ssi) ∧

(case Some ddi of None ⇒ True

| Some ssi ⇒ ssi ∈ set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (dports r)))) by simp

with 2 show ?thesis by blast

qed

show ?thesis

proof

let ?mf = map-option (apsnd (wordNOT o mask o (-) 16) o prefix-to-wordset)

let ?gr = simple-match-to-of-match-single r

(if iface r = ifaceAny then None else Some (p-iface p))

(if proto r = ProtoAny then ProtoAny else Proto (p-proto p))

(?mf ?gr) (=?mf ?di)

note mfu = simple-match-port.simps[of fst (sports r) snd (sports r), unfolded surjective-pairing[of sports r,symmetric]]

simple-match-port.simps[of fst (dports r) snd (dports r), unfolded surjective-pairing[of dports r,symmetric]]

note u = mm[unfolded simple-matches.simps mfu ord-class.atLeastAtMost-iff simple-packet-unext-def simple-packet.simps]

note of-safe-unsafe-match-eq[OF simple-match-to-of-match-generates-prereqs]

from u have ple: fst (sports r) ≤ snd (sports r) fst (dports r) ≤ snd (dports r) by force+

show ?thesis:

unfolding simple-match-to-of-match-def

unfolding custom-simpset

unfolding sndoms-eq-hlp

proof (intro bezI, (intro conjI; ((rule refl)?), goal-cases)

case 2 thus ?case using ple(2) di

apply (simp add: pfm-mask-def prefix-match-dtor-def Set.image_iff

split: option.splits prod.splits uncurry-splits)

apply (erule bezI[rotated])

apply (simp split: prefix-match.splits)

done

next
case 3  thus ?case using p(1) si
  apply(simp add: pfxm-mask-def prefix-match-dtor-def Set.image-iff
    split: option.splits prod.splits uncurry-splits)
  apply(erule bexI[rotated])
  apply(simp split: prefix-match.splits)
  done
next
case 4  thus ?case
  using u i by(clarsimp simp: set-maps split: if-splits)
next
case 1  thus ?case using u u by simp-all (metis match,proto.elims(2))
qed
have dpm: di = Some (PrefixMatch x1 x2) \implies p-dport p & & ~ ~ (mask (16 − x2)) = x1  for x1 x2
proof −
  have *: di = Some (PrefixMatch x1 x2) \implies prefix-match-semantics (the di) (p-dport p) \implies p-dport p & & ~ ~ (mask (16 − x2)) = x1
    by(clarsimp simp: prefix-match-semantics-def pfxm-mask-def word-bw-comm:s fail)
  have **: pfx ∈ set (wordinterval-CIDR-split-prefixmatch ra) \implies prefix-match-semantics pfx a = (a ∈ prefix-to-wordset pfx)
    for pfx ra and a :: 16 word
    by (fact prefix-match-semantics-wordset[OF wordinterval-CIDR-split-prefixmatch-all-valid-Ball THEN bspec, THEN conjunct(1)])
  have [di = Some (PrefixMatch x1 x2); p-dport p ∈ prefix-to-wordset (PrefixMatch x1 x2); PrefixMatch x1 x2 ∈ set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (dports r)))]
    \implies p-dport p & & ~ ~ (mask (16 − x2)) = x1
  using di(1,2)
  using * ** by auto
  thus di = Some (PrefixMatch x1 x2) \implies p-dport p & & ~ ~ (mask (16 − x2)) = x1  using di(1,2) by auto
qed
have spm: si = Some (PrefixMatch x1 x2) \implies p-sport p & & ~ ~ (mask (16 − x2)) = x1  for x1 x2
using si
proof −
  have *: si = Some (PrefixMatch x1 x2) \implies prefix-match-semantics (the si) (p-sport p) \implies p-sport p & & ~ ~ (mask (16 − x2)) = x1
    by(clarsimp simp: prefix-match-semantics-def pfxm-mask-def word-bw-comm:s fail)
  have **: pfx ∈ set (wordinterval-CIDR-split-prefixmatch ra) \implies prefix-match-semantics pfx a = (a ∈ prefix-to-wordset pfx)
    for pfx ra and a :: 16 word
    by (fact prefix-match-semantics-wordset[OF wordinterval-CIDR-split-prefixmatch-all-valid-Ball THEN bspec, THEN conjunct(1)])
  have [si = Some (PrefixMatch x1 x2); p-sport p ∈ prefix-to-wordset (PrefixMatch x1 x2); PrefixMatch x1 x2 ∈ set (wordinterval-CIDR-split-prefixmatch (uncurry WordInterval (sports r)))]
    \implies p-sport p & & ~ ~ (mask (16 − x2)) = x1
  using si(1,2)
  using * ** by auto
  thus si = Some (PrefixMatch x1 x2) \implies p-sport p & & ~ ~ (mask (16 − x2)) = x1  using si(1,2) by auto
qed
show OF-match-fields ?gr p = Some True
unfolding af-safe-unsafe-match-eq[OF simple-match-to-of-match-generates-prereqs[OF mv eq]]
by(cases si; cases di)
(simp-all
  add: simple-match-to-of-match-single-def OF-match-fields-unsafe-def spm
  option2set-def u ipkt prefix-match-dtor-def toprefixmatch-def dpm
  simple-match-dst-alt[OF mv, symmetric] simple-match-src-alt[OF mv, symmetric]
  split: prefix-match.splits)
qed
qed

lemma prefix-match-00 [simp, intro!]: prefix-match-semantics PrefixMatch 0 0 p
  by (simp add: valid-prefix-def zero-prefix-match-all)

lemma simple-match-to-of-matchD:
  assumes eg: gr ∈ set (simple-match-to-of-match r ifs)
  assumes mo: OF-match-fields gr p = Some True
  assumes me: match-iface (oiface r) (p-oiface p)
  assumes mv: simple-match-valid r
  shows simple-matches r p
proof -
  from mv have validpfx:
    valid-prefix (uncurry PrefixMatch (src r)) valid-prefix (uncurry PrefixMatch (dst r))
    \forall pm. toprefixmatch (src r) = Some pm \implies valid-prefix pm
    \forall pm. toprefixmatch (dst r) = Some pm \implies valid-prefix pm
  unfolding simple-match-valid-def valid-prefix-fu-def toprefixmatch-def
  by (simp-all split: uncurry-splits if-splits)
  from mo have mo: OF-match-fields-unsafe gr p
    unfolding OF-safe-unsafe-match-eq[OF simple-match-to-of-match-generates-prereqs[OF mv eg]]
  by simp
  note this[unfolded OF-match-fields-unsafe-def]
  note eg[unfolded simple-match-to-of-match-def simple-match-to-of-match-single-def custom-simpset option2set-def]
  then guess x .. moreover from this[2] guess xa .. moreover from this[2] guess xb ..
  note xx = calculation(1,3) this

  { fix a b xc xa
    fix pp :: 16 word
    have [pp & \& ~\& (pfm-mask xc) = pfsm-prefix xc]
      \implies prefix-match-semantics xc (pp) for xc
      by (simp add: prefix-match-semantics-def word-bw-comms; fail)
    moreover have pp \in wordinterval-to-set (WordInterval a b) \implies a \leq pp \land pp \leq b by simp
    moreover have xc \in set (wordinterval-CIDR-split-prefixmatch (WordInterval a b)) \implies pp \in prefix-to-wordset xc \implies pp \in wordinterval-to-set (WordInterval a b)
      by (subst wordinterval-CIDR-split-prefixmatch) blast
    moreover have [xc \in set (wordinterval-CIDR-split-prefixmatch (WordInterval a b)); xa = Some (pfsm-prefix xc, ~\& (pfsm-mask xc)); prefix-match-semantics xc (pp)] \implies pp \in prefix-to-wordset xc
      apply (subst (asm(1)) prefix-match-semantics-wordset)
      apply (erule wordinterval-CIDR-split-prefixmatch-all-valid-Ball[THEN bspec, THEN conjunct1]; fail)
    apply assumption
  done
  ultimately have [xc \in set (wordinterval-CIDR-split-prefixmatch (WordInterval a b)); xa = Some (pfsm-prefix xc, ~\& (pfsm-mask xc));
    pp & \& ~\& (pfsm-mask xc) = pfsm-prefix xc]
  \implies a \leq pp \land pp \leq b
  by metis
  } note l4port-logic = this

show ?thesis unfolding simple-matches.simps
proof (unfold and-assoc, (rule)+)
show match-iface (iiface r) (p-iiface p)
apply (cases iiface r = iface.Any)
apply (simp add: match-ifaceAny)
using xx(1) mo unfolding xx(4) OF-match-fields-unsafe-def
apply(simp only: if-False set-maps UN-iff)
apply(clarify)
apply(rename-tac a; subgoal-tac match-iface (iiface r) a)
apply(clarsimp simp add: option2set-def fail)
apply(rule ccontr, simp; fail)
done
next
show match-iface (oiface r) (p-oiface p) using me.
next
show simple-match-ip (src r) (p-src p)
using mo unfolding xx(4) OF-match-fields-unsafe-def toprefixmatch-def
by(clarsimp
  simp add: simple-packet-unext-def option2set-def validpfx simple-match-src-alt
  split: if-splits)
next
show simple-match-ip (dst r) (p-dst p)
using mo unfolding xx(4) OF-match-fields-unsafe-def
using xx(1) by(clarsimp
  simp add: singleton-iff simple-packet-unext-def option2set-def prefix-match-semantics-simple-match ball-Un
  split: if-splits)
next
show simple-match-port (sports r) (p-sport p)
using mo xx(2) unfolding xx(4) OF-match-fields-unsafe-def
by(cases sports r) (clarsimp simp add: l4port-logic simple-packet-unext-def	
  option2set-def prefix-match-semantics-simple-match split: if-splits)
next
show simple-match-port (dports r) (p-dport p)
using mo xx(3) unfolding xx(4) OF-match-fields-unsafe-def
by(cases dports r) (clarsimp simp add: l4port-logic simple-packet-unext-def option2set-def
  prefix-match-semantics-simple-match split: if-splits)
next

primrec annotate-rlen where
  annotate-rlen [] = []
  annotate-rlen (a#as) = (length as, a) # annotate-rlen as

lemma annotate-rlen "asdf" = [3, CHR "a"], (2, CHR "s"), (1, CHR "d"), (0, CHR "f") by simp

lemma fst-annotate-rlen-le: (k, a) ∈ set (annotate-rlen l) ⇒ k < length l
  by(induction l arbitrary: k; simp; force)

lemma distinct-fst-annotate-rlen: distinct (map fst (annotate-rlen l))
  using fst-annotate-rlen-le by(induction l) (simp, fastforce)
lemma distinct-annotate-rlen: distinct (annotate-rlen l)
  using distinct-fst-annotate-rlen unfolding distinct-map by blast
lemma in-annotate-rlen: (a,x) ∈ set (annotate-rlen l) ⇒ x ∈ set l
  by(induction l) (simp-all, blast)
lemma map-snd-annotate-rlen: map snd (annotate-rlen l) = l

24
by (induction l) simp-all

lemma sorted-descending (map fst (annotate-rlen l))
  by (induction l; clarsimp) (force dest: fst-annotate-rlen-le)

lemma annotate-rlen l = zip (rev [0..<length l]) l
  by (induction l; simp)

primrec annotate-rlen-code where
  annotate-rlen-code [] = []
  annotate-rlen-code (a # as) = (case annotate-rlen-code as of (r,aas) ⇒ (Suc r, (r, a) # aas))

lemma annotate-rlen-len: fst (annotate-rlen-code r) = length r
  by (induction r) (clarsimp split: prod.splits)+

lemma annotate-rlen-code[code]: annotate-rlen s = snd (annotate-rlen-code s)
  proof (induction s)
    case (Cons ss) thus ?case using annotate-rlen-len[of ss] by (clarsimp split: prod.split)
  qed simp

lemma suc2plus-inj-on: inj-on (of-nat :: nat ⇒ ('l :: len) word) {0..unat (max-word :: 'l word)}
  proof (rule inj-onI)
    let ?mmw = (max-word :: 'l word)
    let ?mstp = (of-nat :: nat ⇒ 'l word)
    fix x y :: nat
    assume x ∈ {0..unat ?mmw} y ∈ {0..unat ?mmw}
    hence se: x ≤ unat ?mmw y ≤ unat ?mmw by simp-all
    assume eq: ?mstp x = ?mstp y
    note f = le-unat-uoi[OF se(1)] le-unat-uoi[OF se(2)]
    show x = y using eq le-unat-uoi se by metis
  qed

lemma distinct-of-nat-list:
  distinct l ⇒ ∀ e ∈ set l. e ≤ unat (max-word :: ('l::len) word) ⇒ distinct (map (of-nat :: nat ⇒ 'l word) l)
  proof (induction l)
    let ?mmw = (max-word :: 'l word)
    let ?mstp = (of-nat :: nat ⇒ 'l word)
    case (Cons a as)
    have distinct as ∀ e ∈ set as. e ≤ unat ?mmw using Cons.prems by simp-all
    note mIH = Cons.IH[OF this]
    moreover have ?mstp a ∉ ?mstp ' set as
    proof
      have representable-set: set as ⊆ {0..unat ?mmw} using ∀ e ∈ set (a # as). e ≤ unat max-word by fastforce
      have a-reprbl: a ∈ {0..unat ?mmw} using ∀ e ∈ set (a # as). e ≤ unat max-word by simp
      assume ?mstp a ∈ ?mstp ' set as
      with inj-on-image-mem-iff[OF suc2plus-inj-on a-reprbl representable-set]
      have a ∈ set as by simp
      with distinct (a # as) show False by simp
    qed
    ultimately show ?case by simp
  qed simp

lemma annotate-first-le-hlp:
  length l < unat (max-word :: ('l :: len) word) ⇒ ∀ e ∈ set (map fst (annotate-rlen l)). e ≤ unat (max-word :: 'l word)
  by (clarsimp simp (meson fst-annotate-rlen-le less_trans nat_less_le))

lemmas distinct-of-prio-hlp = distinct-of-nat-list[OF distinct-fst-annotate-rlen annotate-first-le-hlp]
lemma fst-annotate-rflen: map fst (annotate-rflen l) = rev [0..<length l]
by(induction l) (simp-all)

lemma sorted-word-upt:
  defines[simp]: won ≡ (of-nat :: nat ⇒ ('l :: len) word)
  assumes length l ≤ unat (max-word :: 'l word)
  shows sorted-descending (map won (rev [0..<Suc (length l)]))
using assms
  by(induction l rule: rev-induct; clarsimp)

lemma sorted-annotated:
  assumes length l ≤ unat (max-word :: ('l :: len) word)
  shows sorted-descending (map fst (map (apfst (of-nat :: nat ⇒ ('l :: len) word)) (annotate-rflen l)))
proof
  let ?won = (of-nat :: nat ⇒ ('l :: len) word)
  have sorted-descending (map ?won (rev [0..<Suc (length l)]))
  using sorted-word-upt[OF assms].
  hence sorted-descending (map ?won (map fst (annotate-rflen l))))
  by(simp add: fst-annotate-rflen)
  thus sorted-descending (map fst (map (apfst ?won) (annotate-rflen l))))
  by simp
qed

l3 device to l2 forwarding

definition lr-of-tran-s3 ifs ard = ([(p, b, case a of simple-action.Accept ⇒ [Forward c] | simple-action.Drop ⇒ [])].(p,r,(c,a)) ← ard, b ← simple-match-to-of-match r ifs)

definition oif-ne-iif-p1 ifs ≡ [(simple-match-any (oiface := Iface oif, iiface := Iface iif), simple-action.Accept), oif ← ifs, iif ← ifs, oif ≠ iif]

definition oif-ne-iif-p2 ifs = [(simple-match-any (oiface := Iface i, iiface := Iface i), simple-action.Drop). i ← ifs]

definition oif-ne-iif ifs = oif-ne-iif-p2 ifs @ oif-ne-iif-p1 ifs

definition lr-of-tran-s4 ard ifs = generalized-fw-join ard (oif-ne-iif ifs)

definition lr-of-tran-s1 rt = [(route2match r, output-iface (routing-action r))].r ← rt

definition lr-of-tran-fbs rt fw ifs ≡ let
  gfw = map simple-rule-dtor fw; — generalized simple fw, hopefully for FORWARD
  frt = lr-of-tran-s1 rt; — rt as fw
  prd = generalized-fw-join frt gfw
  in prd

definition pack-OF-entries ifs ard ≡ (map (split3 OFEntry) (lr-of-tran-s3 ifs ard))

definition no-oif-match ≡ list-all (λm. oiface (match-sel m) = ifaceAny)

definition lr-of-tran rt fw ifs ≡
  if ¬ (no-oif-match fw ∧ has-default-policy fw ∧ simple-fw-valid fw ∧ valid-prefixes rt ∧ has-default-route rt ∧ distinct ifs)
  then Inl "Error in creating OpenFlow table: prerequisites not satisfied"
  else (let nrd = lr-of-tran-fbs rt fw ifs;
  ard = map (apfst (of-nat)) (annotate-rflen nrd) — give them a priority
  qed
proof
-
lemma distinct-simple-match-to-of-match-portlist-hlp
qed
∈{
y True
word
proof
-
lemma max-16-word-max[simp]: (a :: 16 word) ≤ 0xffff
proof –
have 0xFFFF = (- 1 :: 16 word) by simp
then show ?thesis by (simp only: simp)
qed

lemma replicate-FT-hlp: x ≤ 16 ∧ y ≤ 16 ⇒ replicate (16 – x) False ⊕ replicate x True = replicate (16 – y) False ⊕ replicate y True
⇒ x = y
proof –
let ?ns = \{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16\}
assume x ≤ 16 ∧ y ≤ 16
hence x ∈ ?ns y ∈ ?ns by(simp; presburger)+
moreover assume replicate (16 – x) False ⊕ replicate x True = replicate (16 – y) False ⊕ replicate y True
ultimately show x = y by simp (elim disjE; simp-all add: numeral-eq-Suc)
qed

lemma mask-inj-hlp1: inj-on (mask :: nat ⇒ 16 word) {0..16}
proof(intro inj-onI, goal-cases)
  case (1 x y)
  from 1(3)
  have oe: of-bl (replicate (16 – x) False ⊕ replicate x True) = (of-bl (replicate (16 – y) False ⊕ replicate y True) :: 16 word)
    unfolding mask-bl of-bl-rep-False .
  have \(\land_z z ≤ 16 ⇒ length (replicate (16 – z) False ⊕ replicate z True) = 16\) by auto
  with 1(1,2)
  have ps: replicate (16 – x) False ⊕ replicate x True ∈ \{bl. length bl = LENGTH(16)\} replicate (16 – y) False ⊕ replicate y True ∈ \{bl. length bl = LENGTH(16)\} by simp-all
  from inj-onD[OF word-bl.Abs-inj-on, OF oe ps]
  show ?case using 1(1,2) by (fastforce intro: replicate-FT-hlp)
qed

lemma distinct-simple-match-to-of-match-portlist-hlp:
  fixes ps :: (16 word × 16 word)
  shows distinct ifs
  proof
    distinct
    (if fst ps = 0 ∧ snd ps = max-word then [None]
    else if fst ps ≤ snd ps
      then map (Some ∘ (λpfz. (pfzm-prefix pfz. ~`` (pfzm-mask pfz))))
         (wordinterval-CIDR-split-prefixmatch (WordInterval (fst ps) (snd ps)))
    else [])
  qed
-
  assume di: distinct ifs
  { define wis where wis = set (wordinterval-CIDR-split-prefixmatch (WordInterval (fst ps) (snd ps)))
  fix x y :: 16 prefix-match
  obtain x m x n y m y n where xym[flip]: x = PrefixMatch x m x n y = PrefixMatch y m y n by(cases x; cases y)

27
assume \( iu \in \text{wis} \) and \( et \in (\text{pfrm-prefix } x, \text{~} (\text{pfrm-mask } x)) = (\text{pfrm-prefix } y, \text{~} (\text{pfrm-mask } y)) \)

hence \( le16 \): \( xn \leq 16 \) and \( yn \leq 16 \) unfolding \( \text{wis-def using wordinterval-CIDR-split-prefixmatch-all-valid-Ball[unfolded Ball-def, THEN spec, THEN mp]} \) by force+

with \( et \) have \( 16 - xn = 16 - yn \) unfolding \( \text{pfrm-mask-def by(auto intro: mask-inj-hlp1[THEN inj-onD])} \)

hence \( x = y \) using \( et le16 \) unfolding \( \text{diff-diff-cancel by simp} \)

} note \( \ast = \text{this} \)

show \( \text{thesis} \)

apply(clarsimp simp add: smtoms-eq-hlp distinct-map wordinterval-CIDR-split-distinct)

apply(subst comp-inj-on-iff [symmetric]; intro inj-onI)

using \( \ast \) by simp-all

qed

lemma \( \text{distinct-simple-match-to-of-match}: \text{distinct ifs} \Rightarrow \text{distinct (simple-match-to-of-match m ifs)} \)

apply(unfold \( \text{simple-match-to-of-match-def Let-def} \))

apply(rule distinct-3compl1)

subgoal by(induction ifs; clarsimp)

subgoal by(fact distinct-simple-match-to-of-match-portlist-hlp)

subgoal by(fact distinct-simple-match-to-of-match-portlist-hlp)

subgoal by(simp-all add: smtoms-eq-hlp)

done

lemma \( \text{inj-inj-on}: \text{inj } F \Rightarrow \text{inj } F \text{ A using } \text{subset-inj-on by auto} \)

lemma \( \text{no-overlaps-lroft-hlp2}: \text{distinct (map fst amr)} \Rightarrow (\forall r. \text{distinct } (\text{fn } r)) \Rightarrow \text{distinct (concat (map (\lambda (p, r, c, a). map (\lambda (b, p, b, f s a c) ) (\text{fn } r)) amr))} \)

by(induction amr; force intro: inj1 inj-onI simp add: distinct-map split: prod.splits)

lemma \( \text{distinct-lroft-s3}: \text{distinct (map fst amr); distinct ifs} \Rightarrow \text{distinct (br-of-tran-s3 ifs amr)} \)

unfolding \( \text{br-of-tran-s3-def} \)

by(erule no-overlaps-lroft-hlp2; simp add: distinct-simple-match-to-of-match)

lemma \( \text{no-overlaps-lroft-hlp3}: \text{distinct (map fst amr)} \Rightarrow (a, a, ab, ac) \in \text{set (br-of-tran-s3 ifs amr)} \Rightarrow (ba, bb, bc) \in \text{set (br-of-tran-s3 ifs amr)} \Rightarrow ac \neq bc \Rightarrow aa \neq ba \)

apply(unfold \( \text{br-of-tran-s3-def} \))

apply(clarsimp)

apply(clarsimp split: simple-action.splits)

apply(metis map-of-eq-Some-iff old.prod.inject option.inject)

apply(metis map-of-eq-Some-iff old.prod.inject option.inject simple-action.distinct(2))

done

lemma \( \text{no-overlaps-lroft-s3-hlp}: \)

[distinct (map fst amr); \( \text{OF-match-fields-unsafe ab p; ab} \neq \text{ad} \Rightarrow \text{ba} \neq \text{bb}; \text{OF-match-fields-unsafe ad p; (ac, ab, ba)} \in \text{set (br-of-tran-s3 ifs amr); (ac, ad, bb)} \in \text{set (br-of-tran-s3 ifs amr)} \)]

\( \Rightarrow \text{False} \)

proof(elim disjE, goal-cases)

case 1

have 4: \( \text{distinct (map fst amr); (ac, ab, x1, x2) \in set amr; (ac, bb, x4, x5) \in set amr; ab} \neq \text{bb} \)

\( \Rightarrow \text{False for } ab \ x1 \ x2 \ bb \ x4 \ x5 \)

by (meson distinct-map-fstD old.prod.inject)

have conjunctSomeProtoAnyD: Some ProtoAny = simple-proto-conjunct a (Proto b) \( \Rightarrow \text{False for a b} \)

using conjunctProtoD by force

have 5:

\( \text{OF-match-fields-unsafe am p; OF-match-fields-unsafe bm p; am} \neq \text{bm;} \)

28
am ∈ set (simple-match-to-of-match ab ifs); bm ∈ set (simple-match-to-of-match bb ifs); ¬ ab ≠ bb]
⇒ False for ab bb am bm

by (clarify | unfold

simple-match-to-of-match def smtoms-eq-hlp Let-def set-concat set-map de-Morgan-conj not-False-eq-True)+
(auto dest: conjunctSomeProtoAnyD cidr split-no-overlaps
split: if-splits
cong: smtoms-eq-hlp)

from / show 2case
using 4 5 by(clarsimp simp add: lr-of-tran-s3-def) blast
qed (metis no-overlaps-lroft-hlp3)

lemma no-overlaps-lroft-s3-hlp: distinct (map fst amr) ⇒ distinct ifs ⇒
no-overlaps OF-match-fields-unsafe (map (split3 OFEntry) (lr-of-tran-s3 ifs amr))
apply (rule no-overlapsI[rotated])
apply (subst distinct-map, rule conjI)
subgoal by (erule (1) distinct-lroft-s3)
subgoal
apply (rule inj-inj-on)
apply (rule injI)
apply (rename-tac x y, case-tac x, case-tac y)
apply (clarsimp simp add: split3-def fail)
done
subgoal
apply (unfold check-no-overlap-def)
apply (clarsimp)
apply (unfold set-map)
apply (clarsimp)
apply (unfold split3-def prod simps flow-entry-match.simps flow-entry-match.sel de-Morgan-conj)
apply (clarsimp simp only:)
apply (erule (1) no-overlaps-lroft-s3-hlp-hlp)
apply simp
apply assumption
apply assumption
apply simp
done
done

lemma lr-of-tran-no-overlaps: assumes distinct ifs shows Inr t = (lr-of-tran rt fw ifs) ⇒ no-overlaps
OF-match-fields-unsafe t
apply (unfold lr-of-tran-def Let-def pack-OF-entries-def)
apply (clarsimp simp split: if-splits)
apply (thin-tac t = -)
apply (drule distinct-of-prio-hlp)
apply (rule no-overlaps-lroft-s3-hlp[rotated])
subgoal by (simp add: assms)
subgoal by (simp add: a-associ)
done

lemma sorted-lr-of-tran-s3-hlp: ∀ x∈ set f. fst x ≤ a ⇒ b ∈ set (lr-of-tran-s3 f s) ⇒ fst b ≤ a
by (auto simp add: lr-of-tran-s3-def)

lemma lr-of-tran-s3-Cons: lr-of-tran-s3 ifs (a#ard) = (}
\[(p, b, \text{case a of simple-action.Accept} \Rightarrow [\text{Forward c}] | \text{simple-action.Drop} \Rightarrow [])\].

\((p, r, (c, a)) \leftarrow [a], b \leftarrow \text{simple-match-to-of-match r ifs}) @ \text{lr-of-tran-s3 ifs ard}

by(clarsimp simp: lr-of-tran-s3-def)

lemma sorted-lr-of-tran-s3: sorted-descending (map fst f) \implies\!\!\implies sorted-descending (map fst (lr-of-tran-s3 s f))
apply(induction f)
subgoal by(simp add: lr-of-tran-s3-def)
apply(clarsimp simp: lr-of-tran-s3-Cons map-concat comp-def)
apply(unfold sorted-descending-append)
apply(simp add: sorted-descending-alt rev-map sorted-lr-of-tran-s3-hlp sorted-const)
done

lemma sorted-lr-of-tran-hlp: (afe-prio \circ split3 OFEntry) = fst by(simp add: fun-eq-iff comp-def split3-def)

lemma lr-of-tran-sorted-descending: Inr r = lr-of-tran rt fw \implies\!\!\implies sorted-descending (map ofe-prio r)
apply(unfold lr-of-tran-def Let-def)
apply(simp split: if-splits)
apply(thin-tac r = -)
apply(unfold sorted-lr-of-tran-hlp pack-OF-entries-def split3-def [abs-def]
fun-app-def map-map comp-def prod.case-distrib)
apply(simp add: fst-def [symmetric])
apply(rule sorted-lr-of-tran-s3)
done

subgoal by(clarsimp simp: lr-of-tran-s3-def)
apply(case-tac a)
apply(rename-tac routing-m metric routing-action)
apply(case-tac routing-m)
apply(simp add: valid-prefix-def pfzm-mask-def prefix-match-semantics-def generalized-sfw-def
lr-of-tran-s1-def route2match-def simple-matches.simps match-ifaceAny match-iface-refl ipset-from-cidr-0
prefix-match-semantics-ipset-from-netmask2)

lemma s1-correct: valid-prefixes rt \implies\!\!\implies has-default-route (rt::('i::len) prefix-routing) \implies\!\!\implies
\exists rm ra. generalized-sfw (lr-of-tran-s1 rt) p = Some (rm,ra) \land ra = output-iface (routing-table-semantics rt (p-dst p))
apply(induction rt)
apply(simp; fail)
apply(drule valid-prefixes-split)
apply(clarsimp simp: lr-of-tran-s1-def route2match-correct)
apply(case_tac a)
apply(rename-tac routing-m metric routing-action)
apply(case_tac routing-m)
apply(simp add: valid-prefix-def pfzm-mask-def prefix-match-semantics-def generalized-sfw-def
lr-of-tran-s1-def route2match-def simple-matches.simps match-ifaceAny match-iface-refl ipset-from-cidr-0
max-word-mask [where 'a = 'i, symmetric, simplified])
done

subgoal for a rt
apply(case_tac a)
apply(rename-tac routing-m metric routing-action)
apply(case_tac routing-m)
apply(simp add: valid-prefix-def pfzm-mask-def prefix-match-semantics-def generalized-sfw-def
lr-of-tran-s1-def route2match-def simple-matches.simps match-ifaceAny match-iface-refl ipset-from-netmask2
lr-of-tran-s1-split generalized-sfw-simps)

done
definition to-OF-action a ≡ (case a of (p, d) ⇒ (case d of simple-action.Accept ⇒ [Forward p] | simple-action.Drop ⇒ []))
definition from-OF-action a = (case a of [] ⇒ (""", simple-action.Drop) | [Forward p] ⇒ (p, simple-action.Accept))

lemma OF-match-linear-not-noD: OF-match-linear γ p ≠ NoAction ⇒ ∃ ome. ome ∈ set oms ∧ γ (ofe-fields ome) p
apply(induction oms)
apply(simp)
apply(simp split: if-splits)
apply blast+ done

lemma s3-noaction-hlp: [simple-match-valid ac; ¬ simple-matches ac p; match-iface (oiface ac) (p-oiface p)] ⇒ OF-match-linear OF-match-fields-safe (map (λ x. split3 OFEntry (x1, x, case ba of simple-action.Accept ⇒ [Forward ad] | simple-action.Drop ⇒ [])) (simple-match-to-of-match ac ifs)) p = NoAction
apply(rule ccontr)
apply(drule OF-match-linear-not-noD)
apply(clarsimp)
apply(rename-tac x)
apply(subgoal-tac all-prerequisites x)
apply(drule simple-match-to-of-matchD)
apply(simp add: split3-def)
apply(subst(asm) of-match-fields-safe-eq2)
apply(simp fail)+ using simple-match-to-of-match-generates-prereqs by blast

lemma aux:\n\langle v = Some x ⇒ the v = x⟩
by simp

lemma s3-correct:\nassumes vsfwm: list-all simple-match-valid (map (fst o snd) ard)
assumes ippkt: p-l2type p = 0x800
assumes iiifs: p-iiface p ∈ set ifs
assumes oiifs: list-all (λ m. oiface (fst (snd m)) = ifaceAny) ard
shows OF-match-linear OF-match-fields-safe (pack-OF-entries ifs ard) p = Action ao \iff (∃ r af. generalized-sfw (map snd ard) p = (Some (r, af)) ∧ (if snd af = simple-action.Drop then ao = [] else ao = [Forward (fst af)]))

unfolding pack-OF-entries-def lr-of-tran-s3-def fun-app-def
using vsfwm oiifs
apply(induction ard)

subgoal by(simp add: generalized-sfw-simps)
apply simp
apply(clarsimp simp add: generalized-sfw-simps split: prod.splits)
apply(intro conjI)

subgoal for ard x1 ac ad ba
apply(clarsimp simp add: OF-match-linear-append split: prod.splits)
apply(drule simple-match-to-of-matchI[rotated])
  apply(rule iiifs)
  apply(rule ippkt)
  apply blast
apply(clarsimp simp add: comp-def)
apply(drule OF-match-linear-match-allsameaction[where γ=OF-match-fields-safe and pri = x1 and
\[ oms = \text{simple-match-to-of-match ac ifs} \quad \text{and} \quad \\text{act = case ba of simple-action.Accept} \Rightarrow [\text{Forward ad}] \quad | \quad \text{simple-action.Drop} \Rightarrow [] \] 

apply(unfold OF-match-fields-safe-def comp-def)
apply(erule aux)
apply(intro iffI)
subgoal
apply(rule exI[where \( x = ac \)])
apply(rule exI[where \( x = ad \)])
apply(rule exI[where \( x = ba \)])
apply(clarsimp simp: split3-def split: simple-action.splits flowtable-behavior.splits if-splits)
done
subgoal
apply(clarsimp)
apply(rename-tac b)
apply(case-tac b)
apply(simp-all)
done
done

subgoal for ar\(d\) z1 ac ad ba
apply(simp add: OF-match-linear-append OF-match-fields-safe-def comp-def)
apply(clarify)
apply(subgoal-tac OF-match-linear OF-match-fields-safe (map (\( \lambda x. \) split3 OFEntry \( x1, x \), case ba of simple-action.Accept \( \Rightarrow [\text{Forward ad}] \quad | \quad \text{simple-action.Drop} \Rightarrow [] \)) (simple-match-to-of-match ac ifs)) \( p = \text{NoAction} \)
apply(simp; fail)
apply(erule (1) s3-noaction-hlp)
apply(simp add: match-ifaceAny; fail)
done
done

case
notes valid-prefix-00[simp, intro!]
begin
lemma lr-of-tran-s1-valid: valid-prefixes rt \( \Rightarrow \) gsfw-valid (lr-of-tran-s1 rt)
unfolding lr-of-tran-s1-def route2match-def gsfw-valid-def list-all-iff
apply(clarsimp simp: simple-match-valid-def valid-prefix-fw-def)
apply(intro conjI)
apply(force)
apply(simp add: valid-prefixes-alt-def)
done
end

lemma simple-match-valid-fbs-rlen: \([\text{valid-prefixes rt}; \text{simple-fw-valid fw}; (a, aa, ab, b) \in \text{set (annotate-rlen (lr-of-tran-fbs rt fw ifs))}] \) \( \Rightarrow \) simple-match-valid aa
proof(goal-cases)
case 1
note 1 [unfolded lr-of-tran-fbs-def Let-def]
have gsfw-valid (map simple-rule-dtor fw) using gsfw-validI 1 by blast
moreover have gsfw-valid (lr-of-tran-s1 rt) using 1 lr-of-tran-s1-valid by blast
ultimately have gsfw-valid (generalized-fw-join (lr-of-tran-s1 rt) (map simple-rule-dtor fw)) using gsfw-join-valid by blast
moreover have (aa, ab, b) \( \in \) set (lr-of-tran-fbs rt fw ifs) using 1 using in-annotate-rlen by fast
ultimately show \( ? \)thesis unfolding lr-of-tran-fbs-def Let-def gsfw-valid-def list-all-iff by fastforce
qed
lemma simple-match-valid-fbs: \([\text{valid-prefixes } rt; \text{simple-fw-valid } fw] \implies \text{list-all } \text{simple-match-valid} (\text{map } \text{fst} (\text{lr-of-tran-fbs} \ rt \ fw \ ifs))\)

proof (goal-cases)

  case 1

  note 1 [unfolded lr-of-tran-fbs-def Let-def]

  have gsfw-valid (map simple-rule-dtor fw) using gsfw-validI 1 by blast
  moreover have gsfw-valid (lr-of-tran-s1 rt) using 1 lr-of-tran-s1-valid by blast
  ultimately have gsfw-valid (generalized-fw-join (lr-of-tran-s1 rt) (map simple-rule-dtor fw)) using gsfw-join-valid by blast
  thus ?thesis unfolding lr-of-tran-fbs-def Let-def gsfw-valid-def list-all-iff by fastforce
qed

lemma lr-of-tran-prereqs: \([\text{valid-prefixes } rt \implies \text{simple-fw-valid } fw \implies \text{lr-of-tran } rt \ fw \ ifs \implies \text{Inr } oft \implies \text{list-all } (\text{all-prerequisites } \circ \text{ofe-fields}) \ oft)\]

unfolding lr-of-tran-def pack-OF-entries-def lr-of-tran-s3-def Let-def
apply (simp add: map-concat comp-def prod.case-distrib split3-def split: if-splits)
apply (clarsimp simp add: list-all-iff)
apply (drule simple-match-valid-fbs-rlen [rotated])
apply (simp add: list-all-iff fail)
apply (clarsimp simp add: list-all-iff fail)
apply (rule simple-match-to-of-match-generates-prereqs; assumption)
done

lemma OF-unsafe-safe-match3-eq:
[\text{list-all } (\text{all-prerequisites } \circ \text{ofe-fields}) \ oft \implies \text{OF-priority-match } \text{OF-match-fields-unsafe } \ oft \implies \text{OF-priority-match } \text{OF-match-fields-safe } \ oft\]

unfolding OF-priority-match-def [abs-def]
proof (goal-cases)

  case 1

  from 1 have \(\\forall \text{packet}. \ l. \text{OF-match-fields-unsafe } (\text{ofe-fields } f) \text{ packet} \ = \ l. \text{OF-match-fields-safe } (\text{ofe-fields } f) \text{ packet}\)

  apply (clarsimp simp add: list-all-iff OF-match-fields-safe-eq)
  using OF-match-fields-safe-eq by (metis mono-tags lifting filter-cong)
  thus ?case bymetis
qed

lemma OF-unsafe-safe-match-linear-eq:
[\text{list-all } (\text{all-prerequisites } \circ \text{ofe-fields}) \ oft \implies \text{OF-match-linear } \text{OF-match-fields-unsafe } \ oft \implies \text{OF-match-linear } \text{OF-match-fields-safe } \ oft\]

unfolding fun-eq-iff
by (induction oft) (clarsimp simp add: list-all-iff OF-match-fields-safe-eq)+

lemma simple-action-ne [simp]:
  \(b \neq \text{simple-action.Accept} \leftrightarrow b = \text{simple-action.Drop}\)
  \(b \neq \text{simple-action.Drop} \leftrightarrow b = \text{simple-action.Accept}\)

using simple-action.exhaust by blast+

lemma map-snd-apfst: \(\text{map } \text{snd} (\text{map } (\text{apfst } x) \ l) = \text{map } \text{snd} \ l\)

unfolding map-map comp-def snd-apfst ..

lemma match-ifaceAny-eq: \(\text{oiface } m = \text{ifaceAny } \implies \text{simple-matches } m \ p = \text{simple-matches } m \ (p(\text{p-oiface } := \text{any}))\)

by (cases m) (simp add: simple-matches.simps match-ifaceAny)

lemma no-oif-matchD: \(\text{no-oif-match } fw \implies \text{simple-fw } fw \ p = \text{simple-fw } fw \ (p(\text{p-oiface } := \text{any}))\)

33
by (induction \( \text{fw} \))
(auto simp add: no-oif-match-def simple-fw-alt dest: match-iifaceAny-eq)

**lemma** \( \text{lr-of-tran-fbs-acceptD} \):
\begin{align*}
& \text{assumes } s1: \text{valid-prefixes } rt \text{ has-default-route } rt \\
& \text{assumes } s2: \text{no-oif-match } \text{fw} \\
& \text{shows } \text{generalized-sfw} (\text{lr-of-tran-fbs } rt \text{ fw ifs}) \ p = \text{Some } (r, \text{ oif}, \text{ simple-action.Accept}) \implies \\
& \text{simple-linux-router-nol12 } rt \text{ fw p} = \text{Some } (p[p-oiface := \text{oif}])
\end{align*}
**proof** (goal-cases)
\begin{align*}
& \text{case } 1 \\
& \text{note } 1 [\text{unfolded lr-of-tran-fbs-def Let-def, THEN generalized-fw-joinD}] \\
& \text{then guess } r1 \ldots \text{ then guess } r2 \ldots \text{ note } r12 = \text{this} \\
& \text{note } s1\text{-correct} (OF s1, \text{ of } p) \\
& \text{then guess } rm \ldots \text{ then guess } ra \ldots \text{ note } rmra = \text{this} \\
& \text{from } r12 \text{ rmra have oifra: oif = ra by simp} \\
& \text{from } r12 \text{ have sfw: simple-fw } \text{fw p} = \text{Decision FinalAllow using simple-fw-iiff-generalized-fw-accept by blast} \\
& \text{note } \text{ifupdateirrel} = \text{no-oif-matchD}(OF s2, \text{ where } any = \text{output-iiface} (\text{routing-table-semantics } rt \text{ (p-dst p)}) \text{ and } p = p, \text{ symmetric}) \\
& \text{show } \text{?case unfolding simple-linux-router-nol12-def by}(simp add: Let-def ifupdateirrel sfw oifra rmra split: Option.bind-splits option.splits)
\end{align*}
qed

**lemma** \( \text{lr-of-tran-fbs-acceptI} \):
\begin{align*}
& \text{assumes } s1: \text{valid-prefixes } rt \text{ has-default-route } rt \\
& \text{assumes } s2: \text{no-oif-match } \text{fw} \\
& \text{shows } \text{simple-linux-router-nol12 } rt \text{ fw p} = \text{Some } (p[p-oiface := \text{oif}]) \implies \\
& \exists r. \text{generalized-sfw} (\text{lr-of-tran-fbs } rt \text{ fw ifs}) \ p = \text{Some } (r, \text{ oif}, \text{ simple-action.Accept})
\end{align*}
**proof** (goal-cases)
\begin{align*}
& \text{from } s2 \text{ have } nud: \forall p. \text{simple-fw } \text{fw p} \neq \text{Undecided by (metis has-default-policy state.distinct(1))} \\
& \text{note } \text{ifupdateirrel} = \text{no-oif-matchD}(OF s2(1), \text{ symmetric}) \\
& \text{case } 1 \\
& \text{from } 1 \text{ have simple-fw } \text{fw p} = \text{Decision FinalAllow by}(simp add: simple-linux-router-nol12-def Let-def nud ifupdateirrel split: Option.bind-splits state.splits final-decision.splits) \\
& \text{then obtain } r \text{ where } r: \text{generalized-sfw} (\text{map simple-rule-dtor } \text{fw}) \ p = \text{Some } (r, \text{ simple-action.Accept}) \text{ using simple-fw-iiff-generalized-fw-accept by blast} \\
& \text{have oif-def: oif = output-iiface (routing-table-semantics } rt \text{ (p-dst p)) using 1 by(cases p) (simp add: simple-linux-router-nol12-def Let-def nud ifupdateirrel split: Option.bind-splits state.splits final-decision.splits)} \\
& \text{note } s1\text{-correct} (OF s1, \text{ of } p) \text{ then guess } rm \ldots \text{ then guess } ra \ldots \text{ note } rmra = \text{this} \\
& \text{show } \text{?case unfolding lr-of-tran-fbs-def Let-def} \\
& \text{apply(rule ez1)} \\
& \text{apply(rule generalized-fw-joinI)} \\
& \text{unfolding oif-def using rmra apply simp} \\
& \text{apply(rule r)} \\
& \text{done}
\end{align*}
qed

**lemma** \( \text{lr-of-tran-fbs-dropD} \):
\begin{align*}
& \text{assumes } s1: \text{valid-prefixes } rt \text{ has-default-route } rt \\
& \text{assumes } s2: \text{no-oif-match } \text{fw} \\
& \text{shows } \text{generalized-sfw} (\text{lr-of-tran-fbs } rt \text{ fw ifs}) \ p = \text{Some } (r, \text{ oif}, \text{ simple-action.Drop}) \implies \\
& \text{simple-linux-router-nol12 } rt \text{ fw p} = \text{None}
\end{align*}
**proof** (goal-cases)
\begin{align*}
& \text{note } \text{ifupdateirrel} = \text{no-oif-matchD}(OF s2(1), \text{ symmetric}) \\
& \text{case } 1
\end{align*}
from \[\{\text{unfolded } \text{lr-of-tran-fbs-def } \text{Let-def}, \ \text{THEN } \text{generalized-fw-joinD}\}\]

obtain \(rr \text{ fr where generalized-sfw } (\text{lr-of-tran-s1 rt}) \ p = \text{Some } (rr, oif) \land\)

\[\text{generalized-sfw } (\text{map } \text{simple-rule-dtor fw}) \ p = \text{Some } (fr, \text{simple-action.Drop}) \land \text{Some } r = \text{simple-match-and } rr \text{ fr by}\]

presburger

hence \(fd: \bigwedge u. \ \text{simple-fw fw } (p|p\text{-oif-face : = } a)) = \text{Decision FinalDeny}\)

\text{unfolding ifupdateirrel}

using \text{simple-fw-iff-generalized-fw-drop by blast}

show \(?\text{thesis}\)

by(clarsimp simp: \text{simple-linux-router-nol12-def } \text{Let-def } fd \text{ split: Option.bind-splits})

qed

lemma \(\text{lr-of-tran-fbs-drop1:}\)

\text{assumes } s2: \text{valid-prefixes rt has-default-route rt}

\text{assumes } s2: \text{no-oif-match fw has-default-policy fw}

\text{shows } \text{simple-linux-router-nol12 rt fw p = None } \Rightarrow

\exists oif; \text{generalized-sfw } (\text{lr-of-tran-fbs rt fw ifs}) \ p = \text{Some } (r, oif, \text{simple-action.Drop})

\text{proof}(\text{goal-cases})

\text{from } s2 \text{ have nud: } \bigwedge p. \ \text{simple-fw fw } p \neq \text{Undecided by } (\text{metis has-default-policy state.distinct}(1))

\text{note } \text{ifupdateirrel} = \text{no-oif-matchD}(OF \ s2(1), \ \text{symmetric})

\text{case } 1

\text{from } 1 \text{ have } \text{simple-fw fw } p = \text{Decision FinalDeny by}(\text{simp add: simple-linux-router-nol12-def Let-def nud ifupdateirrel split: Option.bind-splits state.splits final-decision.split})

\text{then obtain } r \text{ where } r: \text{generalized-sfw } (\text{map } \text{simple-rule-dtor fw}) \ p = \text{Some } (r, \text{simple-action.Drop}) \text{ using simple-fw-iff-generalized-fw-drop by blast}

\text{note } \text{s1-correct}(OF s1, of p) \ \text{then guess } rm \ .. \ \text{then guess } ra .. \ \text{note } rmra = \text{this}

show \ (?case unfolding \ \text{lr-of-tran-fbs-def } \text{Let-def}\)

apply(rule exI)

apply(rule exI[where \(x = ra\)])

apply(rule generalized-fw-joinI)

using rmra apply simp

apply(rule r)

done

qed

lemma \(\text{no-oif-match-fbs:}\)

\text{no-oif-match fw } \Rightarrow \text{list-all } (\lambda m. \text{oiface } (\text{fst } (\text{snd } m))) = \text{ifaceAny} (\text{map } (\text{apfst of-nat}) (\text{annotate-relen } (\text{lr-of-tran-fbs rt fw ifs})))

\text{proof}(\text{goal-cases})

\text{case } 1

\text{have } c: \bigwedge mr \ ar \ mf \ af \ a. \ [(mr, ar) \in \text{set } (\text{lr-of-tran-s1 rt}); (mf, af) \in \text{simple-rule-dtor } \text{set fw}; \text{simple-match-and } mr \ mf

= \text{Some } a] \Longrightarrow \text{oiface } a = \text{ifaceAny}

\text{proof}(\text{goal-cases})

\text{case } (1 mr \ ar \ mf \ af \ a)

\text{have oiface mr = ifaceAny using } 1(1) \text{ unfolding } \text{lr-of-tran-s1-def route2match-def by(clarsimp simp add: Set.image-iff)}

\text{moreover have oiface mf = ifaceAny using } 1(2) \text{ unfolding } \text{no-oif-match-fw}\text{ unfolding } \text{no-oif-match-def}\text{simple-rule-dtor-def[abs-def]}

by(clarsimp simp: list-all-iff split: simple-rule.splits) fastforce

ultimately show \ (?case using 1(3) by(cases a; cases mr; cases mf) (simp add: iface-conjunct-iface.Any split: option.splits))

qed

have \(ba: \text{list-all } (\lambda m. \text{oiface } (\text{fst } m)) = \text{ifaceAny} (\text{lr-of-tran-fbs rt fw ifs})\)

\text{unfolding } \text{lr-of-tran-fbs-def Let-def list-all-iff}

apply(clarify)

apply(subst(asm) generalized-sfw-join-set)

apply(clarsimp)

35
using c by blast
thus \( ? \) case
proof (goal-cases)
case 1
  have \(*\): \((\lambda m. \text{oiface } (\text{fst } (\text{snd } m)) = \text{ifaceAny}) = (\lambda m. \text{oiface } (\text{fst } m) = \text{ifaceAny})\) \(\circ\) \(\text{snd}\) unfolding comp-def ..
  show \(?\) case unfolding \(*\) list-all-map[\{symmetric\}] map-snd-apfst map-snd-annotate-rlen using la .
qed
qed

lemma lr-of-tran-correct:
fixes \( p :: (32, 'a) \) simple-packet-ext-scheme
assumes \( \text{nerr}\) lr-of-tran rt fw ifs = 1\text{hr offt}
and \( \text{ippkt}\) p-l2type \( p = 0x800\)
and \( \text{ifeld}\) p-iface \( p \in \text{set ifs}\)
shows OF-priority-match OF-match-fields-safe oft p = Action \([\text{Forward } \text{oif}] \leftrightarrow \text{simple-linux-router-nol12 rt fw} \) p = (Some (\( p(p \text{oiface} := \text{offt})\)))
  OF-priority-match OF-match-fields-safe oft p = Action [] \leftrightarrow \text{simple-linux-router-nol12 rt fw} \) p = None
  OF-priority-match OF-match-fields-safe oft p \(\neq\) NoAction OF-priority-match OF-match-fields-safe oft p \(\neq\) Undefined
  OF-priority-match OF-match-fields-safe oft p = Action \(\text{ls} \rightarrow \text{length} \text{ls} \leq 1\)
  \(\exists \text{ls} \). \(\text{length} \text{ls} \leq 1\) \(\land\) OF-priority-match OF-match-fields-safe oft p = Action \(\text{ls}\)
proof –
  have s1: valid-prefixes rt has-default-route rt
  and s2: has-default-policy fw simple-fu-valid fu no-of-match fw
  and diffs: distinct ifs
  using nerr unfolding lr-of-tran-def by(simp-all split: if-splits)
  have no-of-match fu using nerr unfolding lr-of-tran-def by(simp split: if-splits)
  note s2 = s2 this
  have unsafe-safe-eq:
    apply(subst OF-unsafe-safe-match3-eq; (rule lr-of-tran-prereqs s1 s2 nerr refl)+)
    apply(subst OF-unsafe-safe-match-linear-eq; (rule lr-of-tran-prereqs s1 s2 nerr refl)+)
  done
  have \( \text{lin}\): OF-priority-match OF-match-fields-safe oft p = OF-match-linear OF-match-fields-safe oft
  using OF-eq(OF lr-of-tran-no-overlaps lr-of-tran-sorted-descending, OF diffs nerr[\{symmetric\}] nerr[\{symmetric\}]]
  unfolding fun-eq-iff unsafe-safe-eq by metis
  let \( \text{yard} = \text{map } (\text{apfst of-atl}) \) (annotate-rlen (lr-of-tran-fbs rt fw ifs))
  have oft-def: oft = pack-OF-entries ifs ?ard using nerr unfolding lr-of-tran-def Let-def by(simp split: if-splits)
  have \( \text{vd}\): list-all simple-match-valid (map (fst o snd) ?ard)
    unfolding fun-app_def map-map[\{symmetric\}] snd-apfst map-snd-apfst map-snd-annotate-rlen using
    simple-match-valid-fbs[OF s1(1) s2(2)] .
  have \( \circ\): list-all (\( \lambda m. \text{oiface } (\text{fst } (\text{snd } m)) = \text{ifaceAny}\)) ?ard using no-of-match-fbs[of s2(3)] .
  have not-unde: \(\forall p. \text{simple-fw fw} \) p \(\neq\) Undecided by (metis has-default-policy s2(1) state.simps(3))
  have w1-1: \(\forall p. \text{OF-match-linear OF-match-fields-safe oft p = Action } [\text{Forward } \text{oif}] \rightarrow \text{simple-linux-router-nol12 rt fw p} \) p = Some (\( p(p \text{oiface} := \text{oif})\))
  \(\land\) \( \text{oif} = \text{output-iface} \) (routing-table-semantics rt (p-dst p))
proof(intro conjI, goal-cases)
case (1 \(\text{oif}\))
  note s3-correct[OF vld ippkt ifvld(1) *, THEN iffD1, unfolded oft-def[\{symmetric\}], OF 1]
  hence \(\exists r\) generalized-sfw (map snd (map (apfst of-atl) (annotate-rlen (lr-of-tran-fbs rt fw ifs)))) p = Some (r, (oif, simple-action.Accept))
    by(clarsimp split: if-splits)
then obtain r where generalized-sfw (lr-of-trans-fbs rt fw ifs) p = Some (r, (oif, simple-action.Accept))
unfolding map-map-comp snd-apfst map-snd-annotate-rlen by blast
thus ?case using lr-of-trans-fbs-acceptD[OF s1 s2(3)] by metis
thus oif = output-iface (routing-table-semantics rt (p-dst p))
by(cases p) (clarsimp simp: simple-linux-router-nol12-def Let-def not-undec split: Option.bind-splits state.splits final-decision.splits)
qed

have w1-2: \A oif. simple-linux-router-nol12 rt fw p = Some (p(p-oiface := oif)) \implies OF-match-linear OF-match-fields-safe

proof (goal-cases)
  case (1 oif)
  note lr-of-trans-fbs-acceptI[OF s1 s2(3) s2(1) this, of ifs] then guess r .. note r = this
  hence generalized-sfw (map snd (map (apfst of-nat) (annotate-rlen (lr-of-trans-fbs rt fw ifs)))) p = Some (r, (oif, simple-action.Accept))

  unfolding map-snd-apfst map-snd-annotate-rlen .
  moreover note s3-correct[OF vld ippkt ifvld(1) *, THEN iffD2, unfolded oif-def[symmetric], of [Forward oif]]
  ultimately show ?case by simp
qed

show w1: \A oif. (OF-priority-match OF-match-fields-safe oif p = Action [Forward oif]) = (simple-linux-router-nol12 rt fw p = Some (p(p-oiface := oif)))

unfolding lin using w1-1 w1-2 by blast

show w2: (OF-priority-match OF-match-fields-safe oif p = Action []) = (simple-linux-router-nol12 rt fw p = None)

unfolding lin

proof (rule iffI, goal-cases)
  case 1
  note s3-correct[OF vld ippkt ifvld(1) *, THEN iffD1, unfolded oif-def[symmetric], OF 1]
  then obtain r oif where oif: generalized-sfw (lr-of-trans-fbs rt fw ifs) p = Some (r, oif, simple-action.Drop)

  unfolding map-snd-apfst map-snd-annotate-rlen by(clarsimp split: if-splits)
  note lr-of-trans-fbs-dropD[OF s1 s2(3) this] 
  thus ?case .

next
  case 2
  note lr-of-trans-fbs-dropI[OF s1 s2(3) s2(1) this, of ifs] then
  obtain r oif where generalized-sfw (lr-of-trans-fbs rt fw ifs) p = Some (r, oif, simple-action.Drop) by blast
  hence generalized-sfw (map snd (map (apfst of-nat) (annotate-rlen (lr-of-trans-fbs rt fw ifs)))) p = Some (r, oif, simple-action.Drop)

  unfolding map-snd-apfst map-snd-annotate-rlen .
  moreover note s3-correct[OF vld ippkt ifvld(1) *, THEN iffD2, unfolded oif-def[symmetric], of []]
  ultimately show ?case by force
qed

have br-determ: \A a. simple-linux-router-nol12 rt fw p = Some a \implies a = p(p-oiface := output-iface (routing-table-semantics rt (p-dst p)))]
by(clarsimp simp: simple-linux-router-nol12-def Let-def not-undec split: Option.bind-splits state.splits final-decision.splits)

show notno: OF-priority-match OF-match-fields-safe oif p \neq NoAction
  apply(cases simple-linux-router-nol12 rt fw p)
  using w2 apply(simp)
  using w1[of output-iface (routing-table-semantics rt (p-dst p))] apply(simp)
  apply(drule br-determ)
  apply(simp)
  done

show notmut: OF-priority-match OF-match-fields-safe oif p = Action is \rightarrow length ls \leq 1
  apply(cases simple-linux-router-nol12 rt fw p)

37
using w2 apply(simp)
using w1[of output-iface (routing-table-semantics rt (p-dst p))] apply(simp)
apply(drule lr-determ)
apply(clarsimp)
done

show ∃ls. length ls ≤ 1 ∧ OF-priority-match OF-match-fields-safe oft p = Action ls
apply(cases OF-priority-match OF-match-fields-safe oft p)
using notmult apply blast
using notno apply blast
using notub apply blast
done
qed
end

theory OF-conv-test
imports
  Iptables-Semantics
  Parser
  Simple-Firewall
  SimpleFw-toString
  Routing
  IpRoute-Parser
  ../../LinuxRouter-OpenFlow-Translation
  ../../OpenFlow-Serialize
begin

parse-iptables-save SQRL-fw=iptables-save
term SQRL-fw
thm SQRL-fw-def
thm SQRL-fw-FORWARD-default-policy-def

value[code] map (λ(c, rs). (c, map (quote-rewrite ◦ common-primitive-rule-toString) rs)) SQRL-fw
definition unfolded = unfold-ruleset-FORWARD SQRL-fw-FORWARD-default-policy (map-of-string-ipv4 SQRL-fw)
lemma map (quote-rewrite ◦ common-primitive-rule-toString) unfolded =
  ["−p icmp −j ACCEPT"],
  ["−i sl−lan −p tcp −m tcp --spts [1024;65535] −m tcp --dpts [80] −j ACCEPT"],
  ["−i sl−wan −p tcp −m tcp --spts [80] −m tcp --dpts [1024;65535] −j ACCEPT"],
  ["−j DROP"] by eval

lemma length unfolded = 4 by eval

value[code] map (quote-rewrite ◦ common-primitive-rule-toString) (upper-closure unfolded)
lemma length (upper-closure unfolded) = 4 by eval

value[code] upper-closure (packet-assume-new unfolded)

lemma length (lower-closure unfolded) = 4 by eval

lemma check-simple-fw-preconditions (upper-closure unfolded) = True by eval
lemma ∀m ∈ get-match’set (upper-closure (packet-assume-new unfolded)). normalized-nnf-match m by eval
\textbf{lemma} \( \forall m \in \text{get-match}\set (\text{optimize-matches abstract-for-simple-firewall (upper-closure (packet-assume-new unfolded)))}. \) 
\text{normalized-nnf-match m by eval}

\textbf{lemma} \text{check-simple-fw-preconditions (upper-closure (optimize-matches abstract-for-simple-firewall (upper-closure (packet-assume-new unfolded))) by eval)}

\textbf{lemma} \text{length (to-simple-firewall (upper-closure (packet-assume-new unfolded)))} = 4 by eval

\textbf{lemma} (lower-closure (optimize-matches abstract-for-simple-firewall (lower-closure (packet-assume-new unfolded))) = lower-closure unfolded 
\quad \text{lower-closure unfolded = upper-closure unfolded} 
\quad (upper-closure (optimize-matches abstract-for-simple-firewall (upper-closure (packet-assume-new unfolded)))) = upper-closure unfolded by eval+

\text{value}[\text{code}] \text{(getPorts (to-simple-firewall (lower-closure (optimize-matches abstract-for-simple-firewall (lower-closure (packet-assume-new unfolded))))))}

\textbf{definition} \text{SQRL-fw-simple} \equiv \text{reundups-rev (to-simple-firewall (upper-closure (optimize-matches abstract-for-simple-firewall (upper-closure (packet-assume-new unfolded)))))}

\text{value}[\text{code}] \text{SQRL-fw-simple}

\textbf{lemma} \text{simple-fw-valid SQRL-fw-simple by eval}

\text{parse-ip-route} \text{SQRL-rtbl-main = ip–route}

\text{value} \text{SQRL-rtbl-main}

\textbf{lemma} \text{SQRL-rtbl-main = [\{routing-match = PrefixMatch 0xA000100 24, metric = 0, routing-action = \{output-iface = "s1–lan", next-hop = None\}\}, 
\quad \{routing-match = PrefixMatch 0xA000200 24, metric = 0, routing-action = \{output-iface = "s1–wan", next-hop = None\}\}, 
\quad \{routing-match = PrefixMatch 0 0, metric = 0, routing-action = \{output-iface = "s1–wan", next-hop = Some 0xA000201\}\}] by eval}

\text{value} \text{dotdecimal-of-ipv4addr 0xA0D2500}

\text{lemma} \text{SQRL-rtbl-main = [ 
\quad rr-ctor (10,0,1,0) 24 "s1–lan" None 0, 
\quad rr-ctor (10,0,2,0) 24 "s1–wan" None 0, 
\quad rr-ctor (0,0,0,0) 0 "s1–wan" (Some (10,0,2,1)) 0 
\ ] by eval}

\textbf{definition} \text{SQRL-rtbl-main-sorted} \equiv \text{rev (sort-key (λr. pfm-length (routing-match r))) SQRL-rtbl-main}

\text{value} \text{SQRL-rtbl-main-sorted}

\textbf{definition} \text{SQRL-ifs ≡ [ 
\quad \{iface-name = "s1–lan", iface-mac = 0x10001\}, 
\quad \{iface-name = "s1–wan", iface-mac = 0x10002\} ]}

\text{value} \text{SQRL-ifs}

\textbf{definition} \text{SQRL-macs ≡ [ 
\quad \{ipv4addr-of-dotdecimal (10,0,1,2), 0x1\}), 
\quad \{ipv4addr-of-dotdecimal (10,0,1,3), 0x2\}), 
\quad \{ipv4addr-of-dotdecimal (10,0,2,1), 0x3\}) 
\ ]}

\textbf{definition} \text{SQRL-ports ≡ [ 
\ ]}

39
lemma let fu = SQRL-fu-simple in no-of-match fu ∧ has-default-policy fu ∧ simple-fu-valid fu by eval

lemma let rt = SQRL-rtbl-main-sorted in valid-prefixes rt ∧ has-default-route rt by eval

lemma let ifs = (map iface-name SQRL-ifs) in distinct ifs by eval

definition of ≡

  case (b-of-tran SQRL-rtbl-main-sorted SQRL-fu-simple (map iface-name SQRL-ifs))
  of (lnr openflow-rules) ⇒ map (serialize-of-entry (the ◦ map-of SQRL-ports)) openflow-rules

lemma of =

  ["priority=11,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-proto=1,nw-dst=10.0.2.0/24,action=output:2"]/,
  "priority=10,hard-timeout=0,idle-timeout=0,in-port=1,dl-type=0x800,nw-proto=0,nw-dst=10.0.2.0/24,tp-src=1024/0xfc00,tp-dst=80/0x80000,acti...

  "priority=9,hard-timeout=0,idle-timeout=0,in-port=2,dl-type=0x800,nw-proto=0,nw-dst=10.0.2.0/24,tp-src=80/0x80000,tp-dst=1024/0xfc00,acti...

  "priority=8,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-dst=10.0.2.0/24,action=output:drop"]/,
  "priority=7,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-proto=1,nw-dst=10.0.1.0/24,action=output:1"]/,
  "priority=6,hard-timeout=0,idle-timeout=0,in-port=1,dl-type=0x800,nw-proto=6,nw-dst=10.0.1.0/24,tp-src=1024/0xfc00,tp-dst=80/0x80000,acti...

  "priority=5,hard-timeout=0,idle-timeout=0,in-port=2,dl-type=0x800,nw-proto=6,nw-dst=10.0.1.0/24,tp-src=80/0x80000,tp-dst=1024/0xfc00,acti...

  "priority=4,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-dst=10.0.1.0/24,action=output:drop"]/,
  "priority=3,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-proto=1,action=output:2"]/,
  "priority=2,hard-timeout=0,idle-timeout=0,in-port=1,dl-type=0x800,nw-proto=6,tp-src=1024/0xfc00,tp-dst=80,action=output:2"]/,
  "priority=1,hard-timeout=0,idle-timeout=0,in-port=1,dl-type=0x800,nw-proto=6,tp-src=80,action=output:2"]/,
"priority=0,hard-timeout=0,idle-timeout=0,dl-type=0x800,action=drop"] by eval

value[code] ofi

end
theory RFC2544
imports
  Iptables-Semantics.Parser
  Routing.IpRoute-Parser
  ../../LinuxRouter-OpenFlow-Translation
  ../../OpenFlow-Serialize
begin

parse-iptables-save SQRL-fw=iptables-save
term SQRL-fw
thm SQRL-fw-def
thm SQRL-fw-FORWARD-default-policy-def
value[code] map (\(\lambda(c,rs)\). (c, map (quote-rewrite \(\circ\) common-primitive-rule-toString) rs)) SQRL-fw
definition unfolded = unfold-ruleset-FORWARD SQRL-fw-FORWARD-default-policy (map-of-string-ipv4 SQRL-fw)

lemma length unfolded = 26 by eval

value[code] unfolded
value[code] (upper-closure unfolded)
value[code] map (quote-rewrite \(\circ\) common-primitive-rule-toString) (upper-closure unfolded)
lemma length (upper-closure unfolded) = 26 by eval

value[code] upper-closure (packet-assume-new unfolded)

lemma length (lower-closure unfolded) = 26 by eval

lemma check-simple-fw-preconditions (upper-closure unfolded) by eval
lemma \(\forall m \in\) get-match'set (upper-closure (packet-assume-new unfolded)). normalized-nnf-match m by eval
lemma \(\forall m \in\) get-match'set (optimize-matches abstract-for-simple-firewall (upper-closure (packet-assume-new unfolded))). normalized-nnf-match m by eval
lemma check-simple-fw-preconditions (upper-closure (optimize-matches abstract-for-simple-firewall (upper-closure (packet-assume-new unfolded)))) by eval
lemma length (to-simple-firewall (upper-closure (packet-assume-new unfolded))) = 26 by eval

lemma (lower-closure (optimize-matches abstract-for-simple-firewall (lower-closure (packet-assume-new unfolded)))) = lower-closure unfolded
  lower-closure unfolded = upper-closure unfolded
  (upper-closure (optimize-matches abstract-for-simple-firewall (upper-closure (packet-assume-new unfolded)))) = upper-closure unfolded by eval+

41
value[code] (getPorts (to-simple-firewall (lower-closure (optimize-matches abstract-for-simple-firewall (lower-closure (packet-assume-new unfolded)))))

definition SQRL-fu-simple ≡ remdups-rev (to-simple-firewall (upper-closure (optimize-matches abstract-for-simple-firewall (upper-closure (packet-assume-new unfolded)))))

value[code] SQRL-fu-simple

lemma simple-fu-valid SQRL-fu-simple by eval

parse-ip-route SQRL-rtbl-main = ip-route

value SQRL-rtbl-main

lemma SQRL-rtbl-main = [\{routing-match = PrefixMatch 0xC6130100 24, metric = 0, routing-action = {output-interface = "ip1", next-hop = None}\},
\{routing-match = PrefixMatch 0xC6130100 24, metric = 0, routing-action = {output-interface = "op1", next-hop = None}\},
\{routing-match = PrefixMatch 0 0, metric = 0, routing-action = {output-interface = "op1", next-hop = Some 0xC6130102}\}]

by eval

lemma SQRL-rtbl-main = [
rr-ctor (198,18,1,0) 24 "ip1" None 0,
rr-ctor (198,19,1,0) 24 "op1" None 0,
rr-ctor (0,0,0,0) 0 "op1" (Some (198,19,1,2)) 0 ]

by eval

definition SQRL-ports ≡ [
("ip1", "1"),
("op1", "2")
]

definition of ≡ case (br-of-tran SQRL-rtbl-main SQRL-fu-simple (map fst SQRL-ports)) of (Inr openflow-rules) ⇒ map (serialize-of-entry (the o map-of SQRL-ports)) openflow-rules

lemma of = [
"priority=27,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-dst=198.18.1.0/24,action=drop",
"priority=26,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-dst=198.19.1.0/24,action=drop",
"priority=25,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.1.1/32,nw-dst=192.18.101.1/32,action=drop",
"priority=24,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.2.2/32,nw-dst=192.18.102.2/32,action=drop",
"priority=23,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.3.3/32,nw-dst=192.18.103.3/32,action=drop",
"priority=22,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.4.4/32,nw-dst=192.18.104.4/32,action=drop",
"priority=21,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.5.5/32,nw-dst=192.18.105.5/32,action=drop",
"priority=20,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.6.6/32,nw-dst=192.18.106.6/32,action=drop",
"priority=19,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.7.7/32,nw-dst=192.18.107.7/32,action=drop",
"priority=18,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.8.8/32,nw-dst=192.18.108.8/32,action=drop",
"priority=17,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.9.9/32,nw-dst=192.18.109.9/32,action=drop",
"priority=16,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.10.10/32,nw-dst=192.18.110.10/32,action=drop",
"priority=15,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.11.11/32,nw-dst=192.18.111.11/32,action=drop",
"priority=14,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.12.12/32,nw-dst=192.18.112.12/32,action=drop",
"priority=13,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.13.13/32,nw-dst=192.18.113.13/32,action=drop",
"priority=12,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.14.14/32,nw-dst=192.18.114.14/32,action=drop",
"priority=11,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.15.15/32,nw-dst=192.18.115.15/32,action=drop",
"priority=10,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.16.16/32,nw-dst=192.18.116.16/32,action=drop",
"priority=9,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.17.17/32,nw-dst=192.18.117.17/32,action=drop",
"priority=8,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.18.18/32,nw-dst=192.18.118.18/32,action=drop",
"priority=7,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.19.19/32,nw-dst=192.18.119.19/32,action=drop",
"priority=6,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.20.20/32,nw-dst=192.18.120.20/32,action=drop",
"priority=5,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.21.21/32,nw-dst=192.18.121.21/32,action=drop",
]
"priority=4,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.21.21/32,nw-dst=192.18.121.21/32,action=drop",
"priority=3,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.22.22/32,nw-dst=192.18.122.22/32,action=drop",
"priority=2,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.23.23/32,nw-dst=192.18.123.23/32,action=drop",
"priority=1,hard-timeout=0,idle-timeout=0,dl-type=0x800,nw-src=192.18.24.24/32,nw-dst=192.18.124.24/32,action=drop",
"priority=0,hard-timeout=0,idle-timeout=0,dl-type=0x800,action=drop"] by eval

value[code] length afi

end
Part II
Documentation

1 Configuration Translation

All the results we present in this section are formalized and verified in Isabelle/HOL [11]. This means that their formal correctness can be trusted a level close to absolute certainty. The definitions and lemmas stated here are merely a repetition of lemmas stated in other theory files. This means that they have been directly set to this document from Isabelle and no typos or hidden assumptions are possible. Additionally, it allows us to omit various helper lemmas that do not help the understanding. However, it causes some notation inaccuracy, as type and function definitions are stated as lemmas or schematic goals.

theory OpenFlow-Documentation

1.1 Linux Firewall Model

We want to write a program that translates the configuration of a Linux firewall to that of an OpenFlow switch. We furthermore want to verify that translation. For this purpose, we need a clear definition of the behavior of the two device types – we need their models and semantics. In case of a Linux firewall, this is problematic because a Linux firewall is a highly complex device that is ultimately capable of general purpose computation. Creating a comprehensive semantics that encompasses all possible configuration types of a Linux firewall is thus highly non-trivial and not useful for the purpose of analysis. We decided to approach the problem from the other side: we created a model that includes only the most basic features. (This implies neglecting IPv6.) Fortunately, many of the highly complex features are rarely essential and even our basic model is still of some use.

We first divided the firewall into subsystems. Given a routing table \( rt \), the firewall rules \( fw \), the routing decision for a packet \( p \) can be obtained by \( \text{routing-table-semantics} \ rt \ (p\text{-dst} \ p) \), the firewall decision by \( \text{simple-fw} \ fw \ p \). We draft the first description of our Linux router model:

1. The destination MAC address of an arriving packet is checked: Does it match the MAC address of the ingress port? If it does, we continue, otherwise, the packet is discarded.
2. The routing decision \( rd \equiv \text{routing-table-semantics} \ rt \ p \) is obtained.
3. The packet’s output interface is updated based on \( rd \).
4. The firewall is queried for a decision: \( \text{simple-fw} \ fw \ p \). If the decision is to Drop, the packet is discarded.
5. The next hop is computed: If \( rd \) provides a next hop, that is used. Otherwise, the destination address of the packet is used.
6. The MAC address of the next hop is looked up; the packet is updated with it and sent.

We decided that this description is best formalized as an abortable program in the option monad:

\[
\text{lemma simple-linux-router} \ rt \ fw \ mlf \ ifl \ p \equiv \{ \\
\quad \text{- } \leftarrow \text{iface-packet-check} \ ifl \ p; \\
\quad \text{let } rd \leftarrow \text{routing-table-semantics} \ rt \ (p\text{-dst} \ p); \\
\quad \text{let } p = p(p\text{-oface} := \text{output-iface} \ rd); \\
\quad \text{let } fd \leftarrow \text{firewall decision} = \text{simple-fw} \ fw \ p; \\
\quad \text{- } \leftarrow \text{case } fd \text{ of Decision FinalAllow } \Rightarrow \text{Some } () \mid \text{Decision FinalDeny } \Rightarrow \text{None}; \\
\quad \text{let } nh = \text{case next-hop } rd \text{ of None } \Rightarrow p\text{-dst} \ p \mid \text{Some } a \Rightarrow a; \\
\quad \text{ma } \leftarrow \text{mlf} \ nh; \\
\quad \text{Some } (p(p\text{-l2dst} := \text{mal})) \\
\}\n\]

\text{unfolding fromMaybe-def[symmetric] by(fact simple-linux-router-def)}

where \( mlf \) is a function that looks up the MAC address for an IP address.

There are already a few important aspects that have not been modelled, but they are not core essential for the functionality of a firewall. Namely, there is no local traffic from/to the firewall. This is problematic since this model can not generate ARP replies — thus, an equivalent OpenFlow device will not do so, either. Furthermore, this model is problematic because it requires access to a function that looks up a MAC address, something that may not be known at the time of time running a translation to an OpenFlow configuration.

\(^{1}\text{Note that we assume a packet model with input and output interfaces. The origin of this is explained in Section 1.1.2}\)
It is possible to circumvent these problems by inserting static ARP table entries in the directly connected devices and looking up their MAC addresses \textit{a priori}. A test-wise implementation of the translation based on this model showed acceptable results. However, we deemed the \textit{a priori} lookup of the MAC addresses to be rather inelegant and built a second model.

\textbf{definition} \textit{simple-linux-router-altered} rt \textit{fw} ifl \( p \equiv \{ \)
\begin{align*}
\text{let } rd &= \text{routing-table-semantics} \text{ rt (p-dst } p); \\
\text{let } p &= p(p\text{-oiface} := \text{output-iface } rd); \\
\text{let } fd &= \text{simple-fw } p; \\
\text{let } rd &= \text{simple-fw } p; \\
\text{let } rd &= \text{simple-fw } p;
\end{align*}

\begin{align*}
\text{None}
\end{align*}
\( p \leftarrow (\text{case } \text{fd} \text{ of } \text{Decision } \text{FinalAllow} \Rightarrow \text{Some } () | \text{Decision } \text{FinalDeny} \Rightarrow \text{None}); \\
\text{Some } p \}
\}

In this model, all access to the MAC layer has been eliminated. This is done by the approximation that the firewall will be asked to route a packet (i.e. be addressed on the MAC layer) \( p \) the destination IP address of the packet causes it to be routed out on a different interface. Because this model does not insert destination MAC addresses, the destination MAC address has to be already correct when the packet is sent. This can only be achieved by changing the subnet of all connected device, moving them into one common subnet\(^2\).

While a test-wise implementation based on this model also showed acceptable results, the model is still problematic. The check \( p\text{-oiface } p = p\text{-iface } p \) and the firewall require access to the output interface. The details of why this cannot be provided are be elaborated in Section 1.3. The intuitive explanation is that an OpenFlow match language to select a routing table. Based on our directive, we only focused on the single most used \texttt{main} routing table.

We define a routing table entry to be a record (named tuple) of a prefix match, a metric and the routing action, which in turn is a record of an output interface and an optional next-hop address.

\textbf{schematic-goal} \( (?\texttt{rtbl-entry} :: (\texttt{a::len}) \texttt{routing-rule}) = () \)
\( \texttt{routing-match} = \texttt{PrefixMatch} \texttt{pfs len, metric } = \texttt{met, routing-action} = (\texttt{output-iface } = \texttt{ofif-string, next-hop } = (\texttt{h :: \texttt{a word option}}) \}) \).

A routing table is then a list of these entries:

\textbf{lemma} \( \texttt{rtbl} :: (\texttt{a :: len}) \texttt{prefix-routing} = (\texttt{rtbl} :: \texttt{a routing-rule list}) \textbf{by rule} \)

Not all members of the type \texttt{prefix-routing} are sane routing tables. There are three different validity criteria that we require so that our definitions are adequate.

- The prefixes have to be 0 in bits exceeding their length.
- There has to be a default rule, i.e. one with prefix length 0. With the condition above, that implies that all its prefix bits are zero and it thus matches any address.
- The entries have to be sorted by prefix length and metric.
The first two are set into code in the following way:

**Lemma valid-prefix (PrefixMatch pfx len) ≡ pfx & & (2 ^ (32 − len) − 1) = (0 :: 32 word)**

by (simp add: valid-prefix-def pfzm-mask-def mask-eq-decr-exp and.commute)

**Lemma has-default-route rt ✼¬ (∃ r ∈ set rt. pfzm-length (routing-match r) = 0)**

by (fact has-default-route-alt)

The third is not needed in any of the further proofs, so we omit it.

The semantics of a routing table is to simply traverse the list until a matching entry is found.

**Schematic-goal routing-table-semantics (rt-entry # rt) dst-addr = (if prefix-match-semantics (routing-match rt-entry) dst-addr then routing-action rt-entry else routing-table-semantics rt dst-addr) by (fact routing-table-semantics.simps)**

If no matching entry is found, the behavior is undefined.

### 1.1.2 iptables Firewall

The firewall subsystem in a linux router is not any less complex than any of the other systems. Fortunately, this complexity has been dealt with in [6, 5] already and we can directly use the result.

In short, one of the results is that a complex iptables configuration can be simplified to be represented by a single list of matches that only support the following match conditions:

- (String) prefix matches on the input and output interfaces.
- A prefix-match on the source and destination IP address.
- An exact match on the layer 4 protocol.
- Interval matches on the source or destination port, e.g. \( p_d \in \{1..1023\} \)

The model/type of the packet is adjusted to fit that: it is a record of the fields matched on. This also means that input and output interface are coded to the packet. Given that this information is usually stored alongside the packet content, this can be deemed a reasonable model. In case the output interface is not needed (e.g., when evaluating an OpenFlow table), it can simply be left blank.

Obviously, a simplification into the above match type cannot always produce an equivalent firewall, and the set of accepted packets has to be over- or underapproximated. The reader interested in the details of this is strongly referred to [6]; we are simply going to continue with the result: `simple-fw`.

One property of the simplification is worth noting here: The simplified firewall does not know state and the simplification approximates stateful matches by stateless ones. Thus, the overapproximation of a stateful firewall ruleset that begins with accepting packets of established connections usually begins with a rule that accepts all packets. Dealing with this by writing a meaningful simplification of stateful firewalls is future work.

### 1.2 OpenFlow Switch Model

In this section, we present our model of an OpenFlow switch. The requirements for this model are derived from the fact that it models devices that are the target of a configuration translation. This has two implications:

- All configurations that are representable in our model should produce the correct behavior wrt. their semantics. The problem is that correct here means that the behavior is the same that any real device would produce. Since we cannot possibly account for all device types, we instead focus on those that conform to the OpenFlow specifications. To account for the multiple different versions of the specification (e.g. [2, 3]), we tried making our model a subset of both the oldest stable version 1.0 [2] and the newest available specification version 1.5.1 [3].

- Conversely, our model does not need to represent all possible behavior of an OpenFlow switch, just the behavior that can be invoked by the result of our translation. This is especially useful regarding for controller interaction, but also for MPLS or VLANs, which we did not model in Section 1.1.

More concretely, we set the following rough outline for our model:

- A switch consists of a single flow table.
- A flow table entry consists of a priority, a match condition and an action list.
The only possible action (we require) is to forward the packet on a port.

We do not model controller interaction.

Additionally, we decided that we wanted to be able to ensure the validity of the flow table in all qualities, i.e. we want to model the conditions ‘no overlapping flow entries appear’, ‘all match conditions have their necessary preconditions’. The details of this are explained in the following sections.

1.2.1 Matching Flow Table entries

Table 3 of Section 3.1 of [2] gives a list of required packet fields that can be used to match packets. This directly translates into the type for a match expression on a single field:

\[
\text{schematic-goal} \quad \text{(field-match :: of-match-field) ∈ } \{\ \text{IngressPort (?x::string), EtherSrc (?as::48 word), EtherDst (?ad::48 word), EtherType (?ti::16 word), VlanId (?ti::16 word), VlanPriority (?ip::16 word), IPv4Src (?pms::32 prefix-match), IPv4Dst (?pmd::32 prefix-match), IPv4Proto (?ipp::8 word), L4Src (?ps::16 word), L4Dst (?pd::16 word), L4Src un-msk (?mm::8 word), L4Dst un-msk (?md::16 word) } \}
\]

Two things are worth additional mention: L3 and L4 “addresses”. The IPv4Src and IPv4Dst matches are specified as “can be subnet masked” in [2], whereas [3] states clearly that arbitrary bitmasks can be used. We took the conservative approach here. Our alteration of L4Src and L4Dst is more grave. While [2] does not state anything about layer 4 ports and masks, [3] specifically forbids using masks on them. Nevertheless, OpenVSwitch [1] and some other implementations support them. We will explain in detail why we must include bitmasks on layer 4 ports to obtain a meaningful translation in Section 1.3.

One of-match-field is not enough to classify a packet. To match packets, we thus use entire sets of match fields. As Guha et al. [7] noted\(^3\), executing a set of given of-match-fields on a packet requires careful consideration. For example, it is not meaningful to use IPv4Dst if the given packet is not actually an IP packet, i.e. IPv4Dst has the prerequisite of EtherType 2048 being among the match fields. Guha et al. decided to use the fact that the preconditions can be arranged on a directed acyclic graph (or rather: an acyclic forest). They evaluated match conditions in a manner following that graph: first, all field matches without preconditions are evaluated. Upon evaluating a field match (e.g., EtherType 2048), the matches that had their precondition fulfilled by it (e.g., IPv4Src and IPv4Src in this example) are evaluated. This mirrors the faulty behavior of some implementations (see [7]). Adopting that behavior into our model would mean that any packet matches against the field match set \{IPv4Dst (PrefixMatch 134744072 32)\} instead of just those destined for 8.8.8.8 or causing an error. We found this to be unsatisfactory.

To solve this problem, we made three definitions. The first, match-no-prereq matches an of-match-field against a packet without considering prerequisites. The second, prerequisites, checks for a given of-match-field whether its prerequisites are in a set of given match fields. Especially:

\[
\text{lemma} \quad \text{prerequisites (VlanPriority pr) m = (∃ id. let v = VlanId id in v ∈ m ∧ prerequisites v m) }
\]

\[
\text{prerequisites (IPv4Proto pr) m = (let v = EtherType 0x0800 in v ∈ m ∧ prerequisites v m) }
\]

\[
\text{prerequisites (IPv4Src a) m = (let v = EtherType 0x0800 in v ∈ m ∧ prerequisites v m) }
\]

\[
\text{prerequisites (IPv4Dst a) m = (let v = EtherType 0x0800 in v ∈ m ∧ prerequisites v m) }
\]

\[
\text{prerequisites (L4Src p msk) m = (∃ proto ∈ \{TCP,UDP,L4-Protocol,SCTP\}. let v = IPv4Proto proto in v ∈ m ∧ prerequisites v m) }
\]

\[
\text{prerequisites (L4Dst p msk) m = prerequisites (L4Src undefined undefined) m by(fact prerequisites.simps)+ }
\]

Then, to actually match a set of of-match-field against a packet, we use the option type:

\[
\text{lemma} \quad \text{OF-match-fields m p = (if } \exists f \in m. ¬\text{prerequisites f m then None else }
\]

\[
\text{if } \forall f \in m. \text{match-no-prereq f p then Some True else Some False) }
\]

\[
\text{by(fact OF-match-fields-alt) }
\]

1.2.2 Evaluating a Flow Table

In the previous section, we explained how we match the set of match fields belonging to a single flow entry against a packet. This section explains how the correct

\(\text{See also: [8, section>2.3]}\)
flow entry from a table can be selected. To prevent to much entanglement with the previous section, we assume an arbitrary match function $\gamma$. This function $\gamma$ takes the match condition $m$ from a flow entry OFEntry priority $m$ action and decides whether a packet matches those.

The flow table is simply a list of flow table entries flow-entry-match. Deciding the right flow entry to use for a given packet is explained in the OpenFlow specification [2], Section 3.4:

Packets are matched against flow entries based on prioritization. An entry that specifies an exact match (i.e., has no wildcards) is always the highest priority. All wildcard entries have a priority associated with them. Higher priority entries must match before lower priority ones. If multiple entries have the same priority, the switch is free to choose any ordering.

We use the term “overlapping” for the flow entries that can cause a packet to match multiple flow entries with the same priority. Guha et al. [7] have dealt with overlapping. However, the semantics for a flow table they presented [7, Figure 5] is slightly different from what they actually used in their theory files. We have tried to reproduce the original inductive definition (while keeping our abstraction $\gamma$), in Isabelle/HOL:

**Lemma** $\gamma$ (ofe-fields $fe$) $p = True \Rightarrow \forall fe' \in set (ft1 \oplus ft2), ofe-prio fe' > ofe-prio fe \rightarrow \gamma (ofe-fields fe') \not\in$ guha-table-semantics $\gamma (ft1 \oplus ft2) p$ (Some (ofe-action $fe$))

$\forall fe \in set ft. \gamma (ofe-fields fe) p = False \Rightarrow$ guha-table-semantics $\gamma ft p$ None by (fact guha-matched guha-unmatched)\footnotemark.

Guha et al. have deliberately made their semantics non-deterministic, to match the fact that the switch “may choose any ordering”. This can lead to undesired results:

**Lemma** $\text{CARD('action)} \geq 2 \Rightarrow \exists ff. \gamma ff p \rightarrow \exists ft (\exists action (a1 \oplus 'action) (a2 \oplus 'action), a1 \neq a2 \land guha-table-semantics \gamma ft p$ (Some $a1) \land guha-table-semantics \gamma ft p$ (Some $a2)$ by (fact guha-table-semantics-ex2res)

This means that, given at least two distinct actions exist and our matcher $\gamma$ is not false for all possible match conditions, we can say that a flow table and two actions exist such that both actions are executed. This can be misleading, as the switch might choose an ordering on some flow table and never execute some of the (overlapped) actions.

Instead, we decided to follow Section 5.3 of the specification [3], which states:

If there are multiple matching flow entries, the selected flow entry is explicitly undefined.

This still leaves some room for interpretation, but it clearly states that overlapping flow entries are undefined behavior, and undefined behavior should not be invoked. Thus, we came up with a semantics that clearly indicates when undefined behavior has been invoked:

**Lemma**

$\text{OF-priority-match \gamma flow-entries} \rightarrow$

let $m = \text{filter} (\lambda x. \gamma (\text{ofe-fields} f) \text{ packet}) \text{ flow-entries}$

$m' = \text{filter} (\lambda x. \forall fo \in set m. \text{ ofe-prio fo} \leq \text{ ofe-prio f}) m$ in

case $m'$ of $[] \Rightarrow \text{NoAction}$

$| [s] \Rightarrow \text{Action (ofe-action s)}$

$| - \Rightarrow \text{Undefined}$

**Unfolding** $\text{OF-priority-match-def} \ldots$

The definition works the following way\footnotemark:

1. The flow table is filtered for those entries that match, the result is called $m$.
2. $m$ is filtered again, leaving only those entries for which no entries with lower priority could be found, i.e. the matching flow table entries with minimal priority. The result is called $m'$.
3. A case distinction on $m'$ is made. If only one matching entry was found, its action is returned for execution. If $m$ is empty, the flow table semantics returns NoAction to indicate that the flow table does not decide an action for the packet. If, not zero or one entry is found, but more, the special value Undefined for indicating undefined behavior is returned.

\footnotetext{Note that the order of the flow table entries is irrelevant. We could have made this definition on sets but chose not to for consistency.}

\footnotetext{This behavior has been deprecated.

The original is written in Coq [4] and we can not use it directly.
The use of Undefined immediately raises the question in which condition it cannot occur. We give the following definition:

\textbf{lemma check-no-overlap} \; \gamma \; ft = (\forall \; a \in \text{set} \; ft. \; \forall \; b \in \text{set} \; ft. \; (a \neq b \land \text{ofo-fields} \; a = \text{ofo-fields} \; b) \rightarrow \neg (\exists \; p. \; \gamma (\text{ofo-fields} \; a) \land \gamma (\text{ofo-fields} \; b) \; p))

\textit{unfolding} check-no-overlap-alt check-no-overlap2-def \textbf{by} force

Together with distinctness of the flow table, this provides the absence of Undefined\footnote{It is slightly stronger than necessary, overlapping rules might be shadowed and thus never influence the behavior.}:

\textbf{lemma} \; \text{[check-no-overlap} \; \gamma \; ft; \; \text{distinct ft]} \; \Longrightarrow \; \text{OF-priority-match} \; \gamma \; ft \; p \neq \text{Undefined} \; \textbf{by} \; \text{(simp add: no-overlapsI no-overlaps-not-undefined)}

Given the absence of overlapping or duplicate flow entries, we can show two interesting equivalences. The first is the equality to the semantics defined by Guha et al.:

\textbf{lemma} \; \text{[check-no-overlap} \; \gamma \; ft; \; \text{distinct ft]} \; \Longrightarrow \; \text{OF-priority-match} \; \gamma \; ft \; p = \text{option-to-ftb} \; d \longleftrightarrow \text{guha-table-semantics} \; \gamma \; ft \; p \; d

\textbf{by} \; \text{(simp add: guha-equal no-overlapsI)}

where option-to-ftb maps between the return type of OF-priority-match and an option type as one would expect.

The second equality for OF-priority-match is one that helps reasoning about flow tables. We define a simple recursive traversal for flow tables:

\textbf{lemma} \; \text{OF-match-linear} \; \gamma \; \emptyset = \text{NoAction}\hspace{1cm} \text{OF-match-linear} \; \gamma \; (a\#as) = (\text{if} \; \gamma (\text{ofo-fields} \; a) \; p \text{ then Action} \; (\text{ofo-action} \; a) \; \text{else} \; \text{OF-match-linear} \; \gamma \; a \; p)

\textbf{by} \; \text{(fact OF-match-linear.simps)+}

For this definition to be equivalent, we need the flow table to be sorted:

\textbf{lemma} \; \text{[no-overlaps} \; \gamma \; f :: \text{sorted-descending} \; (\text{map} \; \text{ofo-prio} \; f)] \; \Longrightarrow \; \text{OF-match-linear} \; \gamma \; f \; p = \text{OF-priority-match} \; \gamma \; f \; p

\textbf{by} \; \text{(fact OF-eq)}

As the last step, we implemented a serialization function for flow entries; it has to remain unverified. The serialization function deals with one little inaccuracy: We have modelled the IngressPort match to use the interface name, but OpenFlow requires numerical interface IDs instead. We deemed that pulling this translation step into the main translation would only make the correctness lemma of the translation more complicated while not increasing the confidence in the correctness significantly. We thus made replacing interface names by their ID part of the serialization.

Having collected all important definitions and models, we can move on to the conversion.

### 1.3 Translation Implementation

This section explains how the functions that are executed sequentially in a linux firewall can be compressed into a single OpenFlow table. Creating this flow table in a single step would be immensely complicated. We thus divided the task into several steps using the following key insights:

- All steps that are executed in the linux router can be formulated as a firewall, more specifically, a generalization of simple-fw that allows arbitrary actions instead of just accept and drop.

- A function that computes the conjunction of two simple-fw matches is already present. Extending this to a function that computes the join of two firewalls is relatively simple. This is explained in Section 1.3.1

#### 1.3.1 Chaining Firewalls

This section explains how to compute the join of two firewalls.

The basis of this is a generalization of simple-fw. Instead of only allowing Accept or Drop as actions, it allows arbitrary actions. The type of the function that evaluates this generalized simple firewall is generalized-sfw. The definition is straightforward:

\textbf{lemma} \; \text{generalized-sfw} \; \emptyset \; p = \text{None} \hspace{1cm} \text{generalized-sfw} \; (a \# as) \; p = (\text{if} \; \text{case} \; a \; \text{of} \; (m,-) \Rightarrow \text{simple-matches} \; m \; p) \; \text{then} \; \text{Some} \; a \; \text{else} \; \text{generalized-sfw} \; as \; p)

\textbf{by} \; \text{(fact generalized-sfw.simps)+}

Based on that, we asked: if \( fw_1 \) makes the decision \( a \) (where \( a \) is the second element of the result tuple from generalized-sfw) and \( fw_2 \) makes the decision \( b \), how can we compute the firewall that makes the decision \( (a, b) \)?

One possible answer is given by the following definition:

\footnote{Note that tuples are right-associative in Isabelle/HOL, i.e., \((a, b, c)\) is a pair of \( a \) and the pair \((b, c)\)}
The data accompanying firewall rule ruleset. The result of the join is a ruleset with the destination IP address. The next step is to join the firewall ruleset with rules that use prefix matches on 
rt into tuples of match and action. For simple-rule-dtor, which just deconstructs simple-rules into tuples of match and action. For rt, we made a firewall ruleset with rules that use prefix matches on the destination IP address. The next step is to join the two rulesets. The result of the join is a ruleset with rules that only match if both, the corresponding firewall rule fur and the corresponding routing rule rr matches. The data accompanying r is the port from rr and the firewall decision from fur. Next, descending priorities are added to the rules using map (apfst word-of-nat) o annotate-rlen. If the number of rules is too large to fit into the $2^{16}$ priority classes, an error is returned. Otherwise, the function pack-OF-entries is used to convert the (16 word \times 32 simple-match \times char list \times simple-action) list to an OpenFlow table. While converting the char list \times simple-action tuple is straightforward, converting the simple-match to an equivalent list of of-match-field set is non-trivial. This is done by the function simple-match-to-of-match.

The main difficulties for simple-match-to-of-match lie in making sure that the prerequisites are satisfied and in the fact that a simple-match operates on slightly stronger match expressions.

- A simple-match allows a (string) prefix match on the input and output interfaces. Given a list of existing interfaces on the router ifs, the function has to insert flow entries for each interface matching the prefix.
- A simple-match can match ports by an interval. Now it becomes obvious why Section 1.2.1 added bitmasks to $L4Src$ and $L4Dst$. Using the algorithm to split word intervals into intervals that can be represented by prefix matches from [6], we can efficiently represent the original interval by a few (32 in the worst case) prefix matches and insert flow entries for each of them.\footnote{It might be possible to represent the interval match more efficiently than a split into prefixes. However, that would produce overlapping matches (which is not a problem if we assign separate priorities) and we did not have a verified implementation of an algorithm that does so.}

The following lemma characterizes simple-match-to-of-match:

- **Lemma:** simple-match-to-of-match:
  - **Assumes:**
    - simple-match-valid r
    - p-interface p ∈ set ifs
    - match-interface (oiface r) (p-oiface p)
    - p-l2type p = 0x800
  - **Shows:** simple-matches r p ←→ (∃ gr ∈ set (simple-match-to-of-match r ifs)). OF-match-fields gr p = Some True
  - **Using:** assms simple-match-to-of-matchD simple-match-to-of-match/ by blast

The assumptions are to be read as follows:
The conclusion then states that the\textit{simple-match}\(r\) matches iff an element of the result of\textit{simple-match-to-of-match} matches. The third assumption is part of the explanation why we did not use \textit{simple-linux-router-altered}: \textit{simple-match-to-of-match} cannot deal with output interface matches. Thus, before passing a generalized simple firewall to \textit{pack-OF-entries}, we would have to set the output ports to \textit{ifaceAny}. A system replace output interface matches with destination IP addresses has already been formalized and will be published in a future version of [5]. For now, we limit ourselves to firewalls that do not do output port matching, i.e., we require \textit{no-oif-match}.

Given discussed properties, we present the central theorem for our translation in Figure 2. The first two assumptions are limitations on the traffic we make a statement about. Obviously, we will never see any packets with an input interface that is not in the interface list. Furthermore, we do not state anything about non-IPv4 traffic. (The traffic will remain unmatched in by the flow table, but we have not verified that.) The last assumption is that the translation does not return a run-time error. The translation will return a run-time error if the rules can not be assigned priorities from a 16 bit integer, or when one of the following conditions on the input data is not satisfied:

\textbf{lemma} \\

\textit{lr-of-tran} \(rt \ fw \ ifs \equiv\) \\
\textit{if} \ ¬ \ (\textit{no-oif-match}\(fw\) \(\land\) \textit{has-default-policy}\(fw\) \(\land\) \textit{simple-fw-valid}\(fw\) \(\land\) \textit{valid-prefixes}\(rt\) \(\land\) \textit{has-default-route}\(rt\) \(\land\) \textit{distinct}\(ifs\)) \\
\textit{then} \text{Inl} \"Error in creating OpenFlow table: prerequisites not satisfied\" \\
\textit{else} ( \\
\hspace{1em} \text{let} \ (\\n\hspace{2em} \text{nfw} = \text{map simple-rule-dtor}\(fw\); \\
\hspace{2em} \text{frt} = \text{map (\text{Ar. (route2match}\(r\), output-iface (routing-action}\(r\))))}\(rt\); \\
\hspace{2em} \text{ndr} = \text{generalized-fw-join}\(frt\)\(nfw\); \\
\hspace{2em} \text{ar} = \text{(map (apfst of-nat) \(\circ\) annotate-rlen)}\(ndr\) \\
\hspace{2em} \text{in} \\
\hspace{3em} \text{if} \ \text{length}\(ndr\) < \text{unat} (\text{-} 1::16) \text{word} \\
\hspace{3em} \text{then} \text{Inr} \ (\text{pack-OF-entries}\(ifs\)\(ar\)) \\
\hspace{3em} \text{else} \text{Inl} \ "Error in creating OpenFlow table: priority number space exhausted" \\
\hspace{1em}) \\
\text{unfolding} \text{Let-def} \text{lr-of-tran-def} \text{lr-of-tran-fbs-def} \text{lr-of-tran-s1-def} \text{comp-def} \text{route2match-def} \text{by force}
Theorem

Fixes
\( p :: (32', 'a) \) simple-packet-ext-scheme

Assumes
\( p\text{-iiface} p \in \text{set ifs} \) and \( p\text{-l2type} p = 0x800 \)
\( lr\text{-of-tran rt fw ifs} = \text{Inr oft} \)

Shows
\( \text{OF-priority-match OF-match-fields-safe oft p} = \text{Action [Forward oif]} \leftrightarrow \text{simple-linux-router-nol12 rt fw p} = \) (Some (p[p\text{-iiface} := oif]))
\( \text{OF-priority-match OF-match-fields-safe oft p} = \text{Action []} \leftrightarrow \text{simple-linux-router-nol12 rt fw p} = \text{None} \)
\( \text{OF-priority-match OF-match-fields-safe oft p} \neq \text{NoAction} \)
\( \text{OF-priority-match OF-match-fields-safe oft p} \neq \text{Undefined} \)
\( \exists ls. \text{length ls} \leq 1 \land \text{OF-priority-match OF-match-fields-safe oft p} = \text{Action ls} \)

Using
\( \text{assms lr-of-tran-correct by simp-all} \)

Figure 2: Central theorem on \( lr\text{-of-tran} \)

\( \exists err. \text{lr\text{-of-tran rt fw ifs} = Inl err} \) unfolding \( lr\text{-of-tran-def} \)
by(simp split: if-splits)

1.3.3 Comparison to Exodus

We are not the first researchers to attempt automated static migration to SDN. The (only) other attempt we are aware of is Exodus by Nelson et al. [10].

There are some fundamental differences between Exodus and our work:

- Exodus focuses on Cisco IOS instead of Linux.
- Exodus is not limited to using a single flow table.
- Exodus requires continuous controller interaction for some of its functions.
- Exodus attempts to support as much functionality as possible and has implemented support for dynamic routing, VLANs and NAT.
- Nelson et al. reject the idea that the translation could or should be proven correct.

2 Evaluation

In Section 1, we have made lots of definitions and created lots of models. How far these models are in accordance with the real world has been up to the vigilance of the reader. This section attempts to alleviate this burden by providing some examples.

2.1 Mininet Examples

The first example is designed to be minimal while still showing the most important properties of our conversion. For this purpose, we used a Linux firewall F, that we want to convert. We gave it two interfaces, and connected one client each. Its original configuration and the ruleset resulting from the translation is shown in Figure 3. (The list of interfaces can be extracted from the routing table; \texttt{s1-lan} received port number 1.) While the configuration does not fulfill any special function (especially, no traffic from the interface \texttt{s1-wan} is permitted), it is small enough to let us have a detailed look. More specifically, we can see how the only firewall rule (Line 2) got combined with the first rule of the routing table to form Line 1 of the OpenFlow rules. This also shows why the bitmasks on the layer 4 ports are necessary. If we only allowed exact matches, we would have \( 2^{15} \) rules instead of just one. Line 2 of the OpenFlow ruleset has been formed by combining the default drop policy with Line 1 of the routing table.
In a similar fashion, Line 2 of the routing rules has also been combined with the two firewall rules. However, as 10.0.2.0/24 from the firewall and 10.0.1.0/24 from the routing table have no common elements, no rule results from combining Line 2 and Line 2. In a similar fashion, the rest of the OpenFlow ruleset can be explained.

We feel that it is also worth noting again that it is necessary to change the IP configuration of the two devices attached to F. Assuming they are currently configured with, e.g., 10.0.1.100/24 and 10.0.2.1/24, the subnet would have to be changed from 24 to 22 or lower to ensure that a common subnet is formed and the MAC layer can function properly.

Next, we show a somewhat more evolved example. Its topology is depicted in Figure 4a. As before, we called the device to be replaced F. It is supposed to implement the simple policy that the clients H1 and H2 are allowed to communicate with the outside world via HTTP, ICMP is generally allowed, any other traffic is to be dropped (we neglected DNS for this example). We used the iptables configuration that is shown in Figure 4b. The routing table is the same as in the first example network.

The topology has been chosen for a number of reasons: we wanted one device which is not inside a common subnet with F and thus requires no reconfiguration for the translation. Moreover, we wanted two devices in a network that can communicate with each other while being overheard by F. For this purpose, we added two clients H1 and H2 instead of just one. We connected them with a broadcasting device.10 Executing our conversion function results in 36 rules11, we decided not to include them here. Comparing to the first example network, the size of the ruleset seems relatively high. This can be explained by the port matches: 1024-65535 has to be expressed by 6 different matches, \( tp_{src}=1024/0xfc00 \), \( tp_{src}=2048/0xf800 \), ... , \( tp_{src}=32768/0x8000 \) (or \( tp_{dst} \) respectively). When installing these rules, we also have to move all of H1, H2 and S1 into a common subnet. We chose 10.0.0.0/16 and updated the IP configuration of the three hosts accordingly. As discussed, the configuration of S2 did not have to be updated, as it does not share any subnet with F. We then tested reachability for TCP 22 and 80 and ICMP. The connectivity between all pairs of hosts (H1,H2,S1 and S2) remained the same compared to before the conversion. This shows that the concept can be made to work.

However, the example also reveals a flaw: When substituting the more complete model of a linux firewall with the simple one in Section 1.1, we assumed that the check whether the correct MAC address is set and the packets are destined for the modelled device would never fail — we assumed that all traffic arriving at a device is actually destined for it. Obviously, this network violates

\[ \text{(c) Resulting OpenFlow rules} \]

Figure 3: Example Network 1 – Configuration

10For the lack of a hub in mininet, we emulated one with an OpenFlow switch.
11If we had implemented some spoofing protection by adding \( -s 10.0.0.0/24 \) to the respective rule, the number of rules would have been increased to 312. This is because a cross product of two prefix splits would occur.
this assumption. We can trigger this in many ways, for example by sending an ICMP ping from H1 to H2. This will cause the generated rule priority=7, icmp, nw_dst=10.0.1.0/24 actions=output:1 (where port 1 is the port facing H1 and H2) to be activated twice. This is obviously not desired behavior. Dealing with this is, as mentioned, future work.

2.2 Performance Evaluation

Unfortunately, we do not have any real-world data that does not use output port matches as required in Section 1.3. There is thus no way to run the translation on the real-world firewall rulesets we have available and obtain a meaningful result. Nevertheless, we can use a real-world ruleset to evaluate the performance of our translation. For this purpose, we picked the largest firewall from the firewall collection from [6]. A significant amount of time is necessary to convert its FORWARD chain including 4946 rules to the required simplified firewall form. Additionally to the simplified firewall, we acquired the routing table (26 entries) from the same machine. We then evaluated the time necessary to complete the translation and the size of the resulting ruleset when using only the first \( n \) simple firewall rules and the full routing table. The result is shown in Figure 5.

12In the pre-parsed and already normalized version we used for this benchmark, it took 45s. The full required time lies closer to 11min as stated in [6].
the firewall match might only contain an output port and the rule can thus only apply for the packets matching a few routing table entries. However, the translation is not aware of that and can thus not remove the combination of the firewall rule and other routing table entries.

In some rules, the conditions above coincide, resulting in 416 (= 16 · 26) rules. To avoid the high number of rules resulting from the port matches, rules that forbid packets with source or destination port 0 could be added to the start of the firewall and the 1-65535 could be removed; dealing with the firewall / routing table problem is part of the future work on output interfaces.

3 Conclusion and Future Work

We believe that we have shown that it is possible to translate at least basic configurations of a linux firewall into OpenFlow rulesets while preserving the most important aspects of the behavior. We recognize that our system has limited practical applicability. One possible example would be a router or firewall inside a company network whose state tables have been polluted by special attack traffic. Our translation could provide an OpenFlow based stateless replacement. However, given the current prerequisites the implementation has on the configuration, this application is relatively unlikely.

For the configuration translation, we have contributed formal models of a linux firewall and of an OpenFlow switch. Furthermore, the function that joins two firewalls and the function that translates a simplified match from [6] to a list of equivalent OpenFlow field match sets are contributions that we think are likely to be of further use.

We want to explicitly formulate the following two goals for our future work:

- We want to deal with output interface matches. The idea is to formulate and verify a destination interface / destination IP address rewriting that can exchange output interfaces and destination IP addressed in a firewall, based on the information from the routing table.\(^\text{13}\)

- We want to develop a system that can provide a stricter approximation of stateful matches so our translation will be applicable in more cases.

\(^{13}\)As of now this has already been implemented, but is not yet fully ready.

References


