SWIFT MAC Protocol: HOL Specification

Adam Biltcliffe
Michael Dales
Sam Jansen
Tom Ridge
Peter Sewell

June 12, 2006

Abstract

This document is typeset from a HOL specification. Further details of the model described in this document can be found in the paper "Rigorous Protocol Design in Practice: An Optical Packet-Switch MAC in HOL", available on the web at http://www.cl.cam.ac.uk/users/pes20/optical/root.pdf.

Rule version:

Contents

Chapter 1

Basic Types and Definitions

1.1 Types

1.1.1 Summary

type_abbrev_mac
type_abbrev_port
type_abbrev_pingid
type_abbrev_fabric_state
type_abbrev_data

1.1.2 Rules

– :
type_abbrev *mac* : *word16*

– :
type_abbrev port : *word16*

– :
type_abbrev *pingid* : *word32*

– :
type_abbrev fabric_state : (port#port)set

– :
type_abbrev *data* : string

1.2 Continuous Time

1.2.1 Summary

type_abbrev_time
NSEC
USEC
MSEC
SEC

1.2.2 Rules

– :
type_abbrev time : *real*

– :
(NSEC : time) = 1/1000000000

Rule version:

– :
$(\mathrm{USEC} : \mathsf{time}) = 1/1000000$

– :
$(\mathrm{MSEC} : \mathsf{time}) = 1/1000$

– :
$(\mathrm{SEC} : \mathsf{time}) = 1$

1.3 Library Functions

1.3.1 Summary

INTERVAL
REMOVE1

1.3.2 Rules

– :
$\mathrm{INTERVAL}(lower : \mathsf{time}, upper) = \{x \mid lower \leq x \wedge x \leq upper\}$

Rule version:

– :
(REMOVE1 $x[\,] = [\,]) \wedge$
(REMOVE1 $x(CONS\ y\ ys) =$ **if** $x = y$ **then** ys **else** REMOVE1 $x\ ys$)

1.4 Parameters

1.4.1 Summary

TRANSMISSION_TIME
SLOP_TIME
MIN_PING_REUSE_TIME
CONNECTED_PING_REPEAT_TIME
UNCONNECTED_PING_REPEAT_TIME
MAX_HOST_THINKING_TIME
ARBITER_WRITEOFF_TIME
REQUEST_REISSUE_TIME
TIMER_GRAININESS

1.4.2 Rules

– :
TRANSMISSION_TIME $= 9 *$ USEC

Description The time taken for a packet of data to pass through the switch.

– :
SLOP_TIME $= 500 *$ NSEC

Description
The fabric state is guaranteed by the arbiter for at least a period of TRANSMISSION_TIME $+2 *$ SLOP_TIME.

Rule version:

– :
MIN_PING_REUSE_TIME : time = 0 ∗ NSEC

Description The amount of time which must pass before a pingid can be reused on the same port.

– :
CONNECTED_PING_REPEAT_TIME = 10 ∗ MSEC

Description The interval between pings sent to a host which the arbiter believes is connected.

– :
UNCONNECTED_PING_REPEAT_TIME = 10 ∗ USEC

Description The interval between 'speculative' pings sent to a port on which the arbiter does not know of a host.

– :
MAX_HOST_THINKING_TIME = 200 ∗ NSEC

Description The maximum 'logic delay' before a host responds to a ping or data-grant message.

– :
ARBITER_WRITEOFF_TIME : time = 100 ∗ MSEC

Description The amount of time which must pass for a host without a ping arriving before the host decides to assume the arbiter has crashed.

– :
REQUEST_REISSUE_TIME = 500 ∗ USEC

Description The amount of time which must pass before a host retransmits a data-request on the assumption that it previously got lost.

– :
TIMER_GRAININESS = 200 ∗ NSEC

Description This is the maximum time which can elapse between when a timer should go off and when it actually does go off (on the next clock cycle).

Chapter 2

Arbiter/ Arbiter

2.1 Messages, Labels

2.1.1 Summary

a2h_msg
h2a_msg
type_abbrev_arbiter_trace

2.1.2 Rules

– **:**
a2h_msg =
  A2H_DATA_GRANT **of** *mac*
| A2H_PING **of** *pingid*
| A2H_MAC_GRANT **of** *mac*
| A2H_MASTER_IS **of** *mac*

– **:**
h2a_msg =
  H2A_DATA_REQUEST **of** *mac*
| H2A_PONG **of** *pingid*

| H2A_MAC_REQUEST(* no info needed *)
| H2A_WHO_IS_MASTER

– :
type_abbrev arbiter_trace : num → arbiter_label

2.2 Functions on Traces

2.2.1 Summary

a_time
a_fabric
!
a_last_ping
a_rtt_estimate

2.2.2 Rules

– :
(a_time(t : arbiter_trace)(0 : num) = (0 : time)) $\wedge$
(a_time $t(SUC\ n)$ =
let d = (**case** $(t\ n)$ **of** A_DUR $x \rightarrow x$ ‖ _ → 0) **in**
d + a_time $t\ n$)

– :
(a_fabric t 0 = {}) $\wedge$
(a_fabric $t(SUC\ n)$ = **case** $(t\ n)$ **of** A_FABRIC $fs \rightarrow fs$ ‖ _ → a_fabric $t\ n$)

Rule version:

– :

$(\forall t. port\_of\_mac (t : \text{arbiter\_trace}) 0 = \emptyset) \wedge$
$(\forall t\ m. port\_of\_mac\ t (SUC\ m) =$
$\quad \textbf{let}\ \ oldmap = port\_of\_mac\ t\ m\ \ \textbf{in}$
$\quad \textbf{case}\ \ t\ m\ \ \textbf{of}$
$\qquad\quad \text{A\_D{\small ARK}}\ p_1 \rightarrow \textbf{rrestrict}\ \ oldmap (\lambda p. \neg (p = p_1))$
$\qquad \|\ \text{A\_A2H}(p_2, \text{A2H\_MAC\_GRANT}\ mac) \rightarrow oldmap \oplus (mac \mapsto p_2)$
$\qquad \|\ \_ \rightarrow \textbf{let}\ \ pings\_timed\_out = \lambda p. p\ \in \textbf{rng}(oldmap) \wedge \textbf{F}\ \ \textbf{in}$
$\qquad\qquad \textbf{rrestrict}\ \ oldmap (\lambda p. \neg pings\_timed\_out\ p))$

– :

$(\text{a\_last\_ping}\ \ t\ id\ 0 = *) \wedge$
$(\text{a\_last\_ping}\ \ t\ id (SUC\ m) = (\textbf{case}\ \ t\ m\ \ \textbf{of}$
$\qquad \text{A\_A2H}(p, \text{A2H\_PING}\ id') \rightarrow (\textbf{case}\ \ id' = id\ \ \textbf{of}\ \ \textbf{T} \rightarrow \uparrow m \ \|\ \textbf{F} \rightarrow \text{a\_last\_ping}\ \ t\ id\ m)$
$\quad \|\ \_ \rightarrow \text{a\_last\_ping}\ \ t\ id\ m))$

– :

(* *rtt_estimate* is calculated per port (arbiter-host connection) and id by keeping a record of the last *PING* sent *)
(* N.B. *PING* and *PONG* messages are identified by *pingid* not by port *)

$(\text{a\_rtt\_estimate}\ \ t\ p\ 0 = *) \wedge$
$(\text{a\_rtt\_estimate}\ \ t\ p (SUC\ m) = (\textbf{case}\ (t\ m)\ \textbf{of}$
$\qquad \text{A\_H2A}(p', \text{H2A\_PONG}\ id) \rightarrow (\textbf{if}\ \ p' = p$
$\qquad\quad \textbf{then}\ (\textbf{case}\ (\text{a\_last\_ping}\ \ t\ id\ m)\ \textbf{of}\ \uparrow m' \rightarrow \uparrow(\text{a\_time}\ \ t\ m - \text{a\_time}\ \ t\ m') \ \|\ * \rightarrow *)$
$\qquad\quad \textbf{else}\ \ \text{a\_rtt\_estimate}\ \ t\ p\ m)$
$\quad \|\ \_ \rightarrow \text{a\_rtt\_estimate}\ \ t\ p\ m))$

2.3 Specification

2.3.1 Summary

grants_correctly_arbitered
starts_pinging
continues_pinging
pings_correctly_spaced

Rule version:

pingids_not_reused_too_soon
data_requests_get_granted
only_talk_to_ports_with_macs
one_mac_per_port
mac_requests_are_granted
one_port_per_mac
slots_not_wasted
arbiter_spec

2.3.2 Rules

– :
grants_correctly_arbitered t =

$\forall n\ psrc\ mac.$
$(t\ n = \text{A\_A2H}(psrc, \text{A2H\_DATA\_GRANT}\ mac)) \implies$
let $rtt\_est = \text{a\_rtt\_estimate}\ t\ psrc\ n$ **in**
case rtt_est **of** $* \rightarrow \mathbf{F} \parallel \uparrow rtt \rightarrow$
$mac \in \mathbf{dom}((port\_of\_mac\ t\ n)) \wedge$
let $pdst = (port\_of\_mac\ t\ n)[mac]$ **in**
let $tn = \text{a\_time}\ t\ n$ **in**
let $low\_time = tn + max(rtt - \text{SLOP\_TIME})0$ **in**
let $high\_time =$
$tn + rtt + \text{TRANSMISSION\_TIME} + \text{SLOP\_TIME}$ **in**
$\exists low\ high.\ \text{a\_time}\ t\ low \leq low\_time \wedge$
$high\_time \leq \text{a\_time}\ t\ high \wedge$
$\forall n.low \leq n \wedge n \leq high \implies (psrc, pdst) \in \text{a\_fabric}\ t\ n$

– :
starts_pinging $ports(t : \text{arbiter\_trace}) =$

$\forall p.\ \mathbf{mem}\ p\ ports \implies$
$\exists n\ pingid.(t\ n = \text{A\_A2H}(p, \text{A2H\_PING}\ pingid)) \wedge$
$\text{a\_time}\ t\ n \leq \text{UNCONNECTED\_PING\_REPEAT\_TIME}$

Rule version:

– :
continues_pinging t =

(* the arbiter repeatedly pings all ports, as long as time increases *)
$(\forall n\ p.$
$(\exists pingid.t\ n = \text{A\_A2H}(p, \text{A2H\_PING}\ pingid)) \wedge$
$(\exists n'.n < n' \wedge$
$(max\ \text{CONNECTED\_PING\_REPEAT\_TIME}\ \text{UNCONNECTED\_PING\_REPEAT\_TIME}) +$
$\text{TIMER\_GRAININESS} \leq \text{a\_time}\ \ t\ n' - \text{a\_time}\ \ t\ n) \implies$
$(\exists n'.n < n' \wedge (\exists pingid.t\ n' = \text{A\_A2H}(p, \text{A2H\_PING}\ pingid))))$

– :
pings_correctly_spaced t =

(* the interval between consecutive pings on a given port is determined by the arbiter's view of the status of the host on that port *)
$(\forall n\ n'\ p\ pingid\ pingid'.$
$n < n' \wedge$
$(t\ n = \text{A\_A2H}(p, \text{A2H\_PING}\ pingid)) \wedge$
$(t\ n' = \text{A\_A2H}(p, \text{A2H\_PING}\ pingid')) \wedge$
$(\forall m.n < m \wedge m < n' \implies \neg\exists pingid''.t\ m = \text{A\_A2H}(p, \text{A2H\_PING}\ pingid'')) \implies$
let $port\_connected\_n = p\ \in \mathbf{rng}((port\_of\_mac\ t\ n))$ **in**
let $port\_connected\_n' = p\ \in \mathbf{rng}((port\_of\_mac\ t\ n'))$ **in**
let $port\_connected\_agree = (port\_connected\_n = port\_connected\_n')$ **in**
let $delta = (\text{a\_time}\ \ t\ n' - \text{a\_time}\ \ t\ n)$ **in**
if $port\_connected\_agree$ **then** (
(* note that it may be possible to change connectedness twice between n and n' *)
let $ping\_period$ = (**if** $port\_connected\_n$
then CONNECTED_PING_REPEAT_TIME
else UNCONNECTED_PING_REPEAT_TIME) **in**
$delta\ \in\ \text{INTERVAL}(ping\_period, ping\_period + \text{TIMER\_GRAININESS}))$
else $delta < (max\ \text{CONNECTED\_PING\_REPEAT\_TIME}$
$\text{UNCONNECTED\_PING\_REPEAT\_TIME})$
$+\ \text{TIMER\_GRAININESS})$

– :
pingids_not_reused_too_soon t =

(* the arbiter does not reuse *pingid*s within MIN_PING_REUSE_TIME *)
(* don't reuse pingids too much - in some long time interval, don't reuse the same ping id for the same port... *)

Rule version:

(* N.B. this doesn't notice whether the port has gone dark in the intervening time, or whether hosts disconnect and reconnect *)

$$(\forall n\ p\ p'\ pingid\ n'.\\ (t\ n = \text{A\_A2H}(p, \text{A2H\_PING}(pingid))) \wedge\\ (t\ n' = \text{A\_A2H}(p', \text{A2H\_PING}(pingid))) \wedge\\ (n' > n)\\ \Longrightarrow\\ \text{a\_time}\ \ t\ n' - \text{a\_time}\ \ t\ n \geq \text{MIN\_PING\_REUSE\_TIME})$$

– :
data_requests_get_granted $t =$

(* if the arbiter receives a request, it eventually sends a grant *)

$$\forall n\ p\ mac.\\ (t\ n = \text{A\_H2A}(p, \text{H2A\_DATA\_REQUEST}\ mac)) \wedge\\ (mac\ \in \mathbf{dom}((port\_of\_mac\ t\ n)))\\ \Longrightarrow\\ \exists m.m > n \wedge (t\ m = \text{A\_A2H}(p, \text{A2H\_DATA\_GRANT}\ mac))$$

– :
only_talk_to_ports_with_macs $t =$

(* the arbiter does not send anything (except mac-grants and pings) to ports with no assigned mac *)

$$(\forall n\ p\ msg.\\ (t\ n = \text{A\_A2H}(p, msg)) \wedge\\ \neg(\exists mac.msg = \text{A2H\_MAC\_GRANT}\ mac) \wedge\\ \neg(\exists pingid.msg = \text{A2H\_PING}\ pingid)\\ \Longrightarrow\\ p\ \in \mathbf{rng}((port\_of\_mac\ t\ n)))$$

– :
one_mac_per_port(t : arbiter_trace) =

(* the arbiter does not send two conflicting macs to the same port without a mac-request or a dark or a timeout on a series of pings inbetween *)

Rule version:

$(\forall p.\forall n\ mac\ n'\ mac'.$
$(t\ n = \text{A\_A2H}(p, \text{A2H\_MAC\_GRANT}\ mac)) \wedge$
$(t\ n' = \text{A\_A2H}(p, \text{A2H\_MAC\_GRANT}\ mac')) \wedge$
$(n < n') \wedge$
$\neg(mac = mac')$
$\implies$
$\exists n''.n < n'' \wedge$
$n'' < n' \wedge$
$((t\ n'' = \text{A\_H2A}(p, \text{H2A\_MAC\_REQUEST})) \vee$
$(t\ n'' = \text{A\_D}{\small \text{ARK}}(p)) \vee$
(* Placeholder for timeout on a series of pings between n and n'. *)
$\mathbf{F}))$

– :
mac_requests_are_granted(t : arbiter_trace) =
(* if the arbiter receives a mac request, it eventually sends a grant *)
$(\forall n\ p.(t\ n = \text{A\_H2A}(p, \text{H2A\_MAC\_REQUEST})) \implies$
$\exists n'\ mac.n' > n \wedge$
$(t\ n' = \text{A\_A2H}(p, \text{A2H\_MAC\_GRANT}\ mac)))$

– :
one_port_per_mac t =
(* the arbiter never assigns the same mac to more than one port *)
$(\forall n\ p\ mac.$
$(t\ n = \text{A\_A2H}(p, \text{A2H\_MAC\_GRANT}\ mac)) \wedge$
$mac\ \in \mathbf{dom}((port\_of\_mac\ t\ n))$
$\implies ((port\_of\_mac\ t\ n[mac]) = p))$

– :
slots_not_wasted = **T** (* performance concern: if you have any pending requests, grant at least one in each slot - want to think about this, since the spec doesn't yet enforce the notion of slot-based grant allocation *)

Rule version:

– :

arbiter_spec *ports*(t : arbiter_trace) =

grants_correctly_arbitered t ∧
starts_pinging *ports* t ∧
continues_pinging t ∧
pings_correctly_spaced t ∧
pingids_not_reused_too_soon t ∧
data_requests_get_granted t ∧
only_talk_to_ports_with_macs t ∧
one_mac_per_port t ∧
mac_requests_are_granted t ∧
one_port_per_mac t

Chapter 3

Hosts

3.1 Messages, Labels

3.1.1 Summary

h_lbl
type_abbrev_host_trace

3.1.2 Rules

```
– :
h_lbl =
      H_TAU
    | H_DUR of time
    | H_DARK
    | H_A2H of a2h_msg
    | H_H2A of h2a_msg
    | H_D2H of d2h_msg
    | H_H2D of h2d_msg
    | H_H2S of h2s_msg(* N.B. implementations will probably ignore the MAC FIXME do we want MAC here? *)
    | H_S2H of s2h_msg
```

– :
type_abbrev host_trace : num → h_lbl

3.2 Functions on Traces

3.2.1 Summary

host_mac	
h_time	
mac_of_data	
pending_output	
h_wedged	identifies when the dataplane should think that the host is wedged

3.2.2 Rules

– :

```
(host_mac t 0 = ∗) ∧
(host_mac t(SUC n) = (let mac = host_mac t n in
                        case t n of
                          H_TAU → mac
                        ‖ H_DUR d → mac
                        ‖ H_DARK → ∗ (* already covered by H2D_WEDGED *)
                        ‖ H_A2H om → (case om of
                                A2H_MAC_GRANT mac′ → ↑ mac′
                              ‖ _x90 → mac)
                        ‖ H_H2A im → mac
                        ‖ H_D2H d2h_msg → (case d2h_msg of
                              D2H_INVALIDATE_MY_MAC → ∗
                              ‖ _ → mac)
                        ‖ H_H2D h2d_msg → (case h2d_msg of
                                H2D_WEDGED → ∗
                              ‖ _x89 → mac)
                        ‖ H_H2S _1 → mac
                        ‖ H_S2H _2 → mac
                       ))
```

Rule version:

– :
(h_time $t(0 : \mathsf{num}) = (0 : \mathsf{time})) \wedge$
(h_time $t(SUC\ n) =$ (**let** $d =$ (**case** $(t\ n)$ **of** H_DUR $x \rightarrow x \parallel \_ \rightarrow 0)$ **in**
$d +$ h_time $t\ n))$

– :
(mac_of_data $t\ 0 = ARB) \wedge$
(mac_of_data $t(SUC\ n) =$
case $t\ n$ **of** (H_A2H(A2H_DATA_GRANT *mac*)) $\rightarrow$ *mac* $\parallel \_ \rightarrow$ mac_of_data $t\ n$)

– :
(pending_output(t : host_trace)*mac* $0 = [\,]) \wedge$
(pending_output(t : host_trace)*mac*$(SUC\ n) =$
let pending_output = (pending_output t *mac* n) **in**
case $t\ n$ **of**
H_D2H(D2H_SEND(mac', *data*)) $\rightarrow$ **if** $mac' = mac$
then pending_output $++$ [*data*] **else** pending_output
$\parallel$ H_H2S(H2S_DATA *data*) $\rightarrow$ **if** mac_of_data $t\ n = mac$
then *TL* pending_output **else** pending_output
$\parallel$ H_DARK $\rightarrow [\,]$
$\parallel \_ \rightarrow$ pending_output)

– identifies when the dataplane should think that the host is wedged :
(h_wedged $t\ 0 =$ **F**) $\wedge$
(h_wedged $t(SUC\ n) =$
let *wedged* = h_wedged $t\ n$ **in**
case $t\ n$ **of**
H_TAU $\rightarrow$ *wedged*
$\parallel$ H_DUR *_1* $\rightarrow$ *wedged*
$\parallel$ H_DARK $\rightarrow$ *wedged*
$\parallel$ H_A2H *_2* $\rightarrow$ *wedged*(* receipt of messages unwedges *)
$\parallel$ H_H2A *_3* $\rightarrow$ *wedged*
$\parallel$ H_D2H *_4* $\rightarrow$ *wedged*
$\parallel$ H_H2D h2d_msg $\rightarrow$ (**case** h2d_msg **of**
H2D_RECV *data* $\rightarrow$ **F** (* i.e. the only way to reset wedged is to receive data *)
$\parallel$ H2D_WEDGED $\rightarrow$ **T**

Rule version:

$\quad\quad \parallel$ H2D_I_AM *mac* $\rightarrow$ *wedged*)
$\quad \parallel$ H_H2S *_6* $\rightarrow$ *wedged*
$\quad \parallel$ H_S2H *_7* $\rightarrow$ *wedged*)

3.3 Specification

3.3.1 Summary

sends_onlyif_grant
only_sends_pending_output
replies_to_pings
sends_requests
responds_to_grants
asks_for_macs
notices_ping_absence
notices_dark
resends_requests
host_spec

3.3.2 Rules

– :
sends_onlyif_grant t =

(* if you send H2S_DATA then you must have previously received A2H_DATA_GRANT strictly within MAX_HOST_THINKING_TIME *)
(* N.B. lack of symmetry with the above- no check for pending output *)
(* $\exists ! n'$ guarantees functional relation between H2S_DATA send and A2H_DATA_GRANT received *)

$\forall n\ data.(t\ n = \text{H\_H2S}(\text{H2S\_DATA}\ data)) \implies$
$\exists(*!*) n'.\exists mac. n' < n\ \wedge$
$(t\ n' = \text{H\_A2H}(\text{A2H\_DATA\_GRANT}\ mac))\ \wedge$
$\text{h\_time}\ \ t\ n - \text{h\_time}\ \ t\ n' < \text{MAX\_HOST\_THINKING\_TIME}$

– :

Rule version:

only_sends_pending_output t =

$\forall n\ data.(t\ n = \text{H\_H2S}(\text{H2S\_DATA}\ data)) \implies$
let $mac = \text{mac\_of\_data}\ t\ n$ **in**
$\exists xs.\ \text{pending\_output}\ t\ mac\ n = (data :: xs)$

– :
replies_to_pings(t : host_trace) =

(* replies to pings strictly before MAX_HOST_THINKING_TIME elapses *)
$\forall n\ pingid.(t\ n = \text{H\_A2H}(\text{A2H\_PING}\ pingid)) \implies$
$\exists m.n < m \land$
$(t\ m = \text{H\_H2A}(\text{H2A\_PONG}\ pingid)) \land$
$\text{h\_time}\ t\ m - \text{h\_time}\ t\ n < \text{MAX\_HOST\_THINKING\_TIME}$

– :
sends_requests(t : host_trace) =

(* if you receive D2H_SEND, then you should send H2A_DATA_REQUEST *)
$\forall n\ mac\ data.(t\ n = \text{H\_D2H}(\text{D2H\_SEND}(mac, data))) \implies$
$\exists n'.n < n' \land (t\ n' = \text{H\_H2A}(\text{H2A\_DATA\_REQUEST}\ mac))$

– :
responds_to_grants(t : host_trace) =

(* if you receive A2H_DATA_GRANT then send H2S_DATA (signal the dataplane) strictly before MAX_HOST_THINKING_TIME *)

$(\forall n\ mac.(t\ n = \text{H\_A2H}(\text{A2H\_DATA\_GRANT}\ mac)) \implies$
let $p = \text{pending\_output}\ t\ mac\ n$ **in**
case p **of**
$[\,] \rightarrow \mathbf{T}$
$\|\ data :: ps \rightarrow$
$\exists n'.n < n' \land$
$(t\ n' = \text{H\_H2S}(\text{H2S\_DATA}\ data)) \land$
$\text{h\_time}\ t\ n' - \text{h\_time}\ t\ n < \text{MAX\_HOST\_THINKING\_TIME})$

Rule version:

$-$ **:**
asks_for_macs(t : host_trace) =

(* if you do have a mac, don't ask for one, contrapositively: if you ask for a mac, you shouldn't have one *)
$\forall n.(t\ n = \text{H\_H2A}(\text{H2A\_MAC\_REQUEST})) \implies (\text{host\_mac}\ \ t\ n = *)$

$-$ **:**
notices_ping_absence t =
$\forall n.$
$(\exists n'.n \leq n' \wedge \text{ARBITER\_WRITEOFF\_TIME} \leq \text{h\_time}\ \ t\ n' - \text{h\_time}\ \ t\ n\ \wedge$
$(\forall m.n \leq m \wedge m < n' \implies \neg\exists \mathit{pingid}.(t\ m = \text{H\_A2H}(\text{A2H\_PING}\ \mathit{pingid})))) \implies$
$\exists n'.n \leq n' \wedge$
$(\textbf{let}\ \ \mathit{delta} = \text{h\_time}\ \ t\ n' - \text{h\_time}\ \ t\ n\ \ \textbf{in}$
$\mathit{delta}\ \in\ \text{INTERVAL}(\text{ARBITER\_WRITEOFF\_TIME},$
$\text{ARBITER\_WRITEOFF\_TIME} + \text{TIMER\_GRAININESS})) \wedge$
$\text{h\_wedged}\ \ t\ n'$

$-$ **:**
notices_dark t =

(* if the line goes dark, send H2D_WEDGED strictly within MAX_HOST_THINKING_TIME *)
$\forall n.(t\ n = \text{H\_D}\textsc{ark}) \implies$
$\exists n'.n < n' \wedge$
$(t\ n' = \text{H\_H2D H2D\_WEDGED}) \wedge$
$\text{h\_time}\ \ t\ n' - \text{h\_time}\ \ t\ n < \text{MAX\_HOST\_THINKING\_TIME}$

$-$ **:**
resends_requests t =

(* if you do not receive A2H_DATA_GRANT strictly before REQUEST_REISSUE_TIME after sending H2A_DATA_REQUEST, then resend H2A_DATA_REQUEST *)

Rule version:

$\forall n\ mac.(t\ n = \mathrm{H\_H2A}(\mathrm{H2A\_DATA\_REQUEST}\ mac)) \wedge$
$(\forall n'.n < n' \wedge \mathrm{h\_time}\ \ t\ n' - \mathrm{h\_time}\ \ t\ n < \mathrm{REQUEST\_REISSUE\_TIME} \implies$
$\quad \neg(t\ n' = \mathrm{H\_A2H}(\mathrm{A2H\_DATA\_GRANT}\ mac))) \implies$
$\exists n'.n < n' \wedge$
$\mathrm{h\_time}\ \ t\ n' - \mathrm{h\_time}\ \ t\ n\ \in\ \mathrm{INTERVAL}(\mathrm{REQUEST\_REISSUE\_TIME},$
$\quad \mathrm{REQUEST\_REISSUE\_TIME} + \mathrm{TIMER\_GRAININESS}) \wedge$
$(t\ n' = \mathrm{H\_H2A}(\mathrm{H2A\_DATA\_REQUEST}\ mac))$

– **:**
host_spec(t : host_trace) =
 sends_onlyif_grant $t \wedge$
 only_sends_pending_output $t \wedge$
 replies_to_pings $t \wedge$
 sends_requests $t \wedge$
 responds_to_grants $t \wedge$
 asks_for_macs $t \wedge$
 notices_ping_absence $t \wedge$
 notices_dark $t \wedge$
 resends_requests t

Chapter 4

Arbiter Channels

4.1 Messages, Labels

4.1.1 Summary

ca_msg
ca_lbl
type_abbrev_ca_trace

4.1.2 Rules

– **:**
ca_msg =
    CA_A2H **of** a2h_msg
  | CA_H2A **of** h2a_msg

– **:**
ca_lbl =
    CA_TAU
  | CA_DUR **of** time
  | CA_A2C **of** time#a2h_msg
  | CA_C2H **of** a2h_msg

| CA_H2C **of** time#h2a_msg
| CA_C2A **of** h2a_msg

– **:**
type_abbrev *ca_trace* : num → ca_lbl

4.2 Functions on Traces

4.2.1 Summary

ca_msgs	channels start off empty

4.2.2 Rules

– **channels start off empty :**
(ca_msgs t 0 = [ ]) ∧
(ca_msgs $t(SUC\ n)$ = **case** $t\ n$ **of**
CA_TAU → ca_msgs $t\ n$
‖ CA_DUR d →
(**let** $f(t', c) = (t' - d, c)$ **in**
map f(ca_msgs $t\ n$))
‖ CA_A2C($t1$, $m1$) → ($t1$, CA_A2H $m1$) :: ca_msgs $t\ n$
‖ CA_C2H $m2$ → REMOVE1(0, CA_A2H $m2$)(ca_msgs $t\ n$)
‖ CA_H2C($t3$, $m3$) → ($t3$, CA_H2A $m3$) :: ca_msgs $t\ n$
‖ CA_C2A $m4$ → REMOVE1(0, CA_H2A $m4$)(ca_msgs $t\ n$))

4.3 Specification

4.3.1 Summary

arbiter_channel_spec

Rule version:

4.3.2 Rules

$\vdash$ **:**

arbiter_channel_spec t =
$\forall n.$**let** ms = ca_msgs t n **in**
($\forall m.$ **mem** (0, CA_A2H m) ms = (t n = CA_C2H m)) $\wedge$
($\forall m.$ **mem** (0, CA_H2A m) ms = (t n = CA_C2A m))

Rule version:

Chapter 5

Switches

5.1 Messages, Labels

5.1.1 Summary

s_lbl
type_abbrev_switch_trace

5.1.2 Rules

– :
s_lbl =
S_Tau
| S_Dur **of** time
| S_Fabric **of** fabric_state
| S_S2H **of** port#s2h_msg
| S_H2S **of** port#h2s_msg

– :
type_abbrev switch_trace : num → s_lbl

5.2 Functions on Traces

5.2.1 Summary

switch_time
switch_fabric_state

5.2.2 Rules

$$
\begin{array}{l}
\vdash: \\
(\text{switch\_time}\ \ t(0:\mathsf{num}) = (0:\mathsf{time})) \wedge \\
(\text{switch\_time}\ \ t(SUC\ n) = (\textbf{let}\ \ d = (\textbf{case}\ (t(SUC\ n))\ \textbf{of}\ \ \text{S\_DUR}\ x \to x \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \|\ \_ \to 0)\ \textbf{in} \\
\qquad\qquad\qquad\qquad\qquad d + \text{switch\_time}\ \ t\ n))
\end{array}
$$

$$
\begin{array}{l}
\vdash: \\
(\text{switch\_fabric\_state}\ \ t\ 0 = \{\}) \wedge \\
(\text{switch\_fabric\_state}\ \ t(SUC\ n) = (\\
\quad \textbf{case}\ \ t\ n\ \ \textbf{of} \\
\qquad\quad \text{S\_FABRIC}\ fs \to fs \\
\qquad \|\ \_ \to \text{switch\_fabric\_state}\ \ t\ n \\
))
\end{array}
$$

5.3 Specification

5.3.1 Summary

switch_spec

5.3.2 Rules

Rule version:

– :

switch_spec t =

(* Messages are only received if the fabric state is set appropriately. *)
$(\forall n\ psrc\ data.(t\ n = \mathrm{S\_H2S}(psrc, \mathrm{H2S\_DATA}\ data)) \implies$
$\exists pdst.(psrc, pdst) \in (\mathrm{switch\_fabric\_state}\ \ t\ n)) \wedge$

(* Messages are only sent if the fabric state is set appropriately. *)
$(\forall n\ pdst\ data.(t\ n = \mathrm{S\_S2H}(pdst, \mathrm{S2H\_DATA}\ data)) \implies$
$\exists psrc.(psrc, pdst) \in (\mathrm{switch\_fabric\_state}\ \ t\ n)) \wedge$

(* A message is sent only if it is received, and the fabric state stays constant for transmission time. *)
$(\forall n'\ psrc\ pdst\ data.$
$(t\ n' = \mathrm{S\_S2H}(pdst, \mathrm{S2H\_DATA}\ data)) \implies$
$\exists n.n < n' \wedge$
$(t\ n = \mathrm{S\_H2S}(psrc, \mathrm{H2S\_DATA}\ data)) \wedge$ (* Message received ... *)
$(\mathrm{switch\_time}\ \ t\ n' - \mathrm{switch\_time}\ \ t\ n = \mathrm{TRANSMISSION\_TIME}) \wedge$ (* ... TRANSMISSION_TIME ago *)
$(\forall m.n < m \wedge m < n' \implies \neg \exists fs.t\ m = \mathrm{S\_F{\small ABRIC}}\ fs)$ (* Fabric stays constant between receive and send *)
$) \wedge$

(* If a message is received, and the fabric state stays constant for transmission time, it is sent. *)
$(\forall n\ psrc\ data.$
(* Want to allow some Taus before the message has to be sent *)
$(t\ n = \mathrm{S\_H2S}(psrc, \mathrm{H2S\_DATA}\ data)) \wedge$
$(\exists n'.(\mathrm{TRANSMISSION\_TIME} = \mathrm{switch\_time}\ \ t(n+n') - \mathrm{switch\_time}\ \ t\ n) \wedge$
$(\forall m.m < n' \implies \neg \exists fs.t(n+m) = \mathrm{S\_F{\small ABRIC}}\ fs)) \implies$

(* N.B. could well be different n', with preceding S_FABRIC change. *)
$(\exists n'\ pdst.(\mathrm{TRANSMISSION\_TIME} = \mathrm{switch\_time}\ \ t(n+n') - \mathrm{switch\_time}\ \ t\ n) \wedge$
$(t(n+n') = \mathrm{S\_S2H}(pdst, \mathrm{S2H\_DATA}\ data)))$
$)$

Chapter 6

Switch Channels

6.1 Messages, Labels

6.1.1 Summary

cs_msg
cs_lbl
type_abbrev_cs_trace

6.1.2 Rules

– :
cs_msg =
  CS_S2H **of** s2h_msg
  | CS_H2S **of** h2s_msg

– :
cs_lbl =
  CS_Tau
  | CS_Dur **of** time
  | CS_H2CS **of** time#h2s_msg
  | CS_CS2S **of** h2s_msg

| CS_S2CS **of** time#s2h_msg
| CS_CS2H **of** s2h_msg

− **:**
type_abbrev *cs_trace* : num → cs_lbl

6.2 Specification

6.2.1 Rules

Chapter 7

Networks

7.1 Labels

7.1.1 Summary

n_lbl
type_abbrev_net_trace

7.1.2 Rules

– :
n_lbl =
N_Tau
| N_Dur **of** time
| N_Fabric **of** fabric_state
| N_Dark **of** port
| N_A2CA **of** port#a2h_msg#time
| N_CA2H **of** port#a2h_msg
| N_H2CA **of** port#h2a_msg#time
| N_CA2A **of** port#h2a_msg
| N_D2H **of** port#d2h_msg
| N_H2D **of** port#h2d_msg
| N_H2CS **of** port#h2s_msg#time
| N_CS2S **of** port#h2s_msg
| N_S2CS **of** port#s2h_msg#time
| N_CS2H **of** port#s2h_msg

– :
type_abbrev net_trace : num → n_lbl

7.2 Functions on Net Traces

7.2.1 Summary

net_project_port	
net_time	
net_fabric	
last_ping	
net_rtt_estimate	
net_mac	
net_pending_output	pending output, presumably for a given port, per destination mac

7.2.2 Rules

– :
(* Most definitions are per port, so first project the trace onto the port before applying the relevant function. *)

net_project_port $t\ p\ n =$ (**case** $t\ n$ **of**
  N_TAU → N_TAU
  ∥ N_DUR d → N_DUR d
  ∥ N_FABRIC fs → N_TAU
  ∥ N_DARK p' → (**case** $p' = p$ **of** **T** → N_DARK p ∥ **F** → N_TAU)
  ∥ N_A2CA(p', a2h_msg, r) → (**case** $p' = p$ **of** **T** → $t\ n$ ∥ **F** → N_TAU)
  ∥ N_CA2H(p', a2h_msg) → (**case** $p' = p$ **of** **T** → $t\ n$ ∥ **F** → N_TAU)
  ∥ N_H2CA(p', h2a_msg, r) → (**case** $p' = p$ **of** **T** → $t\ n$ ∥ **F** → N_TAU)
  ∥ N_CA2A(p', h2a_msg) → (**case** $p' = p$ **of** **T** → $t\ n$ ∥ **F** → N_TAU)
  ∥ N_D2H(p', d2h_msg) → (**case** $p' = p$ **of** **T** → $t\ n$ ∥ **F** → N_TAU)
  ∥ N_H2D(p', h2d_msg) → (**case** $p' = p$ **of** **T** → $t\ n$ ∥ **F** → N_TAU)
  ∥ N_H2CS(p', h2s_msg, r) → (**case** $p' = p$ **of** **T** → $t\ n$ ∥ **F** → N_TAU)
  ∥ N_CS2S(p', h2s_msg) → (**case** $p' = p$ **of** **T** → $t\ n$ ∥ **F** → N_TAU)

Rule version:

$\parallel$ N_S2CS(p', s2h_msg, r) $\rightarrow$ (**case** $p' = p$ **of** **T** $\rightarrow t\ n \parallel$ **F** $\rightarrow$ N_TAU)
$\parallel$ N_CS2H(p', s2h_msg) $\rightarrow$ (**case** $p' = p$ **of** **T** $\rightarrow t\ n \parallel$ **F** $\rightarrow$ N_TAU))

– **:**
(net_time $t(0 : \mathsf{num}) = (0 : \mathsf{time})) \wedge$
(net_time $t(SUC\ n) = ($
let $d =$ **case** $t\ n$ **of** N_DUR $x \rightarrow x \parallel \_ \rightarrow (0 : \mathsf{time})$ **in**
$d +$ net_time $t\ n))$

– **:**
(net_fabric $t\ 0 = \{\}) \wedge$
(net_fabric $t(SUC\ n) =$ **case** $(t\ n)$ **of** N_FABRIC $fs \rightarrow fs \parallel \_ \rightarrow$ net_fabric $t\ n)$

– **:**
(last_ping $t\ id\ 0 = *) \wedge$
(last_ping $t\ id(SUC\ m) =$ (**case** $t\ m$ **of**
N_A2CA(p, A2H_PING id', r) $\rightarrow$ (**case** $id' = id$ **of** **T** $\rightarrow \uparrow m \parallel$ **F** $\rightarrow$ last_ping $t\ id\ m$)
$\parallel \_ \rightarrow$ last_ping $t\ id\ m$))

– **:**
(* *rtt_estimate* is calculated per port (arbiter-host connection) and id by keeping a record of the last *PING* sent *)

(net_rtt_estimate $t\ 0 = *) \wedge$
(net_rtt_estimate $t(SUC\ m) =$ (**case** $(t\ m)$ **of**
N_H2CA(p', H2A_PONG id, r) $\rightarrow$
(**case** (last_ping $t\ id\ m$) **of** $\uparrow m' \rightarrow \uparrow$(net_time $t\ m -$ net_time $t\ m'$) $\parallel * \rightarrow *$)
$\parallel \_ \rightarrow$ net_rtt_estimate $t\ m$))

Rule version:

– **:**
(* mac for a given port, according to messages sent by arbiter and host events D2H_INVALIDATE_MY_MAC and H2D_WEDGED *)

(net_mac t 0 = ∗) ∧
(net_mac $t(SUC\ n)$ =
  (**let** *mac* = net_mac $t\ n$ **in**
    case $t\ n$ **of**
        N_DARK p' → ∗ (* already covered by H2D_WEDGED *)
      ‖ N_A2CA(p', a2h_msg, r) →
          (**case** a2h_msg **of** A2H_MAC_GRANT mac' → ↑ mac' ‖ *_x90* → *mac*)
      ‖ N_D2H(p', d2h_msg) →
          (**case** d2h_msg **of** D2H_INVALIDATE_MY_MAC → ∗ ‖ _ → *mac*)
      ‖ N_H2D(p', h2d_msg) → (**case** h2d_msg **of** H2D_WEDGED → ∗ ‖ *_x89* → *mac*)
      ‖ *_1* → *mac*))

– **pending output, presumably for a given port, per destination mac :**
(net_pending_output t *mac* 0 = [ ]) ∧
(net_pending_output t *mac*$(SUC\ n)$ =
  (**let** *oldq* = (net_pending_output t *mac* n) **in**
    case $(t\ n)$ **of**
        N_D2H(p, D2H_SEND(*mac*, *data*)) → *oldq* + +[*data*]
        (* notion of pending output if host sends an arbitrary message? *)
      ‖ N_H2CS(p, H2S_DATA *data*, r) → (**case** **hd** *oldq* = *data* **of** **T** → *TL oldq* ‖ **F** → *oldq*)
      ‖ N_DARK p → [ ] (* N_DARK clears pending output *)
      ‖ _ → *oldq*))

7.3 Projections from Net Traces

7.3.1 Summary

n_to_a
arbiter_trace
n_to_h
host_trace
n_to_s
switch_trace
arbiter_channel_trace

Rule version:

7.3.2 Rules

– :
n_to_a l = **case** l **of**
N_TAU → A_TAU
‖ N_DUR d → A_DUR d
‖ N_FABRIC fs → A_FABRIC fs
‖ N_DARK p_1 → A_DARK p_1
‖ N_A2CA(p_2, *a2h_msg2*, *t2*) → A_A2H(p_2, *a2h_msg2*)
‖ N_CA2H *_3* → A_TAU
‖ N_H2CA *_4* → A_TAU
‖ N_CA2A(*p5*, *h2a_msg5*) → A_H2A(*p5*, *h2a_msg5*)
‖ N_D2H *_6* → A_TAU
‖ N_H2D *_7* → A_TAU
‖ N_H2CS *_8* → A_TAU
‖ N_CS2S *_9* → A_TAU
‖ N_S2CS *_10* → A_TAU
‖ N_CS2H *_11* → A_TAU

– :
arbiter_trace(t : net_trace) = n_to_a o t

– :
n_to_h p l = **case** l **of**
N_TAU → H_TAU
‖ N_DUR *dur* → H_DUR *dur*
‖ N_FABRIC *_4* → H_TAU
‖ N_DARK *p8* → **if** p = *p8* **then** H_DARK **else** H_TAU
‖ N_A2CA *_1* → H_TAU
‖ N_CA2H(p_1, *m1*) → **if** $p = p_1$ **then** H_A2H *m1* **else** H_TAU
‖ N_H2CA(p_2, *m2*, *t2*) → **if** $p = p_2$ **then** H_H2A *m2* **else** H_TAU
‖ N_CA2A(p_3, *m3*) → H_TAU
‖ N_D2H(p_4, *m4*) → **if** $p = p_4$ **then** H_D2H *m4* **else** H_TAU
‖ N_H2D(*p5*, *m5*) → **if** p = *p5* **then** H_H2D *m5* **else** H_TAU
‖ N_H2CS(*p6*, *m6*, *t6*) → **if** p = *p6* **then** H_H2S *m6* **else** H_TAU
‖ N_CS2H(*p7*, *m7*) → **if** p = *p7* **then** H_S2H *m7* **else** H_TAU

Rule version:

$\|$ N_CS2S _2 → H_TAU
$\|$ N_S2CS _3 → H_TAU

− :
host_trace $p(t:$ net_trace$) = ($n_to_h $p)o\ t$

− :
n_to_s $l =$ **case** l **of**
N_TAU → S_TAU
$\|$ N_DUR d → S_DUR d
$\|$ N_FABRIC fs → S_FABRIC fs
$\|$ N_DARK p_1 → S_TAU
$\|$ N_A2CA _2 → S_TAU
$\|$ N_CA2H _3 → S_TAU
$\|$ N_H2CA _4 → S_TAU
$\|$ N_CA2A _5 → S_TAU
$\|$ N_D2H _6 → S_TAU
$\|$ N_H2D _7 → S_TAU
$\|$ N_H2CS _8 → S_TAU
$\|$ N_CS2S($p9$, h2s_msg) → S_H2S($p9$, h2s_msg)
$\|$ N_S2CS($p10$, s2h_msg, $t10$) → S_S2H($p10$, s2h_msg)
$\|$ N_CS2H _11 → S_TAU

− :
switch_trace(t : net_trace) = n_to_s $o\ t$

− :
arbiter_channel_trace $h(t:$ net_trace$) = \lambda n.$**case** $t\ n$ **of**
N_DUR dur → CA_DUR dur
$\|$ N_A2CA($h1, m1, t1$) → **if** $h = h1$ **then** CA_A2C($t1, m1$) **else** CA_TAU
$\|$ N_CA2A($h2, m2$) → **if** $h = h2$ **then** CA_C2A $m2$ **else** CA_TAU

Rule version:

$\|$ N_H2CA($h3, m3, t3$) $\rightarrow$ **if** $h = h3$ **then** CA_H2C($t3, m3$) **else** CA_TAU
$\|$ N_CA2H($h4, m4$) $\rightarrow$ **if** $h = h4$ **then** CA_C2H $m4$ **else** CA_TAU
$\|$ _ $\rightarrow$ CA_TAU

7.4 Specification

7.4.1 Summary

spec
example_trace
fromNum16
fromNum32

7.4.2 Rules

– :
spec *ports*(t : net_trace) =

arbiter_spec *ports*(arbiter_trace t) $\wedge$
switch_spec(switch_trace t) $\wedge$
($\forall p.$ **mem** p *ports* $\implies$
host_spec(host_trace p t) $\wedge$
arbiter_channel_spec(arbiter_channel_trace p t))

– :
example_trace = [N_DUR(10 $*$ USEC),
N_A2CA(**n2w** 0, A2H_PING(**n2w** 901), 84 $*$ NSEC),
N_DUR(10 $*$ USEC),
N_A2CA(**n2w** 0, A2H_PING(**n2w** 902), 84 $*$ NSEC),
N_DUR(85 $*$ NSEC),
N_CA2H(**n2w** 0, A2H_PING(**n2w** 902)),
N_H2CA(**n2w** 0, H2A_PONG(**n2w** 902), 84 $*$ NSEC),
N_H2CA(**n2w** 0, H2A_MAC_REQUEST, 84 $*$ NSEC),
N_DUR(85 $*$ NSEC),
N_CA2A(**n2w** 0, H2A_PONG(**n2w** 902)),

N_Dur(80 $*$ NSEC),
N_CA2A(**n2w** 0, H2A_MAC_REQUEST),
N_A2CA(**n2w** 0, A2H_MAC_GRANT(**n2w** 50), 84 $*$ NSEC)]

– **:**
fromNum16 = *word16* $*fromNum*

– **:**
fromNum32 = *word32* $*fromNum*