ENGINEERING TCP/IP WITH LOGIC

Hannes Mehnert*, *robur.io*, @h4nnes

based on work by Peter Sewell*, Michael Norrish^, Tom Ridge*
earlier contributors are Steve Bishop*, Matthew Fairbairn*, Michael Smith*, and Keith Wansbrough*
* while at *University of Cambridge*, ^ *NICTA*

Bob 2018, 23th February 2018

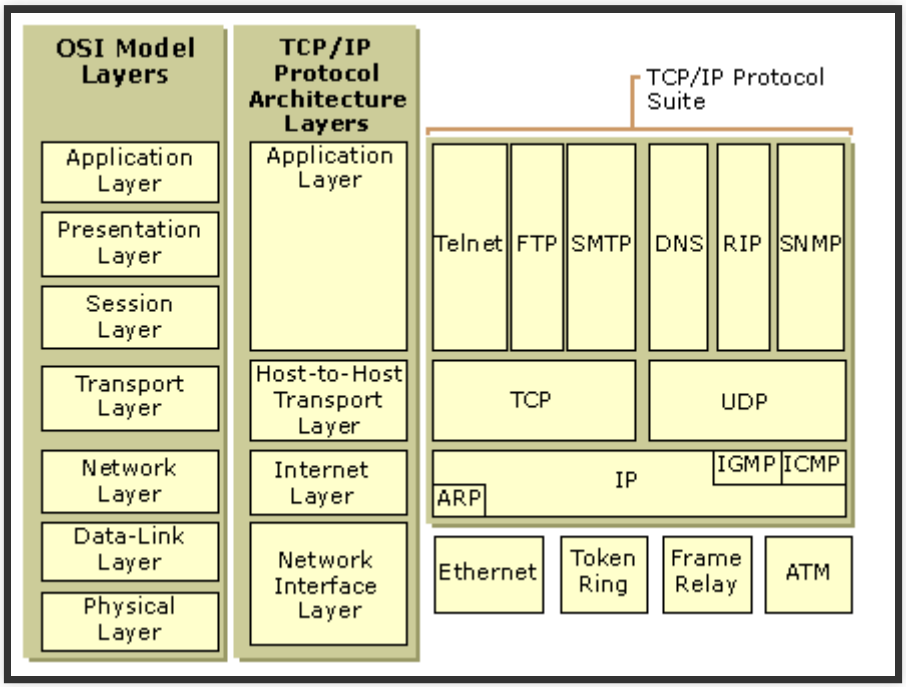
ABOUT ME

- Programmer (Turbo Pascal, C, Perl, Haskell, Dylan, Visual Basic, Python, C++, Java, Scala, Common Lisp, Coq, Idris, Emacs Lisp, JavaScript, Agda, OCaml)
- FreeBSD since 4.5 (2002), some Linux
- PhD in mechanised verification of the correctness of Java programs (using separation logic) at ITU Copenhagen
- PostDoc at University of Cambridge with Peter Sewell
- MirageOS (see Bob 2015 keynote) core team member
- Since 2018 non-profit robur.io to put MirageOS into production
- Looking for funding and contracts!

NETWORK PROGRAMMING

- Variety of protocols (IP, ICMP, UDP, TCP etc)
- Features: concurrency, packet loss, host failure, timeouts
- Sockets API
- Described in RFCs using informal prose and pseudocode
- Ambiguous and incomplete descriptions
- Protocols are hard to design and implement correctly
- Testing conformance against the standards is challenging
- Many obscure corner cases and failure semantics requires considerable expertise

TCP/IP


OSI Model Layers
Application Layer
Presentation Layer
Session Layer
Transport Layer
Network Layer
Data-Link Layer
Physical Layer
TCP/IP Protocol Architecture Layers
Application Layer
Host-to-Host Transport Layer
Internet Layer
Network Interface Layer
TCP/IP Protocol Suite
Telnet
FTP
SMTP
DNS
RIP
SNMP
TCP
UDP
IP
IGMP
ICMP
ARP
Ethernet
Token Ring
Frame Relay
ATM

WHAT IS TCP/IP?

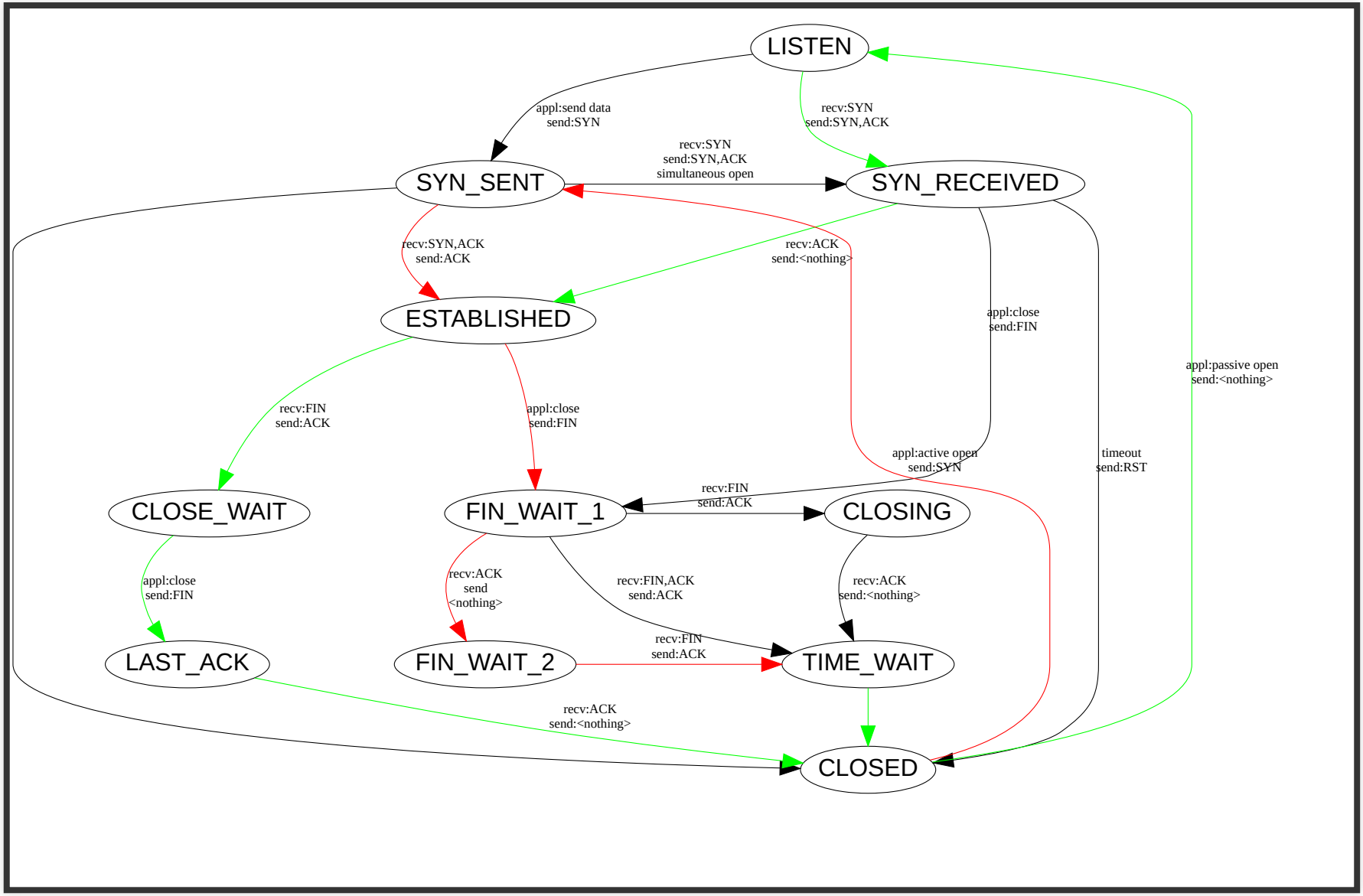
- Main protocol suite used for the Internet
- Internet Protocol (IP) RFC 760, Jan 80 - later RFC 791, Sep 81
  - connectionless, best-effort for packet-switched networks
- Internet Control Message Protocol (ICMP) RFC 792, Sep 81
  - error messages and organisational information
- User Datagram Protocol (UDP) RFC 768, Aug 80
  - connectionless, unreliable, integrity for messages
- Transmission Control Protocol (TCP) RFC 793, Sep 81
  - reliable, ordered, error-checked delivery of byte streams

- Reliable, ordered, error-checked delivery of byte streams

WHAT IS TCP?

- Sockets API: socket, bind, listen, accept, listen, connect, send, receive, shutdown, close
- Segments transmitted via Ethernet
- Connection setup and teardown
- Retransmission of lost segments
- Window size controls congestion
- Window is negotiated continuously

TCP STATE MACHINE



IMPLEMENTATION ISSUES

- "Mystery of hanging S3 downloads", "The many ways of handling TCP RST packets" https://www.snellman.net/blog
- Complex: described in dozens RFCs, complex state machine
- Extensible: TCP selective acknowledgement, TCP fast open, IPv6
- Security: everywhere
- Congestion control: loss vs delay, more bandwidth, shared medium (3G, wireless)
- Testing: huge test space (1200 bit TCP state + 190 bit per segment), try deployed stacks on the Internet

WHAT IS A BUG IN TCP/IP?

- May manifest as error in connection setup or teardown
- Or just introducing higher delay or less bandwidth (small windows)
- Interoperability with deployed stacks is crucial! Even if an RFC is violated
- Security: amplification, off-the-path attackers (blind window, LAND), DoS, common implementation pitfalls

FORMAL METHODS TO THE RESCUE

- Clear, accessible to a broad community and easy to modify
- Unambiguous, characterising exactly what behaviour is specified
- Sufficiently loose, characterising exactly what is not specified
- Directly usable as a basis for conformance testing
- Validated by getting used as a test oracle

HISTORY OF NETWORK SEMANTICS

- Started as research project at University of Cambridge in 2000 (FreeBSD 4.6, Linux 2.4, ~9kloc HOL and 17kloc comments)
- UDP Calculus: Rigorous Semantics for Real Networking (TACS 2001)
- Rigorous specification and conformance testing techniques for network protocols, as applied to TCP, UDP, and Sockets (SIGCOMM 2005)
- Engineering with Logic: HOL Specification and Symbolic-Evaluation Testing for TCP Implementations (POPL 2006)
- A rigorous approach to networking: TCP, from implementation to protocol to service (FM 2008)
- Engineering with Logic: Rigorous Test-Oracle Specification and Validation for TCP/IP and the Sockets API (JACM draft

and Validation for TCP/IP and the Sockets API (JACM draft Nov 2017)
- 11 person years of work, 386 pages specification
- Revival in 2016 with help from Michael Norrish

MODEL

- Developed in HOL4
- Label transition system
- Host state and label to new state
- Label: duration, segment send or received, state change
- Internal tau-transitions: arriving packet is not processed immediately, but put into queue
- Configuration parameters (sequence number, ..) via existentially quantified variables
- SML executable with backtracking to validate traces

EXAMPLE RULE: BIND_5

bind_5 **all: fast fail** **Fail with EINVAL: the socket is already bound to an address and does not support rebinding; or socket has been shutdown for writing on FreeBSD**

$$h\ \langle\!\langle ts := ts \oplus (tid \mapsto (\text{RUN})_d) \rangle\!\rangle$$

$$\xrightarrow{tid{\cdot}\text{bind}(fd, is_1, ps_1)}$$

$$h\ \langle\!\langle ts := ts \oplus (tid \mapsto (\text{RET(FAIL EINVAL)})_{\text{sched\_timer}}) \rangle\!\rangle$$

1. $fd \in \mathbf{dom}(h.fds) \wedge$
2. $fid = h.fds[fd] \wedge$
3. $h.files[fid] = \text{FILE}(\text{FT\_SOCKET}(sid), ff) \wedge$
4. $h.socks[sid] = sock \wedge$
5. $(sock.ps_1 \neq * \vee$
6. (bsd_arch $h.arch \wedge sock.pr = \text{TCP\_PROTO}(tcp\_sock) \wedge$
7. $(sock.cantsndmore \vee$
8. $tcp\_sock.cb.bsd\_cantconnect)))$

Description From thread *tid*, which is in the RUN state, a bind(fd, is_1, ps_1) call is made where *fd* refers to a socket *sock*. The socket already has a local port binding: $sock.ps_1 \neq *$, and rebinding is not supported.
A $tid{\cdot}\text{bind}(fd, is_1, ps_1)$ transition is made, leaving the thread state RET(FAIL EINVAL).

Variations

FreeBSD	This rule also applies if *fd* refers to a TCP socket which is either shut down for writing or has its *bsd_cantconnect* flag set.

WHAT IS A TEST?

- Tthee autotest implemented in OCaml, ad-hoc, large rule coverage
- Now using packetdrill (2013), which does expect-based testing

RCV-SYN-WITHOUT-DATA-CLOSED-IPV4.PKT

```
 0.00 socket(..., SOCK_STREAM, IPPROTO_TCP) = 3
+0.00 setsockopt(3, SOL_SOCKET, SO_DEBUG, [1], 4) = 0
+0.00 bind(3, ..., ...) = 0
+0.00 getsockopt(3, SOL_SOCKET, SO_RCVBUF, [65536], [4]) = 0
// Now it is in the CLOSED state.
+0.10 < S  17:17(0) win 32767
+0.00 > R. 0:0(0) ack 18 win 0
+0.00 close(3) = 0
```

WHAT IS A TRACE?

- Series of POSIX system calls or TCP fragments
- Possible injection of TCP fragments from remote host
- DTrace instrumenatation outputs a trace:
  - Duration in ms
  - Socket calls
  - TCP segments on wire
  - TCP control block structure

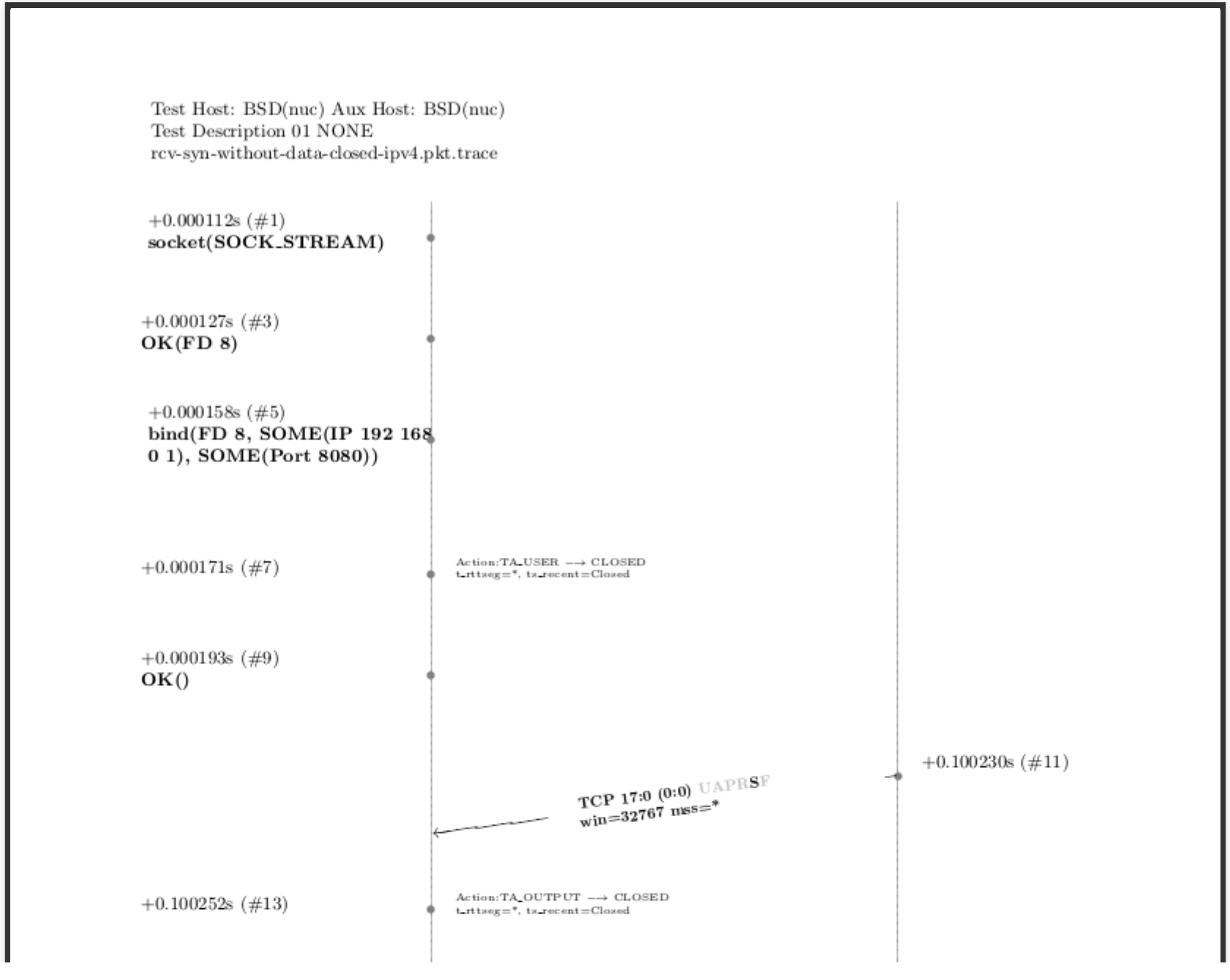
HOLTCP.D (700 LINES)

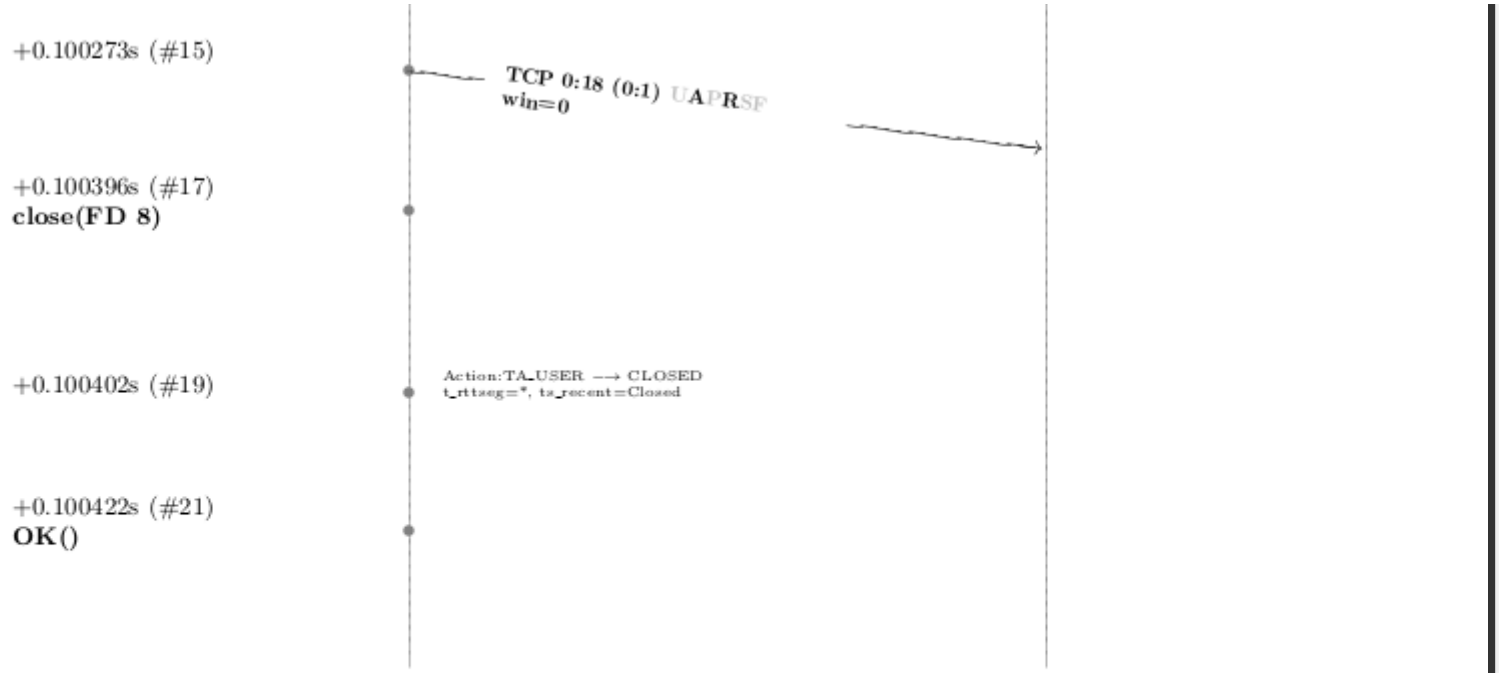
```
#define act execname == "packetdrill" && self->started == 1

int ts;
int step;

#define dur()                       \
  this->dur = timestamp - ts ;    \
  this->us  = this->dur / 1000;   \
  this->s   = this->us / 1000000; \
  this->us  = this->us % 1000000; \
  ts  = timestamp ;               \
  printf("(* Merge Index: %d *)\n", step); \
  step = step + 1 ;                        \
  printf("Lh_epsilon(duration %d %06d);\n", this->s, this->us); \
  printf("(* Merge Index: %d *)\n", step); \
  step = step + 1 ;

syscall::socket:entry
```

RCV-SYN-WITHOUT-DATA-CLOSED-IPV4.PKT.TRACE

```
(* HOST *)
initial_host (IP 192 168 0 1) (TID 19494) (FreeBSD_4_6_RELEASE) F [( NONE
(* TSOH *)
(* BEGIN *)
(* BASETIME *)
abstime 1493299013 650354405
(* EMITESAB *)
(* Merge Index: 0 *)
Lh_epsilon(duration 0 000112);
(* Merge Index: 1 *)
Lh_call(TID 19494, socket(SOCK_STREAM));
(* Merge Index: 2 *)
Lh_epsilon(duration 0 000015);
(* Merge Index: 3 *)
Lh_return(TID 19494, OK(FD 8));
(* Merge Index: 4 *)
Lh_epsilon(duration 0 000031);
(* Merge Index: 5 *)
```

RCV-SYN-WITHOUT-DATA-CLOSED-IPV4.PKT.TRACE.PDF




+0.100273s (#15)
TCP 0:18 (0:1) UAPRSF
win=0
+0.100396s (#17)
close(FD 8)
+0.100402s (#19)
Action:TA_USER → CLOSED
t_rttseg=*, ts_recent=Closed
+0.100422s (#21)
OK()

HOL Trace: rcv-syn-without-data-closed-ipv4.pkt.trace

[Show/hide variables and constraints.]

==Working on trace file rcv-syn-without-data-closed-ipv4.pkt.trace [plain] [ps]
==Date: 2017-10-16 T 17:34:34 Z (Mon)

```
(* Test Host: BSD(nuc) Aux Host: BSD(nuc) *)
(* Test Description 01 NONE *)
```

==Simplifying host and labels from disk ... done

```
==Step 0 at <2017-10-16 T 17:34:35 Z (Mon)> 1508175276:
 attempting time passage with duration 7 / 62500
CPU time elapsed : 3.172 seconds(unwind: 0.000)
```

==Successful transition of epsilon_1

```
==Step 1 at <2017-10-16 T 17:34:38 Z (Mon)> 1508175278:
 Lh_call (TID 19494,socket SOCK_STREAM)
initial: 0.010s (#poss: 6)
==Attempting socket_1 -- pre_host -- post_host -- phase2 -- ctxtclean
CPU time elapsed : 0.519 seconds (unwind: 0.000)
Label                      #calls     real     user   system       gc
```

==Successful transition of socket_1

```
==Step 2 at <2017-10-16 T 17:34:39 Z (Mon)> 1508175279:
 attempting time passage with duration 3 / 200000
CPU time elapsed : 4.227 seconds(unwind: 0.000)
```

==Successful transition of epsilon_1

```
==Step 3 at <2017-10-16 T 17:34:43 Z (Mon)> 1508175284:
 Lh_return (TID 19494,TL_err (OK (TL_fd (FD 8))))
initial: 0.010s (#poss: 2)
==Attempting return_1 -- pre_host -- post_host -- phase2 -- ctxtclean
CPU time elapsed : 0.292 seconds (unwind: 0.000)
Label                      #calls     real     user   system       gc
```

==Successful transition of return_1

```
==Step 4 at <2017-10-16 T 17:34:43 Z (Mon)> 1508175284:
```

ONGOING WORK

- More tests
- Validating more stacks
- More features in model (congestion control, SACK)
- TCP/IP implementation in OCaml
- Test coverage: model and stacks

RESULTS

- Roughly 3 dozen anomalies in FreeBSD implementation (2005)
  - see Section 9 http://www.cl.cam.ac.uk/~pes20/Netsem/tr.pdf
- Slowly re-checking and fixing upstream
- Revival of HOL model lead to various rule fixes
- Enhanced state machine diagram

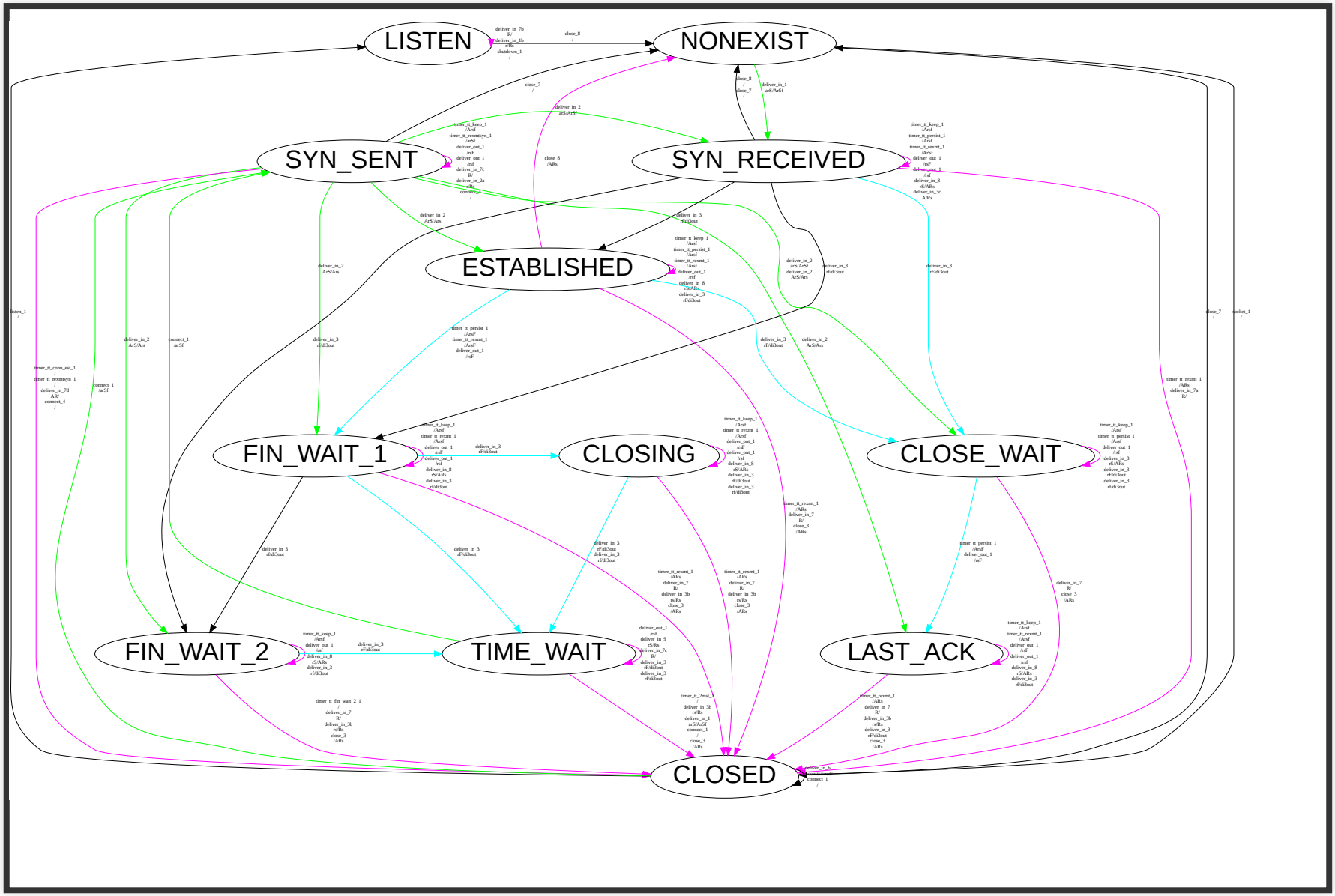
A8: RESPONSE TO SYN, FIN SEGMENTS.

In the SYN SENT state, it is possible to receive a FIN along with the required SYN. In the case of a SYN, FIN , ACK being received, BSD will ACK both the SYN and the FIN , moving into CLOSE_WAIT , which is perfectly reasonable behaviour. If, however, a SYN, FIN segment is received (a simultaneous open), BSD incorrectly bypasses the SYN_RECEIVED state and moves directly into CLOSE WAIT without waiting for our SYN to be acknowledged. See deliver_in_2, deliver_in_3.

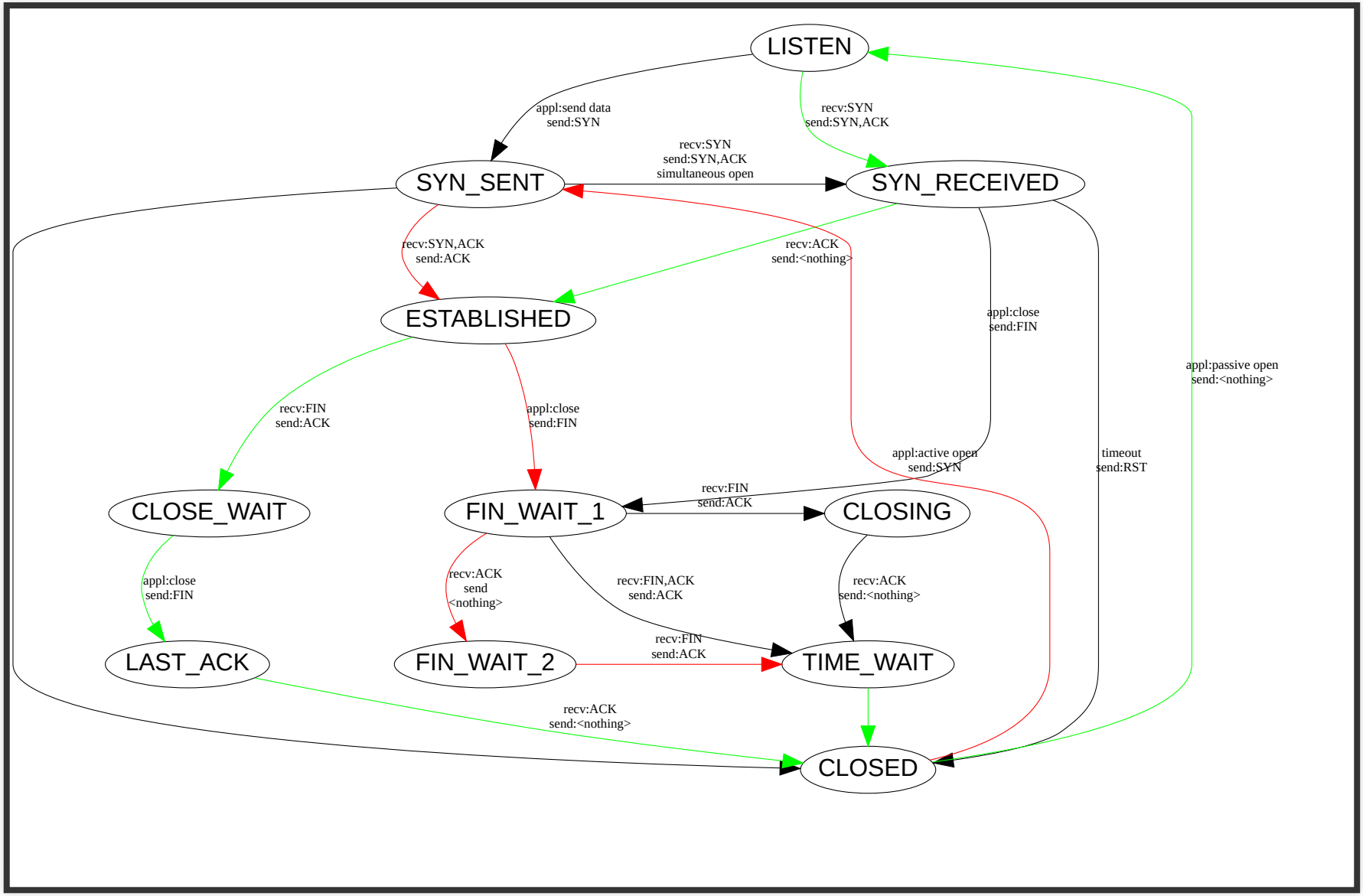
A10: WINDOW OF NO RTT CACHE UPDATES

After 2^32 packets, there is a 16 packet window during which time, if the TCP connection is closed, the RTT values will not be cached in the routing table entry. This is because of an overflow/wraparound problem in `t_rttupdated`. Impact: Very rarely, after the closure of 1 in 2^28 connections, the round-trip time estimator will be less accurate that it might be, adversely affecting the performance of a subsequent connection.

TCP STATE MACHINE



STEVENS STATE MACHINE



CONCLUSION

- Formalising real-world protocols is possible, but lots of work
- Artifacts include a readable specification with typesetted transition rules
- Discovered various subtle bugs
- Coverage of test suite
- Interested in testing your TCP stack: get instrumentation ready!
- JACM draft http://www.cl.cam.ac.uk/~pes20/Netsem/paper3.pdf

Advertisement:

- MirageOS retreat 7th-18th March in Marrakesh http://retreat.mirage.io
- https://mirage.io
- http://robur.io blog https://hannes.nqsb.io