Validating Formal Descriptions of TCP/IP

Experimental Formal Semantics

Michael Norrish
Michael.Norrish@nicta.com.au

NICTA

September 2007

Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

Background

- This is joint work with Peter Sewell, Keith Wansbrough, Tom Ridge, Steve Bishop, Andrei Serjantov, Michael Fairbairn and Michael Smith (all at University of Cambridge).
- Project began in 2001(?), with work on UDP by Sewell, Serjantov and Wansbrough.
- UDP is a simple protocol, but work on a detailed semantics became overwhelming for pen-and-paper techniques.
- I joined the project to see if mechanical support might be helpful (hence the choice of HOL4...)

Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

Work on UDP

- The mechanised semantics for UDP was not **too** large: I proved a simple safety property in HOL.
- By hand, we also proved
  - a timing property
  - correctness properties for a heart-beat program built using the sockets interface to UDP
- Next step was clear: move to TCP.

1. Introduction
   - Beginning a TCP Specification
2. The Segment Level Specification
   - Use of HOL
3. Specification Validation
   - As a Theorem Proving Problem
4. The High Level Specification
5. Conclusion

Motivation

TCP is critical Internet infrastructure.

Time spent specifying it is time well-spent:

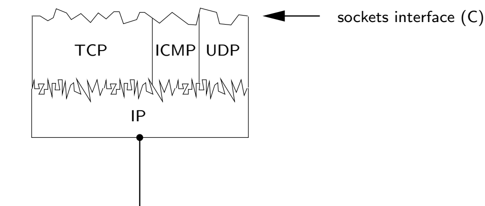
- Users of the API know what to expect
- Future implementors have a better idea of what they have to do
- It can be studied mathematically, not just empirically.

...implementing TCP correctly is very difficult
—Vern Paxson (SIGCOMM'97)

To which we add:

Using TCP and the Sockets API correctly is also difficult

Networking: The TCP/IP Protocols



IP (Internet Protocol): unreliable asynchronous small messages, delivered to IP addresses such as 128.34.1.14.

UDP (User Datagram Protocol): as above, but delivered to IP address/Port pairs.

TCP (Transmission Control Protocol): duplex streams, with retransmission, flow control, congestion control, etc. Messages between IP/Port pairs.

Networking: The Sockets API

The Sockets API gives programmers levers with which to control of the various internet protocols:

- Expression in C is ghastly (`ntohs`, `struct inaddr *...`)
- Even stripped of C-isms, there are a plethora of confusing entry-points (bind, listen, accept, connect, close, socket, dupfd. . .)
- Specified in POSIX and the various implementations (including Windows)

Post Hoc Specification

We cannot tell the world's TCP and OS implementors what to do.

Our specification must reflect not only the existing "specifications":

- RFCs, POSIX, ...

But also current practice, as enshrined in representative implementations: Windows, BSD and Linux.

Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

Writing a Huge Specification

This project was a test of theoretical machinery:

- An operational semantics with almost 200 reduction rules (no recursion though)
- Handling of non-determinism
  - timing: modelling the interleaving of asynchronous and synchronous system calls, as well as packet arrival and dispatch
  - choices of values: under-specified behaviours cause semantic states to take on constrained values for attributes
- Quantified time: TCP is full of timed quantities, and counters representing the passage of time.

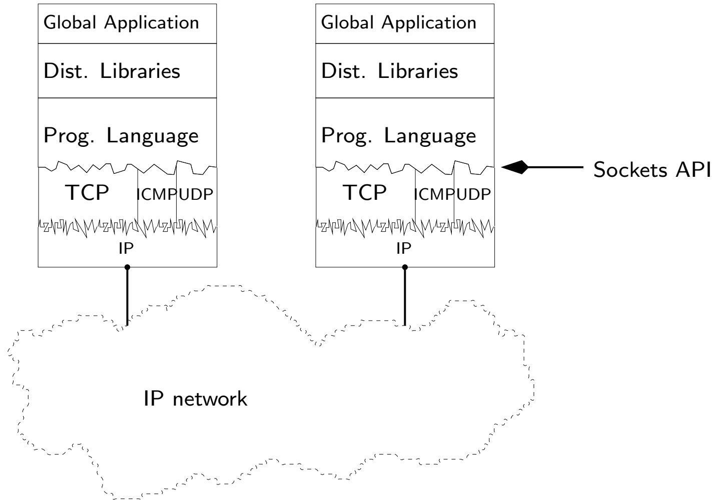
Writing a Huge Specification

In addition to "inherent complexities", we contended with 20–30 years of haphazard code evolution, resulting in

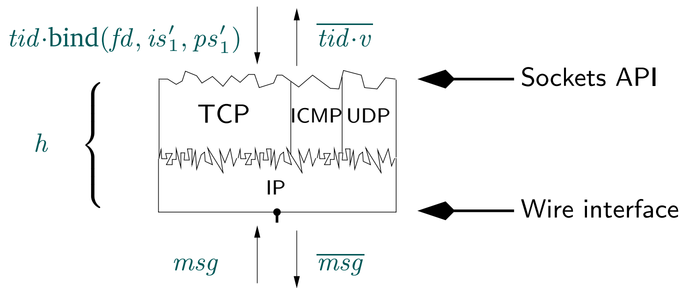
- the warped sockets API;
- specifications allowing a great deal of implementation latitude;
- numerous ugly corner cases

Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

Specification: Where to Cut?



Specification: Where to Cut?



The specification describes the evolution of **hosts**, which are involved in six sorts of behaviours:

system calls **in**; syscall returns **out**; messages **in**; messages **out**;

time elapsing; internal/unobservable state-changes.

Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

Specification Language

The specification is written in higher-order logic.

Expressive:

- Supports natural, mathematical idiom
- Rich types (lists, sets, finite-maps, $\mathbb{N}$, $\mathbb{R}$)
  - $+$ user defined types (records, algebraic types)
- Captures non-determinism and under-specification easily

Clear:

- Has well-defined semantics
- Easy to write (non-expert CS people picked it up in a week or so)

Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

The Segment Level Specification

Our segment level specification (the "low level spec") describes

- the "on the wire" behaviour of hosts: the packets they emit
- what programmers using the sockets API can expect
- the internal (hidden) state of hosts supporting the above

Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

The Segment Level Specification

Our segment level specification (the "low level spec") describes

- the "on the wire" behaviour of hosts: the packets they emit
- what programmers using the sockets API can expect
- the internal (hidden) state of hosts supporting the above

We have

- validated the specification against real implementations
- found bugs in the implementations

Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

Important Types: Segments

$$
\begin{array}{ll}
\text{tcpSegment} = \langle & \\
\quad \mathit{is1} : \text{ip option}; & \textit{(* source IP *)} \\
\quad \mathit{is2} : \text{ip option}; & \textit{(* destination IP *)} \\
\quad \mathit{ps1} : \text{port option}; & \textit{(* source port *)} \\
\quad \mathit{ps2} : \text{port option}; & \textit{(* destination port *)} \\
\quad \mathit{seq} : \text{tcp_seq_local}; & \textit{(* sequence number *)} \\
\quad \mathit{URG}, \mathit{ACK} : \text{bool}; & \\
\quad \mathit{PSH}, \mathit{RST} : \text{bool}; & \\
\quad \mathit{SYN}, \mathit{FIN} : \text{bool}; & \\
\quad \mathit{win} : \text{word16}; & \textit{(* window size (unsigned) *)} \\
\quad \mathit{mss} : \text{word16 option}; & \textit{(* maximum segment size *)} \\
\quad \ldots & \\
\rangle &
\end{array}
$$

Important Types: Hosts

$$
\begin{array}{ll}
\mathsf{host} = \langle & \\
\quad arch : \mathsf{arch}; & \textit{(* architecture *)} \\
\quad privs : \mathsf{bool}; & \\
\quad socks : \mathsf{sid} \mapsto \mathsf{socket}; & \\
\quad ts : \mathsf{tid} \mapsto \mathsf{hostThreadState\ timed}; & \textit{(* threads *)} \\
\quad listen : \mathsf{sid\ list}; & \\
\quad iq : \mathsf{msg\ list\ timed}; & \textit{(* messages in *)} \\
\quad oq : \mathsf{msg\ list\ timed}; & \textit{(* messages out *)} \\
\quad ticks : \mathsf{ticker} & \\
\quad \ldots & \\
\rangle &
\end{array}
$$

Specification: A Sample Rule

Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

bind_5 **rp_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**

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

$$\xrightarrow{tid \cdot \mathrm{bind}(fd, is1, ps1)} \quad h \langle\!\langle ts := ts \oplus (tid \mapsto (\mathsf{Ret}(\mathsf{FAIL\ EINVAL}))_{\mathrm{sched_timer}}) \rangle\!\rangle$$

$$\begin{aligned}
& fd \in \mathbf{dom}(h.fds) \wedge fid = h.fds[fd] \wedge \\
& h.files[fid] = \mathsf{File}(\mathsf{FT_Socket}(sid), ff) \wedge \\
& h.socks[sid] = sock \wedge \\
& (sock.ps1 \neq * \vee \\
& (\mathrm{bsd_arch}\ h.arch \wedge sock.pr = \mathsf{TCP_PROTO}(tcp_sock) \wedge \ldots))
\end{aligned}$$

Specification: A More Complicated Rule

$deliver_in_1$ **tcp: network nonurgent**

Passive open: receive SYN, send SYN,ACK

$h\ \langle\!|socks := socks \oplus [(sid, sock)];$ (* listening socket *)
$iq := iq;$ (* input queue *)
$oq := oq|\!\rangle$ (* output queue *)

$\xrightarrow{\tau}$

$h\ \langle\!|socks := socks \oplus$
(* listening socket *)
$[(sid, \mathrm{SOCK}(\uparrow fid, sf, is_1, \uparrow p_1, is_2, ps_2, es, csm, crm,$
$\mathrm{TCP_Sock}(\mathrm{LISTEN}, cb, \uparrow lis', [], *, [], *, \mathrm{NO_OOB})));$
(* new connecting socket *)
$(sid', \mathrm{SOCK}(*, sf', \uparrow i_1, \uparrow p_1, \uparrow i_2, \uparrow p_2, *, csm, crm,$
$\mathrm{TCP_Sock}(\mathrm{SYN_RCVD}, cb'', *, [], *, [], *, \mathrm{NO_OOB})))];$
$iq := iq';$
$oq := oq'|\!\rangle$

(* check first segment matches desired pattern; unpack fields *)
$\mathrm{dequeue_iq}(iq, iq', \uparrow(\mathrm{TCP}\ seg)) \wedge$
$(\exists win_\ ws_\ mss_\ PSH\ URG\ FIN\ urp\ data\ ack.$
$seg =$
$\langle\!|\ is_1 := \uparrow i_2; is_2 := \uparrow i_1; ps_1 := \uparrow p_2; ps_2 := \uparrow p_1;$
$seq := tcp_seq_flip_sense(seq : \mathrm{tcp_seq_foreign});$
$ack := tcp_seq_flip_sense(ack : \mathrm{tcp_seq_local});$
$URG := URG; ACK := \mathbf{F}; PSH := PSH;$
$RST := \mathbf{F}; SYN := \mathbf{T}; FIN := FIN;$
$win := win_; ws := ws_; urp := urp; mss := mss_; ts := ts;$
$data := data$
$|\!\rangle \wedge$
$\mathbf{w2n}\ win_ = win \wedge$ (* type-cast from word to integer *)
$\mathbf{option_map}\ \mathbf{ord}\ ws_ = ws \wedge$
$\mathbf{option_map}\ \mathbf{w2n}\ mss_ = mss) \wedge$

(* IP addresses are valid for one of our interfaces *)
$i_1 \in \mathrm{local_ips}\ h.ifds \wedge$
$\neg(\mathrm{is_broadormulticast}\ h.ifds\ i_1) \wedge \neg(\mathrm{is_broadormulticast}\ h.ifds\ i_2) \wedge$

(* sockets distinct; segment matches this socket; unpack fields of socket *)
$sid \notin (\mathbf{dom}(socks)) \wedge sid' \notin (\mathbf{dom}(socks)) \wedge sid \neq sid' \wedge$
$\mathrm{tcp_socket_best_match}\ socks(sid, sock) seg\ h.arch \wedge$
$sock = \mathrm{SOCK}(\uparrow fid, sf, is_1, \uparrow p_1, is_2, ps_2, es, csm, crm,$
$\mathrm{TCP_Sock}(\mathrm{LISTEN}, cb, \uparrow lis, [], *, [], *, \mathrm{NO_OOB})) \wedge$

(* socket is correctly specified (note BSD listen bug) *)
$((is_2 = * \wedge ps_2 = *) \vee$
$(\mathrm{bsd_arch}\ h.arch \wedge is_2 = \uparrow i_2 \wedge ps_2 = \uparrow p_2)) \wedge$
$(\mathbf{case}\ is_1\ \mathbf{of}\ \uparrow i1' \rightarrow i1' = i_1 \parallel * \rightarrow \mathbf{T}) \wedge$
$\neg(i_1 = i_2 \wedge p_1 = p_2) \wedge$

(* (elided: special handling for TIME_WAIT state, 10 lines) *)

(* place new socket on listen queue *)
$\mathrm{accept_incoming_q0}\ lis\ \mathbf{T} \wedge$
(* (elided: if drop_from_q0, drop a random socket yielding q0') *)
$lis' = lis\ \langle\!|\ q_0 := sid' :: q_0'|\!\rangle \wedge$

(* choose MSS and whether to advertise it or not *)
$advmss \in \{n \mid n \geq 1 \wedge n \leq (65535 - 40)\} \wedge$
$advmss' \in \{*; \uparrow advmss\} \wedge$

(* choose whether this host wants timestamping; negotiate with other side *)
$tf_rcvd_tstmp' = \mathbf{is_some}\ ts \wedge$
$(\mathbf{choose}\ want_tstmp :: \{\mathbf{F}; \mathbf{T}\}.$
$tf_doing_tstmp' = (tf_rcvd_tstmp' \wedge want_tstmp)) \wedge$

(* calculate buffer size and related parameters *)
$(rcvbufsize', sndbufsize', t_maxseg', snd_cwnd') =$
$\mathrm{calculate_buf_sizes}\ advmss\ mss * (\mathrm{is_localnet}\ h.ifds\ i_2)$
$(sf.n(\mathrm{SO_RCVBUF}))(sf.n(\mathrm{SO_SNDBUF}))$
$tf_doing_tstmp'\ h.arch \wedge$
$sf' = sf\ \langle\!|\ n := \mathrm{funupd_list}\ sf.n[(\mathrm{SO_RCVBUF}, rcvbufsize');$
$(\mathrm{SO_SNDBUF}, sndbufsize')]|\!\rangle \wedge$

(* choose whether this host wants window scaling; negotiate with other side *)
$req_ws \in \{\mathbf{F}; \mathbf{T}\} \wedge$
$tf_doing_ws' = (req_ws \wedge \mathbf{is_some}\ ws) \wedge$
$(\mathbf{if}\ tf_doing_ws'\ \mathbf{then}$
$rcv_scale' \in \{n \mid n \geq 0 \wedge n \leq \mathrm{TCP_MAXWINSCALE}\} \wedge$
$snd_scale' = \mathbf{option_case}\ 0\ \mathbf{I}\ ws$
$\mathbf{else}$
$rcv_scale' = 0 \wedge snd_scale' = 0) \wedge$

(* choose initial window *)
$rcv_window \in \{n \mid n \geq 0 \wedge$
$n \leq \mathrm{TCP_MAXWIN} \wedge$
$n \leq sf.n(\mathrm{SO_RCVBUF})\} \wedge$

(* record that this segment is being timed *)
$(\mathbf{let}\ t_rttseg' = \uparrow(\mathrm{ticks_of}\ h.ticks, cb.snd_nxt)\ \mathbf{in}$

(* choose initial sequence number *)
$iss \in \{n \mid \mathbf{T}\} \wedge$

(* acknowledge the incoming SYN *)
$\mathbf{let}\ ack' = seq + 1\ \mathbf{in}$

(* update TCP control block parameters *)
$cb' =$
$cb\ \langle\!|\ tt_keep := \uparrow((())_{\mathrm{slow_timer\ TCPTV_KEEP_IDLE}});$
$tt_rexmt := \mathrm{start_tt_rexmt}\ h.arch\ 0\ \mathbf{F}\ cb.t_rttinf;$
$iss := iss; irs := seq;$
$rcv_wnd := rcv_window; tf_rxwin0sent := (rcv_window = 0);$
$rcv_adv := ack' + rcv_window; rcv_nxt := ack';$
$snd_una := iss; snd_max := iss + 1; snd_nxt := iss + 1;$
$snd_cwnd := snd_cwnd'; rcv_up := seq + 1;$
$t_maxseg := t_maxseg'; t_{advmss} := advmss';$
$rcv_scale := rcv_scale'; snd_scale := snd_scale';$
$tf_doing_ws := tf_doing_ws';$
$ts_recent := \mathbf{case}\ ts\ \mathbf{of}$
$* \rightarrow cb.ts_recent \parallel$
$\uparrow(ts_val, ts_ecr) \rightarrow (ts_val)^{\mathrm{TIMEWINDOW}}_{\mathrm{kern_timer\ dtsinval}};$
$last_ack_sent := ack';$
$t_rttseg := t_rttseg';$
$tf_req_tstmp := tf_doing_tstmp';$
$tf_doing_tstmp := tf_doing_tstmp'$
$|\!\rangle) \wedge$

(* generate outgoing segment *)
$\mathbf{choose}\ seg' :: \mathrm{make_syn_ack_segment}\ cb'$
$(i_1, i_2, p_1, p_2)(\mathrm{ticks_of}\ h.ticks).$

(* attempt to enqueue segment; roll back specified fields on failure *)
$\mathrm{enqueue_or_fail}\ \mathbf{T}\ h.arch\ h.rttab\ h.ifds[\mathrm{TCP}\ seg']oq$
$(cb$
$\langle\!|\ snd_nxt := iss;$
$snd_max := iss;$
$t_maxseg := t_maxseg';$
$last_ack_sent := \mathrm{tcp_seq_foreign}\ 0w;$
$rcv_adv := \mathrm{tcp_seq_foreign}\ 0w$
$|\!\rangle)cb'(cb'', oq')$

Specification Tools

Somewhat more than just “vanilla” HOL4.

I wrote very little of the specification.

Specifiers who were new to HOL wrote the specification. They:

- were happy with straight HOL syntax (didn’t want to develop their own custom language and parse this to HOL);
- developed a type-setting tool that was fed stereotyped HOL script (would have surely been happy with Isabelle’s facilities)
- blithely wrote in a style that was elegant, mathematical and hard to execute

Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

Stress-Testing HOL4

Experience was a great opportunity to test and improve the implementation:

- Error-reporting was improved.
  E.g., Keith *"welcome-to-the-1980s"* Wansbrough implemented line-number reporting on parse errors.

- Scaling problems were addressed:
  - 150KB string literals
  - records with 40+ fields became possible
    (avoid nasty $O(n^2)$ behaviours)
  - de Bruijn indices for term abstractions dropped
    (and other kernel experiments)

- Spent a lot of time with the simplifier

How Do We Know It's Right?

It would be surprising if a logical entity of this size did not contain errors.

The specification is not really executable

- we did toy with the idea of trying to execute it in concert with specially slowed-down real implementations

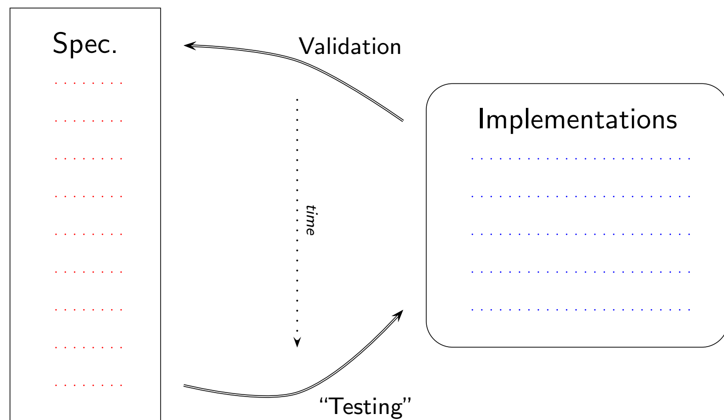
But we can **test** implementations against it.

Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

Specification Validation and Testing

Initially, a **post hoc** specification must be held to be in error if it doesn't capture what the implementations do.

Once refined, implementations can be validated against **it**.



Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

Validation/Testing Problem

Observe two machines communicating.

Record trace of events (with time-stamps).

Check that trace of real-world events is compatible with specification.

Validation/Testing Problem

Observe two machines communicating.

Record trace of events (with time-stamps).

Check that trace of real-world events is compatible with specification.

Disagreement indicates

- specification is wrong (initially)
- a possible bug (later)

Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

This is “Just” Testing

Testing programs with simple specifications is easy.

For example, simple observation + calculation can determine if a program for calculating square roots has behaved correctly.

Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

This is "Just" Testing

Testing programs with simple specifications is easy.

For example, simple observation + calculation can determine if a program for calculating square roots has behaved correctly.

With many tools (software model-checking, ESC/Java etc) the programs analysed have implicit specification:

> *Does not do anything bad.*

Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

This is “Just” Testing

Testing programs with simple specifications is easy.

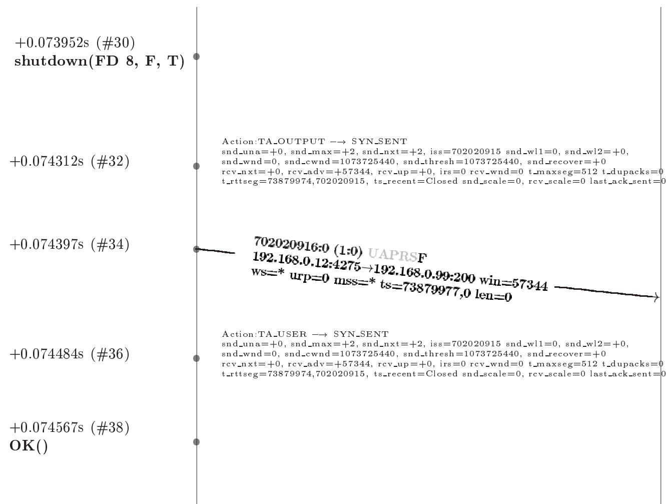
For example, simple observation + calculation can determine if a program for calculating square roots has behaved correctly.

With many tools (software model-checking, ESC/Java etc) the programs analysed have implicit specification:

Does not do anything bad.

With a specification as complicated as TCP’s, testing is a hard problem.

Sample Trace Fragment



Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

Outline

1 Introduction
   Beginning a TCP Specification

2 The Segment Level Specification
   Use of HOL

3 Specification Validation
   As a Theorem Proving Problem

4 The High Level Specification

5 Conclusion

The Trace Checking Problem

Problem: Given an initial host h_0 and a captured trace $\ell_1, \ldots \ell_n$, determine if

$$\exists h_1 \ldots h_n.\ h_0 \xrightarrow{\tau}{}^{*} \xrightarrow{\ell_1} h_1 \xrightarrow{\tau}{}^{*} \xrightarrow{\ell_2} h_2 \ldots \xrightarrow{\tau}{}^{*} \xrightarrow{\ell_n} h_n$$

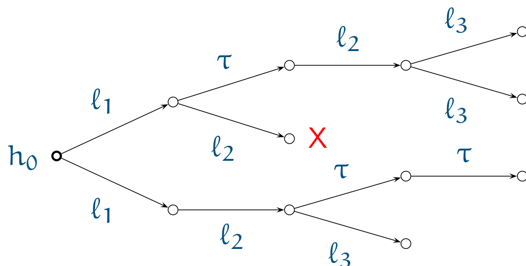
The reduction relation, defining $\xrightarrow{\ell}$ (including $\ell = \tau$), is given by the people writing the specification, and is subject to change as validation proceeds.

This is "existential model checking": can a path of the specified type be exhibited by the given model?

Why Validation is Hard

Specification is not (can not be) deterministic.

Though we know the initial state, and the observed behaviours, there may be multiple paths possible



The X indicates a state from which no τ or ℓ_3 transition is possible.

Backtracking is required from such a state.

More Non-determinism: Accumulating Constraint Sets

In addition to branching ("competition" between many reduction rules), non-determinism arises within a single rule.

E.g.:

$$\frac{0 \leq \mathrm{i} < \mathrm{h}.\mathit{fld}_2}{\mathrm{h} \xrightarrow{\ell} \mathrm{h}\langle \mathit{fld}_1 := \mathrm{i} \rangle}$$

Every transition becomes associated with a set of constraints that have to be true for that transition to have occurred.

States come to have symbolic values.

Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

Accumulating Constraint Sets

The result is the proof of a sequence of theorems :

$$\Gamma_0 \vdash h_0 \xrightarrow{\ell_0} h_1$$
$$\Gamma_0 \cup \Gamma_1 \vdash h_1 \xrightarrow{\ell_1} h_2$$
$$\Gamma_0 \cup \Gamma_1 \cup \Gamma_2 \vdash h_2 \xrightarrow{\ell_2} h_3$$
$$\ldots$$

with each Γ_i being the set of constraints associated with the ith transition.

The growing constraint sets have to be checked for satisfiability at each step.

(This is computationally horrific.)

Accumulating Constraint Sets

The result is the proof of a sequence of theorems :

$$\Gamma_0 \vdash h_0 \xrightarrow{\ell_0} h_1$$
$$\Gamma_0 \cup \Gamma_1 \vdash h_1 \xrightarrow{\ell_1} h_2$$
$$\Gamma_0 \cup \Gamma_1 \cup \Gamma_2 \vdash h_2 \xrightarrow{\ell_2} h_3$$
$$\ldots$$

with each Γ_i being the set of constraints associated with the ith transition.

The growing constraint sets have to be checked for satisfiability at each step.

At end of process can "ground" complete trace by finding values satisfying all constraints.

Typical Constraints

```
pending (cb'_0_t_rttvar =
           MAX (1 / 1600)
               ((1 + (abs delta - 1 / 40) * 10) / 40))
pending (cb'_0_t_srtt = MAX (1 / 3200)
                            ((2 + delta * 5) / 40))
pending (delta = r / 100 - 1 / 100 - 1 / 20)
pending (r = real_of_int (ticks_of ticks'22 -
                            SEQ32 Tstamp 152410675w))
MIN 64 (MAX 1 (cb'_0_t_srtt + 4 * cb'_0_t_rttvar)) = 1
r <= 6
3 <= 160 * cb'_0_t_rttvar
19 <= 400 * cb'_0_t_srtt
100 <= 25 * r
800 * cb'_0_t_rttvar <= 19
20 * cb'_0_t_srtt <= 1
```


Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

Typical Constraints

```
pending (cb'_0_t_rttvar =
           MAX (1 / 1600)
               ((1 + (abs delta - 1 / 40) * 10) / 40))
pending (cb'_0_t_srtt = MAX (1 / 3200)
                            ((2 + delta * 5) / 40))
pending (delta = r / 100 - 1 / 100 - 1 / 20)
pending (r = real_of_int (ticks_of ticks'22 -
                            SEQ32 Tstamp 152410675w))
MIN 64 (MAX 1 (cb'_0_t_srtt + 4 * cb'_0_t_rttvar)) = 1
r <= 6
3 <= 160 * cb'_0_t_rttvar
19 <= 400 * cb'_0_t_srtt
100 <= 25 * r
800 * cb'_0_t_rttvar <= 19
20 * cb'_0_t_srtt <= 1
```

What is `pending` all about?

Lazy Handling of Equality Constraints

Many constraints in a Γ_i are equalities on freshly introduced variables. (Often in turn derived from `let` expressions in the specification.)

`pending` is used to "tag" equalities that shouldn't be eliminated, like

```
v = if P then e1 else e2
```

Only substitute out "values". E.g.,

- `v = 4`
- `v = SOME 4`
- `v = h :: t`

In `h :: t`, sub-expressions `h` and `t` may still be complicated.
— Only substitute out the "cons", inventing fresh variable bindings for `h` and `t`

Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

Handling High-Level Specification Idioms

Specification often seemed to be written to cause me maximum pain:

- Use of complicated set comprehensions
- Use of non-injective patterns
- General use of a declarative style
- . . .

In all cases, it's important to **keep** the original specification.

- If I force the specifiers to write Prolog programs, we lose HOL's advantages.
- The response is to **prove** logical equivalences, "translating" the high-level expressions into more executable forms.

Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

Non-injective Pattern-Matching

Specifiers like to write things like:

$$\frac{\text{side conditions}}{\mathrm{h}\langle \mathit{fld}_1 := v_1 \rangle \xrightarrow{\ell} \mathrm{h}\langle \mathit{fld}_1 := v_2;\ \mathit{fld}_2 := 10 \rangle}$$

When we try to discover if our current host can make this transition, there is no unique h to choose.

Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

Non-injective Pattern-Matching

Specifiers like to write things like:

$$\frac{\text{side conditions}}{h\langle \mathit{fld}_1 := v_1\rangle \xrightarrow{\ell} h\langle \mathit{fld}_1 := v_2;\ \mathit{fld}_2 := 10\rangle}$$

When we try to discover if our current host can make this transition, there is no unique h to choose.

For records (like hosts), programmatically rewrite above to

$$\frac{\text{side conditions}}{\begin{array}{c}\langle \mathit{fld}_1 := v_1;\ \mathit{fld}_2 := v_2';\ \cdots \mathit{fld}_n := v_n\rangle \xrightarrow{\ell} \\ \langle \mathit{fld}_1 := v_2;\ \mathit{fld}_2 := 10;\ \cdots \mathit{fld}_n := v_n\rangle\end{array}}$$

Non-injective Pattern-Matching & (Finite) Maps

If a specifier writes

$$\frac{\text{side conditions}}{\mathrm{h}\langle ts := tids \oplus (\mathrm{t} \mapsto \nu)\rangle \xrightarrow{\ell} \mathrm{h}\langle ts := tids \oplus (\mathrm{t} \mapsto \nu')\rangle}$$

we can't expand the ts map into a function with a fixed domain as we did with records.

Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

Non-injective Pattern-Matching & (Finite) Maps

If a specifier writes

$$\frac{\text{side conditions}}{\mathrm{h}\langle ts := tids \oplus (\mathrm{t} \mapsto v)\rangle \xrightarrow{\ell} \mathrm{h}\langle ts := tids \oplus (\mathrm{t} \mapsto v')\rangle}$$

we can't expand the ts map into a function with a fixed domain as we did with records.

Rewrite (again, programmatically!) to

$$\frac{\text{side conditions} \quad \mathrm{t} \in \mathrm{dom}(\mathrm{f}) \quad \mathrm{f}(\mathrm{t}) = v}{\mathrm{h}\langle ts := \mathrm{f}\rangle \xrightarrow{\ell} \mathrm{h}\langle ts := \mathrm{f} \oplus (\mathrm{t} \mapsto v')\rangle}$$

(Some finite map idioms are not this simple.)

Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

The Streams Level Specification

- The specification at the segment level is
  - **good** for implementors
  - **bad** (difficult) for users
- TCP implements an abstraction: two reliable streams between the local and remote end-points
- Can we provide a validated streams level specification of TCP/IP?

First, Write Your Streams Level Specification

The Cambridge specifiers wrote a specification that:

- replaces segment movements with reads and writes to pairs of streams
- drastically reduces the complexity of hosts (TCP control blocks disappear)
- retains accurate descriptions of error cases in the Sockets API

Second, Abstract From Low to High

The next step is to write two abstraction functions:

- One, *low-state* $\longrightarrow$ *high-state* removes extraneous detail from hosts, and calculates the state of the streams
- Another, *low-label* $\longrightarrow$ *high-label* translates low-level observations into high-level observations.
  - Socket calls remain the same;
  - Packet movements on the network become unobservable τ transitions

Third, Validate High-Level Specification

Every trace that is to be validated at the high level has already been validated at the low level.

Consequently:

- have a sequence of low-level hosts witnessing the trace
- know which reduction rules were chosen at the low level
  (high-level rules were written to correspond)

So, high-level validation is much simpler...

Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

High-Level Validation Problem

Given concrete h_0, h, ℓ and the particular rule to check ($\textsc{SomeRule}$), determine if

$$\frac{}{h_0 \xrightarrow{\ell} h}\ \textsc{SomeRule}$$

- There are no symbolic values in h.
- Can find witnesses for any existential variables in $\textsc{SomeRule}$'s side conditions immediately

Results

We ran approx. 2000 segment level tests on each of Windows, BSD and Linux.

Over 90% of traces now succeed. (Focus on BSD's TCP.)

So, we're confident our specification is

- Precise
- Accurate

We also found what must be bugs...

Sample TCP Bugs

- incorrect RTT estimate after repeated retransmission timeouts
- `TCPSHAVERCVDFIN` wrong — so can `SIGURG` a closed connection
- initial retransmit timer miscalculation
- simultaneous open responds with ACK instead of SYN,ACK
- receive window updated even for bad segment
- urgent pointer not updated in fastpath (so after 2GB, won't work for 2GB)
- shutdown state changes in pre-established states
- (Linux) sending options in a SYN,ACK that were not received in SYN

Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

Using Our Technology in Other Settings

Work on TCP is well-motivated.

But, the approach is well-suited to the design and testing of new protocols too.

Using Our Technology in Other Settings

Work on TCP is well-motivated.

But, the approach is well-suited to the design and testing of new protocols too.

In fact, starting by writing a formal specification may make things easier:

- Can write spec. to be executable
- Avoids dangerous hand-waving
- Gives implementors a clear target

Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion

Functional Specifications

Writing a full functional specification can be a **massive** undertaking.

Circumstances when it can be worthwhile:

- Critical applications
- When multiple (interoperating?) implementations are expected

An ideal specification

- Is mechanically manipulable
- Is written in expressive language
- Does not specify too much
- Has a clear meaning

Mechanising a Specification

Specifications should be

- written in advance; and
- with mechanisation in mind

This is not difficult, and focussing the mind early is of great benefit to a design.

A mechanised specification then allows efficient testing of a new implementation.

This was done successfully by Ridge, Sewell, Dales and Jansen in the design of a new optical switch (with Intel Research).

Introduction
Beginning a TCP Specification
The Segment Level Specification
Use of HOL
Specification Validation
As a Theorem Proving Problem
The High Level Specification
Conclusion