CS423: Reliable Transmission and Protocol Fundamentals

George Varghese

January 27, 2000
Global State of a Protocol

- Global state is the state of every node and every link in the system.
- State of every node is value of all relevant variables. State of link is the sequence of frames currently stored on link.
- Think of protocol that executes by choosing a next state transition and changing global state
• Can choose any one of multiply enabled transitions (assuming no time-bounds). Can create an infinite number of global states.

• Want to prove protocol correct across all possible executions. How?
Invariants

• Invariants describe possible states of a protocol. In example:
  
  – **If** $M$ in forward link or both links empty **then** $i = j$.
  – **Else if** $A$ in reverse link . . .
Proving Invariants

- Need only examine *small* number of transitions instead of *infinite* number of executions. Possible to automate.

- Proof by induction. Assume true initially (goal of protocol initialization). All transitions must preserve invariant.
Band Invariant for Stop-and-wait

Two Bands of equal values

\[ y = x \text{ or } x = y+1 \]
Band Proof

*Receiving a non-duplicate frame*

- How many cases in proof? Subcases?
- Stop and wait only compares numbers; so enough to check LSB. ⇒ alternating bit.
Band Proof-2

Receiving a Valid Ack

- Implies that when sender changes number to $i + 1$, the forward link has only $i$ and reverse link has only $i + 1$.
- This is why sender’s transmission is guaranteed to work.
Latency and Throughput

- **Throughput**: jobs completed per second. System owners want to maximize this.

- **Latency**: worst-case time to complete a job. Users want to minimize.

- **Propagation Delay**: Time for transmitted bit to reach receiver. Contrast to transmission rate.

- **Pipe Size**: Transmission Rate * Round-trip Propagation Delay. Need to pipeline if pipe size is large. Alternating bit does not.
Sliding Window Protocols

- Sender can send a window of outstanding frames before getting any acks. Lower window $L$, can send up to $L + w - 1$.

- Receiver has a receive sequence number $R$, next number it expects. $L$ and $R$ are initially 0.

- Sender retransmits all frames in current window until it gets an ack. Ack numbered $R$ implicitly acks all numbers $< R$.

- Two variants: receiver accepts frames in order only (go-back-n) or buffers out-of-order frames (selective reject)
Go-back n code

Send (s, m)

The sender can send this frame if:

m corresponds to s-th data item
given to sender by client AND
L \leq s \leq L + w - 1

Receive(r, Ack)

On receipt:
L := R

Receive(s, m)

On receipt:
If s = R then
R := s + 1
deliver data m to client.

Send(r, Ack)

r must equal R

Selective Reject Code
Send \( (s,m) \)

The sender can send this frame if:
- \( m \) corresponds to \( s \)-th data item given to sender by client AND
- \( L \leq s \leq L + w - 1 \) AND
- \( s \) has not been acked.

Receive \( (r, \text{List}, \text{Ack}) \)

On receipt:
- \( L = R \)
- Mark numbers in List as acked

Receive \( (s,m) \)

On receipt:
- If \( s \geq R \) then
  - Mark \( s \) as acked; store \( m \)
- While \( R \) not acked do
  - Deliver data at position \( R \)
  - \( R := R + 1 \)
Send(r, List, Ack)

r must equal R

List contains acked numbers > R
Protocol Design Lesson 1

- First design simple protocols; then optimize using what you know of protocol invariants.

- For stop-and-wait, we got away with a space of two numbers $w + 1$. Can we do same for sliding window? Depends on sliding window variant used.
Protocol Design Lesson 2

• Understand what parts of protocol need to be specified for correctness. Some parts can be left to implementors to optimize for their own implementations. Separate out details from main idea.

• For example: how to manage timers, when to send acks.
Implementation and Other Details

- Timers: works regardless of values, but needed for performance. So calculate round-trip delay.
- Need only one timer (for lowest outstanding number) in Go-back-\(n\). Need one for each window element in Sel Reject.
- In selective reject, have to send an ack with \(R\) and a bit-map of numbers greater than \(R\) that have been received.
- Piggybacking: to reduce frames sent.
How naive restarts can fail
Protocol Design Lesson 3

• Impossibility results tell us what we need to change to get our job done, not just what we can’t do.

• In order to solve crash impossibility: either use non-volatile memory, timers, or random numbers.

• Another fundamental impossibility result is 2 generals.
Design Lessons

- Use invariants to understand and design.
- Start simple. Optimize later.
- Separate out details from essence.
- Know what’s impossible and change assumptions to get job done.
Thus comparisons between $s$ and $R$ can be done mod $m$, $m > w$ and still get the same answers. See text for a counterexample when $m=w$
SELECTIVE REJECT MODULUS

\[ D(s) \]

Since receiver must buffer n packets ahead, simple equality testing is no longer enough. Must be able to tell apart the last \( w \) frames from next \( w \) frames. Needs modulus \( m \geq 2w \).

See text for counterexample if \( m < 2w \)
FLOW CONTROL

Windows provide static flow control. Can provide dynamic flow control if receiver acks indicate what receiver will buffer.

- Flow control without error recovery
  - Credits
  - Rate Control (sender does not send > R frames/sec)
INITIALIZING LINK PROTOCOLS
(in the face of link and node crashes)

EXAMPLE: SETM = RESTART in HDLC
WHAT MAKES PROTOCOLS HARD?

Asynchrony and Faults

LUNCH IN 5 MINUTES?

IF LINE CAN FAIL FOR UP TO 5 MINUTES, IT IS IMPOSSIBLE TO PREVENT THE POSSIBILITY OF EMBARRASSMENT

Can’t know state of remote nodes at specific times. Luckily we don’t need this for most useful protocols.