Transport Layer Part III

Vibha Patil – vp3fr
March 3, 2020

1. Pipeline Protocols Overview:

1.1 Go-Back-N (GBN)

We have a window: Notion of pipelining and cumulative acknowledgements.
If we send packets 0, 1 2, 3, but there is a loss on packet 2, then we'll get acks for 0 and 1, and send packets 4 and 5, but will get an ack for 1 again since the receiver gets packet 3 but ignores it because it still hasn't gotten packet 2.
Sender can have up to **N** un-acked packets in the pipeline, **N** being the window size.
Receiver only sends cumulative ack, doesn't ack packet if there is a gap – ignore packets that are after a missing one.
Sender has timer **for oldest un-acked packet** and when timer expires, **it retransmits ALL un-acked packets**

1.2 Selective Repeat

Limit the pipeline but send individual acks for each packet instead of cumulative ack and maintain a timer **for each un-acked packet**. When timer expires, transmit **only that un-acked packet.**
So, it individually acknowledges all correctly received packets.
Buffers packets, as needed, for eventual in-order delivery to upper layer
Sender only resends packets for which ack not received. Sender timer for each un-acked packet

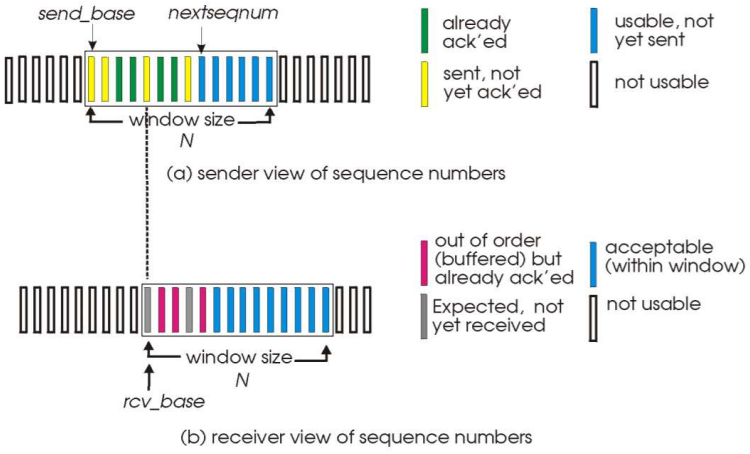

Figure 1 – Sender vs. Receiver Windows for Selective Repeat

Sender Window:

N consecutive seq#s
Has flags for acked, send but un-acked, usable but not sent, and unusable packets.

Acknowledge all packets base-n. so you'll have a cumulative ack for the base packet so the sender will be able to advance its window.

Data from above: if next available seq# in window, send pkt
Timeout(n): resent pkt n, restart timer – unique timer for every packet

ACK(n) in [sendbase, sendbase+N]: mark pkt n as received, if n smallest un-acked pkt, advance window base to next un-acked seq#

Receiver window:

It increments when the packet is received and ack is sent, but not when the ack is received. So, it's buffered because it's out of order.

Pkt n in [recvbase, rcvbase+N-1]
Send ACK(n)
Out-of-order: buffer
In order: deliver (deliver buffered, in order pkts), advance winto do next not-yet received pkt.

Pkt n in [rcvbase-N, rcvbase-1] (prev window):
Ack(n)

Otherwise, ignore!!

Strategy for window size: if you can't do the buffer fast enough, window is made smaller. For TCP, that is negotiated over time. Every time a timer expires, it sends that packet.

Selective repeat – dilemma:
If pkt 0, 1, 2 are sent, but all of the acks get lost, then the recv window moves and there's a new 0 in its window, but the sender sends pkt 0 again because the ack was never received – duplicate data received in the wrong order. **Thus: seq#s must be half the window size.**

2. TCP – overview

Point-to-point: one sender and one receiver
Reliable, in order byte steam: no message boundaries
Pipelined: TCP congestion and flow control set window size.
Full duplex data: bidirectional data flow in same connection. MSS: Maximum segment size.
Connection-oriented: so handshaking (exchange of control msgs), inits sender, receiver state before data exchange
Flow controlled: sender will not overwhelm the receiver. Trick the window size so there are fewer things in flight.

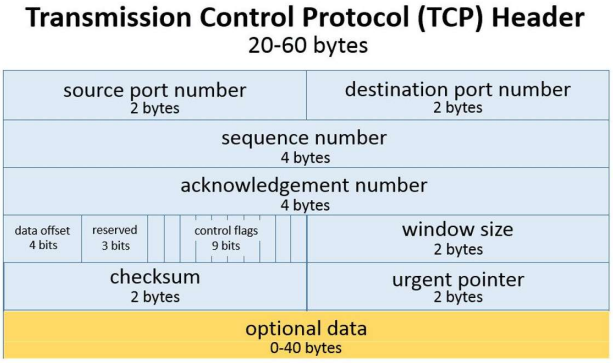

Figure 2 – Transmission Control Protocol Header

TCP segment structure:

Src port #, dst port #, src ip addr, dst ip addr.
Full duplex approach
Unique relationship between seq# and ack#
Certain flags: URG (urgent data), ACK (Ack# valid), PSH (push data now), RST, SYN, FIN (connection established, setup, and teardown commands)

Each TCP Packet has a header that contains the information in the format specified in the TCP Segment structure and Figure 2.

TCP seq numbers and ack numbers:
Telnet – ssh that isn't secure – communication. User types 'C' on Host A (seq= **42**, ack=79, data=c), Host B 'acks' receipt of C and echos back C to Host A (seq=**79**, ack = **43**, data =c – advance the ack), Host A 'acks' receipt of echoed C to Host B (seq=43, ack=**80** – advance ack) – seq# is what I'm going to send and ack# is what I expect to get.

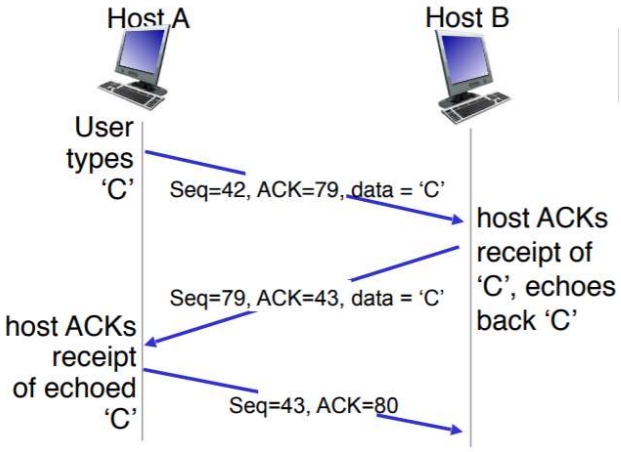

Figure 3 – Sequence and Ack numbers

Sequence numbers: A byte stream number that takes up the four bytes in the TCP header. Used to identify consecutive packets

Acknowledgment numbers: sequence number of next packet expected from other side/the packet that should be received next)

3. Round Trip Time

How do we set the TCP timer?
It has to be longer than RTT but RTT varies.

If it's too short, then we get premature timeouts – unnecessary retransmissions
If it's too long, then we get slow reaction to segment loss.

How to estimate RTT?
SampleRTT: measured time from segment transmission until ACK receipt. ignore retransmissions
However, Sample RTT will vary, so we want to estimate RTT "smoother" by taking the average of several recent measurements, not just current SampleRTT.

$$DevRTT = (1-\beta)*DevRTT + \beta*|SampleRTT-EstimatedRTT|$$

(typically, $\beta = 0.25$)

$$TimeoutInterval = EstimatedRTT + 4*DevRTT$$

estimated RTT

"safety margin"

Figure 4 – DeviationRTT Formula

EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT

- ➔ Exponential weighted moving average
- ➔ Influence of past sample decreases exponentially fast
- ➔ **Usually: α = 0.125**

The timeout interval is the EstimatedRTT plus safety margin (using largest EstimatedRTT and largest safety margin)
Estimate Sample RTT deviation from Estimated RTT
DevRTT formula will be given during an exam but it is as follows on Figure 4.
TimeoutInterval = EstimatedRtt + 4*DevRTT -> the 4 is just the safety margin.
** We don't have to set the timer in this way for the hw **

4. TCP – Reliable Data Transfer

Creates a reliable delivery service on top of IP's unreliable service.

- Pipelined segments cumulative acks, single retransmission timer

Retransmissions triggered by:

- Timeout events
- Duplicate acks

Initially consider simplified TCP sender:

- Ignore duplicate acks
- Ignore flow control, congestion control

TCP fast retransmission - After sender receipt of triple duplicate ack.
TCP Sender Events – There are three events for the sender: Data received from app, a timeout occurs, or an ack is received.

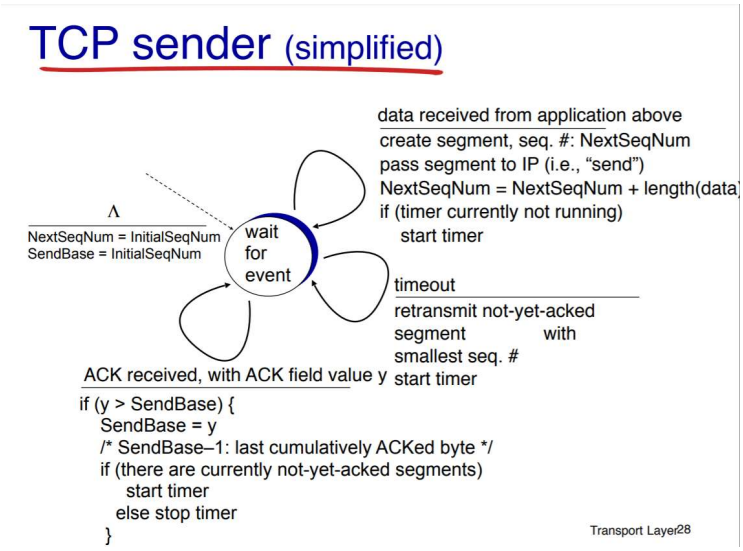

Figure 5 – TCP Sender

Received Data:
In the case of receiving data from the app, the sender will create a TCP packet with a sequence number (seq# is byte-stream number of the first data byte in the segment/packet). It will then activate the timer if it isn't already running – the timer can be thought to be a timer for the oldest un-acked segment. With it running, we can know which segment has not received a corresponding ack. When the **timeout condition** has been reached (some sort of timeout interval for the timer), the sender proceeds to the timeout event. When an **ack is received**, the sender proceeds to the acknowledgment received event.

Timeout Event:
In the event of a timeout, the sender will retransmit the timed-out packet and restart the timer, waiting for the timeout condition or the ack.

Acknowledgment received:
In the event of an acknowledgment, if the ack acknowledges previously un-acked segments, it will update the un-acked segment and start the timer if it is still waiting on acks for other segments.

To see various retransmission scenarios, refer to slide 29 and 30 from the class slides.

5. Other

Elon Musk has a new internet - Spacex Starlink.
Broadcast protocol – broadcast signal to all systems.
For HW: Implement Go Back N, not the timer.
Book: Network Algorithmics – algorithms of networks.