

Transports and TCP

Adolfo Rodriguez
CPS 214

DUKE

Host-to-Host vs. Process-to-Process Communication

- Until now, we have focused on delivering packets between arbitrary *hosts* connected to Internet
 - Routing protocols
 - IP best effort delivery model
 - Scalability and robustness through hierarchy and soft state
 - Transition to arbitrary *processes* communicating together
 - One goal: provide illusion that all processes located on one large computer
 - Can address (*name*) and reliably communicate with any process
- Ports

DUKE

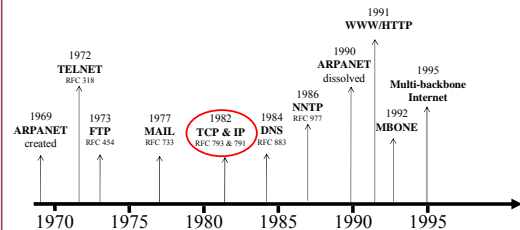
UDP

- User Datagram Protocol (UDP)
 - Simple demultiplexing
 - No guarantees about reliability, in-order delivery
 - Thin veneer on top of IP adds src/dest port numbers
 - 16 bit port number allows for identification of 65536 unique communication endpoints per host
 - Note that a single process can utilize multiple ports
 - IP addr + port number uniquely identifies all Internet endpoints



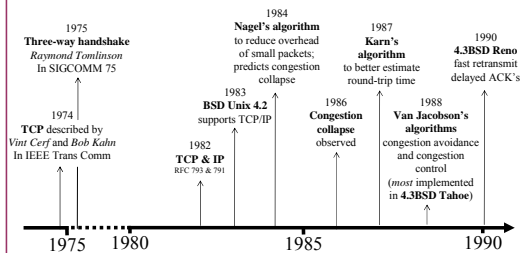
DUKE

A Brief Internet History



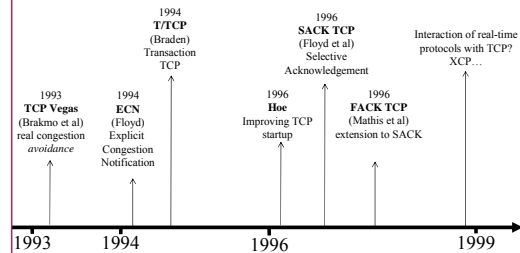
DUKE

TCP Timeline



DUKE

TCP: After 1990



DUKE

TCP

- Transmission Control Protocol (TCP)
 - Reliable in-order delivery of byte stream
 - Full duplex (endpoints simultaneously send/receive)
e.g., single socket for web browser talking to web server
 - Flow-control
To ensure that sender does not overrun *receiver*
Fast server talking to slow client
 - Congestion control
Keep the sender from overrunning the *network*
Many simultaneous connections across routers (cross traffic)

DUKE

TCP

- Utilize sliding window protocol, plus:
 - Need for connection establishment (no dedicated cable)
 - Varying round trip times over life of connection
Different paths, different levels of congestion
 - Ready for *very* old packets
 - Delay-bandwidth product highly variable
Amount of available buffer space at receivers also variable
 - Sender has no idea what links will be traversed to receiver
Must dynamically estimate changing end-to-end characteristics

DUKE

TCP Flavors

- TCP Tahoe
 - Jacobson's implementation of congestion control (AIMD)
- TCP Reno
 - Fast recovery
 - Fast retransmit
 - Delayed ACK's
- TCP Vegas
 - Source-based congestion *avoidance* rather than control
 - TCP Reno needs to cause congestion to determine available bandwidth

DUKE

TCP Header Format

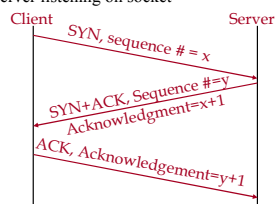
0	4	10	16	31
SrcPort		DestPort		
SequenceNum				
Acknowledgment				
HdrLen	0	Flags	AdvertisedWindow	
Checksum		UrgPtr		
Options (variable)				
Data				

- Without options, TCP header 20 bytes
 - Thus, typical Internet packet minimum of 40 bytes

DUKE

TCP Connection Establishment

- Exchange necessary information to begin communication
- Three-way handshake
 - E.g., server listening on socket



DUKE

TCP Connection Teardown

- Closing process sends a FIN message
 - Waits for ACK of FIN to come back
 - This side of the connection is now closed
- Each side of a TCP connection can independently close the connection
 - Thus, possible to have a half duplex connection

DUKE

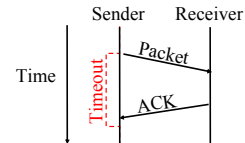
Reliable Transmission

- How do we send a packet reliably when it can be lost?
- Two mechanisms
 - Acknowledgements
 - Timeouts
- Simplest reliable protocol: Stop and Wait

DUKE

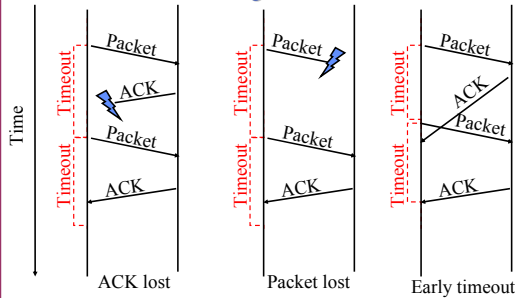
Stop and Wait

Send a packet, stop and wait until acknowledgement arrives



DUKE

Recovering From Error



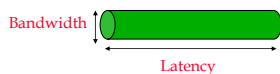
DUKE

Problems with Stop and Wait

- How to recognize a duplicate transmission?
 - Solution: put sequence number in packet
- Performance
 - Unless Latency-Bandwidth product is very small, sender cannot fill the pipe
 - Solution: sliding window protocols

DUKE

Keeping the Pipe Full

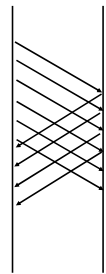


- Bandwidth-Delay product measures network capacity
- How much data can you put into the network before the first byte reaches receiver
- Stop and Wait: 1 data packet per RTT
 - Ex. 1.5-Mbps link with 45-ms RTT
 - Stop-and-wait: 182 Kbps
- Ideally, send enough packets to fill the pipe before requiring first ACK

DUKE

How Do We Keep the Pipe Full?

- Send multiple packets without waiting for first to be ACK'd
- Reliable, unordered delivery:
 - Send new packet after each ACK
 - Sender keeps list of unack'd packets; resends after timeout
- Ideally, first ACK arrives immediately after pipe is filled
 - Opens up another "slot"



DUKE

TCP Flow Control

- TCP is a sliding window protocol
 - For window size n , can send up to n bytes without receiving an acknowledgement
 - When the data is acknowledged then the window slides forward
- Each packet advertises a window size in TCP header
 - Indicates number of bytes the receiver is willing to get
- Original TCP always sent entire window immediately
 - Too bursty?

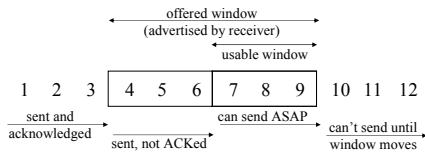
DUKE

Sliding Window

- Receivers buffer later packets until prior packets arrive
 - For out of order delivery
- Sender must prevent buffer overflow at receiver
 - Flow control
- Solution: sliding window
 - Circular buffer at sender and receiver
 - Packets in transit \leq buffer size
 - Advance when sender and receiver agree packets at beginning have been received

DUKE

Visualizing the Window

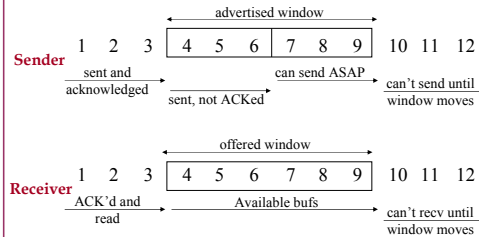


Left side of window advances when data is acknowledged.
Right side controlled by size of window advertisement.

DUKE

Visualizing the Window: Example

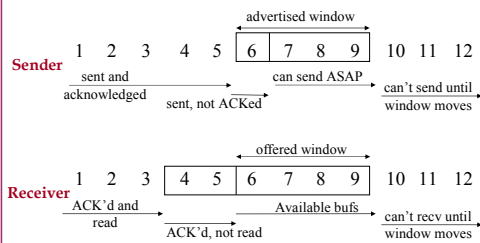
Initial State, Receiver has 6 slots to buffer packets
Packets 4, 5, 6 sent, but not yet received



DUKE

Visualizing the Window: Example

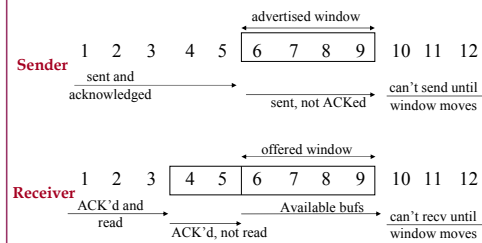
Receiver to Sender → ACK 5, Window 4



DUKE

Visualizing the Window: Example

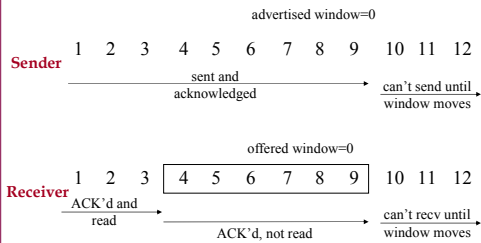
Sender to Receiver → Send 7, 8, 9



DUKE

Visualizing the Window: Example

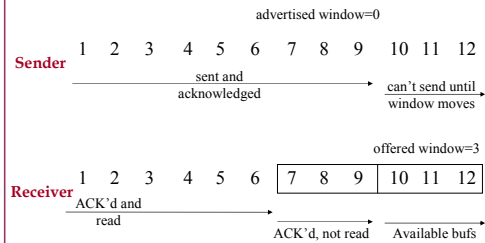
Receiver to Sender → ACK 9, Window 0



DUKE

Visualizing the Window: Example

Receiver App reads packets 4, 5, 6
But sender has no way of knowing that more room is available!



DUKE

Options for Sender Discovery of Increased Advertised Window

- Receiver sends duplicate ACK with a larger advertised window
 - Complicates receiver design
 - TCP design philosophy: keep receiver simple
Also explains slow deployment of SACK, NACK, etc.
- Sender periodically transmits a 1-byte packet
 - If no space available at receiver → packet dropped, no ACK
 - If additional space became available → ACK contains new advertised window
- NOTE: advertised window in bytes, not packets

DUKE

Sequence Numbers

- TCP uses 32-bit sequence number
 - TCP assumes that packet will not live in Internet for > 1 min
 - On 622 Mbps link, can wrap 32-bit sequence number in 55 seconds
Gbps links becoming common
 - Why is this a problem?

DUKE

Sequence Numbers

- TCP uses 32-bit sequence number
 - TCP assumes that packet will not live in Internet for > 1 min
 - On 622 Mbps link, can wrap 32-bit sequence number in 55 seconds
Gbps links becoming common
 - Proposal: extend sequence number with timestamp to distinguish between old and new incarnations of packets

DUKE

Advertised Window

- TCP uses a 16-bit advertised window field (flow control)
 - Specifies number of bytes that can be sent from sender to receiver
 - Recall "keeping pipe full" to obtain available bandwidth
 - 16-bit field translates to max 64KB advertised window
 - For 100 ms RTT T3 link (45 Mbps), delay-bandwidth product is 549 KB
Advertised window not large enough to keep pipe full
Poor bandwidth utilization

DUKE

Advertised Window

- TCP uses a 16-bit advertised window field (flow control)
 - Specifies number of bytes that can be sent from sender to receiver
 - Recall “keeping pipe full” to obtain available bandwidth
 - 16-bit field translates to max 64KB advertised window
 - For 100 ms RTT T3 link (45 Mbps), delay-bandwidth product is 549 KB
 - Advertised window not large enough to keep pipe full
 - Poor bandwidth utilization
 - Proposal: advertised window specifies chunks larger than byte granularity

DUKE

Adaptive Retransmission for Reliable Delivery

- TCP retransmits packet if ACK not received within timeout period
 - Necessary for reliability on top of “best-effort” IP
- Round trip time varies with congestion, route changes, ...
 - If timeout too small, useless retransmits
 - If timeout too big, low utilization
- TCP: estimate RTT by timing ACKs
 - Exponential weighted moving average
 - Factor in RTT variability

DUKE

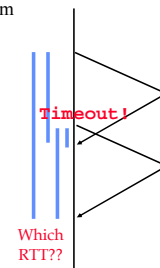
Retransmission

- How long a timeout to set?
- Original TCP: Estimate round-trip time R
 - $R = \alpha R + (1 - \alpha)M$
 - α is a smoothing factor of 0.9
 - Places much more weight on historical result
 - Smooth out outlying measurements
 - M is measured round-trip time (ACK's to data)
- Timeout at $R\beta$, where β is a delay variance factor of 2.0
 - Conservative: do not retransmit until two RTT's have passed
 - Jacobson's TCP modifications allows for varying β

DUKE

Retransmission Ambiguity

- How do we distinguish first ACK from retransmitted ACK?
 - First send to first ACK
 - What if ACK dropped?
 - Last send to last ACK
 - What if last ACK dropped?



DUKE

Retransmission Ambiguity: Solutions

- TCP: Karn-Partridge
 - Ignore RTT estimates for retransmitted packets
 - Double timeout on every retransmission
 - Exponential backoff similar to Ethernet for congestion avoidance
- Add sequence #'s to retransmissions (retry #1, retry #2)
- TCP proposal: Add timestamp into packet header; ACK returns timestamp

DUKE

Jacobson's RTT estimator

- Problem:
 - Original TCP does not adapt to wide variance in RTT
 - Uses fixed β of 2.0
- Need to account for both estimate RTT and variance
- Jacobson:
 - Low variance \rightarrow estimate RTT sufficient
 - High variance \rightarrow estimate RTT could be far off
- Solution:
 - $\text{Timeout} = \mu * \text{EstimateRTT} + \phi * \text{Deviation}$
 - $\mu=1, \phi=4$

DUKE

Can We Shortcut Timeout?

- If packets usually arrive in order, out of order signals a drop
 - Negative ACK (NACK)
Receiver requests missing packet
 - Selective ACK (SACK)
Receiver describes state of receive window
 - Fast retransmit
Sender detects missing ACK from multiple duplicate ACKs
Recall: receiver ACKs highest sequence # received in order
Triple duplicate ACKs for fast retransmission (shortcut timeout)
- How is retransmission timer related to congestion control?

DUKE

Transport Protocol Summary

- TCP designed to connect arbitrary hosts on the Internet
 - Difficult to determine link characteristics
 - Difficult to determine receiving host characteristics
 - Both host/network characteristics change over time
Network becomes more congested → larger RTT
Receiving becomes overloaded → smaller advertised window
- TCP provides
 - Reliable, in-order delivery of byte stream
 - Flow control
 - Congestion control

DUKE

Silly Window Syndrome

- Problem: (Clark, 1982)
 - If receiver advertises small increases in the receive window then the sender may waste time sending lots of small packets
e.g., application reads small number of bytes, freeing up small amount of kernel buffer space
- Solution:
 - Receiver must not advertise small window increases

DUKE

Nagel's algorithm (self-clocking)

- Small packet problem:
 - Don't want to send a 41 byte packet for each keystroke
IP 20 bytes, TCP 20 bytes, keystroke 1 byte
 - How long should OS/app buffer keystrokes?
- Solution:
 - Only one outstanding small segment not yet ACK'd
e.g., telnet cannot echo last character until ACK'd anyway
- Can turn of with TCP_NODELAY option
 - What's the story with these options anyway?

DUKE