Lecture 6

Transport Layer

(Computer Communication Networks)

CS 35201
Spring 2020

H. Peyravi
Department of Computer Science
Kent State University

§6.0.0 Contents

The contents of this lecture have been composed from various resources including those listed at the reference section.

§6.0.0 Glossaries

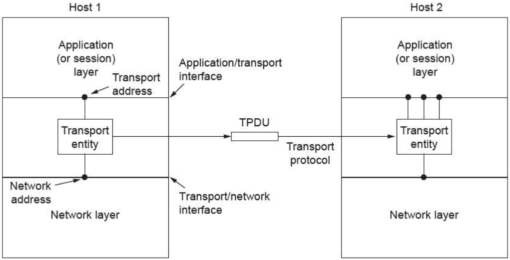
ACK	Acknowledgment 18, 23
AIMD	Additive Increase, Multiplicative Decrease 40
AQM	Active Queue Management 52–55
ARED	Adaptive RED 52
ARP	Address Resolution Protocol 17
ARQ	Automatic Repeat Request 18
CA	Congestion AvoidanceC 35, 38
CC	Congestion Control 8, 9, 13, 35, 40, 55
CRC	Cyclic Redundancy Check 50
DLC	Data Link Control 26
DNS	Domain Name System 16, 17
E2E	End-to-End 38
EC	Error Control 9
ECN	Explicit Congestion Notification 57
EWMA	Exponentially Weighted Moving Average 57
FC	Flow Control 7–9, 13, 54
FTP	File Transfer Protocol 55
IP	Internet Protocol 5, 15, 17, 19, 25
LAN	Local Area Network 19
RED	Random Early Detection 52, 56
RTT	Round Trip Time 26, 40, 41, 43, 45, 47
SNMP	Simple Network Management Protocol 16, 17
TCP	Transport Control Protocol 7, 10, 15, 17, 19, 25, 36, 38–40, 43, 50, 52, 56
TFTP	Trivial File Transfer Protocol 17
TO	Time Out 28, 45, 47
TPDU	Transport Protocol Data Unit 9
UDP	User Data Protocol 7, 10, 15–17, 36

Part I

Transport Layer

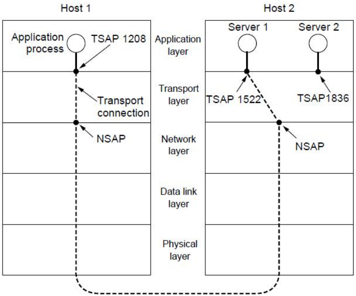
§6.1.0 Transport Layer I

1. Provides logical communication between application processes
   - Not hosts
2. Provides reliable connection-oriented services ⇒ TCP
3. Provides unreliable connection-less-oriented services ⇒ UDP
4. Provides Multiplexing/Demultiplexing of application services
   - Based on port# and Internet Protocol (IP) addresses
5. Provides parameters for specifying quality of services
6. Runs on end systems, but relies on network services ⇓



§6.1.1 Elements of Transport Protocols

- Functions
  1. Addressing
  2. Connection establishment
  3. Connection release
  4. Error control and flow control
  5. Multiplexing
  6. Crash recovery
- Transport service access point
  - Each layer has its own way of dealing with addresses
  - In Internet, a transport service access point is an IP address with a port number



§6.1.2 Transport Protocols

- Point-to-Point communication ⇒ between processes
  1. Connection-oriented ⇒ e.g. Transport Control Protocol (TCP)
     - Reliable data transfer
     - In-order delivery
     - Flow Control (FC)
     - Congestion Control
  2. Connectionless-oriented ⇒ e.g. User Data Protocol (UDP)
     - Unreliable
     - Unordered
     - No handshaking ⇒ independent segment handling
- Application Needs
  - Real-time
  - Bandwidth guarantees
  - Reliable multicast

§6.1.3 Transport Architectural Requirements

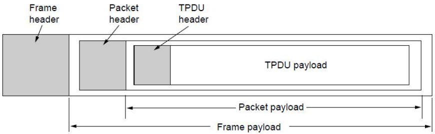
- Procedures to exchange data between devices
  - Can be complex
- High degree of cooperation between end systems

Example 6.1 (File Transfer)

- Transport needs:
  1. A data communication path ⇒ points of transfer
  2. An activation mechanism to activate the path
  3. A mechanism to determine the status of the destination
  4. A mechanism for the destination file management to store the file
  5. A format conversion mechanism
  6. A few other details such Congestion Control (CC) and FC

§6.1.4 Transport Layer Operations

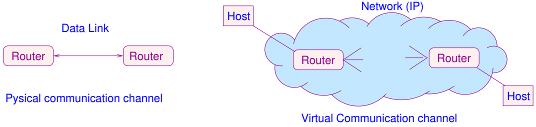
- Messages are encapsulated as Transport Protocol Data Units (TPDUs) to the network layer

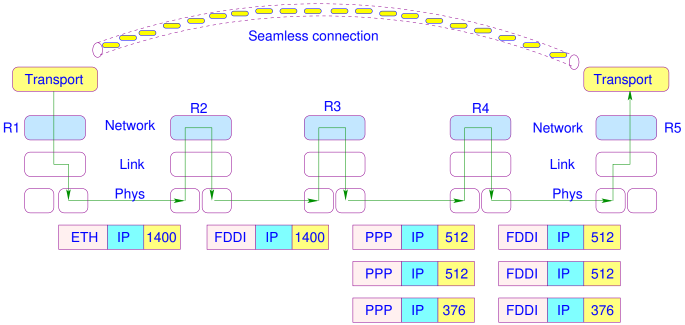


  - The network layer is in the hands of carriers
  - Transport layer is in the hand of host (computer)
    - Has no say on what the carrier actually offers
- A real hard part is establishing and releasing connections
  1. Symmetric: one side sends a disconnect request, and waits for the other to acknowledge that the connection is closed
  2. Asymmetric: one side just closes the connection, no Acks
     - Simple, but unacceptable ⇒ data may be lost *How?*
- Transport protocols strongly resemble those in the data link layer
  - Error Control (EC)
  - FC/CC
  - Reliability ⇒ sequencing
- Messages are encapsulated as TPDUs to the network layer

§6.1.5 Major differences with L2

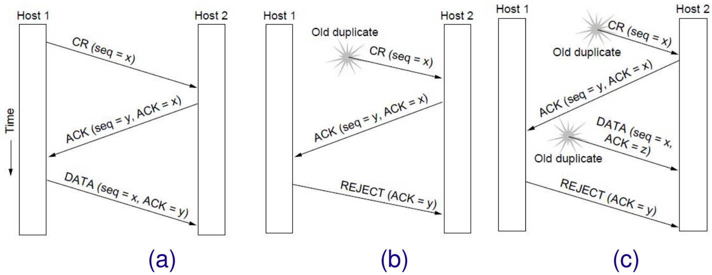
- Explicit addressing
- Establishing, maintaining, and releasing connections
  - TCP, not UDP
- Many connections (applications) require different solutions
  - Rates, losses, etc.





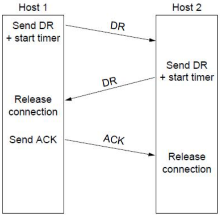
§6.1.6 Connection Establishment (3-Way Handshaking)

1. Host 1 chooses a sequence number, x, and sends a CONNECTION REQUEST TPDU containing it to host 2.
2. Host 2 replies with an ACK TPDU acknowledging x and announcing its own initial sequence number, y.
3. Finally, host 1 acknowledges host 2's choice of an initial sequence number in the first data TPDU that it sends

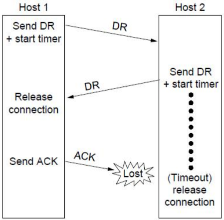


(a) Normal Operation
(b) Old duplicate CONNECTION REQUEST appearing out of nowhere
(c) Duplicate CONNECTION REQUEST and duplicate ACK

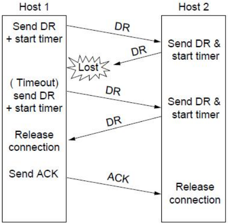
§6.1.7 Connection Release



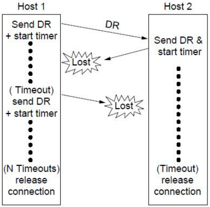
Normal operation



Final Ack lost



Response lost



Response lost and subsequent DR's lost

§6.1.8 Common Transport Protocols

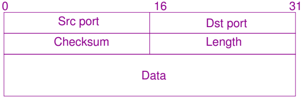
1. UDP provides just integrity and demux
2. TCP adds:
    - Connection-oriented
    - Reliable and ordered delivery
    - end-to-end
    - Byte-stream
    - Full duplex
    - FCCC

Part II

UDP vs TCP

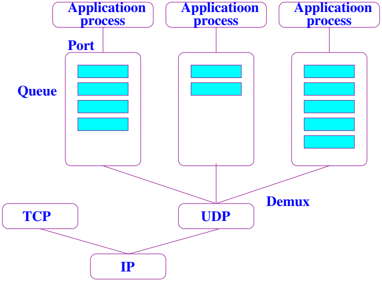
§6.2.0 User Datagram Protocol (UDP)

1. Bare bone Internet transport protocol ⇒ IP + Port #
   - Endpoints are identified by ports
   - Best effort service between source and destination



2. Connection-less
   - No connection setup/tear down between Src/Dst
   - No handshaking see /etc/services on Unix
   - Checksum is optional
     - UDP segments can be lost ⇒ Unreliable ⇒ loss
     - UDP can be delivered out of order
   - A segment is a sequence of 16-bit integer
     - Checksum= addition (1's complement) of 16-bit integers *Why?*
   - Each UDP segment handled independently
3. UDP is trade-off between reliability and speed
   - Not necessarily a good idea *Why?*
   - Small segment header
4. State-less
   - No state information ⇒ less overhead ⇒ faster then TCP
   - No congestion or flow control
   - No flow control, no Ack, no error recovery

§6.2.1 Simple UDP Multiplexing/Demultiplexing



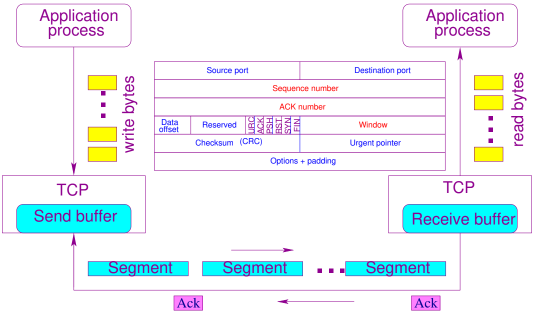
- UDP is often used for streaming multimedia applications
  - Loss tolerant
  - Rate sensitive
- Other UDP usage *Why?*
  - Domain Name System (DNS)
  - Simple Network Management Protocol (SNMP)
- Reliability is obtained by application layer
  - Application specification error recovery

§6.2.2 UDP Applications

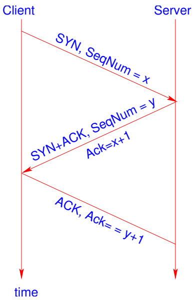
- Applications
  - Inward data collection
  - Outward data dissemination
  - Request/Reply
  - Real-time data
  - Used by SNMP, DNS, Trivial File Transfer Protocol (TFTP) etc.
- Application message encapsulated and sent to IP
  - Can result in fragmentation *How?*
- Implications
  - Responsiveness
    - TCP responds to congestion, UDP does not
  - Source Quench
    - UDP sources ignore source quench due to congestion
  - ARP Flooding (Poisoning)
    - When UDP datagram is fragmented, each fragment may generate an Address Resolution Protocol (ARP) request *How?*
    - Security concern
  - Datagram Truncation
    - A destination may truncate a large UDP datagram
    - TCP does not have this problem

§6.2.3 Transport Control Protocol (TCP)

Operation



3-Way Handshake



- A reliable transport layer protocol that preserves
  - Zero loss of bytes
    - Link Automatic Repeat Request (ARQ)
  - Zero duplication of bytes
    - Acknowledgment (ACK) (sliding window)
- Octets of the data stream are numbered sequentially

§6.2.4 TCP Header

- **Windows:** $\Rightarrow$ Operate at the octet level
  - Flow Window (W_F) $\Rightarrow$ controlled by the rate of Acks
    - Flow control: keep sender from overrunning the receiver *How?*
      - ☆ Receiver controls $\Rightarrow$ Hold ACKs
  - Congestion Window (W_C) $\Rightarrow$ Sliding Window
    - The number of bytes that the receiver is prepared to accept *How?*
      - ☆ Advertised window (W_A) $\Rightarrow \min\{W_F, W_C\}$ $W_A \leq W_C \leq W_F = W$
    - Congestion control: keep sender from overrunning network
- Connection-oriented $\Rightarrow$ Full duplex
  - Sending process writes some number of bytes
  - TCP breaks into segments and sends via IP
    - ☆ Further fragmentation possible by Local Area Networks (LANs)
- Checksum
  - Pseudo header + TCP header + Data
- Each connection identified with 4-tuple:

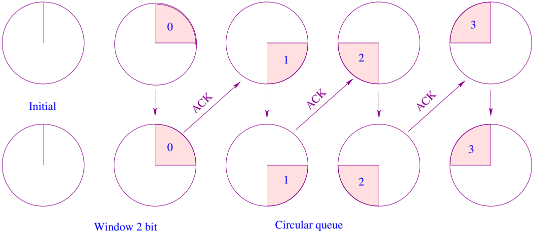
$$< SrcPort, SrcIPAddr, DstPort, DstIPAddr >$$

- Sliding window + flow control:

$$Ack, SeqNum, AdverWindow$$

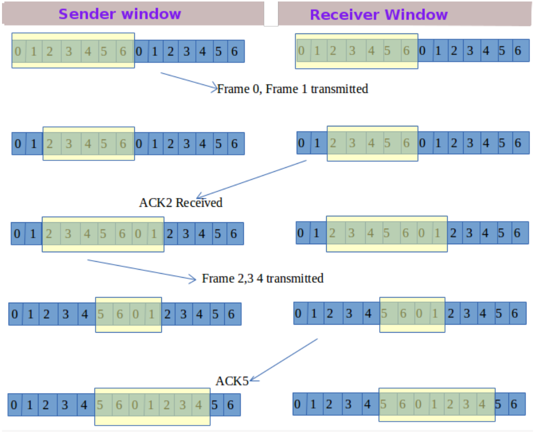
§6.2.5 TCP Window I

- The sender/receiver each has two windows:
  - Sending window
  - Receiving window
- It is a duplex mechanism
  - One can send/receive at the same time
- Windows don't have to be the same size
  - Window size = 4, but uses Stop-and-Wait protocol



§6.2.5 TCP Window II

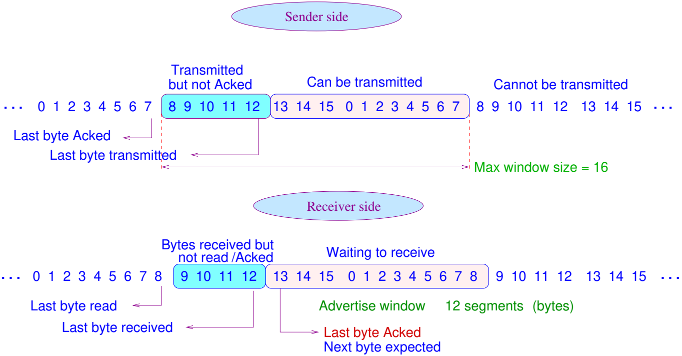
Recall Window from Data Link Layer



- Sending ACK2 = the receiver is expecting segment 2

§6.2.5 TCP Window III

- Sender keeps three pointers
  - Last byte Acked
  - Last byte Sent(transmitted)
  - Max Window
- Receiver keeps three pointers
  - Last byte Read
  - Last byte Received
  - Next Byte Expected



Sender side

Transmitted but not Acked

Can be transmitted

Cannot be transmitted

··· 0 1 2 3 4 5 6 7 | 8 9 10 11 12 | 13 14 15 0 1 2 3 4 5 6 7 | 8 9 10 11 12 13 14 15 ···

Last byte Acked

Last byte transmitted

Max window size = 16

Receiver side

Bytes received but not read /Acked

Waiting to receive

··· 0 1 2 3 4 5 6 7 8 | 9 10 11 12 | 13 14 15 0 1 2 3 4 5 6 7 8 | 9 10 11 12 13 14 15 ···

Last byte read

Last byte received

Advertise window 12 segments (bytes)

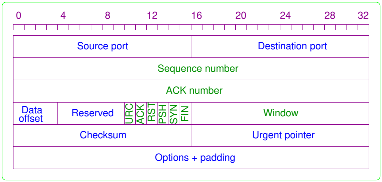
Last byte Acked
Next byte expected

Advertised window size = Maximum window size – (Last byte received –Last byte read)

§6.2.6 TCP Segment Format

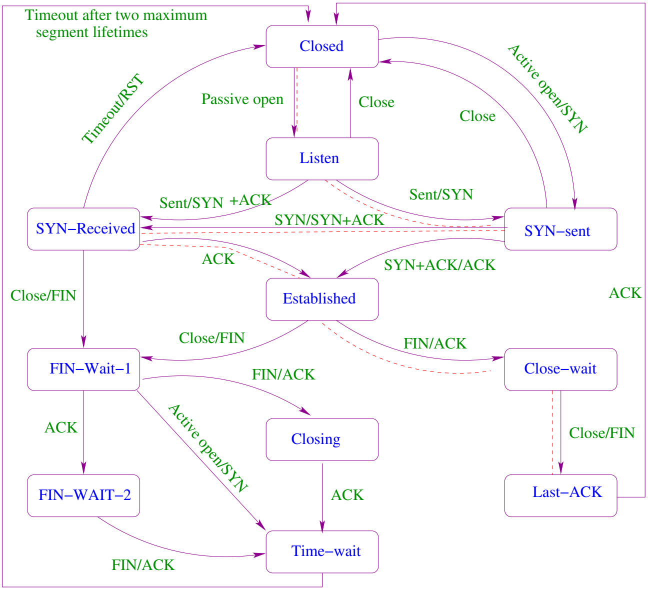
- The sender/receiver each has two windows:
- The windows move to the right not necessarily buy 1 *Why?*
- Cumulative Acks ⇒ source of bursty
  - Can work with Go-back-N retransmission protocol
    - If a byte is missing, the receiver can ACK it to be re-transmitted
  - Can not identify which block of data is lost
    - Duplicate Ack can identify the lost block
- Initially
  - Sender buffer size: `MaxSendBuffer`
  - Receive buffer size: `MaxRcvBuffer`

Segment Format



- `SYN`: Connection establishing
- `FIN`: Connection terminating
- `RESET`: Receiver is confused
- `PUSH`: Sender invokes push
- `URG`: segment contains urgent data
- `ACK`: Acknowledgment

§6.2.7 TCP Finite State Diagram



§6.2.8 TCP Significance

1. Many applications run on TCP/IP suite
2. TCP/IP protocol engine becomes critical
    - Many believe boosting hardware ⇒ better network performance
    - TCP has total control of applications
    - TCP is a complex protocol
        - Depends on many network elements
    - Unless TCP is optimized, hardware alone cannot do much
3. Emergence of new technologies and environment
    - Proliferation of different networking technologies
        - Wireless, satellite, optical, etc.
    - TCP algorithms for one environment may not work best with another
    - A huge body or literature
4. IP convergence: Many non TCP/IP industries converging to TCP/IP
    - Cellular
5. Understanding TCP/IP fundamental performance is essential

§6.2.9 End-to-End Issues

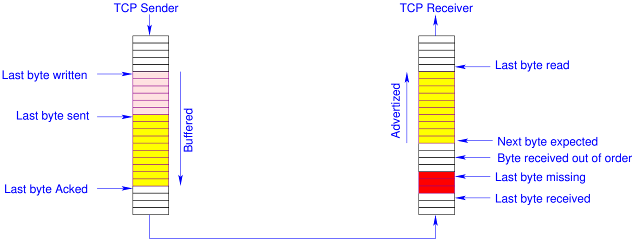
- Based on the sliding window (Data Link Control (DLC)), but very different
- Potentially connects many different hosts
  - Explicit connection establishment/termination
- Potentially different Round Trip Time (RTT)
  - Need adaptive timeout mechanism
- Potentially long delay in network
  - Need to be prepared for arrival of very old packets
- Potentially different capacity at destination
  - Buffering accommodation at the receiver
- Potentially different network capacity ⇒ congestion
  - Need to be prepared for network congestion

Part III

TCP Flow and Congestion Control

§6.3.0 TCP Flow Control

- Receiver tells Sender how much it can receive *How?*
  - ▶ 16 bit *Advertize Window*



- Receiving side: ⇒ in control of flow by:
  1. Holding off Ack ⇒ not for too long *Why?*
     - Could cause retransmission by sender if held too long
     - Cumulative Acks cause traffic burstiness *How?*
       - ☆ Adjusting Cumulative Acks to reduce burstiness may cause buffer overflow
  2. Slow the sender by persisting *Advertize Window*=0
     - Blocked sender if $(LastByteReceived - LastByteRead) + y > MaxSendBufferSize$
- Sending side
  1. Adjusts its Time Out (TO) by the rate of Acks arriving
     - It reduces un-necessary re-transmissions

§6.3.1 RTT and Sequence Number Wrap-around

- Sequence Number Wrap Around: 32-bit $SequenceNum$
- Time to Wrap-around
  - 1 Mbps $(2^{20}\ bps) \Rightarrow \frac{2^{32}}{2^{20}/2^3\ B/s} = 2^{15}\ s \approx$ 9.1 hours
  - 10 Mbps $\Rightarrow \frac{2^{32}}{10\times 2^{20}/2^3\ B/s} = 2^{15}\ s \approx$ 0.91 hour
  - 1 Gbps $\Rightarrow \frac{2^{32}}{2^{30}/2^3\ B/s} = 2^5\ s = 32$ sec.
- Wrap-around puts a limit how much we can fill the pipe
  - Each byte has a sequence number
  - Cumulative $\Rightarrow$ send several bytes $\Rightarrow$ how many?
- Bytes in Transit: 16-bit $AdvertisedWindow$
  - You can't ask for more 2^{16} bytes $\Rightarrow$ how many?
  - This will put a limit on how much you can fill the pipe
  - keeping the pipe full, assume RTT= 100 ms cross-country delay in US
    - You can't ask for more 2^{16} bytes $\Rightarrow$ then how many?
    - For 1 Mbps $Delay \times Bandwidth = 2^{20}/8\ B/s \times 0.1\ s = 13,107$ B
    - For 1 Gbps $Delay \times Bandwidth = 2^{30}/8\ B/s \times 0.1\ s = 13,421,772$ B
  - In $AdvertisedWindow$ receiver can ask for maximum 2^{16} segments $\Rightarrow$ 65,536 bytes
  - How do we solve this problem (not being able to fill the pipe)
    - Send larger segments (more than one byte)
    - 2^{16} segments, each 8 bytes $\Rightarrow$ 8×2^{16} segments = 6,524,288 bytes

§6.3.2 TCP Congestion Control

- Congestion: when traffic load $>$ network capacity
  - Traffic aggregate load exceeds the capacity
- Effects
  - Packet loss (or multiple packet losses)
    - Retransmission
    - Wasted resources due to packet loss
  - Reduced throughput $\Rightarrow$ low throughput $\Rightarrow$ low link utilization
  - Long delays $\Rightarrow$ high queuing delay
  - Congestion Collapse

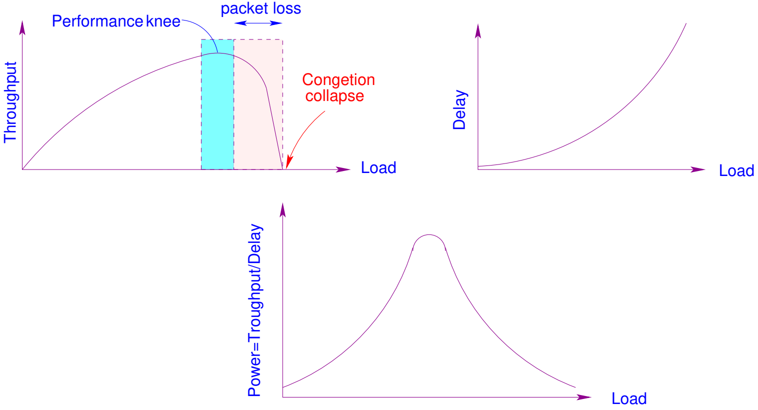
Definition 6.3.2.1 (Congestion collapse)

Congestion Collapse occurs when the network gets to a point in which it only forwards retransmissions while most of the retransmitted packets are delivered only half-way before being discarded again due to congestion

- Congestion collapse due to
  - Unnecessarily retransmitted packets
  - Un-delivered or unusable packets

§6.3.3 Congestion Collapse

- Saturation point



§6.3.4 What to Do with Congestion

- Congestion Control remains a critical issue given
  - Network growing size
  - Demand bandwidth and speed
  - Increasingly integrated services
  - Aggregate traffic
- Despite all research efforts, there is no perfect solution
- Current solutions
  - Mainly intuition driven
  - Non-linear simple schemes
  - Exhibits poor performance ⇒ not strongly responsive to
    - Oscillation, chaotic behavior, fairness
  - Ineffective, particularly for non-elastic (delay/loss sensitive) traffic
    - Multimedia
  - Not scalable

§6.3.5 Two Strategies at Saturation Point

1. Discard any incoming packet if no buffer available
2. Saturated routers exercises flow control over neighbors
   - Called back-pressure
   - May cause congestion to propagate throughout network
     - It passes congestion to other nodes
   - Oscillation

§6.3.6 What Makes CC a Difficult Problem?

1. A very complex system
2. Deals with a very large distributed system
3. Diverse in term of
    - Traffic carried
    - Application requirements
    - QoS requirements
4. Large feed-back delays $\Rightarrow$ large bandwidth $\times$ delay
5. Unpredictable user behavior in terms of
    - Timing
    - Space $\Rightarrow$ locality
    - Size $\Rightarrow$ bursts
6. Lack of appropriate dynamic models
    - Mathematically a difficult problem
7. Fairness provisioning without computational complexity

§6.3.7 Solutions

- CC ⇒ congestion recovery
  - Reactive approach after congestion is detected
- Congestion AvoidanceC (CA)
  - Proactive approach before congestion occurs
- CC involves statically or dynamically limiting demand-capacity mismatch
- Static solutions such as additional buffer, links, and processors are not effective *Why?*

§6.3.8 War Between Mice and Elephants

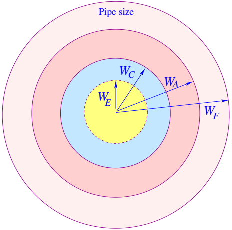
- TCP traffic is still dominant ⇒ 90%
    - But UDP is growing in a faster pace
- 50-70% of TCP are short-lived connections
    - A large number connections are short in terms of traffic they carry
    - A small fraction of connections carry a large portion of traffic ⇒ elephants
    - A large fraction of connections carry a small portion of traffic ⇒ mice

Part IV

TCP Congestion Control Approaches

§6.4.0 TCP Congestion Approaches

- End-to-End (E2E) congestion control
- Implicit feed-back ⇒ time-out, duplicate ACKs, etc.
- Uses window based flow control



1. Advertize Windo Window(W_A)
   - Advertized by the reciever in the hearder of TCP segment
2. Flow Window(W_F) ⇒ Min $\{Pipe_{size}, W_A\}$
   - Determined by the source
3. Congestion Window ($W_C \leq W_F$)
   - Number of segments which won't congestion
4. Effective Window ($W_E \leq W_C$)
   - Number of segments actually sent beased on all information

- Many variations of TCP
  - TCP Tahoe, TCP Reno, TCP Vegas, etc.
  - TCP uses Slow start + CA

How?

§6.4.1 How TCP Handles Congestion?

- A TCP host inject a segment(s) into the networks without reservation
- It reacts (adjust rate) by observing events
    - Time-out
    - Rate of ACKs arrivals
    - Duplicate ACKs
    - Triple duplicate ACKs ⇒ 4 ACKs
- It determines how much capacity is available
- Rate of ACKs arrival determine how much a source can transmit
    - TCP is self-clocking *How?*
        - Sender is clocked (controlled) by the receiver through ACK rate

§6.4.2 Additive Increase/Multiplicative Decrease (AIMD) I

- Additive Increase, Multiplicative Decrease (AIMD) is a feedback mechanism used in TCP
- AIMD combines linear growth of the congestion window with an exponential reduction when a congestion takes place
- Multiple flows using AIMD congestion control will eventually converge to use equal amounts of a contended link
- Let $W_C(t)$ be the sending rate during time slot t, then

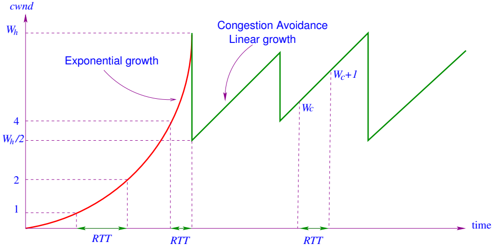
$$W_C(t+1) = \begin{cases} W_C(t) + a & \text{If congestion is not detected} \\ W_C(t) \times d & \text{If congestion is detected} \end{cases}$$

$$a > 0, \quad 0 < d < 1 \qquad (1)$$

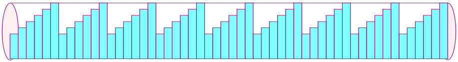
- It has been shown that AIMD is necessary for stability of TCP CC
- Accurate time-out is important to
  - Prevent unnecessary retransmission
  - Prevent under-utilization
- Time-out is a function of average and standard deviation of RTT

§6.4.3 Slow Start I

- Two phases
  1. Congestion detection ⇒ exponential growth of W_C
     - W_C grows exponentially ⇒ effectively $W_C = 2 \times W_C$ every RTT
  2. Congestion Avoidance ⇒ linear growth of W_C
     - W_C grows linearly $W_C = W_C + 1$

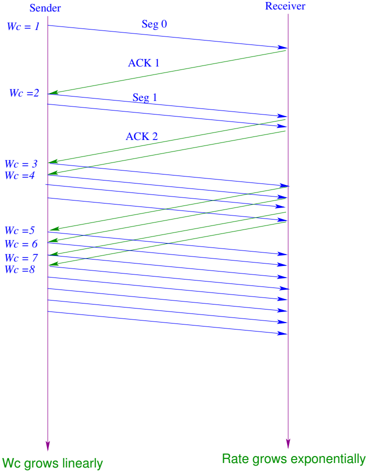


- Does it do the job?



- Cold Start (beginning of a connection), $W_C = 1$ ⇓
- Dead/broken connection results in advertised window goes to zero ⇓

§6.4.3 Slow Start II



§6.4.3 Slow Start III

Slow Start Algorithm

- Slow Start has two phases
  1. Exponential growth phase
     - Start with $W_C = 1$
     - For each ACK, $W_C := W_C + 1$ segment
       - ☆ It increases W_C by the number of segments acknowledged
       - ☆ W_C is effectively doubled per RTT
       - ☆ Note on cumulative ACKs
     - Until it reaches the threshold W_h, i.e., $W_C \geq W_h$ or an ACK is lost after a time out
       - ☆ Then $W_h = W_C$; $W_C = W_C/2$,
       - ☆ Enters Congestion Avoidance phase ⇒ linear growth ⇒ step 2
       - ☆ Problem: When a segment is lost TCP assumes that it is due to congestion
         - ➛ Not always true *Why?*
  2. Congestion avoidance phase (linear growth)
     - For each successful ACK $W_C := W_C + 1/W_C$ *How?*
     - For each RTT $W_C := W_C + 1$ Until a segment is lost
- Slow Start is quite aggressive than congestion avoidance phase
  - ▶ Could drive the network into saturation *How?*
  - ▶ However, slow start is slower than sending the full advertised window of segments

§6.4.3 Slow Start IV

- Current W_C can be used as congestion threshold W_h (ssthresh)
  ⇒ implicit information

Algorithm 1: Slow Start

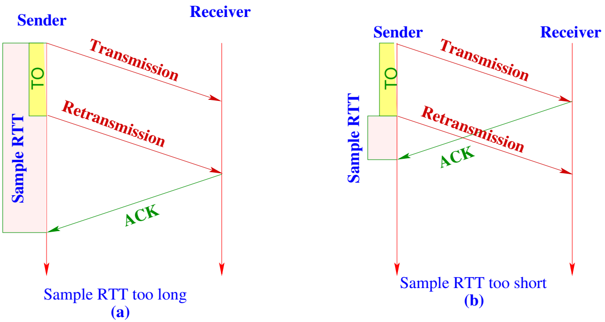
```
input  : $W_c = 1$, $W_h = \infty$
for every ACK do
    if $W_c < W_h$ then
        $W_c \leftarrow W_c + 1$;    // 1 max segment size (MSS) per ACK
                                      // Additive increase
    else
        $W_c \leftarrow W_c + 1/W_c$;          // Congestion avoidance
    end
end
for every time-out do
    $W_h \leftarrow W_c/2$;                     // Multiplication decrease
    $W_c \leftarrow 1$;                                   // Slow Start
end
```

§6.4.4 Adaptive Retransmission

- Measure `SampleRTT` for each segment/ACK pair
  - Needed to set the TO
- RTT is never accurate due to dynamic load
  - We estimate its average, but not arithmetic average ⇒ aged samples
- Compute weighted average of RTT
  - $EstimatedRTT = \alpha \times EstimatedRTT + \beta \times SampleRTT$
    - $\alpha + \beta = 1$
    - $0.8 \leq \alpha \leq 0.9$
    - $0.1 \leq \beta \leq 0.2$
- Set timeout based on `EstimatedRTT`
  - `TimeOut = 2 × EstimatedRTT`

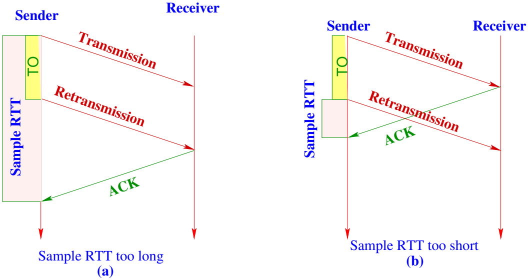
§6.4.5 RTT Flaw

- Does ACK really acknowledges a transmission?
- How many retransmissions had taken place before ACK arrived?



  - Wrong samples
  - Non-representative
- How do we fix this?

§6.4.6 Karn/Partridge Algorithm



- Which Sample?
  - In (a) sample should be for the second attempt *Why?*
  - In (b) sample should be for the first attempt *Why?*
- Karn/Partridge suggest
  - Do not sample RTT when retransmitting
  - Double TO after each retransmission
    - Similar to back-off algorithm
- Karn/Partridge algorithm was introduced (87) when the Internet was not suffering the current congestion
- Jacobson/Karels came up with a new calculation for average RTT

§6.4.7 Jacobson/Karels Algorithm

$$\begin{aligned} \mathit{Difference} &= \mathit{SampleRTT} - \mathit{EstimatedRTT} \\ \mathit{EstimatedRTT} &= \mathit{EstimatedRTT} + \delta \times \mathit{Difference} \\ \mathit{Deviation} &= \mathit{Deviation} + \delta \mid \mathit{Difference} - \mathit{Deviation} \mid \qquad 0 < \delta < 1 \end{aligned}$$

- Consider variance when setting timeout value

$$\mathit{TimeOut} = \mu \times \mathit{EstimatedRTT} + \phi \times \mathit{Deviation} \qquad \mu = 1 \quad \text{and} \quad \phi = 4$$

- Notes: accurate timeout is important in congestion control

§6.4.8 TCP Performance Metrics

1. Round trip delay
    - Access delay + Propagation Delay + Transmission Delay + Queuing Delay
    - One way delay ⇒ Asymmetric?
    - Maximum tolerable delay
2. Delay variation (or jitter)
3. Packet loss rate
4. Bandwidth
5. Throughput variation
    - Variability in the received bandwidth over time
6. File transfer time
7. Fairness
8. Resource consumption ⇒ CPU cycles, memory, battery, etc.

§6.4.9 TCP Limitations

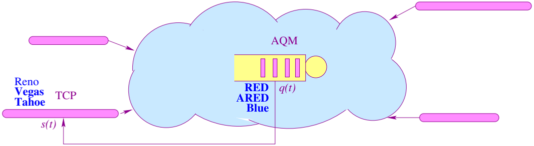
1. Limitation of existing transport protocols
   - Depends on the application domain
     - Some need reliability and ordered delivery
     - Some are sensitive to delay
2. TCP was designed with certain assumptions
   - Segment loss is due to congestion $\Rightarrow$ may not be true
   - Could be link/Cyclic Redundancy Check (CRC) error
3. TCP Performance depends on $delay \times bandwidth$
   - Could lead to inefficient link utilization
4. Standard TCP does not scale well in large $delay \times bandwidth$
   - Optical and grid applications

Part V

Packet Based Congestion Control

§6.5.0 CC Techniques

- So far, we controlled congestion with flow (W_F, W_C, W_A)
  - Not scalable
  - Does not distinguish between
    - Packet drop as a result of congestion
    - Packet drop as a result CRC error
- Other measures of network control performance
  - Delay, loss, and throughput are not enough
  - Must look at the robustness, complexity, etc.
- Packet level $\Rightarrow$ doesn't look into flows
  - Packet are too microscopic
  - Coarser grained models needed
  - Resurgence of fluid models
- Algorithms
  - Window Flow Control
    - TCP: Tahoe, Reno, Vegas
    - Receiver window flow control
    - Self Clocking , Vegas
  - Active Queue Management (AQM) Congestion Control
    - Random Early Detection (RED), Adaptive RED (ARED), etc.



§6.5.1 Internet Congestion Control

- Current CCs result in unsatisfactory performance
- Many CC algorithms have been proposed based on AQM
  - Executed by routers
- Network algorithm ⇒ What to feedback?
  - Implicit/Explicit congestion notification
- Source algorithms ⇒ How to react?
  - Adjusts data rate accordingly

§6.5.2 Four Basic Components of CC

1. Packet scheduler
2. Buffer management
3. Feedback ⇒ best possible explicit feedback
4. Source algorithms ⇒ end-system, FC

- Major problems with AQM
  - Parameter setting
  - The insensitivity to the input traffic load variation.
  - The mismatch between macroscopic and microscopic behavior of queue length dynamics

§6.5.3 Active Queue Management (AQM)

- AQM Provide CC information by marking probabilistically
- Issues
  - How to measure congestion?
  - How to embed congestion measures?
  - How to feed-back congestion information?
- Traditional voice model based on Poisson process fails
  - Aggregated traffic smooths out
- Internet traffic; File Transfer Protocol (FTP), Web, video are bursty and self-similar
- To cope with the bursty traffic, intelligent CC are based on
  - FIFO queue management
  - Per-flow queue management
  - Scheduling
- Traditional (tail drop) has two drawbacks
  - Lock-out ⇒ a few flows monopolize the queue
    - As a result of tail drop
  - Full queue may persist for a long time

Early drop may fix the problem ⇒ AQM

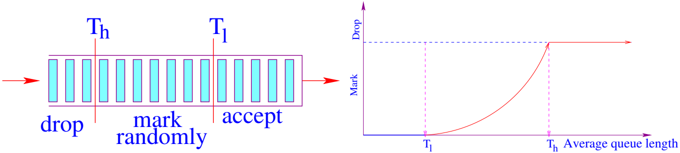
§6.5.4 Random Early Detection (RED) I

- TCP (Tahoe, Reno, Vegas) detect congestion after a buffer at an intermediate router is full
- RED provides a warning to sources ⇒
  - It detects incipient congestion
- Congestion measure ⇒ Average queue length

$$Q(t+1) = [Q(t) + \lambda(t) - \mu(t)]^{+}$$

- RED objectives
  - Congestion avoidance ⇒ proactive rather than reactive
    - Detect the onset of congestion, rather than react to it
    - Maintain the optimal region of high throughput and low delay
  - Avoid bias against bursty traffic
    - Probabilistic drop/mark
  - Preventing global synchronization or traffic oscillation
  - Bounded average queue length
  - Accommodating an equitable distribution of packet loss
    - Fairness
  - Providing a lower delay or a lower jitter (delay variation)
- p-linear probability function

§6.5.4 Random Early Detection (RED) II



- Uses Exponentially Weighted Moving Average (EWMA) *Why?*
- Probabilistically drop packets
- Probabilistically mark packets
  - Marking require Explicit Congestion Notification (ECN) bit (RFC 2481)

$$\overline{Q} = \begin{cases} (1 - W_q) \times \overline{Q} + W_q \times L_q & \text{if} \quad L_q \neq 0 \\ (1 - W_q)^m \times \overline{Q} & \text{otherwise,} \end{cases} \tag{2}$$

  - W_q determines how rapidly $\overline{Q}$ w.r.t L_q,
  - m accounts for the periods when the queue has been empty
- It uses two thresholds T_h and T_ℓ to control the growth of the queue

§6.5.4 Random Early Detection (RED) III

$$\begin{array}{lll} \text{If} & \overline{Q} < T_\ell & \text{accept the packet} \\ \text{If} & \overline{Q} \geq T_h & \text{mark/drop the packet} \\ \text{If} & T_\ell \leq \overline{Q} < T_h & \text{mark/drop probabilistically with, } P_a \end{array} \qquad (3)$$

- $\overline{Q}$ is calculated at the packet arrival times
  - Rather than at a fixed time interval *Why?*
- m estimates the number of packets that could have been transmitted during an idle period.
- Within the region $[T_\ell, T_h)$, the probability of packet discard depends on the proximity of $\overline{Q}$ to T_h and it is bounded by P_{max}
- When $T_\ell \leq \overline{Q} < T_h$, P_b increases linearly from zero to P_{max}

$$P_b = P_{max} \times \frac{(\overline{Q} - T_\ell)}{(T_h - T_\ell)} \qquad (4)$$

§6.6.0 References

[Tanenbaum and Wetherall, 2011] Tanenbaum, A. S. and Wetherall, D. J. (2011).
Computer Networks: 5th Edition.
Prentice Hall PTR.

§6.7.0 Suggested Exercises From the Text I

8 Why does the maximum packet lifetime, T, have to be large enough to ensure that not only the packet but also its acknowledgements have vanished?

15 Why does UDP exist? Would it not have been enough to just let user processes send raw IP packets?

22 What is the total size of the minimum TCP MTU, including TCP and IP overhead but not including data link layer overhead?

23 Datagram fragmentation and reassembly are handled by IP and are invisible to TCP. Does this mean that TCP does not have to worry about data arriving in the wrong order?

28 The maximum payload of a TCP segment is 65,495 bytes. Why was such a strange number chosen?

32 If the TCP round-trip time, RTT, is currently 30 msec and the following acknowledgements come in after 26, 32, and 24 msec, respectively, what is the new RTT estimate using the Jacobson algorithm? Use $\alpha = 0.9$.

33 A TCP machine is sending full windows of 65,535 bytes over a 1-Gbps channel that has a 10-msec one-way delay. What is the maximum throughput achievable? What is the line efficiency?

34 What is the fastest line speed at which a host can blast out 1500-byte TCP payloads with a 120-sec maximum packet lifetime without having the sequence numbers wrap around? Take TCP, IP, and Ethernet overhead into consideration. Assume that Ethernet frames may be sent continuously.

36 In a network whose max segment is 128 bytes, max segment lifetime is 30 sec, and has 8-bit sequence numbers, what is the maximum data rate per connection?

42 Calculate the bandwidth-delay product for the following networks: (1) T1 (1.5 Mbps), (2) Ethernet (10 Mbps), (3) T3 (45 Mbps), and (4) STS-3 (155 Mbps). Assume an RTT of 100 msec. Recall that a TCP header has 16 bits reserved for Window Size. What are its implications in light of your calculations?

43 What is the bandwidth-delay product for a 50-Mbps channel on a geostationary satellite? If the packets are all 1500 bytes (including overhead), how big should the window be in packets?