The Transport Layer

Ken Birman

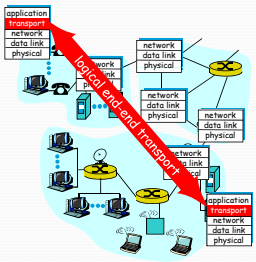
TCP: The network transport protocol

- Although Internet supports other protocols, TCP is by far the most commonly used
  - In fact many firewalls block everything except TCP
  - Some companies, like Amazon.com, have corporate policies that require all products they deploy to run over TCP – if you want to sell to them, and you run on UDP, you need to recode that part of the product
- In some ways the Internet has evolved to "talk to TCP" and wouldn't work if people didn't use TCP, or at least mostly use TCP

2

Purpose of this layer

- Interface end-to-end applications and protocols
  - Turn best-effort IP into a usable interface
- Data transfer b/w processes:
  - Compared to end-to-end IP
- We will look at two:
  - TCP
  - UDP (even if less common...)



3

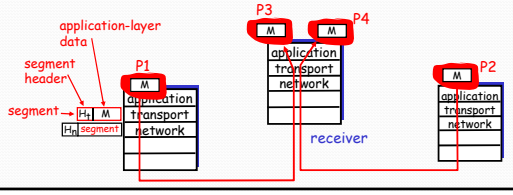
UDP

- **U**nreliable **D**atagram **P**rotocol
- Best effort data delivery between processes
  - No frills, bare bones transport protocol
  - Packet may be lost, out of order
- Connectionless protocol:
  - No handshaking between sender and receiver
  - Each UDP datagram handled independently

4

UDP Functionality

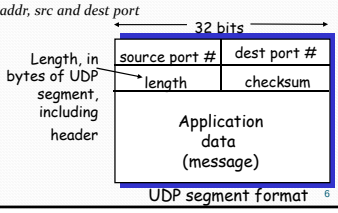
- Multiplexing/Demultiplexing
  - Using ports
- Checksums (optional)
  - Check for corruption



5

Multiplexing/Demultiplexing

- Multiplexing:
  - Gather data from multiple processes, envelope data with header
  - Header has src port, dest port for multiplexing
    - Why not process id?
- Demultiplexing:
  - Separate incoming data in machine to different applications
  - Demux based on *sender addr, src and dest port*



6

Implementing Ports

- As a message queue
  - Append incoming message to the end
  - Much like a mailbox file
- ***If queue full, message can be discarded silently***
- When application reads from socket
  - OS removes some bytes from the head of the queue
- If queue empty, application blocks waiting

7

UDP Checksum

- Over the headers and data
  - Ensures integrity end-to-end
  - 1's complement sum of segment contents
- UDP checksum is optional in UDP
- If checksum is non-zero, and receiver computes another value:
  - ***Silently drop the packet, no error message detected***

8

UDP Discussion

- Why UDP?
  - No delay in connection establishment
  - Simple: no connection state
  - Small header size
  - No congestion control: can blast packets
- Uses:
  - Streaming media, DNS, SNMP
  - Could add application specific error recovery
- Cons:
  - Casual about dropping packets
  - Often blocked by firewalls

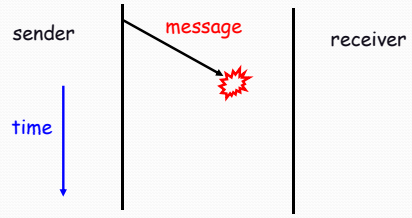
9

TCP

- **T**ransmission **C**ontrol **P**rotocol
  - Reliable, in-order, process-to-process, two-way byte stream
- Different from UDP
  - Connection-oriented, "punches hole in firewalls"
  - Error recovery: Packet loss, duplication, corruption, reordering
  - A number of applications require this guarantee
    - Web browsers use TCP
- Cons?
  - Awkward for voice applications because it has no notion of time-critical data delivery

10

Handling Packet Loss

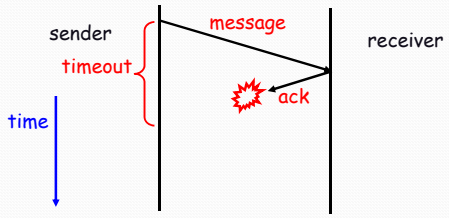


There are a number of reasons why the packet may get lost:
- router congestion, lossy medium, etc.

How does sender know of a successful packet send?

11

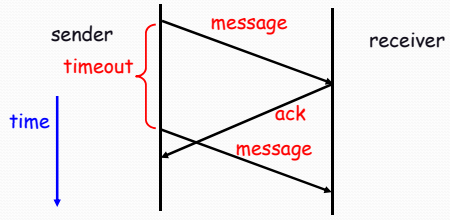
Lost Acks



What if packet/ack is lost?

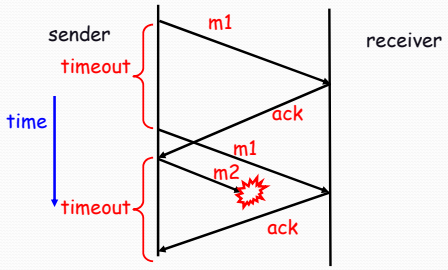
12

Delayed ACKs



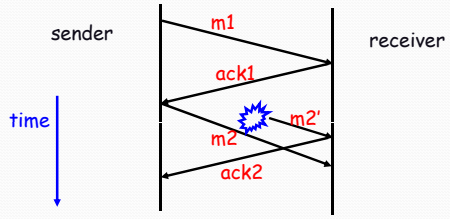
What will happen here? Due to congestion, small timeout, …
Delayed ACKs ⇒ duplicate packets

Delayed ACKs



How to solve this scenario?

Insertion of Packets

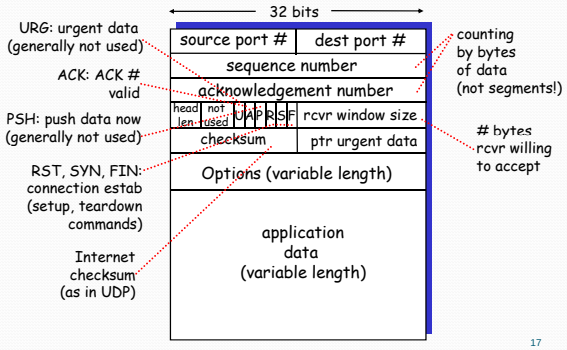


m2' could be from an old expired session!

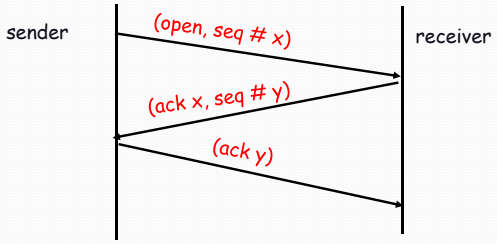
Message Identifiers

- Each message has <message id, session id>
  - Message id: uniquely identifies message in sender
  - Session id: unique across sessions
- Message ids detect duplication, reordering
- Session ids detect packet from old sessions
- TCP's sequence number has similar functionality:
  - Initial number chosen randomly
  - Unique across packets
  - Incremented by length of data bytes

TCP Packets

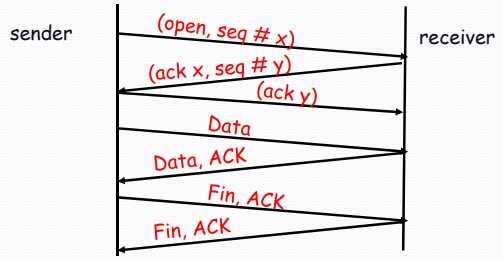


TCP Connection Establishment



TCP is connection-oriented. Starts with a 3-way handshake.
Protects against duplicate SYN packets.

TCP Usage



19

TCP timeouts

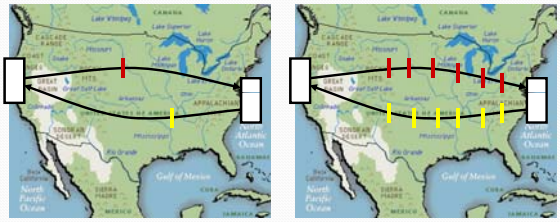
- What is a good timeout period ?
  - Want to improve throughput without unnecessary transmissions

NewAverageRTT = (1 - α) OldAverageRTT + α LatestRTT
NewAverageDev = (1 - α) OldAverageDev + α LatestDev
where LatestRTT = (ack_receive_time – send_time),
LatestDev = |LatestRTT – AverageRTT|,
α = 1/8, typically.
Timeout = AverageRTT + 4*AverageDev

- Timeout is thus a function of RTT and deviation

20

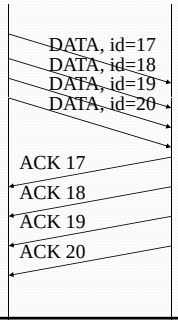
TCP Windows



- Multiple outstanding packets can increase throughput

21

TCP Windows



- Can have more than one packet in transit
- Especially over fat pipes, e.g. satellite connection
- Need to keep track of all packets within the window
- Need to adjust window size

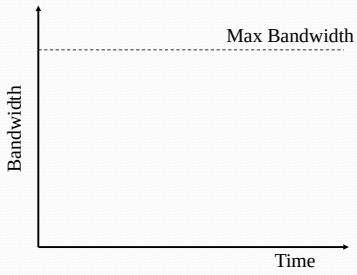
22

TCP Congestion Control

- TCP increases its window size when no packets dropped
- It halves the window size when a packet drop occurs
  - A packet drop is evident from the acknowledgements
- Therefore, it slowly builds to the max bandwidth, and hover around the max
  - It doesn't achieve the max possible though
  - Instead, it shares the bandwidth well with other TCP connections
- This linear-increase, exponential backoff in the face of congestion is termed *TCP-friendliness*

23

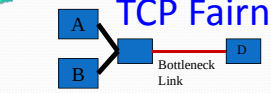
TCP Window Size

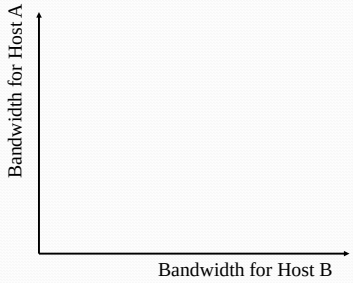


- Linear increase
- Exponential backoff
- Assuming no other losses in the network except those due to bandwidth

24

TCP Fairness





- Want to share the bottleneck link fairly between two flows

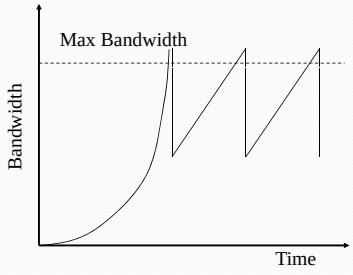
25

TCP Slow Start

- Linear increase takes a long time to build up a window size that matches the link bandwidth*delay
- Most file transactions are not long enough
- Consequently, TCP can spend a lot of time with small windows, never getting the chance to reach a sufficiently large window size
- Fix: Allow TCP to build up to a large window size initially by doubling the window size until first loss

26

TCP Slow Start

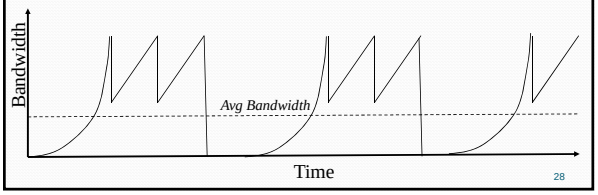


- Initial phase of exponential increase
- Assuming no other losses in the network except those due to bandwidth
- Backoff when max is exceeded

27

Thought question

- What would happen if an application is bursty and sends data with long delays between each send?
- Answer: it will repeatedly trigger TCP slow start, which can greatly reduce the average throughput



28

Cases where TCP breaks down

- Wireless networks
  - These often lose packets
  - But TCP "interprets" the loss to mean that congestion has occurred, even if the real issue is that the signal strength is poor in one or both directions
- Effect?
  - TCP slows down even though the smart thing to do would be to aggressively retransmit data but to run at full speed!

29

Cases where TCP breaks down

- Wide area networks
  - Can have very high speeds, like 40 Gbits/sec
  - And can have high latency, like 100ms
- Problem?
  - TCP runs out of buffer space: to run flat out needs 2*RTT*speed worth of space (= 1 Gigabyte in our example)
  - TCP recovers missing packets by asking the *sender* to retransmit… lost packet "stalls" connection for 1.5 RTTs (send… wait for nack…. resend… finally gets through)
  - Even a few lost packets can collapse throughput

30

Cases where TCP breaks down

- Voice over IP (VOIP)
  - Here the issue is that VOIP protocols really require steady 56kBit data rates with low delay
  - If data would be delayed longer than 100ms, better to drop the data than to delay *subsequent* data
- TCP weaknesses
  - Has no idea that it is carrying VOIP and can shut down to less than 56kBits rather easily
  - Won't drop data even if the receiver no longer wants that data because it was delayed for too long

31

Cases where TCP breaks down

- Prioritization
  - Suppose a network is carrying a mixture of more critical traffic, such as VOIP traffic, and less critical traffic, such as file downloads and email
  - If the network gets busy we would prefer to selectively drop data from the file downloads (anyhow, they probably account for more of the load)
  - But Internet tends to drop data indiscriminately
- Impact?
  - Lacking a sense of prioritization, unimportant traffic can interfere with critical traffic

32

So what should we do?

- Some enterprises are leasing *personal lambdas*
  - These are dedicated network connections that run at 10Gbit or 40Gbit ("lambda" == "optical wavelength)
  - Uncongested… my data routes directly to my dest
- Doing this reduces the risk that contention will cause TCP to misbehave.
  - Can also run large numbers of side-by-side TCP streams
  - And can carefully configure TCP with extra sender buffer space (but that space is in the kernel…)
  - An expensive way to work around the issue

33

TCP Summary

- Reliable ordered message delivery
  - Connection oriented, 3-way handshake
- Transmission window for better throughput
  - Timeouts based on link parameters
- Congestion control
  - Linear increase, exponential backoff
- Fast adaptation
  - Exponential increase in the initial phase
- But has a number of issues
  - Mostly relate to applications that are poor fits with the basic TCP model but are running on TCP anyhow

34

TCP variants

- These days there are many flavors of TCP
  - Some just work by letting the user change parameters
    - Can modify buffer (window) size
    - Can disable "slow start"
    - Can disable "keep alive" test
  - Others use non-standard flow control schemes
- Most applications stick to "out of the box" TCP and don't even modify parameters
  - Worries about making the application non-standard

35

TCP and End to End

- Thought question (to end on):
  - The Internet routers drop packets to "signal" congestion
  - TCP obediently backs off: "TCP friendly" behavior
- But *is this really consistent with end-to-end argument?*
  - Like a dialog between the network layer and TCP
  - Doesn't end-to-end tell us that the network layer shouldn't really care what runs over it?

36