

Transport Layer

Mike Freedman
COS 461: Computer Networks

<http://www.cs.princeton.edu/courses/archive/spr14/cos461/>

IP Protocol Stack: Key Abstractions



- Transport layer is where we “pay the piper”
 - Provide applications with good abstractions
 - Without support or feedback from the network

2

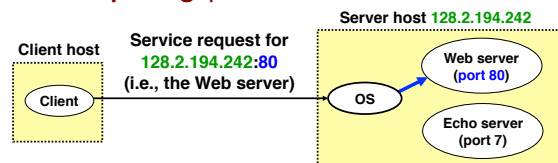
Transport Protocols

- Logical communication between processes
 - Sender divides a message into segments
 - Receiver reassembles segments into message
- Transport services
 - (De)multiplexing packets
 - Detecting corrupted data
 - Optionally: reliable delivery, flow control, ...

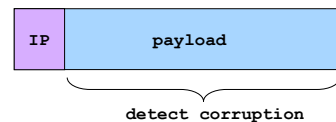
3

Two Basic Transport Features

- Demultiplexing: port numbers



- Error detection: checksums



4

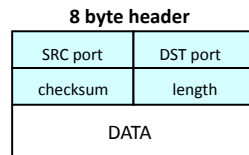
User Datagram Protocol (UDP)

- **Lightweight communication between processes**

- Send and receive messages
- Avoid overhead of ordered, reliable delivery
 - No connection setup delay, no in-kernel connection state

- **Used by popular apps**

- Query/response for DNS
- Real-time data in VoIP



5

Advantages of UDP

- **Fine-grain control**

- UDP sends as soon as the application writes

- **No connection set-up delay**

- UDP sends without establishing a connection

- **No connection state**

- No buffers, parameters, sequence #s, etc.

- **Small header overhead**

- UDP header is only eight-bytes long

6

Transmission Control Protocol (TCP)

- **Stream-of-bytes service**

- Sends and receives a stream of bytes

- **Reliable, in-order delivery**

- Corruption: checksums
- Detect loss/reordering: sequence numbers
- Reliable delivery: acknowledgments and retransmissions

- **Connection oriented**

- Explicit set-up and tear-down of TCP connection

- **Flow control**

- Prevent overflow of the receiver's buffer space

- **Congestion control**

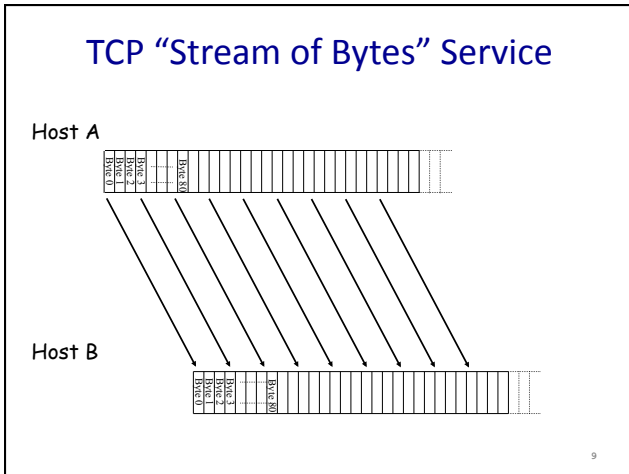
- Adapt to network congestion for the greater good

7

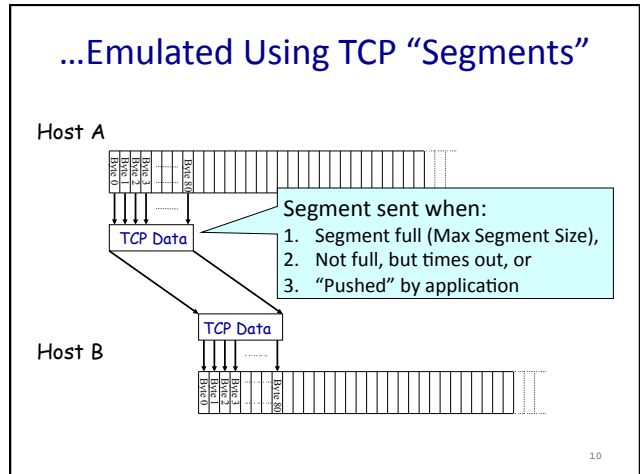
Breaking a Stream of Bytes into TCP Segments

8

TCP "Stream of Bytes" Service

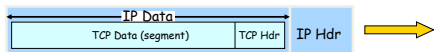


...Emulated Using TCP "Segments"

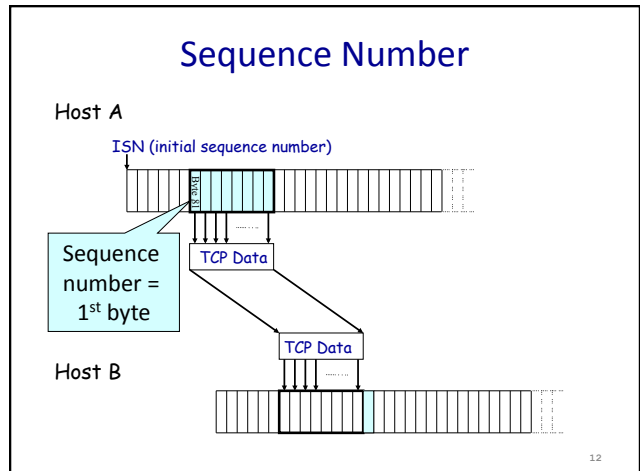


TCP Segment

- **IP packet**
 - No bigger than Maximum Transmission Unit (MTU)
 - E.g., up to 1500 bytes on an Ethernet link
- **TCP packet**
 - IP packet with a TCP header and data inside
 - TCP header is typically 20 bytes long
- **TCP segment**
 - No more than Maximum Segment Size (MSS) bytes
 - E.g., up to 1460 consecutive bytes from the stream



Sequence Number



Initial Sequence Number (ISN)

- Sequence number for the very first byte
 - E.g., Why not a de facto ISN of 0?
- Practical issue: reuse of port numbers
 - Port numbers must (eventually) get used again
 - ... and an old packet may still be in flight
 - ... and associated with the new connection
- So, TCP must change the ISN over time
 - Set from a 32-bit clock that ticks every 4 microsec
 - ... which wraps around once every 4.55 hours!

13

Reliable Delivery on a Lossy Channel With Bit Errors

14

Challenges of Reliable Data Transfer

- Over a perfectly reliable channel: Done
- Over a channel with bit errors
 - Receiver detects errors and requests retransmission
- Over a lossy channel with bit errors
 - Some data missing, others corrupted
 - Receiver cannot easily detect loss
- Over a channel that may reorder packets
 - Receiver cannot easily distinguish loss vs. out-of-order

15

An Analogy

- Alice and Bob are talking
 - What if Alice couldn't understand Bob?
 - Bob asks Alice to repeat what she said
- What if Bob hasn't heard Alice for a while?
 - Is Alice just being quiet? Has she lost reception?
 - How long should Bob just keep on talking?
 - Maybe Alice should periodically say "uh huh"
 - ... or Bob should ask "Can you hear me now?"



16

Take-Aways from the Example

- **Acknowledgments from receiver**
 - Positive: “okay” or “uh huh” or “ACK”
 - Negative: “please repeat that” or “NACK”
- **Retransmission by the sender**
 - After *not* receiving an “ACK”
 - After receiving a “NACK”
- **Timeout by the sender (“stop and wait”)**
 - Don’t wait forever without some acknowledgment

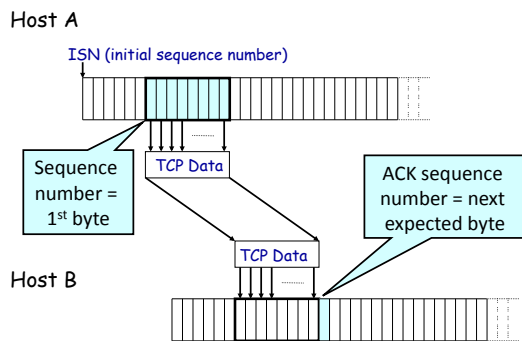
17

TCP Support for Reliable Delivery

- **Detect bit errors: checksum**
 - Used to detect corrupted data at the receiver
 - ...leading the receiver to drop the packet
- **Detect missing data: sequence number**
 - Used to detect a gap in the stream of bytes
 - ... and for putting the data back in order
- **Recover from lost data: retransmission**
 - Sender retransmits lost or corrupted data
 - Two main ways to detect lost packets

18

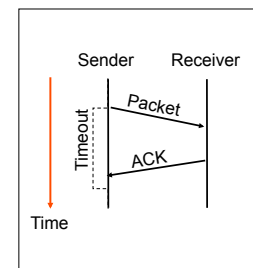
TCP Acknowledgments



19

Automatic Repeat reQuest (ARQ)

- **ACK and timeouts**
 - Receiver sends ACK when it receives packet
 - Sender waits for ACK and times out
- **Simplest ARQ protocol**
 - Stop and wait
 - Send a packet, stop and wait until ACK arrives



20

Quick TCP Math

- Initial Seq No = 501. Sender sends 4500 bytes successfully acknowledged. Next sequence number to send is:
(A) 5000 (B) 5001 (C) 5002
- Next 1000 byte TCP segment received. Receiver acknowledges with ACK number:
(A) 5001 (B) 6000 (C) 6001

21

Quick TCP Math

- Initial Seq No = 501. Sender sends 4500 bytes successfully acknowledged. Next sequence number to send is:
(A) 5000 (B) 5001 (C) 5002
- Next 1000 byte TCP segment received. Receiver acknowledges with ACK number:
(A) 5001 (B) 6000 (C) 6001

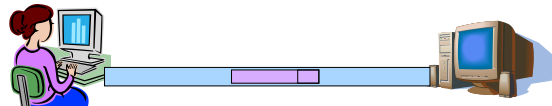
22

Flow Control: TCP Sliding Window

23

Motivation for Sliding Window

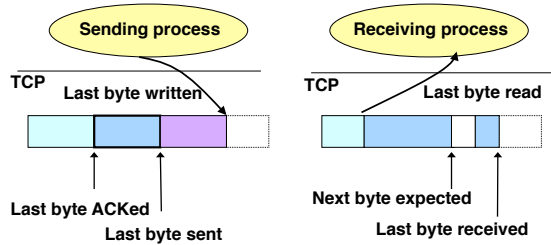
- Stop-and-wait is inefficient
 - Only one TCP segment is “in flight” at a time
- Consider: 1.5 Mbps link with 50 ms round-trip-time (RTT)
 - Assume TCP segment size of 1 KB (8 Kbits)
 - 8 Kbits/segment at 50 msec/segment → 160 Kbps
 - That’s 11% of the capacity of 1.5 Mbps link



24

Sliding Window

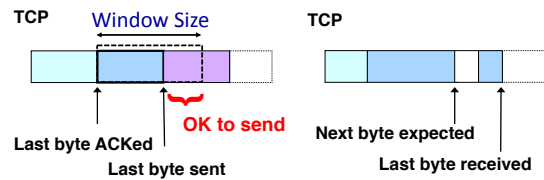
- Allow a larger amount of data “in flight”
 - Allow sender to get ahead of the receiver
 - ... though not too far ahead



25

Sliding Window

- Receive window size
 - Amount that can be sent without acknowledgment
 - Receiver must be able to store this amount of data
- Receiver tells the sender the window
 - Tells the sender the amount of free space left

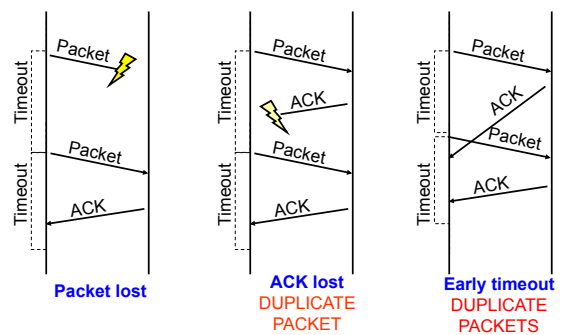


26

Optimizing Retransmissions

27

Reasons for Retransmission



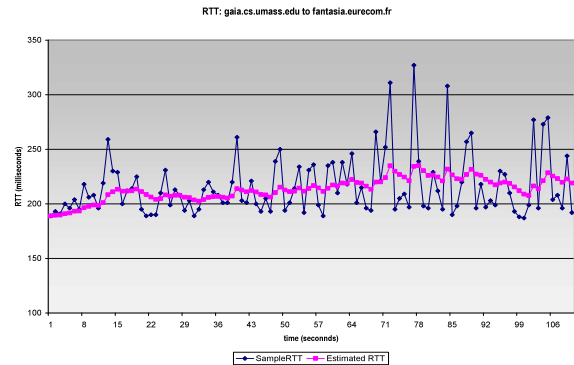
28

How Long Should Sender Wait?

- **Sender sets a timeout to wait for an ACK**
 - Too short: wasted retransmissions
 - Too long: excessive delays when packet lost
- **TCP sets timeout as a function of the RTT**
 - Expect ACK to arrive after a “round-trip time”
 - ... plus a fudge factor to account for queuing
- **But, how does the sender know the RTT?**
 - Running average of delay to receive an ACK

29

Example RTT Estimation



Still, timeouts are slow (\approx RTT)

- **When packet n is lost...**
 - ... packets $n+1$, $n+2$, and so on may get through
- **Exploit the ACKs of these packets**
 - ACK says receiver is still awaiting n th packet
 - Duplicate ACKs suggest later packets arrived
 - Sender uses “duplicate ACKs” as a hint
- **Fast retransmission**
 - Retransmit after “triple duplicate ACK”

31

Effectiveness of Fast Retransmit

- **When does Fast Retransmit work best?**
 - High likelihood of many packets in flight
 - Long data transfers, large window size, ...
- **Implications for Web traffic**
 - Most Web transfers are short (e.g., 10 packets)
 - So, often there aren't many packets in flight
 - Making fast retransmit is less likely to “kick in”
 - Forcing users to click “reload” more often...

32

Effectiveness of Fast Retransmit

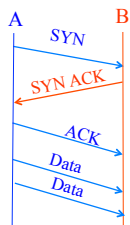
- When does Fast Retransmit work best?
 - (A) Short data transfers
 - (B) Large window size
 - (C) Small RTT networks

33

Starting and Ending a Connection: TCP Handshakes

34

Establishing a TCP Connection



Each host tells its ISN to the other host.

- Three-way handshake to establish connection
 - Host A sends a **SYN** (open) to the host B
 - Host B returns a SYN acknowledgment (**SYN ACK**)
 - Host A sends an **ACK** to acknowledge the SYN ACK

35

TCP Header

Flags: SYN
FIN
RST
PSH
URG
ACK

Source port		Destination port	
Sequence number			
Acknowledgment			
HdrLen	0	Flags	Advertised window
Checksum		Urgent pointer	
Options (variable)			
Data			

36

Step 1: A's Initial SYN Packet

Flags: SYN
FIN
RST
PSH
URG
ACK

A's port		B's port	
A's Initial Sequence Number			
Acknowledgment			
20	0	Flags	Advertised window
Checksum		Urgent pointer	
Options (variable)			

A tells B it wants to open a connection...

37

Step 2: B's SYN-ACK Packet

Flags: SYN
FIN
RST
PSH
URG
ACK

B's port		A's port	
B's Initial Sequence Number			
A's ISN plus 1			
20	0	Flags	Advertised window
Checksum		Urgent pointer	
Options (variable)			

**B tells A it accepts, and is ready to hear the next byte...
... upon receiving this packet, A can start sending data**

38

Step 3: A's ACK of the SYN-ACK

Flags: SYN
FIN
RST
PSH
URG
ACK

A's port		B's port	
Sequence number			
B's ISN plus 1			
20	0	Flags	Advertised window
Checksum		Urgent pointer	
Options (variable)			

**A tells B it is okay to start sending
... upon receiving this packet, B can start sending data**

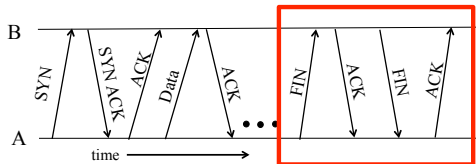
39

SYN Loss and Web Downloads

- Upon sending SYN, sender sets a timer
 - If SYN lost, timer expires before SYN-ACK received
 - Sender retransmits SYN
- How should the TCP sender set the timer?
 - No idea how far away the receiver is
 - Some TCPs use default of 3 or 6 seconds
- Implications for web download
 - User gets impatient and hits reload
 - ... Users aborts connection, initiates new socket
 - Essentially, forces a fast send of a new SYN!

40

Tearing Down the Connection



- **Closing (each end of) the connection**
 - Finish (FIN) to close and receive remaining bytes
 - And other host sends a FIN ACK to acknowledge
 - Reset (RST) to close and not receive remaining bytes

41

Sending/Receiving the FIN Packet

- **Sending a FIN: close()**
 - Process is done sending data via socket
 - Process invokes “close()”
 - Once TCP has sent all the outstanding bytes...
 - ... then TCP sends a FIN
- **Receiving a FIN: EOF**
 - Process is reading data from socket
 - Eventually, read call returns an EOF

42

Conclusions

- **Transport protocols**
 - Multiplexing and demultiplexing
 - Checksum-based error detection
 - Sequence numbers
 - Retransmission
 - Window-based flow control

43