The Transmission Control Protocol (TCP): Lecture 1

Internet Protocols

CSC / ECE 573

Fall, 2005

N. C. State University

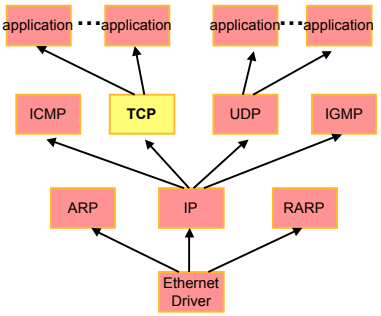
Today's Lecture

I. TCP overview

II. The TCP Header

III. Connection establishment and termination

What Layer is TCP?



TCP OVERVIEW

TCP (RFC 793, RFC 1122)

- TCP processes (sends and receives) data as an unstructured stream of bytes
  - *segment* = the unit of data transfer in TCP
  - transmitted in a single IP datagram
  - TCP divides data stream into segments (mostly) independently of application program writes

TCP (RFC 793, RFC 1122) (cont'd)

- TCP establishes an end-to-end *connection* (*flow*, *virtual circuit*, …)
  - the state of this connection is maintained by the endpoints, not by the network
  - pluses? minuses?
- The connection is full duplex (bi-directional)
  - concurrent transfer in both directions is possible
  - connection establishment sets up both directions
  - however, may terminate each direction individually

What TCP Provides

1. Addressing
   - delivery of data to correct application (by port #'s, same as UDP)
2. Reliable delivery
   - receiving application must get all the data sent by sending application, bit errors detected by checksum
3. In-order delivery
   - data must be delivered to the receiving application in the same order sent by the sending application

What TCP Provides (cont'd)

4. Flow control
   - sending transmission rate should not exceed the receiving application's maximum processing rate
5. Congestion control
   - sending transmission rate should not exceed the speed of slowest link on the path to the receiver
6. Segmentation
   - data should be sent in segments whose size provides the highest throughput

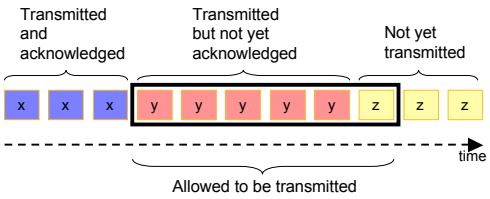
How is Reliable Transfer Accomplished?

1. Mandatory checksum on TCP header and data
   - single bit errors will be detected
2. Acknowledgments
   - so you know the data was received
3. Timeouts
   - if you didn't get acknowledgment, retransmit the data
4. Sliding windows
   - handles flow control and congestion control

TCP Sliding Windows

- Window size limits how much data can be sent without an acknowledgment
- Allows efficient transmission
  - sender and receiver don't have to operate in "lock-step"
  - (compare to go-back-N DLC automatic repeat request (ARQ) protocol)

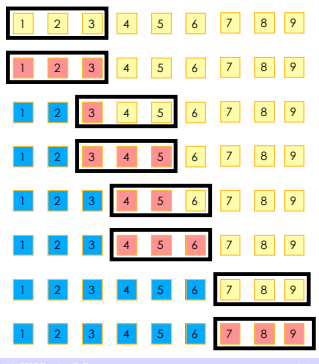
Flow Control: Segment Status



Sequence of TCP segments to transmit

Sliding Window Example (Fixed Window Size)

- Sender S and Receiver R negotiate a window size of 3; S has 9 segments ready to send as soon as possible
- S sends segments 1, 2, 3
- R acks 1 and 2
- S sends segments 4 and 5
- R acks 3
- S sends segment 6
- R acks 4, 5, 6
- S sends segments 7, 8, 9



TCP Ports and Endpoints

- A TCP port identifies the sending or receiving service
- An *endpoint* = <IP address, Port #>
  - source endpoint = <SrcIPAddr, SrcPort>
  - destination endponit = <DestIPAddr, DestPort>

Identifying TCP Connections

- The *5-tuple* identifies a single TCP *connection:*
  1. Source IP address
  2. Source Port #
  3. Destination IP address
  4. Destination Port #
  5. Protocol type (i.e., TCP)

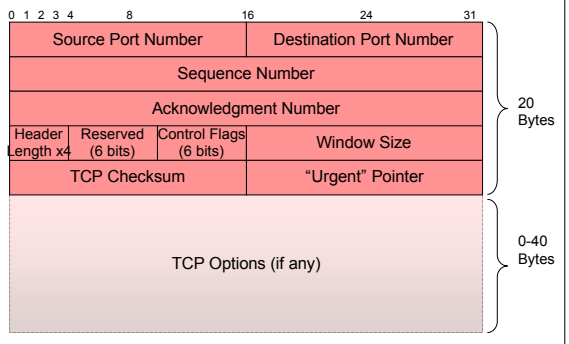
  (1, 2: Source endpoint; 3, 4: Destination endpoint)
- TCP connections from the same host may have the same destination IP address and port
  - why needed? how tell them apart?

Identifying TCP Connections (cont'd)

- For purposes of identifying a connection, doesn't matter which is the source, which is the destination
  - why not?

TCP HEADER

Header Format



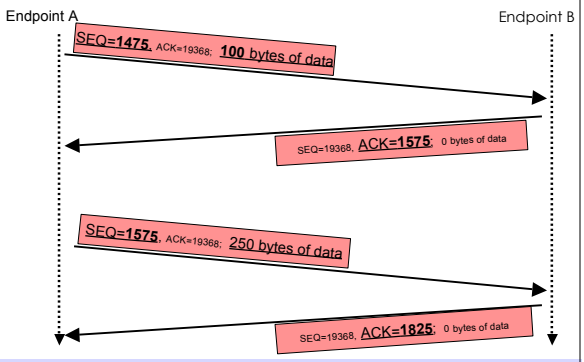
Sequence Numbers

- Sequence Number (from sender to receiver)
  - bytes being sent are numbered sequentially
  - Sequence Number of a segment = sequence number of the first data byte in this segment
- Sequence Number starts from a "randomly" selected value
- When Sequence Number exceeds 2^{32}-1, "wraps around" and starts over from 0

Acknowledgment Numbers

- ACK Number (from receiver to sender)
  - Sequence Number of the next data byte the receiver expects to receive
  - = Sequence Number of last byte actually received + 1
- ACK Number is valid only when ACK Flag = 1
  - if ACK Flag = 0, there is no acknowledgment in this segment

Example

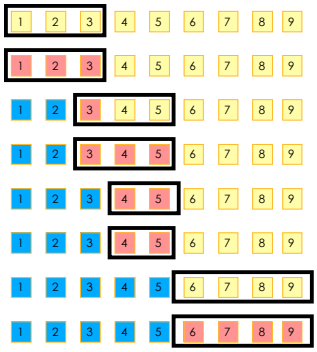


Window Size Field

- Window Size
  - number of bytes the receiver is willing to accept, starting with byte specified in ACK Number
  - 16 bits long; max value = 65535 (bytes)
- Example
  - for ACK Number = 5400, Window Size = 500, sender is allowed to transmit bytes 5400…5899 to the receiver
  - what if Sender has already sent bytes 5400..5649?
- Why do we need to vary the size of the window (i.e., why not just have a fixed size)?

Sliding Window Example (Variable Window Size)

- Sender S and Receiver R negotiate a window size of 3
- S sends segments 1, 2, 3
- R acks 1 and 2
- S sends segments 4 and 5
- R acks 3, advertises window size = 2
- S cannot send segment 6
- R acks 4, 5, advertises window size = 4
- S sends segments 6, 7, 8, 9



Other Fields

- Header Length x 4
  - length of header (in bytes) ÷ 4
  - Since field is 4 bits long, header length cannot exceed 60 bytes (15 x 4)
- Checksum
  - similar to UDP computation (including pseudo-header)
  - use is mandatory
- Can a TCP segment be longer than a single IP datagram???

Other Fields (cont'd)

- Urgent Pointer
  - index (offset) of last byte of *urgent data*
  - *(more later)*
- TCP Options
  - most common option is Maximum Segment Size
  - *(more on options later)*

Description of Control Flags

Flag	Description
URG	The value of the Urgent Pointer is valid
ACK	The value of the Acknowledgment Number is valid
PSH	Push the data, i.e., pass data in this segment to the receiver as quickly as possible (more later)
RST	The connection must be reset
SYN	Synchronize the sequence numbers during connection establishment
FIN	The sender has no more data to transmit

TCP Segments That Have No Data

- Possible?
- Why needed?

CONNECTION ESTABLISHMENT AND TERMINATION

Connection Establishment

- Remember: TCP is full duplex (simultaneous transmission in both directions)
- Example: A wants to converse with B (and they can't see each other or even tell if the other is present and unoccupied)
- Scenario 1
  - A says "How are you, B?" to B
  - 2-way connection established? Is it safe for A to start talking? Is it safe for B to start talking?

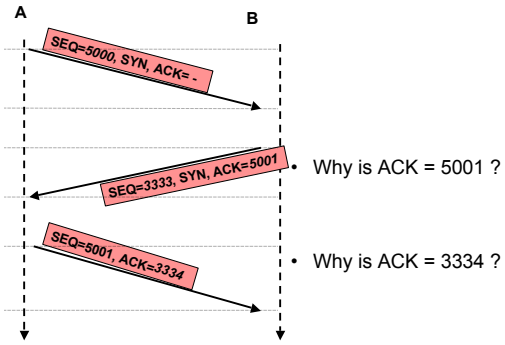
Connection Establishment (cont'd)

- Scenario 2
  - A says "How are you, B?" to B
  - B says "Good, A, how about yourself?" to A
  - 2-way connection established? Safe for A to start talking? Safe for B to start talking?

Connection Establishment (cont'd)

- Scenario 3
  - A sends "How are you, B?" to B
  - B sends "Good, A, how about yourself?" to A
  - A sends "Great, B! Listen, I want to talk about…" to B
  - 2-way connection established? Safe for A to start talking? Safe for B to start talking?
- The *TCP 3-way handshake*
  - A sends "SYN" to B
  - B sends "SYN+ACK" to A
  - A sends "ACK" to B

Three-Way Handshake



- Why is ACK = 5001 ?
- Why is ACK = 3334 ?

Connection Termination

- To avoid data loss, no side should completely disconnect until it is convinced that the other side is also prepared to disconnect
- Scenario 1
  - one side closes, doesn't wait for the other
  - A: "That's all for me, B, see you later"
  - has B stopped listening for something from A? is it safe for A to stop listening to B? Can B keep talking?

Connection Termination (cont'd)

- Scenario 2
  - A: "That's all for me, B, see you later"
  - B: "Right, A, don't forget to …"
  - has B stopped listening for something from A? is it safe for A to stop listening to B? Can B keep talking?
- Scenario 3
  - A: "That's all for me, B, see you later"
  - B: "Right, A, that's all for me, see you later"
  - has B stopped listening for something from A? is it safe for A to stop listening to B? Can B keep talking?

TCP Connection Termination

- Scenario 4
  - A: "That's all for me, B, see you later"
  - B: "Right, A, that's all for me, see you later"
  - A: "Right, B"
  - has B stopped listening for something from A? is it safe for A to stop listening to B? Can B keep talking?
- The TCP connection termination
  - A sends "FIN" to B (but B may still be sending data)
  - B sends "FIN+ACK" to A (B no longer sending)
  - A sends "ACK" to B
- What if the last ACK is lost?

Connection Reset

- A connection can also be aborted with a RST segment (hard reset)
  - reserved for error conditions, not normal termination
- "Non-graceful"
  - no waiting for the other side to acknowledge
  - all queued data is discarded

Problem of "Overlapping" Connections

- Delivery of a packet across an Internet may take a substantial amount of time
- Any problems?
  - A and B establish a connection
  - A asks B "Do I have more than $100 in account **X**?"
  - Answer="Yes", sequence # = **19347**
  - A and B close connection
  - 30 seconds later, A and B establish a new connection
  - A asks B "Do I have more than $100 in account **Y**"
  - Answer="Yes", sequence # = **19347**
  - A says "withdraw $100 from account **Y**"

Problem of "Overlapping" Connections

- Delivery of a packet across an Internet may take a substantial amount of time
- Any problems?
  - A and B establish a connection
  - A asks B "Do I have more than $100 in account **X**?"
  - Answer="Yes", sequence # = **19347**
  - A and B close connection
  - **.5** seconds later, A and B establish a new connection
  - A asks B "Do I have more than $100 in account **Y**"
  - Answer="**No**", sequence # = **19347**
  - **??**A says "withdraw $100 from account **Y**"**??!**

Solutions

- Choose a different Initial Sequence Number (*ISN*) with each connection establishment
- Bound the maximum packet lifetime
  - let *MSL* = max packet (segment) delivery time (in seconds, not the same as TTL)
- Use long enough Sequence Numbers
  - Time to "wrap around" or repeat should be > 2**MSL*
  - allows sufficient time for data to be ACKed before Sequence Number is used again
  - what determines "long enough"?

Solutions (cont'd)

- After connection termination: wait for time 2**MSL* before reusing the same "5-tuple"
  - allow time for old packets to be delivered or discarded

Summary

1. TCP provides reliable, in-order delivery
2. TCP provides flow and congestion control, based on sliding windows
3. TCP is bi-directional, and connection-oriented (state maintained by end hosts, not network)
4. Connection establishment requires a 3-way handshake
5. Connection termination requires FIN+ACK in both directions
   - can be done in each direction individually

Summary (cont'd)

6. Care must be taken to avoid overlapping connections problem

Next Lecture

- TCP, lecture 2