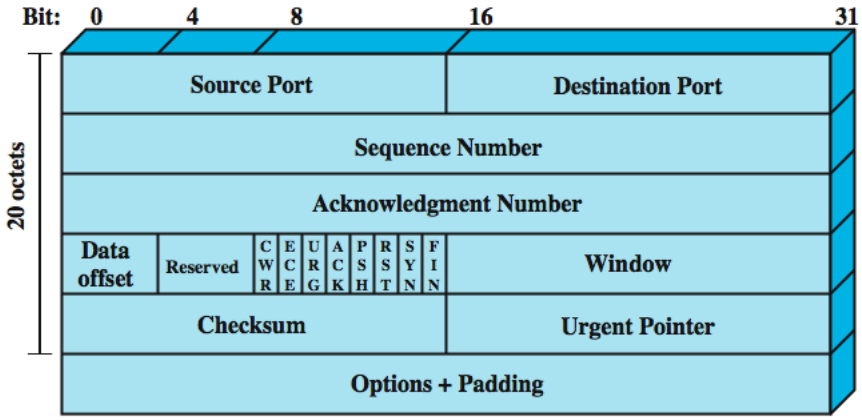
Transmission Control Protocol (TCP)

EECS 3214

TCP Services

- Transmission Control Protocol (RFC 793)
  - connection oriented, reliable communication
  - over reliable and unreliable (inter)networks
- If the underlying network is unreliable (IP)
  - segments may get lost
  - segments may arrive out of order
- Connection establishment and termination
- Flow control
- Reliable delivery
- Congestion control

TCP Header



3

Issues to Consider

- ordered delivery
- flow control
- connection establishment
- connection termination
- retransmission strategy
- duplication detection
- failure recovery

4

Ordered Delivery

- segments may arrive out of order.
- hence we number segments sequentially.
- TCP numbers each octet sequentially.
- each segment is numbered by the first octet number in the segment.

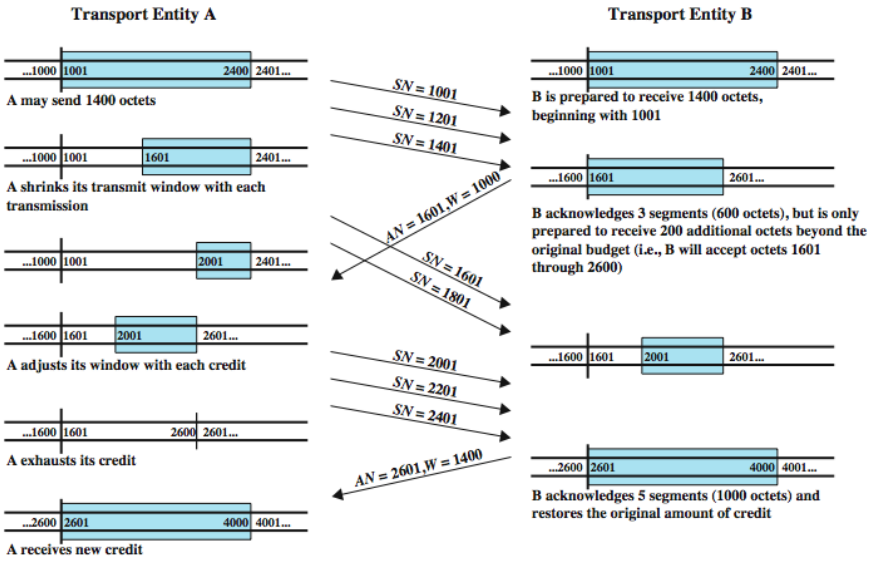
5

TCP Flow Control

- uses a credit scheme
- each octet has a sequence number
- each transport segment has in header
  - sequence number (SN)
  - ACK number (AN)
  - window size (W)
- sends sequence number of first octet in segment
- ACK includes (AN=*i*, W=*j*) which means
  - all octets through SN=*i*-1 acknowledged, want *i* next
  - permission to send additional window of W=*j* octets

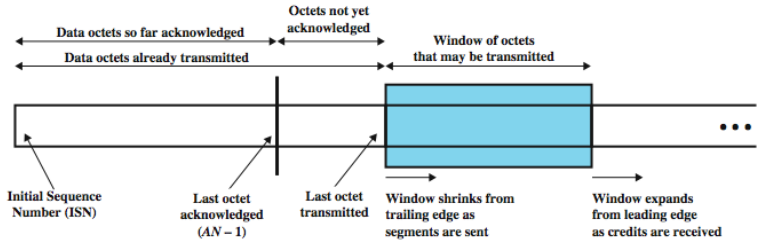
6

Credit Allocation

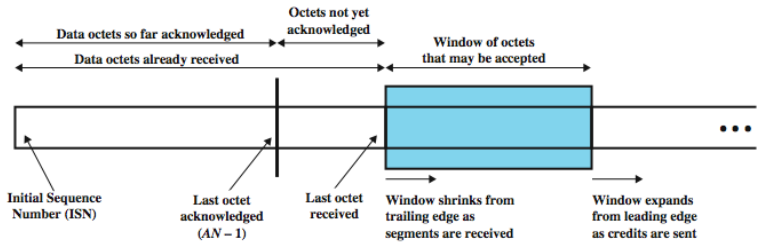


7

Sending and Receiving Perspectives



(a) Send sequence space



(b) Receive sequence space

8

Issues to Consider

- ordered delivery
- flow control
- connection establishment
- connection termination
- retransmission strategy
- duplication detection
- failure recovery

9

Connection Establishment and Termination

- required by connection-oriented transport protocols like TCP
- need connection establishment and termination procedures to
  - allow each end to know the other exists
  - allow negotiation of optional parameters
  - trigger allocation of transport entity resources

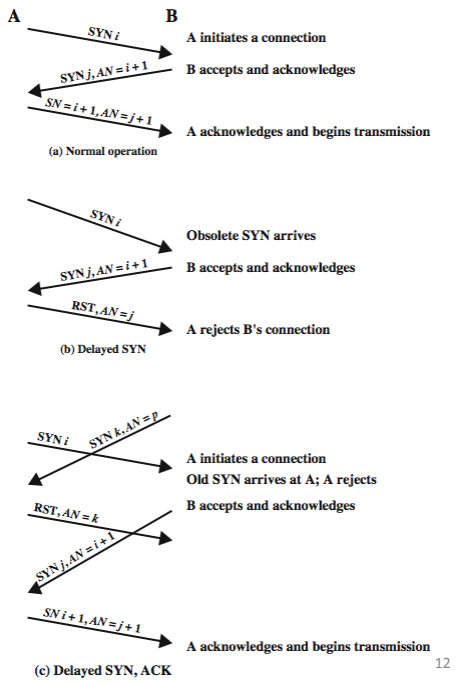
10

TCP Connection Establishment

- three way handshake
  - SYN, SYN-ACK, ACK
- connection determined by source and destination sockets
- can only have a single connection between any unique pairs of ports
- but one port can connect to multiple ports

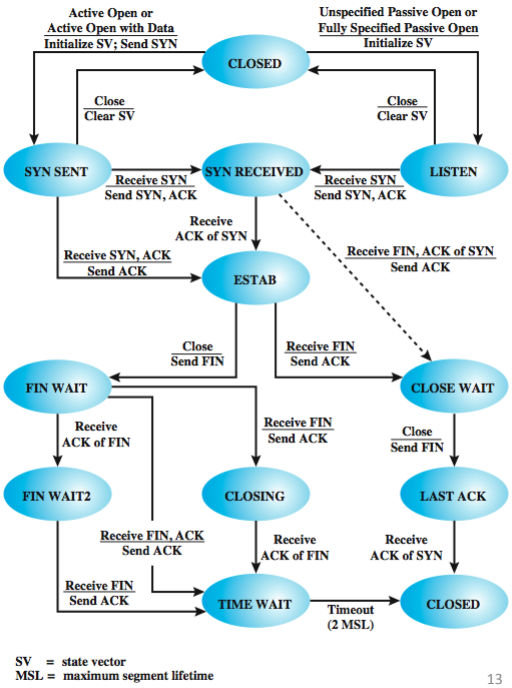
11

TCP Three Way Handshake: Examples



12

TCP Three Way Handshake: State Diagram



13

Issues to Consider

- ordered delivery
- flow control
- connection establishment
- connection termination
- retransmission strategy
- duplication detection
- failure recovery

14

Connection Termination

- also needs 3-way handshake
- misordered segments could cause the following:
  - entity in CLOSE WAIT state sends the last data segment, followed by a FIN
  - but the FIN arrives before the last data segment
  - receiver accepts FIN, closes connection, loses data
- need to associate sequence number with FIN
- receiver waits for all segments before FIN sequence number

15

Connection Termination: Graceful Close

- also have problems with loss of segments and obsolete segments
- need graceful close which will:
  - send ***FIN i*** and receive ***AN i+1***
  - receive ***FIN j*** and send ***AN j+1***
  - wait twice maximum expected segment lifetime before "closed".

16

Issues to Consider

- ordered delivery
- flow control
- connection establishment
- connection termination
- retransmission strategy
- duplication detection
- failure recovery

17

Retransmission Strategy

- retransmission of segments needed because
  - segments may be damaged in transit
  - segments fail to arrive
- transmitter does not know of failure
- receiver must acknowledge successful receipt
  - can use cumulative acknowledgement for efficiency
- If a segment does not arrive successfully, no ACK will be issued ➔ retransmission.
- There is a timer associated with each segment sent.
- Timer expires before ACK arrives ➔ retransmit.

18

Accept Policy

- segments may arrive out of order
- accept in order
  - only accept segments in order
  - discard out of order segments
  - simple implementation, but burdens network
- accept in window
  - accept all segments within receive window
  - reduce transmissions
  - more complex implementation with buffering

19

Retransmit Policy

- TCP sender has a queue of segments transmitted but not acknowledged
- Sender will retransmit if it does not receive an ACK within a given time
  - **first only** - single timer, send the front segment when timer expires; efficient for traffic, considerable delays
  - **batch** - single timer, send all segments when timer expires; has unnecessary retransmissions
  - **individual** - timer for each segment; lower delay, more efficient for traffic, but complex

20

Retransmit Policy (2)

- effectiveness depends in part on receiver's accept policy
  - accept in order: batch
  - accept in window: first-only or individual

21

Acknowledgement Policy

- immediate
  - send empty ACK (no data) for each accepted segment
  - simple, at cost of extra transmissions
- cumulative
  - piggyback ACK on suitable outbound data segments unless persist timer expires
  - if persist timer expires, send empty ACK
  - typically used in practice
  - more complex (processing, estimating RTT)
  - but fewer transmissions

22

Issues to Consider

- ordered delivery
- flow control
- connection establishment
- connection termination
- retransmission strategy
- duplication detection
- failure recovery

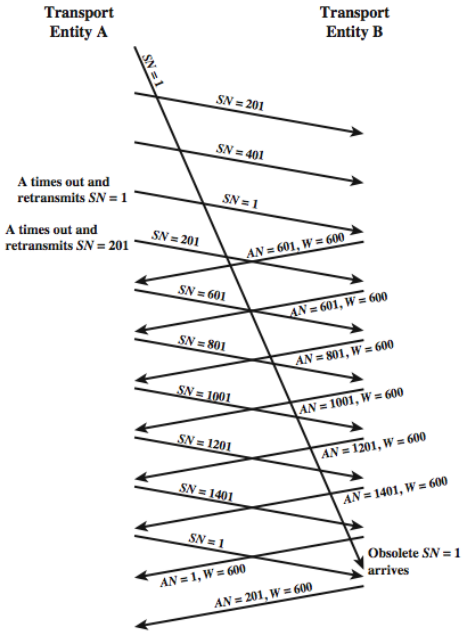
23

Duplication Detection

- if an ACK is lost, the segment is duplicated and re-transmitted
- receiver must recognize duplicates
- if duplicate received prior to closing connection
  - receiver assumes previous ACK lost and sends a new ACK for the duplicate
  - sender must not get confused by multiple ACKs
  - need a sequence number space large enough to not recycle sequence numbers within the maximum lifetime of a segment

24

Incorrect Duplicate Detection



25

Issues to Consider

- ordered delivery
- flow control
- connection establishment
- connection termination
- retransmission strategy
- duplication detection
- failure recovery

26

Failure Recovery

- The still active side can close the connection using a keep-alive timer.
  - Timer expires: closes the connection, and signals an abnormal close to the upper layer.
- The failed side restarts quickly:
  - The failed side returns an *RST i* to every segment *i* that it receives.
  - The other side performs an abnormal termination.

27

TCP Congestion Control

28

TCP Congestion Control

- flow control also used for congestion control
  - recognize increased transit times and dropped packets
  - react by reducing flow of data
- RFC 1122 and 2581 detail extensions
  - Tahoe, Reno and New Reno implementations
- two categories of extensions:
  - retransmission timer management
  - window management

29

Retransmission Timer Management

- static timer likely too long or too short
- hence estimate round trip delay by observing pattern of delay for recent segments
- set retransmission timer to a value a bit greater than estimated RTT: $RTO(k) = RTT(k) + \Delta$
- to estimate RTT:
  - simple average over a number of segments
  - exponential average using time series (RFC 793)
- assume no loss, the RTT of each segment is used in the following calculations

30

Computing RTT

- Simple average

$$r(K+1) = \frac{1}{K+1}\sum_{i=1}^{K+1} RTT(i)$$

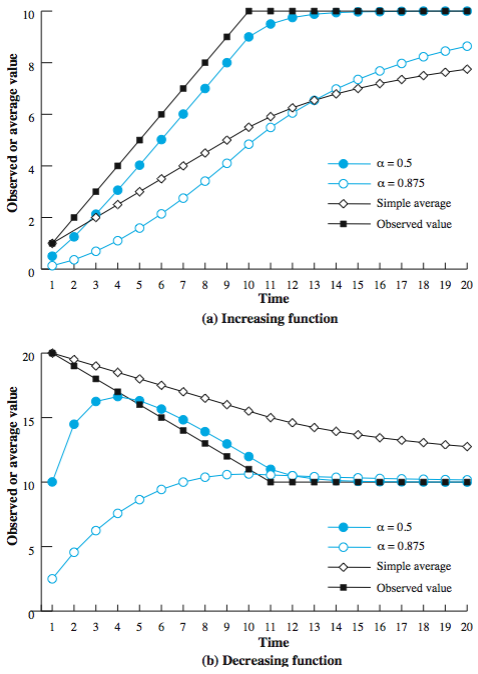
$$r(K+1) = \frac{K}{K+1}r(K) + \frac{1}{K+1}RTT(K+1)$$

- Exponential average

$$r(K+1) = a \times r(K) + (1-a) \times RTT(K+1)$$
$$0 < a < 1$$

31

Use of Exponential Averaging



Implementation of TCP Congestion Control Measures

Measure	RFC 1122	TCP Tahoe	TCP Reno	NewReno
RTT Variance Estimation	√	√	√	√
Exponential RTO Backoff	√	√	√	√
Karn's Algorithm	√	√	√	√
Slow Start	√	√	√	√
Dynamic Window Sizing on Congestion	√	√	√	√
Fast Retransmit		√	√	√
Fast Recovery			√	√
Modified Fast Recovery				√

33

RTT Variance Estimation

- Jacobson's algorithm
- To "smooth out" high variance in RTTs caused by
  - dynamic network traffic
  - receiver using cumulative acknowledgements
- Then calculates RTT and RTO

34

Implementation of TCP Congestion Control Measures

Measure	RFC 1122	TCP Tahoe	TCP Reno	NewReno
RTT Variance Estimation	√	√	√	√
Exponential RTO Backoff	√	√	√	√
Karn's Algorithm	√	√	√	√
Slow Start	√	√	√	√
Dynamic Window Sizing on Congestion	√	√	√	√
Fast Retransmit		√	√	√
Fast Recovery			√	√
Modified Fast Recovery				√

35

Why Karn's Algorithm?

- Scenario: a segment C1 times out and is retransmitted as C2.
- Sender receives an ACK: 2 possibilities
  - case1: to acknowledge C1
  - case 2: to acknowledge C2
- Sender cannot distinguish between the 2 cases.
- If case 2 is true, and $RTT = t_{C1} - t_{ACK}$ then RTT would be too long.
- If case 1 is true, and $RTT = t_{C2} - t_{ACK}$ then RTT would be too short.

36

Karn's Algorithm

- Do not use the measured RTT of a retransmitted segment to update $r(K+1)$.
- Calculate the backoff RTO using *binary exponential backoff* when a retransmission occurs.
- Use the backoff RTO value for succeeding segments until an ACK arrives for a segment that has not been retransmitted,
  - at which point use Jacobson's algorithm (averaging formula) to compute new *RTT* and future RTO values.

37

Exponential RTO Backoff

- timeout probably due to congestion
  - dropped packet or long round trip time
- hence maintaining same *retransmission timeout* (RTO) timer is not good idea
- better to increase RTO each time a segment is re-transmitted
  - RTO = $q \times$ RTO
  - commonly $q = 2$ (binary exponential backoff)
  - as in Ethernet CSMA/CD

38

Implementation of TCP Congestion Control Measures

Measure	RFC 1122	TCP Tahoe	TCP Reno	NewReno
RTT Variance Estimation	√	√	√	√
Exponential RTO Backoff	√	√	√	√
Karn's Algorithm	√	√	√	√
Slow Start	√	√	√	√
Dynamic Window Sizing on Congestion	√	√	√	√
Fast Retransmit		√	√	√
Fast Recovery			√	√
Modified Fast Recovery				√

39

Window Management: Slow Start

- `awnd = min( credit, cwnd )`
- larger windows allow connections to send more data
- at start, limit TCP to 1 segment: `cwnd` = 1
- assume no loss, for every ACK, slide window forward by 1 and also increase `cwnd` by 1, doubling window size in the next "round"
- exponential growth during "slow start"
- when does "slow start" stop?

40

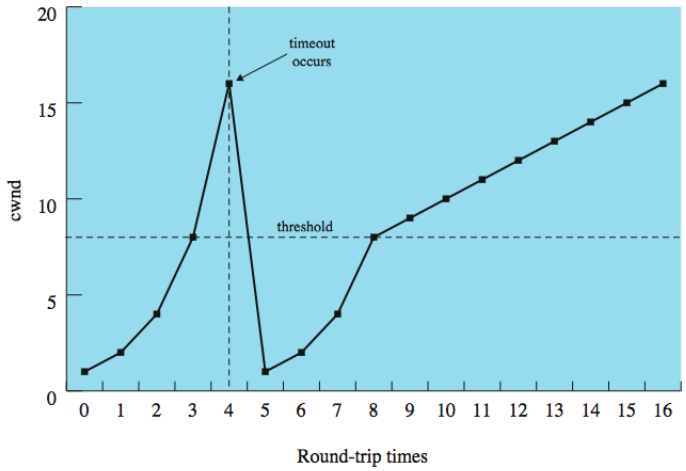
Dynamic Window Sizing on Congestion

- when a timeout occurs, assume congestion
- set slow start *threshold* to half current congestion window: `ssthresh` = cwnd / 2
- set window size to 1 and slow start until reaching threshold
- beyond threshold, increase window size by 1 for each RTT with no loss (congestion avoidance)

- *Note*: threshold can become quite small for successive packet losses.

41

Window Management



42

Implementation of TCP Congestion Control Measures

Measure	RFC 1122	TCP Tahoe	TCP Reno	NewReno
RTT Variance Estimation	√	√	√	√
Exponential RTO Backoff	√	√	√	√
Karn's Algorithm	√	√	√	√
Slow Start	√	√	√	√
Dynamic Window Sizing on Congestion	√	√	√	√
Fast Retransmit		√	√	√
Fast Recovery			√	√
Modified Fast Recovery				√

43

Fast Retransmit

- retransmit timer RTO rather longer than RTT
- if a segment is lost TCP slow to retransmit
  - accept in order: many segments may be lost
  - accept in window: first segment lost may cause buffer overflow at receiver
- fast retransmit
  - receiver: if receive a segment out of order, issue an ACK for the last in-order segment correctly received. Repeat this until the missing segment arrives.
  - sender: if receive 4 ACKs for the same segment then immediately retransmit (without waiting for timer to expire), since it is likely to have been lost,

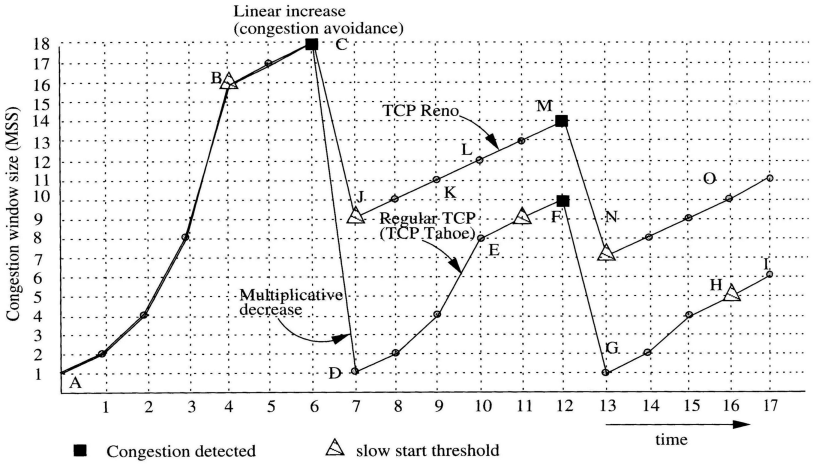
44

Fast Recovery

- Setting `cwnd` = 1 upon a loss is unnecessarily conservative (low throughput).
- Avoid slow start.
- Fast recovery:
  - retransmit the lost segment
  - cut `cwnd` in half
  - then increase `cwnd` linearly

45

Window Management Examples



46

When does “slow start” stop?

`awnd = min( credit, cwnd )`

- when reaching `credit`
- when a time out occurs
- when reaching `ssthresh` (if one exists)
- when receiving triple duplicate ACKs

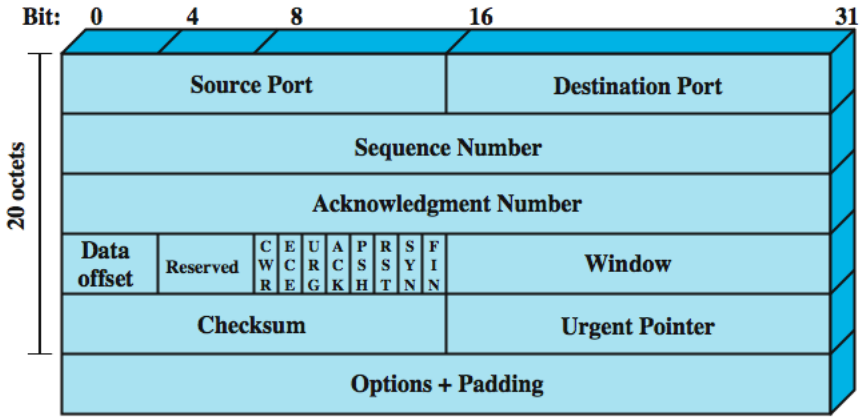
47

Transport Protocol Timers

Retransmission timer	Retransmit an unacknowledged segment
2MSL (maximum segment lifetime) timer	Minimum time between closing one connection and opening another with the same destination address
Persist timer	Maximum time between ACK/CREDIT segments
Retransmit-SYN timer	Time between attempts to open a connection
Keepalive timer	Abort connection when no segments are received

48

TCP Header



49

Reading

- Chapter “Transport Protocols”, William Stallings’ book

50