TCP
Transport Layer vs. Network Layer

• Provide *logical communication* between app’ processes

• Transport protocols run in end systems

• Transport vs. network layer services:
  – network layer
    • data transfer between end systems
  – transport layer
    • data transfer between processes
    • relies on, enhances, network layer services
Transport Layer Services and Protocols

- Transport services
  - multiplexing/demultiplexing
  - flow control
  - reliable data transfer
  - congestion/rate control

- Transport protocols in the Internet
  - UDP
  - TCP
Internet Protocol (IP)

- Packets may be delivered out-of-order
- Packets may be lost
- Packets may be duplicated
Transmission Control Protocol (TCP)

- Reliable ordered delivery
- Implements congestion avoidance and control
- Reliability achieved by means of retransmissions if necessary
- End-to-end semantics
  - Acknowledgements sent to TCP sender confirm delivery of data received by TCP receiver
  - Ack for data sent only after data has reached receiver
TCP Basics

- Cumulative acknowledgements

- An acknowledgement ack’s all contiguously received data

- TCP assigns byte sequence numbers
- For simplicity, we will assign packet sequence numbers

- Also, we use slightly different syntax for acks than normal TCP syntax
  - In our notation, \textit{ack} $i$ acknowledges receipt of packets through packet $i$
Cumulative Acknowledgements

- A new cumulative acknowledgement is generated only on receipt of a **new in-sequence** packet.
Delayed Acknowledgements

• An ack is delayed until
  – another packet is received, or
  – delayed ack timer expires (200 ms typical)

• Reduces ack traffic

```
Delayed Acknowledgements

• An ack is delayed until
  – another packet is received, or
  – delayed ack timer expires (200 ms typical)

• Reduces ack traffic

New ack not produced on receipt of packet 36, but on receipt of 37
```
Duplicate Acknowledgements

- A **dupack** is generated whenever an **out-of-order** segment arrives at the receiver.

(Above example assumes *delayed acks*)
Duplicate Acknowledgements

• Duplicate acks are **not delayed**

• Duplicate acks may be generated when
  – a packet is **lost**, or
  – a packet is delivered **out-of-order (OOO)**
Number of dupacks depends on how much OOO a packet is
Window Based Flow Control

• Sliding window protocol
• Window size minimum of
  – receiver’s advertised window - determined by available buffer space at the receiver
  – congestion window - determined by the sender, based on feedback from the network
Window Based Flow Control

Sender’s window

Ack 5

Sender’s window
Ack Clock

• TCP window flow control is “self-clocking”

• New data sent when old data is ack’d

• Helps maintain “equilibrium”
Window Based Flow Control

• Congestion window size bounds the amount of data that can be sent per round-trip time

• Throughput $\leq \frac{W}{RTT}$
Ideal Window Size

• Ideal size = delay * bandwidth
  – delay-bandwidth product

• What if window size < delay*bw ?
  – Inefficiency (wasted bandwidth)

• What if > delay*bw ?
  – Queuing at intermediate routers
    • increased RTT due to queuing delays
  – Potentially, packet loss
How does TCP detect a packet loss?

- Retransmission timeout (RTO)
- Duplicate acknowledgements
Detecting Packet Loss Using Retransmission Timeout (RTO)

• At any time, TCP sender sets retransmission timer for only one packet

• If acknowledgement for the timed packet is not received before timer goes off, the packet is assumed to be lost

• RTO dynamically calculated
Retransmission Timeout (RTO) calculation

- **RTO** = mean + 4 mean deviation

  - Standard deviation $\sigma$: $\sigma^2 = \text{average of } (\text{sample} - \text{mean})^2$
  - Mean deviation $\delta = \text{average of } |\text{sample} - \text{mean}|$
  - Mean deviation is more conservative: $\delta \geq \sigma$

- Large variations in the RTT increase the deviation, leading to larger RTO
Timeout Granularity

- RTT is measured as a discrete variable, in multiples of a “tick”

- 1 tick = 500 ms in many implementations

- smaller tick sizes in more recent implementations (e.g., Solaris)

- RTO is at least 2 clock ticks
Exponential Backoff

• Double RTO on each timeout

Packet transmitted

Time-out occurs before ack received, packet retransmitted

T1

T2 = 2 * T1

Timeout interval doubled
Fast Retransmission

• Timeouts can take too long
  – how to initiate retransmission sooner?

• Fast retransmit
Detecting Packet Loss Using Dupacks

Fast Retransmit Mechanism

• Dupacks may be generated due to
  – packet loss, or
  – out-of-order packet delivery

• TCP sender assumes that a packet loss has occurred if it receives three **dupacks** consecutively

| 12 | 8 | 11 | 10 | 9 | 7 |

3 dupacks are also generated if a packet is delivered at least 3 places beyond its in-sequence location

Fast retransmit useful only if lower layers deliver packets “almost ordered” ---- otherwise, unnecessary fast retransmit
Congestion Avoidance and Control

Slow Start

• initially, congestion window size $cwnd = 1$ MSS (maximum segment size)
• slow start phase ends when window size reaches the slow-start threshold
• $cwnd$ grows **exponentially** with time during slow start
  – factor of 1.5 per RTT if every other packet ack’d
  – factor of 2 per RTT if every packet ack’d
  – Could be less if sender does not always have data to send
Congestion Avoidance

• On each new ack,
• $cwnd$ increases **linearly** with time during congestion avoidance
  – $1/2$ MSS per RTT if every other packet ack’d
  – 1 MSS per RTT if every packet ack’d
Example assumes that acls are not delayed
Congestion Control

• On detecting a packet loss, TCP sender assumes that network congestion has occurred

• On detecting packet loss, TCP sender drastically reduces the congestion window

• Reducing congestion window reduces amount of data that can be sent per RTT
  – throughput may decrease
Congestion Control -- Timeout

• On a timeout, the congestion window is reduced to the initial value of **1 MSS**

• The slow start threshold is set to half the window size before packet loss
  – more precisely,
    \[ ssthresh = \text{maximum of } \min(cwnd, \text{receiver’s advertised window})/2 \text{ and } 2 \text{ MSS} \]

• **Slow start** is initiated
After timeout

$cwnd = 20$

$ssthresh = 8$

$ssthresh = 10$
Congestion Control - Fast retransmit

• Fast retransmit occurs when multiple (>= 3) dupacks come back

• **Fast recovery** follows fast retransmit

• Different from timeout: slow start follows timeout
  – timeout occurs when no more packets are getting across
  – fast retransmit occurs when a packet is lost, but latter packets get through
  – ack clock is still there when fast retransmit occurs
  – no need to slow start
Fast Recovery

• $ssthresh = \min(cwnd, \text{receiver’s advertised window})/2$
  (at least 2 MSS)
• retransmit the missing segment (fast retransmit)
• $cwnd = ssthresh + \text{number of dupacks}$
• when a new ack comes: $cwnd = ssthresh$
  – enter congestion avoidance

Congestion window cut into half
After fast retransmit and fast recovery window size is reduced in half.
TCP Reno

- Slow-start
- Congestion avoidance
- Fast retransmit
- Fast recovery
Impact of transmission errors on TCP performance
Random Errors

• If number of errors is small, they may be corrected by an error correcting code
• Excessive bit errors result in a packet being discarded, possibly before it reaches the transport layer
Random Errors May Cause Fast Retransmit

Example assumes delayed ack - every other packet ack’d
Random Errors May Cause Fast Retransmit

Example assumes delayed ack - every other packet ack’d
Random Errors May Cause Fast Retransmit

Duplicate acks are not delayed
Random Errors May Cause Fast Retransmit
Random Errors May Cause Fast Retransmit

3 duplicate acks trigger fast retransmit at sender
Random Errors May Cause Fast Retransmit

• Fast retransmit results in
  – retransmission of lost packet
  – reduction in congestion window by half

• Reducing congestion window in response to errors is unnecessary

• Reduction in congestion window reduces the throughput
Sometimes Congestion Response May be Appropriate in Response to Errors

• On a CDMA channel, errors occur due to *interference from other user*, and due to *noise* [Karn99pilc]
  – Interference due to other users is an indication of congestion. If such interference causes transmission errors, it is appropriate to reduce congestion window
  – If noise causes errors, it is not appropriate to reduce window

• When a channel is in a bad state for a *long duration*, it might be better to let TCP backoff, so that it does not unnecessarily attempt retransmissions while the channel remains in the bad state [Padmanabhan99pilc]
Next

• We consider errors for which reducing congestion window is an inappropriate response
Burst Errors May Cause Timeouts

• If wireless link remains unavailable for extended duration, a window worth of data may be lost — driving through a tunnel
• Timeout results in slow start
• Slow start reduces congestion window to 1 MSS, reducing throughput
• Reduction in window in response to errors unnecessary
Random Errors May Also Cause Timeout

• Multiple packet losses in a window can result in timeout when using TCP-Reno
Impact of Transmission Errors

• TCP cannot distinguish between packet losses due to congestion and transmission errors
• Unnecessarily reduces congestion window
• Throughput suffers
TCP/Reno Throughput as a Function of Loss Rate

Given mean packet loss rate $p$, mean round-trip time $RTT$, packet size $S$

\[ \Delta W = \begin{cases} \frac{1}{W} & \text{if the packet is not lost} \\ -\frac{W}{2} & \text{if packet is lost} \end{cases} \]

mean of $\Delta W = (1 - p) \frac{1}{W} + p(-\frac{W}{2}) = 0$

mean of $W = \sqrt{\frac{2(1-p)}{p}} \approx \frac{1.4}{\sqrt{p}}$, when $p$ is small

throughput $\approx \frac{1.4S}{RTT \sqrt{p}}$, when $p$ is small
Example

\[ \text{throughput} \approx \frac{1.4S}{RTT \sqrt{p}} \]

- Assume a TCP flow with \( S = 1000 \) bits, \( RTT = 10 \) ms
- Assume the link bandwidth is 11 Mbps
- If there is only one flow, then the (congestion) loss rate to fully utilize the link

\[ 11Mbps \approx \frac{1.4S}{RTT \sqrt{p}} \]

\[ \Rightarrow p = (0.0127)^2 = 0.016\% \]

- Assume a wireless network with packet (corruption) loss rate of 4%,
  - then the maximum achievable rate using TCP/Reno is 700 Kbps
- If we increase \( RTT = 200 \) ms (satellite)
  - the rate is 35 Kbps
Discussion

• Does TCP perform really that bad in wireless?
How to solve the problem?

• Approaches to improve TCP performance
  – Classification
  – Discussion of selected approaches
Classification of Schemes to Improve Performance of TCP in Presence of Transmission Errors
Techniques to Improve TCP Performance in Presence of Errors: **Classification 1**

Classification based on nature of actions taken to improve performance

- **Hide error losses from the sender**
  - if sender is unaware of the packet losses due to errors, it will not reduce congestion window

- **Let sender know, or determine, cause of packet loss**
  - if sender knows that a packet loss is due to errors, it will not reduce congestion window
Techniques to Improve TCP Performance in Presence of Errors: Classification 2

Classification based on where modifications are needed

- At the sender node only
- At the receiver node only
- At intermediate node(s) only
- Combinations of the above