Midterms Summary

50.012 Networks, Elective 2019

Tey Siew Wen

06 Feb 2020

Week	Topic	Specific Sections
1 – 2	Internet Network, Application Layer	Lecture 1: Transport Layer, Lecture 2: HTTP, Lecture 3: WebAPI
3	Multimedia Networking and CDN	Lecture 5: Network Applications, Lecture 6: CDN and PNP
4 – 5	Transport Layer Principles Reliable Data Transfer and Congestion Control	Lecture 7: RDT, Lecture 8: RDT Pipelines, Lecture 9: TCP and RDT Principles, Lecture 10: Congestion Control
6	TCP	Lecture 11: TCP Wrapup

Resources for practice

Problem sets: http://gaia.cs.umass.edu/kurose_ross/interactive/index.php

Learning Checklist

Week 1 – 2: Internet Overview and Application Layer

Overview

- ☐ Internet Structure: ISPs
- ☐ The Internet Protocol Stack: APTNLP
- ☐ Network Performance: Throughput, Delay and Loss Probability
- ☐ 4 Sources of Packet Delay
- ☐ Space-Time Diagrams
- ☐ Circuit Switching vs. Packet Switching

Application Layer

- ☐ Application Architectures
- ☐ Process Communication
  - ☐ Addressing processes with *identifiers*
  - ☐ What are Sockets
- ☐ Internet Transport Protocol Services: TCP vs. UDP
  - ☐ Socket Programming: Establishment and Usage
  - ☐ Protocol definitions
- ☐ Electronic Mail Application Protocols
  - ☐ SMTP Interaction and DATA Format
- ☐ HTTP
  - ☐ Web API
    - ☐ URI, URL, URNs
    - ☐ Architectural Style: REST
  - ☐ Idempotence and safe methods

Week 3: Multimedia Networking

- ☐ Streaming Stored Media
  - ☐ Role of UDP
  - ☐ DASH: Dynamic Adaptive Streaming over HTTP
  - ☐ Interpreting plots for data transmission and playout rate
- ☐ VoIP (Voice-over-IP)
  - ☐ Adaptive Playout Delay: Average Estimate of Packet Delay, Average Deviation of Delay, Playout time for packets in talk spurts
  - ☐ Recovery From Packet Loss: Simple Forward Error Correction (FEC)
- ☐ Streaming Live Media
- ☐ Content Distribution Networks (CDN) and concerns of CDN Operators
  - ☐ CDN Server Placement
  - ☐ CDN Server Selection
  - ☐ CDN Content Routing
  - ☐ CDN Content Replication

Week 4: Transport Layer

- □ Multiplexing/Demultiplexing
- □ UDP
- □ Reliable Data Transfer
- □ Flow Control
- □ Congestion Control
- □ TCP Connection Management

1 Lecture 1: Transport Layer

Transport service requirements

- Data loss
- Throughput
- Time sensitivity

1.1 Protocols

1.1.1 TCP/UDP

TCP	UDP
Reliable transport	Unreliable data transfer between sending/receiving process
Flow control: sender won't overwhelm receiver	NIL
Congestion control: throttle sender when network overloaded	NIL
Connection-oriented: setup required between client/server	No need

For both TCP & UDP, there is no encryption of data. Hence we have SSL (Secure Sockets Layer)/ TLS (Transport Layer Security) for providing an encrypted TCP connection, ensuring data integrity and end-point authentication. Apps can use SSL/ TLS APIs to do so.

1.2 Client-server Architecture

Server is always on with permanent IP address. Hosted in data centers for scaling. Clients communicate with the server and may be intermittently (irregularly) connected with the server. Could have dynamic IP addresses. Clients usually do not communicate directly with each other.

1.3 Peer-to-peer Architecture

- Server is not always on
- Arbitrary End Systems directly communicate: Good for file sharing, overlay-routing
- Principles:
    - Fault-tolerant
    - Fate-sharing: It's okay to fail if it's your own mistake?
- Self Scalability
    - Peers request service from other peers and provide service in return
    - New peers bring new service capacity and new service demands
- Challenges:
    - Peers are intermittently connected and change IP addresses
    - Chunk Poisoning

1.4 Electronic Mail

3 major components:

1. User Agents
2. Mail Servers
    i. Mailbox: contain incoming messages for user
    ii. Message Queue: Messages to be sent (outgoing)
3. Simple Mail Transfer Protocol: SMTP
    - uses TCP for sending emails from client to server via port 25
    - has 3 phases of transfer: handshake, transfer, closure.
    - Requires message to be in 7-bit ASCII
    - Uses CRLF.CRLF for determining end of message

Compared to HTTP, SMTP is a push rather than a pull server. Both have ASCII command/res interaction, status codes.

User Agent → SMTP → Sender Mail Server → SMTP → Receiver's Mail Server → Mail Access Protocol → User Agent

Mail server: forward mail from sender to receiver mail server. If both the client and the receiver mail server are offline, the sender's mail server will keep retrying to send the mail until it works. Back then the mail servers are not so reliable to on all the time.

1.4.1 Mail Access Protocol

Mail Access Protocol is used for retrieval from server

- POP (Post Office Protocol)
- IMAP (Internet Mail Access Protocol)
- HTTPs

1.5 Processes

Definition of a Process: A program running within the host. It must have an identifier that includes IP address, port numbers associated with the process on host. e.g. HTTP: 80, Mail: 25

Types of Process Communication

- Inter-process communication: Dependent on OS
    - unless the process is on the application layer, then it is controlled by the app developer.
- Host-to-host communication: Exchange Messages
    - Messages are sent/received via sockets.

Types of Processes

- Client Process: Initiate Communication
- Server Process: Waits to be contacted

1.6 Message Segmentation

- Reducing end-to-end delay.
- More efficient recovery from bit error. Otherwise the whole message needs to be retransmitted.
- Huge message may block other smaller packets
- Header overhead linear to the no. of packets
- Cause new problems e.g. out-of-order arrival of packets

2 Lecture 2: HTTP

Each HTTP message designed to be *self-contained*:

- it bring as much detail as the server needs to serve that request
- server does not maintain state

Protocols that maintain state are complex

- Past History must be maintained
- If server/client crash, the state that is stored on either host will be inconsistent
- however doing so is likely to improve performance

2.1 HTTP Methods

Safe Methods e.g. GET, HEAD:

- enable caching and loading distribution
- does not modify resources on server

Idempotent Methods e.g. Multiple DELETE:

- *Definition*: An effort that can be applied multiple times without changing the result beyond the initial application.
  - Handle lost confirmations by re-sending
  - May modify resources on the server
  - Can be executed multiple times without changing outcome
- Counter e.g. Multiple POST

2.2 Proxy

Definition: An entity authorized to act on behalf of another e.g. an intermediatory server performing requests for us

- Serve as a single point access of control to enforce security protocols

Common traits of proxies:

- Single access of control
- Load Balancing: Distribute incoming requests to a cluster of servers, all provide the same kind of service

2.2.1 Forward Proxy

When a client makes a connection attempt to that file transfer server on the Internet, its requests usually have to *pass through the forward proxy first*, where a firewall will be behind it.

1. Depending on the forward proxy's settings, a request can be allowed or denied.
2. If allowed, then the request is forwarded to the firewall and then to the file transfer server.

3. From the point of view of the file transfer server, it is the proxy server that issued the request, not the client. So when the server responds, it addresses its response to the proxy.
4. When the forward proxy receives the response, it recognizes it as a response to the request that went through earlier. And so it in turn sends that response to the client that made the request.

Applications:

- Content Logging & Eavesdropping
- Accessing Services Anonymously

2.2.2 Reverse Proxies

The reverse proxy does the exact opposite of what a forward proxy does. It accepts requests from external clients on behalf of servers stationed behind it. The firewall is between the client and reverse proxy instead of being in between the forward proxy and the servers.

1. Depending on the reverse proxy's settings, a request can be allowed or denied.
2. From the perspective of the client, it is the reverse proxy that is providing file transfer services.

Applications:

- A/B testing, Multivariate testing
- Distribute load

3 Lecture 3: Web API

Application programming interface (API) specifies how 2 software components should interact.

Types of API Protocols

3.1 Simple Object Access Protocol (SOAP)

Simple Object Access Protocol (SOAP) is the specific protocol for XML-based data exchange, popular in enterprise M-M communication.

- However, specific protocols seem to add overhead in many cases.

Web service definition language (WSDL) is used to specify the available services to clients.

- Client-side functions to call API can be automatically generated
- Auto-completion for API calls

3.2 Representational State Transfer (REST)

Representational State Transfer (REST) is not a protocol, but rather an architectural style.

- Pros: Simple, scalable, general, high performance
- Cons: No-built-in ACID & Under-fetching/Over-fetching
  - ACID: Atomicity, Consistency, Isolation, Durability
  - Over-fetching: You might get more data than you need, but the end point is designed to give you that specific data.
  - Underfetching: You might get less data than you need, because there is no end point designed to give you the data you need from that server.

3.2.1 Resources in REST

- Types
  i. Collections
  ii. Instances
- Referenced in the HTTP header

Simple Static Settings	Dynamic Settings
Each resource corresponds to single file.	Server will interpret the URL as parameters, dynamically create content for provided parameters. Content at resource URL may not exist yet.

3.2.2 Running HTTP Requests with curl

At any point of time, if you want to understand more about the flags that you pass into curl to test your http requests, run `curl -help`. Examples of sending a get request and giving the `-v` flag to show more information on exchanged messages. For a patch request, you can run: `curl -H "Content-Type: application/json" -X PATCH -d '"title":"test"' http://jsonplaceholder.typicode.com/todos/199`

The result will be:

```
{
    "userId": 10,
    "id": 199,
    "title": "test",
    "completed": true
}
```

PUT vs POST

- PUT: Used to update an existing resource. Reply will be 200.
- POST: Used to create an element in a collection. Reply will be 201 with URL of created element.

3.3 Multipurpose Internet Mail Extensions (MIME)

Multipurpose Internet Mail Extensions (MIME) is an Internet standard that extends the format of email messages to support text in character sets other than ASCII, as well attachments of audio, video, images, and application programs. Message bodies may consist of multiple parts, and header information may be specified in non-ASCII character sets.

4 Lecture 5: Network Applications

4.1 Streaming stored video

- Use Redundancy within and between images to decrease # bits required to encode image
  - Spatial (within image)
  - Temporal (from one image to next)
- Encoding Rate
  - CBR (Constant Bit Rate): Fixed encoding rate
  - VBR (Variable Bit Rate): Changes as amount of spatial, temporal coding changes
- Challenges
  - Continuous playout constraint: Once client playout begins, playback must match original timing
    - Network Delay e.g. queue delay are variable → need client side buffer to match playout requirements
  - Client Interactivity: Allow pause, fast-forward, rewind and jump through video
  - Video packets may be lost, need to retransmit
    - Packets may be received slower than it is being sent, so some packets might be skipped.

4.1.1 Streaming multimedia: DASH

DASH stands for Dynamic, Adaptive Streaming over HTTP (DASH).

Other adaptive solutions: Apple's HTTP Live Streaming (HLS) solution, Adobe Systems HTTP Dynamic Streaming, Microsoft Smooth Streaming

- Server
  - encodes video file into multiple versions
  - each version stored, encoded at a different rate
  - manifest file: provide URLs for different versions
- Client
  - Handles most of the streaming logic:
    - When to request chunk
    - What encoding rate to request
    - Where to request

4.2 Voice over IP (VoIP)

VoIP end-end-delay requirement: <150 ms good, >400ms bad

- For delay jitter, we aim to minimize client playout delay
- Receiver attempts to playout each chunk exactly q msecs after chunk is generated
  - Large q: less packet loss
  - Small q: better interactive experience

4.2.1 Adaptive Playout Delay

Adaptively Estimate Packet Delay

Exponentially Weighted Moving Average (EWMA)

$$d_i = (1-\alpha)d_{i-1} + \alpha(r_i - t_i)$$

where:

- d_i is the delay estimate after i-th packet
- α is a small constat
- r_i is the time received, t_i is the time sent
- so $r_i - t_i$ is the measured delay of i-th packet

Average deviation of delay

$$v_i = (1-B)v_{i-1} + B|r_i - t_i - d_i|$$

d_i, v_i are calculated for every received packet, but used only at the start of talk spurt.

- 1st packet in talkspurt: $\text{playout} - \text{time}_i = t_i + d_i + Kv_i$
  - longer playback delay
- Remaining packets are played out periodically

5 Lecture 6: CDN and PNP

5.1 Content Distribution Networks (CDN)

Problem Background: Streaming Content to hundred of thousands of simultaneous users

Possible solutions:

1. Single mega server: doesn't scale
    - point of network congestion
    - long path to distant clients
    - single point of failure
    - multiple copies of vid sent over outgoing link
2. Store/serve multiple copies of videos at multiple geographically distributed sites (CDN)

CDN Operators stores copies of content at CDN Nodes.

5.1.1 CDN Server

Types of CDN

- Commercial CDN: e.g. Akamai, Cloudflare
- Content provider's own CDN: e.g. Google, netflix
- Telco CDN

Choice of Placement

- Push CDN servers deep into many access ISPs so that they are close to users
- Smaller no. of larger clusters in IXPs near access ISPs

Server Selection

Points of consideration:

- Geographically close
- Performance: Real-time measurement
- Load-balancing
- Cost: CDN may need to pay its provider ISP
- Fault-tolerance

Routing the Content

After selection, we still have a routing problem. We can route content access in 3 different ways:

1. DNS-based
2. Application Driven
    - Multiple connection setup, name lookups
3. Routing (anycast)-based

Content Replication

Mechanisms:

- Push: Use of off-peak bandwidth optimization
- Pull: More Adapative

e.g. Netflix

- has prepared content
- so push content to CDN during off-peak hours whose CDN servers pull & cache content by user demand.

e.g. Youtube

- people can upload content anytime
- so CDN servers pull content by user demand at all time

5.2 Peer-to-peer architecture

- Arbitrary end systems directly communicate as peers
- Peers are intermittently connected and may change IP addresses

e.g. VoIP (Skype), Multimedia Streaming (Kankan.com), File Distribution (BitTorrent)

5.2.1 File distribution time (lower bound)

- d_{min}: min client download rate
- $\frac{F}{d_{min}}$: min client download time
- $\frac{F}{u_s}$: server upload time for one copy of file

Client-server

- Server must sequentially upload N file copies (to N clients)
- Each client must download file copy

Time to distribute F to N clients:

$$D_{c-s} \geq \max\left(N\frac{F}{u_s}, \frac{F}{d_{min}}\right)$$

P2P

- Server must upload at least one copy
- Each client must download one file copy
- Clients as aggregate must download NF bits

Time to distribute F to N clients

$$D_{p2p} \geq \max\left(\frac{F}{u_s}, \frac{F}{d_{min}}, N\frac{F}{u_s + \sum u_i}\right)$$

Another simplification of the third term of the equation:

$$N\frac{F}{u_s + \sum u_i} = \frac{F}{u_s/N + u}$$

6 Lecture 7: Reliable Data Transport

6.1 Principles of Reliable Data Transport

1. Physical channels are never completely reliable
    - Wireless links subjected to interference
    - Transmission noise → lead to bit errors
    - Routers & Switches may drop packets due to buffer overflow
2. Reliable communication relies on **detection and retransmissions as necessary**.
    - Receiver: Acknowledges packets from the sender.
    - Sender: Transmits & retransmits based on information provided by the receiver. If the sender times out (ack/nack not received within a certain time-frame). An action will be triggered.
    - Packets: Contain sequence numbers

6.1.1 Model for Reliable Communication

- Provides send/receive methods for applications to transfer packets
- Fields in packet header is used to coordinate with peer

6.2 Basic RDT Protocols

- RDT 1.0: Assume reliable channel
- RDT 2.0: Considers that channel may be corrupted.
- RDT 2.1: RDT 2.0 + 1-bit `seq_num` for identifying packet loss.
- RDT 2.2: RDT 2.1 + Stop and wait.
- RDT 3.0: RDT 2.2 + timeouts

6.2.1 RDT 2.0 & 2.1

Assumptions

1. Channel may corrupt but never lose packets.
    - Receiver always get something whenever a packet is sent
    - Sender always receives an ack after sending a packet (Feedback)
2. Receiver can always detect if packet has been corrupted.
    - Checksum to detect bit errors
        - May be flawed as the checksum could still tally after modification to data

Communication Flow

- Sender: Transmits one packet at a time and waits for ack/nack
- Receiver: Sends ack when it receives packet correctly, and sends nack when it receives erroneous packet.

Challenges

- Corruption: The reality is that packets, acks, nacks can all get corrupted.
  - This means that the sender may not be able to tell which packet was received correctly or not.
  - If sender retransmits the packet, the receiver could deliver the same packet to the application twice. If it transmits a new one, the receiver can fail to deliver a packet to the application.
- Lost Packets
  - Receiver does not know that a packet was sent to it, so it won't send an ack response
  - Sender left waiting for ack, but ack never comes :(

Hence the 1-bit `seq_num` is introduced in RDT2.2, such that nack is not necessary anymore.

6.2.2 RDT 2.2

Stop and wait (Alternating-bit protocol)

- Sender sends one packet and waits for receiver response

Importance of Sequence Numbers (`seq_num`)

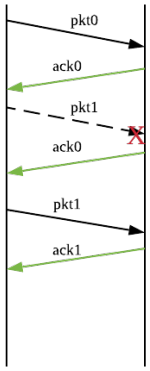
- `seq_num` in ack identifies received packet(s)
- Size of `seq_num`: Determines num of packets that can be sent before acknowledgements must be received

Sender	Receiver
Adds `seq_num` to packet	Must check if received packet's `seq_num` != previously stored `seq_num`
Must check if received ACK matches correct `seq_num`	Receiver cannot know if its last ACK received OK at sender

Scenarios

Case: Corrupted packet

- Receiver not receiving pkt1 as intended

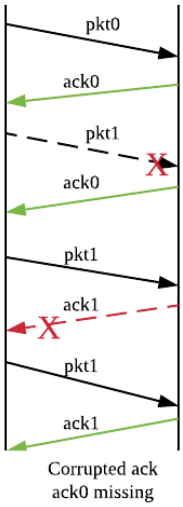


Case: Corrupted ack

- Sender not receiving ack0, so it retransmit pkt0

Case: Corrupted packet + Corrupted ack

- 1st packet and its ack not corrupted
- 2nd packet is corrupted so it is sent a 2nd time.
- During the 2nd time, the packet is not corrupted but the ack is corrupted



6.2.3 RDT 3.0

Previously RDT 2.2 did not address the issue of lost packets. There are a few ways of recovering from lost packets:

1. Keep sending packet repeatedly until sender gets an ack
    - Sender and receiver both will do extra work
2. Sender waits for ack for a specified time, then re-sends the packet if ack fails to arrive in time (timeout)
    - Works well if maximum ack delay is known and **does not change**. Otherwise will lead to premature timeout.

Features of RDT 3.0

RDT3.0 is the correct way of implemeting RDT but inefficient.

- Involves timeout before retransmission
- Sends only 1 packet at a time
- works well if delay between sender & receiver is small
- inefficient if $RTT \gg t_{pkt}$ where $t_{pkt} = \frac{L}{C}$

Example of ineffiency:
Consider a 1 Gb/s link with 10 ms delay in each direction, where RDT 3.0 sends only 1 packet every 20ms. The link is actually capable of sending 20 million bits in 20ms, so for typical packet sizes, only tiny fraction of link's capacity is used.

Calculating the Performance of RDT 3.0 Important variables

- $t_{pkt} = \frac{L}{C}$
- t_{out}: RTT
- C: link speed
- L: average packet size
- q: packet loss/corruption probability

If given p, the time to travel from sender to receiver and p', the time to travel from receiver to sender,

$$q = p + (1-p)p'$$

Expected time between successful transmissions (T_{succ})

$$\begin{aligned} T_{succ} &= \sum_{k=0}^{\infty}(k+1)(RTT + t_{pkt})(q^k)(1-q)(RTT + t_{pkt})\frac{1}{(1-q)^2} \\ &= \frac{RTT + t_{pkt}}{1-q} \end{aligned}$$

Throughput

$$\frac{L}{T_{succ}} = C(1-q)\left(1 + \frac{RTT}{t_{pkt}}\right)$$

Thoroughput improves if

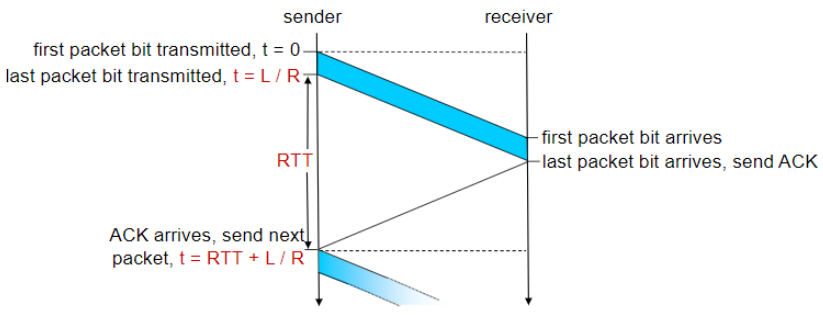
- corruption probabiliy/loss get smaller
- RTT gets smaller compared to t_{pkt}

Utilization

Fraction of time sender is busy sending

$$U = \frac{D}{RTT + D}$$

Space-time diagram



$$U_{sender} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

Requirements of Timeout

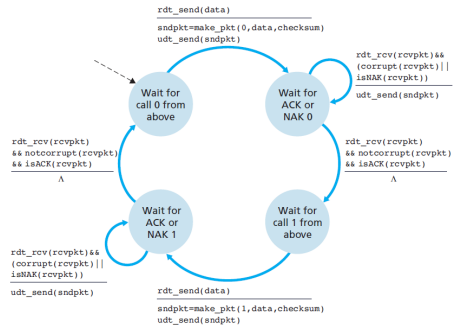
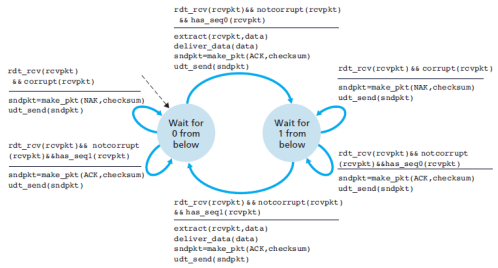
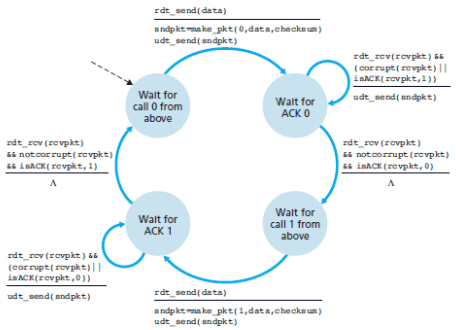
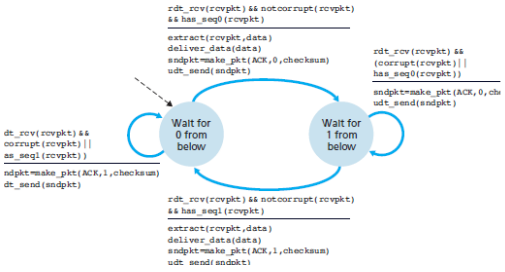
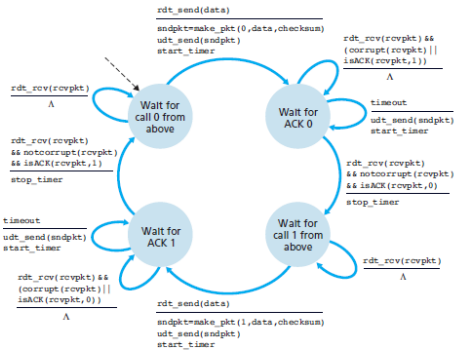
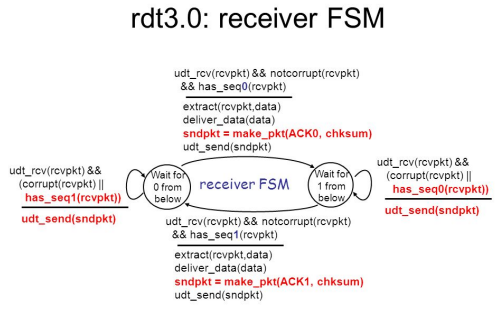
$$\text{Timeout} \geq RTT \text{ where } RTT = d_{nodal(data)} + d_{nodal(ack)}$$

Recall that $d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$.

Challenges

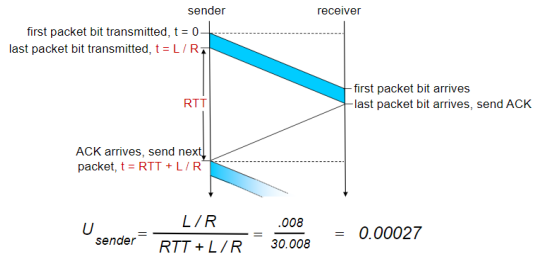
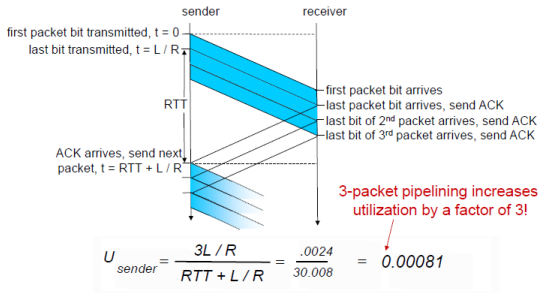
- Premature timeout: sender receives ack late, after it has retransmitted the packet that it did not receive ack for.
- Estimating RTT: Components are unknown and variable

6.2.4 Finite State Machine Diagrams

Protocol	Sender	Receiver
RDT 2.1	 Figure 3.11 ♦ rdt2.1 sender	 Figure 3.12 ♦ rdt2.1 receiver
RDT 2.2	 Figure 3.13 ♦ rdt2.2 sender	
RDT 3.0	 Figure 3.15 ♦ rdt3.0 sender	

7 Lecture 8: RDT Pipelines

Pipelining allows for increased utilization of the link.

Without pipeline	With pipeline
	

7.1 Consequences of pipelining

- The range of `seq_num` must be increased, since each in-transit packet must have a unique number and there may be multiple, in-transit, unack packets. (not counting retransmissions)
- Minimally, the sender will have to buffer packets that have been transmitted but not ack yet.

There are 2 basic approaches towards pipelined error recovery:

1. Go-Back-N (GBN): Interactive Animation
2. Selective Repeat (SR): Interactive Animation

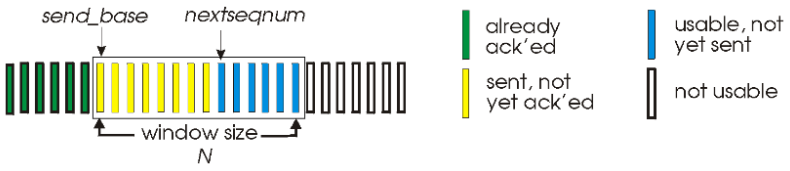
7.2 Types of Pipeline Protocols

7.2.1 Go-Back-N (GBN)

Sender

The sender is allowed to transmit multiple packets without waiting for an ack, where:

- `N`: the no. unack packets in the pipeline / the window size.
- `base`: the `seq_num` of the oldest unack packet.
- `nextseqnum`: smallest unused `seq_num`, the index of the next packet to be sent.
- `seq_num_space`: range of $[0, 2^k - 1]$
- `seq_num` with k number of bits for the packet sequence number field in the packet header.



This results in the following array of packets at the sender's side: The result will be:

```
data = seq_num_space
ack_pkts = data[0:base]
window = data[base:base+N]
sent_unack_pkts = data[base:nextseqnum]
avail_pkts_to_send = data[nextseqnum:base+N]
outside_window = data[base+N::]
```

As such, it is referred to the **sliding-window protocol**.

Receiver

The receiver always send ack for pkt with highest in-order `seq_num`. E.g. If the receiver receives packets 1,2,4,5 (pkt 3 is lost)

- Assuming window size is 5. (packets 1-5)
- It will keep resending the ack for pkt 2 and discard packets above 3 (4-5).
- Base is updated to 3, so after the timeout, the sender will resend packets 3-7.
- If these packets are successfully received, then the sender can update base to 8 and send 8-12th packets.

7.2.2 Selective Repeat (SR)

Comparison with GBN

Similarities	Differences
Fixed window size	-
Initialize timeout for packet at sender's side	Timeout initalized for each packet in SR, instead of one timeout for one packet in GBN
Sender allowed to transmit multiple packets without waiting for an ack	For out of order packets due to lost packets, they are buffered instead in SR instead of being discarded.

Limitations of small range of `seq_num`

- The receiver may intepret duplicate data as new data.
- The `seq_num_size` should be $2n$ to avoid this problem.

8 Lecture 9: TCP and RDT Principles

8.1 Transmission Control Protocol (TCP)

8.1.1 TCP Segment Structure

- Cumulative ACK: TCP sends an ACK with `seq_num` of next byte expected from the other side, instead of replying which packet it has received
    - Similar to Go-back-N
- Out-of-order packets: TCP will buffer, and not discard them.

8.1.2 Round-trip time (RTT) and Timeout

Estimating RTT

- $SampleRTT$: an average of recent measurements of time from segment transmission until ACK receipt
    - Ignore retransmissions

Exponential Moving Average Equation:

$$EstimatedRTT = (1-\alpha) \cdot EstimatedRTT + \alpha \cdot SampleRTT$$

- Influence of past samples decreases exponentially fast
- Typically $\alpha = 0.125$

Estimating SampleRTT

$$DevRTT = (1-\beta)DevRTT + \beta|SampleRTT - EstimatedRTT|$$

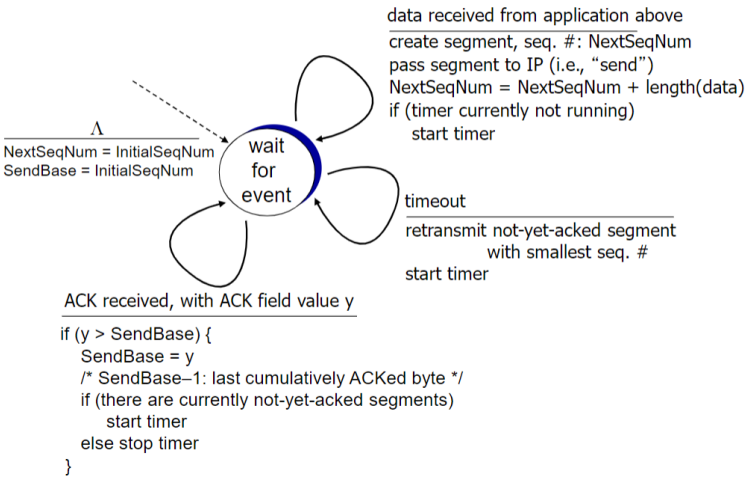
- Typically $\beta = 0.25$

Timeout Interval

$$TimeoutInterval = EstimatedRTT + 4 \times DevRTT$$

- $4 \times DevRTT$ is a safety margin
- Too short will result in premature timeout and unnecessary retransmissions
- Too long will result in slow reactions to segment loss

8.1.3 TCP RDT



Fast retransmit

- If sender receives 3 duplicate (extra) ACKs for same data, it will resend unACKed segment with smallest `seq_num`.

8.2 Other Reliable Data Transfer (RDT) Protocols

8.2.1 Go-Back-N (GBN)

- Requirement for k-bit `seq_num` in packet header: $2^k > N$
- Receiver window size: 1 (`expected_seq_num`).
- Relationship among `expected_seq_num`, `send_base`, `next_seq_num`:
  - `send_base` $\leq$ `next_seq_num` $\leq$ `expected_seq_num`

8.2.2 Selective Repeat (SR)

`rcv_base` = `next_seq_num` if all packets up to `next_seq_num` are received

- Requirement for k-bit `seq_num` in packet header : $2^k > 2N$
- Receiver window size: 1 (`expected_seq_num`).
- Relationship among `expected_seq_num`, `send_base`, `next_seq_num`:
  - `send_base` < `next_seq_num` < `expected_seq_num`

9 Lecture 10: Congestion Control

9.1 Principles of Flow Control

- Receiver controls sender so the sender won't overflow receiver's buffer by transmitting too much/-too fast
  - Application may remove data from TCP socket buffers
- Receiver includes a `rwnd` (receiver window) value in TCP header of receiver-to-sender segments
  - `RcvBuffer`

9.2 Principles of Congestion Control

- Congestion Control ≠ Flow Control!
- Mainfestations
  - Buffer Overflow at Routers: Lost Packets
  - Queueing in Router buffers: Long Delay

When packets are lost, any upstream transmission capacity used for that packet is wasted.

9.2.1 Scenario 1: One Router w/ Infinite Buffers

- Assuming no retransmission

9.2.2 One Router w/ Finite Buffers

Assumptions for idealized case

- Sender knows when router buffers available
- Sender sends only when router buffers available

Transfer rates

- $\lambda_{\text{in}} = \lambda_{\text{out}}$: Application-layer input = output
- $\lambda'_{\text{in}} \geq \lambda_{\text{in}}$: Transport-layer input includes retransmissions

9.2.3 TCP Congestion Controls

Increase sender's transmission rate until loss occurs

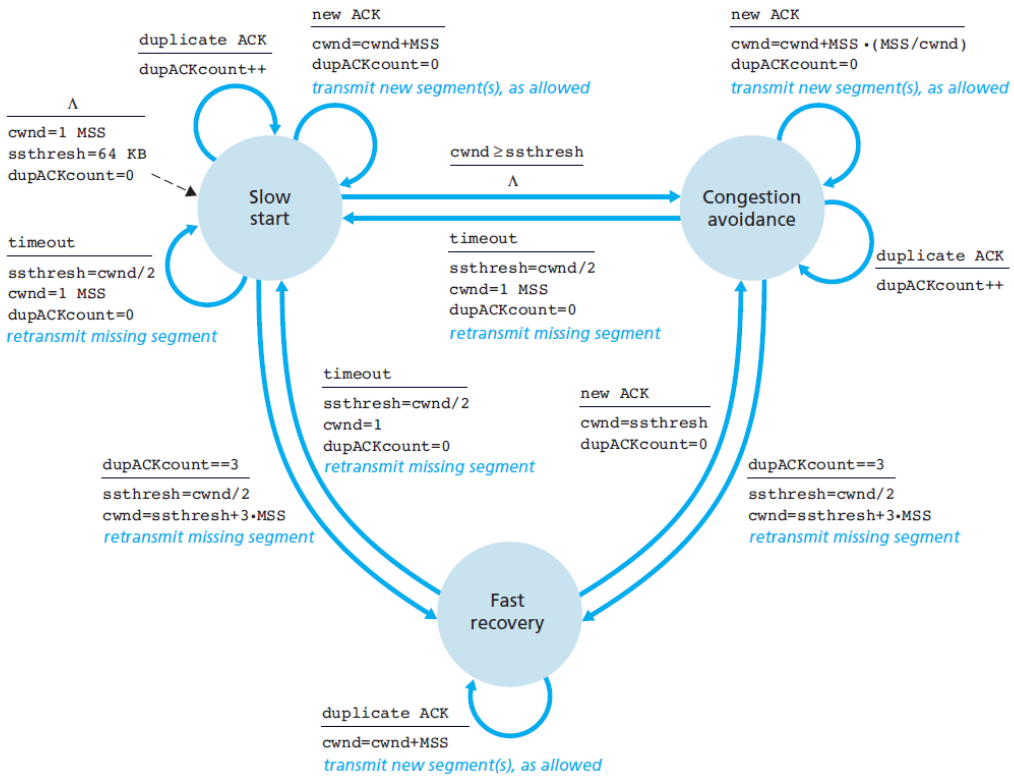
- Additive Increase: Increase cwnd (congestion window) by 1 MSS every RTT until loss detected
- Multiplicate Increase: Reduce cwnd in half after loss

10 Lecture 11: TCP Wrapup

10.1 TCP Congestion Control

A summary code for the sections discussed below:

```
cwnd = MSS
while connected:
if time < slow_start_duration:
    cwnd *= 2
    if receive_ACK:
    cwnd += MSS
else: # congestion-avoidance state
    if receive_ACK:
    cwnd += MSS*(MSS/cwnd)
```



10.1.1 Start Connection: Slow Start

When connection begins, increase rate exponentially until first loss event. (Initial rate is slow but ramps up very fast)

1. Initial cwnd: 1 MSS (maximum segment size)
2. Double cwnd every RTT
    - It is doubled with the formula: $cwnd = cwnd + MSS * \frac{cwnd}{mss}$
3. Increment cwnd for every ACK received

10.1.2 Congestion-avoidance state

Window grows exponentially in **slow start** to threshold, then grows linearly during the congestion-avoidance (CA) state.

- The inexponential increase is switched to linear when `cwnd` gets to half of its value before timeout.
- On loss event, `ssthresh`=0.5∗`cwnd` before loss event.

In the CA state, the congestion window is increased by $\frac{1}{k}$.

- where $k = \frac{cwnd}{mss}$

10.1.3 Explicit Congestion Notification (ECN)

Network-assisted congestion control

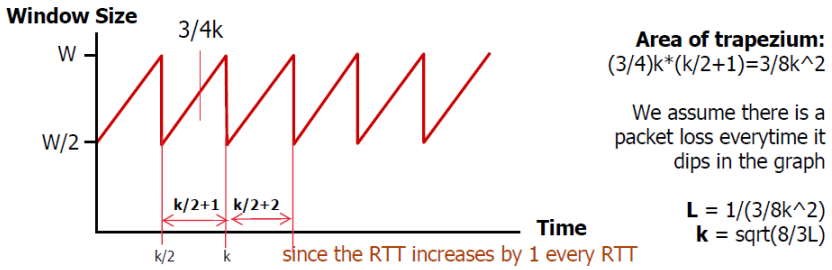
- 2 bits in IP header (ToS field) marked by network router to indicate congestion
- Receiver sets ECE bit on ACK to notify sender of congestion

10.1.4 Calculating TCP Throughput

Ignoring slow start and assuming there is always data to send,

$$\text{TCP Throughput} = \frac{3}{4} * \frac{W}{RTT}$$

- where W: window size in bytes where loss occurs



Calculating Segment Loss Probability, L

e.g. given 1500 byte segments, 100ms RTT, 10Gbps throughput, requires average 83,333 in-flight segments

$$10^7 = \frac{1.22 \times 1500}{100 \times 10^{-3} \times \sqrt{L}}$$
$$L = 2 \times 10^{-10}$$

10.1.5 TCP Fairness

- Goal: For n TCP sessions sharing same bottleneck link of bandwidth R, each should have $\frac{R}{K}$ rate.
- Implementation via Additive Increase and Multiplicate Decrease