CS 438 Final Review

ACM @ UIUC

September 26, 2024

Disclaimers and Logistics

- **Disclaimer:** We have not seen the exam. We have no idea what the questions are. However, we've taken the course and reviewed material/practice exams, so we have **suspicions** as to what the questions will be like.
- This review session is being recorded. Recordings and slides will be distributed on Piazza and the ACM site at the end.
- **Agenda:** We'll review all topics likely to be covered, working through some examples that may look like exam questions as we go, then review individual topics by request.
  - Questions are designed to be written in the same style as previous exams but to be *slightly* harder, so don't worry if you don't get everything right away!
- We're just planning to cover second-half material, but have slides for the entire course- please let us know what you want to cover
- Please let us know if we're going too fast/slow, not speaking loud enough/speaking too loud, etc.
- If you have a question anytime during the review session, please ask! Someone else almost surely has a similar question.
- We'll provide a feedback form at the end of the session.

Table of Contents

1. Foundations
2. Application Layer
3. Transport Layer
4. Network Layer
5. Link Layer
6. Security
7. Practice Questions and Feedback

Foundations I: Resource Sharing

- A **network** is just a set of **elements** (servers, routers, etc.) connected together, that implements a set of **protocols** for the purpose of **sharing resources** at the end hosts
  - Visualized as graph with elements as nodes and **links** as the edges connecting them

Foundations I: Resource Sharing

- A **network** is just a set of **elements** (servers, routers, etc.) connected together, that implements a set of **protocols** for the purpose of **sharing resources** at the end hosts
  - Visualized as graph with elements as nodes and **links** as the edges connecting them
  - Only one goal- deliver the data between edge nodes

Foundations I: Resource Sharing

- A **network** is just a set of **elements** (servers, routers, etc.) connected together, that implements a set of **protocols** for the purpose of **sharing resources** at the end hosts
  - Visualized as graph with elements as nodes and **links** as the edges connecting them
  - Only one goal- deliver the data between edge nodes
- **Statistical Multiplexing**: combining demands to share resources without overbuilding capacity, as aggregate of peak demand is much bigger than peak of aggregate demand. Two approaches:

Foundations I: Resource Sharing

- A **network** is just a set of **elements** (servers, routers, etc.) connected together, that implements a set of **protocols** for the purpose of **sharing resources** at the end hosts
  - Visualized as graph with elements as nodes and **links** as the edges connecting them
  - Only one goal- deliver the data between edge nodes
- **Statistical Multiplexing**: combining demands to share resources without overbuilding capacity, as aggregate of peak demand is much bigger than peak of aggregate demand. Two approaches:
  - **Reservations/Circuit Switching**: Source sends call request, path between source and destination reserved + blocked off, communication happens, then circuit teardown. Used in some telephone + ATM protocols. Can share a channel using FDM (split frequencies) or TDM (round-robin whole resource)

Foundations I: Resource Sharing

- A **network** is just a set of **elements** (servers, routers, etc.) connected together, that implements a set of **protocols** for the purpose of **sharing resources** at the end hosts
  - Visualized as graph with elements as nodes and **links** as the edges connecting them
  - Only one goal- deliver the data between edge nodes
- **Statistical Multiplexing**: combining demands to share resources without overbuilding capacity, as aggregate of peak demand is much bigger than peak of aggregate demand. Two approaches:
  - **Reservations/Circuit Switching**: Source sends call request, path between source and destination reserved + blocked off, communication happens, then circuit teardown. Used in some telephone + ATM protocols. Can share a channel using FDM (split frequencies) or TDM (round-robin whole resource)
  - **Packets/Datagrams**: Packets contain data (body) + information on how/where to send it and where it came from (headers). No underutilization/blocked connections/setup costs and can route around link failures, but no guarantees on availiability/delay, and overhead from headers. Used basically everywhere.

Foundations II: Internet as IPC

- Main problem: **inter-process communication**, where processes may not be on the same machine

End-To-End Story

Foundations II: Internet as IPC

- Main problem: **inter-process communication**, where processes may not be on the same machine

End-To-End Story

- Program opens **socket** which allows it to connect to **network stack**

Foundations II: Internet as IPC

- Main problem: **inter-process communication**, where processes may not be on the same machine

End-To-End Story

- Program opens **socket** which allows it to connect to **network stack**
- **DNS** maps name of target to **address**

Foundations II: Internet as IPC

- Main problem: **inter-process communication**, where processes may not be on the same machine

End-To-End Story

- Program opens **socket** which allows it to connect to **network stack**
- **DNS** maps name of target to **address**
- Network stack embeds address and **port** of source/destination in datagram headers

Foundations II: Internet as IPC

- Main problem: **inter-process communication**, where processes may not be on the same machine

End-To-End Story

- Program opens **socket** which allows it to connect to **network stack**
- **DNS** maps name of target to **address**
- Network stack embeds address and **port** of source/destination in datagram headers
- **Routers** create **routing tables** to decide which **outgoing link** to send packets along (knowing only local information). When link is free, forward packet to next router

Foundations II: Internet as IPC

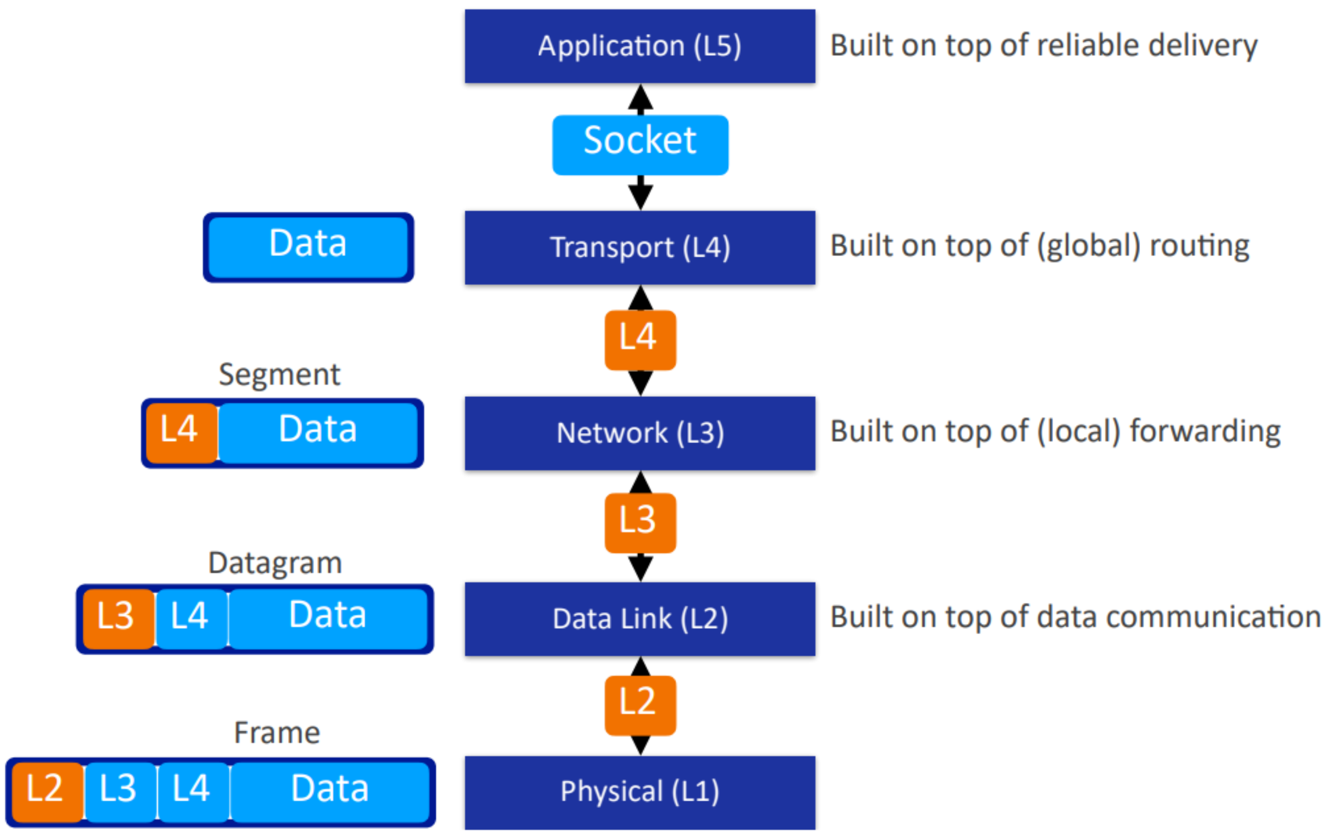
- Main problem: **inter-process communication**, where processes may not be on the same machine

End-To-End Story

- Program opens **socket** which allows it to connect to **network stack**
- **DNS** maps name of target to **address**
- Network stack embeds address and **port** of source/destination in datagram headers
- **Routers** create **routing tables** to decide which **outgoing link** to send packets along (knowing only local information). When link is free, forward packet to next router
- When packet arrives at destination, forward to correct application

- Goal: Nodes shouldn't have to worry about the implementation details of other nodes, just the correct decision locally (modularity)

Foundations III: Layering



Foundations IV: Desiderata

- **Data Loss**: some applications (video/audio) can tolerate it, others (file transfer/ssh/etc.) can't

Foundations IV: Desiderata

- **Data Loss**: some applications (video/audio) can tolerate it, others (file transfer/ssh/etc.) can't
  - **Packet Error Rate** = $\frac{N}{P}$ for N unrecoverable bit errors for P packets. Packets can be recovered through error-correcting coding, depending on schema.

Foundations IV: Desiderata

- **Data Loss**: some applications (video/audio) can tolerate it, others (file transfer/ssh/etc.) can't
  - **Packet Error Rate** = $\frac{N}{P}$ for N unrecoverable bit errors for P packets. Packets can be recovered through error-correcting coding, depending on schema.
  - Example: if bit error rate is 10^{-6}, packet 10kb, no ECC, then PER is $1 - (1 - 10^{-6})^{10240} = 1.0\%$

Foundations IV: Desiderata

- **Data Loss**: some applications (video/audio) can tolerate it, others (file transfer/ssh/etc.) can't
  - **Packet Error Rate** = $\frac{N}{P}$ for N unrecoverable bit errors for P packets. Packets can be recovered through error-correcting coding, depending on schema.
  - Example: if bit error rate is 10^{-6}, packet 10kb, no ECC, then PER is $1 - (1 - 10^{-6})^{10240} = 1.0\%$
- **Bandwidth**: some applications (video streaming) require some amount to be effective, but "elastic apps" can use whatever bandwidth they get

Foundations IV: Desiderata

- **Data Loss**: some applications (video/audio) can tolerate it, others (file transfer/ssh/etc.) can't
  - **Packet Error Rate** = $\frac{N}{P}$ for N unrecoverable bit errors for P packets. Packets can be recovered through error-correcting coding, depending on schema.
  - Example: if bit error rate is 10^{-6}, packet 10kb, no ECC, then PER is $1-(1-10^{-6})^{10240} = 1.0\%$
- **Bandwidth**: some applications (video streaming) require some amount to be effective, but "elastic apps" can use whatever bandwidth they get
  - **Throughput**: Number of *total* correctly delivered bits in unit time

Foundations IV: Desiderata

- **Data Loss**: some applications (video/audio) can tolerate it, others (file transfer/ssh/etc.) can't
  - **Packet Error Rate** = $\frac{N}{P}$ for N unrecoverable bit errors for P packets. Packets can be recovered through error-correcting coding, depending on schema.
  - Example: if bit error rate is 10^{-6}, packet 10kb, no ECC, then PER is $1 - (1 - 10^{-6})^{10240} = 1.0\%$
- **Bandwidth**: some applications (video streaming) require some amount to be effective, but "elastic apps" can use whatever bandwidth they get
  - **Throughput**: Number of *total* correctly delivered bits in unit time
  - **Goodput**: Number of *application-layer* correctly delivered bits in unit time

Foundations IV: Desiderata

- **Data Loss**: some applications (video/audio) can tolerate it, others (file transfer/ssh/etc.) can't
  - **Packet Error Rate** = $\frac{N}{P}$ for N unrecoverable bit errors for P packets. Packets can be recovered through error-correcting coding, depending on schema.
  - Example: if bit error rate is 10^{-6}, packet 10kb, no ECC, then PER is $1 - (1 - 10^{-6})^{10240} = 1.0\%$
- **Bandwidth**: some applications (video streaming) require some amount to be effective, but "elastic apps" can use whatever bandwidth they get
  - **Throughput**: Number of *total* correctly delivered bits in unit time
  - **Goodput**: Number of *application-layer* correctly delivered bits in unit time
- **Timing**: some applications need packets to be delivered right away (video games/live conferencing/etc.). **Per Packet Latency** = Transmit time + Propagation delay + Process time + Queuing Delay. Assuming packet length L, transmit rate R bps

Foundations IV: Desiderata

- **Data Loss**: some applications (video/audio) can tolerate it, others (file transfer/ssh/etc.) can't
  - **Packet Error Rate** = $\frac{N}{P}$ for N unrecoverable bit errors for P packets. Packets can be recovered through error-correcting coding, depending on schema.
  - Example: if bit error rate is 10^{-6}, packet 10kb, no ECC, then PER is $1 - (1 - 10^{-6})^{10240} = 1.0\%$
- **Bandwidth**: some applications (video streaming) require some amount to be effective, but "elastic apps" can use whatever bandwidth they get
  - **Throughput**: Number of *total* correctly delivered bits in unit time
  - **Goodput**: Number of *application-layer* correctly delivered bits in unit time
- **Timing**: some applications need packets to be delivered right away (video games/live conferencing/etc.). **Per Packet Latency** = Transmit time + Propagation delay + Process time + Queuing Delay. Assuming packet length L, transmit rate R bps
  - **Transmit Time**: Time used by transmitter to write packet to wire ($\frac{L}{R}$)

Foundations IV: Desiderata

- **Data Loss**: some applications (video/audio) can tolerate it, others (file transfer/ssh/etc.) can't
  - **Packet Error Rate** = $\frac{N}{P}$ for N unrecoverable bit errors for P packets. Packets can be recovered through error-correcting coding, depending on schema.
  - Example: if bit error rate is 10^{-6}, packet 10kb, no ECC, then PER is $1 - (1 - 10^{-6})^{10240} = 1.0\%$
- **Bandwidth**: some applications (video streaming) require some amount to be effective, but "elastic apps" can use whatever bandwidth they get
  - **Throughput**: Number of *total* correctly delivered bits in unit time
  - **Goodput**: Number of *application-layer* correctly delivered bits in unit time
- **Timing**: some applications need packets to be delivered right away (video games/live conferencing/etc.). **Per Packet Latency** = Transmit time + Propagation delay + Process time + Queuing Delay. Assuming packet length L, transmit rate R bps
  - **Transmit Time**: Time used by transmitter to write packet to wire ($\frac{L}{R}$)
  - **Propagation Delay**: Time for packet to travel from transmitter to receiver ($\frac{\text{distance}}{\text{velocity}}$)

Foundations IV: Desiderata

- **Data Loss**: some applications (video/audio) can tolerate it, others (file transfer/ssh/etc.) can't
  - **Packet Error Rate** = $\frac{N}{P}$ for N unrecoverable bit errors for P packets. Packets can be recovered through error-correcting coding, depending on schema.
  - Example: if bit error rate is 10^{-6}, packet 10kb, no ECC, then PER is $1 - (1 - 10^{-6})^{10240} = 1.0\%$
- **Bandwidth**: some applications (video streaming) require some amount to be effective, but "elastic apps" can use whatever bandwidth they get
  - **Throughput**: Number of *total* correctly delivered bits in unit time
  - **Goodput**: Number of *application-layer* correctly delivered bits in unit time
- **Timing**: some applications need packets to be delivered right away (video games/live conferencing/etc.). **Per Packet Latency** = Transmit time + Propagation delay + Process time + Queuing Delay. Assuming packet length L, transmit rate R bps
  - **Transmit Time**: Time used by transmitter to write packet to wire ($\frac{L}{R}$)
  - **Propagation Delay**: Time for packet to travel from transmitter to receiver ($\frac{\text{distance}}{\text{velocity}}$)
  - **Process Time**: Time required for router to read header + decide route

Foundations IV: Desiderata

- **Data Loss**: some applications (video/audio) can tolerate it, others (file transfer/ssh/etc.) can't
  - **Packet Error Rate** = $\frac{N}{P}$ for N unrecoverable bit errors for P packets. Packets can be recovered through error-correcting coding, depending on schema.
  - Example: if bit error rate is 10^{-6}, packet 10kb, no ECC, then PER is $1-(1-10^{-6})^{10240}=1.0\%$
- **Bandwidth**: some applications (video streaming) require some amount to be effective, but "elastic apps" can use whatever bandwidth they get
  - **Throughput**: Number of *total* correctly delivered bits in unit time
  - **Goodput**: Number of *application-layer* correctly delivered bits in unit time
- **Timing**: some applications need packets to be delivered right away (video games/live conferencing/etc.). **Per Packet Latency** = Transmit time + Propagation delay + Process time + Queuing Delay. Assuming packet length L, transmit rate R bps
  - **Transmit Time**: Time used by transmitter to write packet to wire ($\frac{L}{R}$)
  - **Propagation Delay**: Time for packet to travel from transmitter to receiver ($\frac{\text{distance}}{\text{velocity}}$)
  - **Process Time**: Time required for router to read header + decide route
  - **Queuing Delay**: Time that a packet waits in queue because link is busy. In expectation, proportional to $\frac{La}{r}$ with a packets in queue.

Foundations V: Signal Processing

- **Frequency Band** refers to the range of frequencies used for a signal. **Bandwidth** is the width of this band, is proportional to data rate.
- Frequency bands may be divided up into smaller **channels** for simultaneous communication

Foundations V: Signal Processing

- **Frequency Band** refers to the range of frequencies used for a signal. **Bandwidth** is the width of this band, is proportional to data rate.
- Frequency bands may be divided up into smaller **channels** for simultaneous communication
- **Carrier Frequency**: Fixed (higher) frequency used to carry signal. Options include **Amplitude Shift Keying**, **Frequency Shift Keying**

Foundations V: Signal Processing

- **Frequency Band** refers to the range of frequencies used for a signal. **Bandwidth** is the width of this band, is proportional to data rate.
- Frequency bands may be divided up into smaller **channels** for simultaneous communication
- **Carrier Frequency**: Fixed (higher) frequency used to carry signal. Options include **Amplitude Shift Keying**, **Frequency Shift Keying**
- **Signal to Interference and Noise Ratio**: $\frac{P_{\text{signal}}}{P_{\text{noise}}+P_{\text{interference}}}$. Bit error rate is a function of this.

Theorem (Shannon Capacity)

$C = B \log_2(1 + SINR)$

- *Capacity (C) in bits per second*
- *Bandwidth (B) in Hz*

Application Layer: Transport Protocols

- Applications can choose between TCP and UDP

Application Layer: Transport Protocols

- Applications can choose between TCP and UDP
- TCP
  - Uses connection setup between processes
  - Provides a **reliable transport** guarantee w.r.t. correctness/order/duplication
  - **Flow control**: sender won't overwhelm receiver
  - **Congestion control**: Will slow down to avoid network overload
  - No guarantees on timing, bandwidth

Application Layer: Transport Protocols

- Applications can choose between TCP and UDP
- TCP
  - Uses connection setup between processes
  - Provides a **reliable transport** guarantee w.r.t. correctness/order/duplication
  - **Flow control**: sender won't overwhelm receiver
  - **Congestion control**: Will slow down to avoid network overload
  - No guarantees on timing, bandwidth
- UDP
  - Unreliable data transfer between sender and receiver. No fancy control/ordering systems.

Application Layer: Transport Protocols

- Applications can choose between TCP and UDP
- TCP
  - Uses connection setup between processes
  - Provides a **reliable transport** guarantee w.r.t. correctness/order/duplication
  - **Flow control**: sender won't overwhelm receiver
  - **Congestion control**: Will slow down to avoid network overload
  - No guarantees on timing, bandwidth
- UDP
  - Unreliable data transfer between sender and receiver. No fancy control/ordering systems.
- Most internet protocols (HTTP/FTP/SMTP/etc.) are built on TCP, but a lot of video streaming/VoIP/trading systems use UDP

Application Layer: DNS

- A **distributed database** implemented as a hierarchy of **name servers** to **resolve** domain names as IP addresses at the application layer
  - 13 **root** DNS servers, thousands of TLD DNS servers (.com/.edu/.org/.uk), **authoritative** DNS servers set up by organization or service provider providing an authoritative source for organization's servers (web, mail, etc.)

Application Layer: DNS

- A **distributed database** implemented as a hierarchy of **name servers** to **resolve** domain names as IP addresses at the application layer
  - 13 **root** DNS servers, thousands of TLD DNS servers (.com/.edu/.org/.uk), **authoritative** DNS servers set up by organization or service provider providing an authoritative source for organization's servers (web, mail, etc.)
- **Iterative Querying**: Server either returns the record, or an address of a DNS server who might

Application Layer: DNS

- A **distributed database** implemented as a hierarchy of **name servers** to **resolve** domain names as IP addresses at the application layer
  - 13 **root** DNS servers, thousands of TLD DNS servers (.com/.edu/.org/.uk), **authoritative** DNS servers set up by organization or service provider providing an authoritative source for organization's servers (web, mail, etc.)
- **Iterative Querying**: Server either returns the record, or an address of a DNS server who might
- **Recursive Querying**: Server returns record, asking other servers if needed

Application Layer: DNS

- A **distributed database** implemented as a hierarchy of **name servers** to **resolve** domain names as IP addresses at the application layer
  - 13 **root** DNS servers, thousands of TLD DNS servers (.com/.edu/.org/.uk), **authoritative** DNS servers set up by organization or service provider providing an authoritative source for organization's servers (web, mail, etc.)
- **Iterative Querying**: Server either returns the record, or an address of a DNS server who might
- **Recursive Querying**: Server returns record, asking other servers if needed
- When any DNS server learns a mapping, it **caches** it (which times out and disappears eventually). Most common TLD servers are often cached locally, meaning root name server unusual.

Application Layer: DNS

- A **distributed database** implemented as a hierarchy of **name servers** to **resolve** domain names as IP addresses at the application layer
  - 13 **root** DNS servers, thousands of TLD DNS servers (.com/.edu/.org/.uk), **authoritative** DNS servers set up by organization or service provider providing an authoritative source for organization's servers (web, mail, etc.)
- **Iterative Querying**: Server either returns the record, or an address of a DNS server who might
- **Recursive Querying**: Server returns record, asking other servers if needed
- When any DNS server learns a mapping, it **caches** it (which times out and disappears eventually). Most common TLD servers are often cached locally, meaning root name server unusual.
- 4 types of records: A (hostname `name` is IP address `value`), NS (authoritative name server for `name` can be found at `value`), CNAME (the "real" name for alias `name` is the canonical name `value`), MX (the mailserver for `name` has name `value`)

Application Layer: DNS

- A **distributed database** implemented as a hierarchy of **name servers** to **resolve** domain names as IP addresses at the application layer
  - 13 **root** DNS servers, thousands of TLD DNS servers (.com/.edu/.org/.uk), **authoritative** DNS servers set up by organization or service provider providing an authoritative source for organization's servers (web, mail, etc.)
- **Iterative Querying**: Server either returns the record, or an address of a DNS server who might
- **Recursive Querying**: Server returns record, asking other servers if needed
- When any DNS server learns a mapping, it **caches** it (which times out and disappears eventually). Most common TLD servers are often cached locally, meaning root name server unusual.
- 4 types of records: A (hostname `name` is IP address `value`), NS (authoritative name server for `name` can be found at `value`), CNAME (the "real" name for alias `name` is the canonical name `value`), MX (the mailserver for `name` has name `value`)
- Inserting Records: Provide registrar with name and IP of authorative name server, registrar inserts NS record for auth server name and A record for auth server IP

Application Layer: Architectures

- **Client-Server**
  - Server: always-on host with constant address, Clients: communicate only with servers and not with each other, may disconnect/reconnect, change IP addresses.

Application Layer: Architectures

- **Client-Server**
  - Server: always-on host with constant address, Clients: communicate only with servers and not with each other, may disconnect/reconnect, change IP addresses.
- **Peer-to-Peer**
  - No always-on server, peers might disconnect, change addresses. Scalable, but sometimes difficult to manage. Examples: CHORD, Gnutella

Application Layer: Architectures

- **Client-Server**
  - Server: always-on host with constant address, Clients: communicate only with servers and not with each other, may disconnect/reconnect, change IP addresses.
- **Peer-to-Peer**
  - No always-on server, peers might disconnect, change addresses. Scalable, but sometimes difficult to manage. Examples: CHORD, Gnutella
- Many services use hybrid (ex: video conferencing/instant messaging: users directly connect with each other but use central server to register/look up *where* users are)

Application Layer: HTTP

- Web pages consist of **objects** (HTML, images, javascript, etc.), each of which has a URL

Application Layer: HTTP

- Web pages consist of **objects** (HTML, images, javascript, etc.), each of which has a URL
- **HTTP uses a client-server model** where clients request + render, servers send objects in response to requests.

Application Layer: HTTP

- Web pages consist of **objects** (HTML, images, javascript, etc.), each of which has a URL
- **HTTP uses a client-server model** where clients request + render, servers send objects in response to requests.
- **HTTP uses TCP**. Clients connect to port 80 of host, messages are exchanged between brower + server.

Application Layer: HTTP

- Web pages consist of **objects** (HTML, images, javascript, etc.), each of which has a URL
- **HTTP uses a client-server model** where clients request + render, servers send objects in response to requests.
- **HTTP uses TCP**. Clients connect to port 80 of host, messages are exchanged between brower + server.
- **HTTP is "stateless"**. Servers maintain no information about previous requests.

Application Layer: HTTP

- Web pages consist of **objects** (HTML, images, javascript, etc.), each of which has a URL
- **HTTP uses a client-server model** where clients request + render, servers send objects in response to requests.
- **HTTP uses TCP**. Clients connect to port 80 of host, messages are exchanged between brower + server.
- **HTTP is "stateless"**. Servers maintain no information about previous requests.
- **Non-Persistent HTTP** (HTTP/1.0): Connection closes after one response. What's the total response time?

Application Layer: HTTP

- Web pages consist of **objects** (HTML, images, javascript, etc.), each of which has a URL
- **HTTP uses a client-server model** where clients request + render, servers send objects in response to requests.
- **HTTP uses TCP**. Clients connect to port 80 of host, messages are exchanged between brower + server.
- **HTTP is "stateless"**. Servers maintain no information about previous requests.
- **Non-Persistent HTTP** (HTTP/1.0): Connection closes after one response. What's the total response time? **2 $\times$ RTT + file transmit time** per object.

Application Layer: HTTP

- Web pages consist of **objects** (HTML, images, javascript, etc.), each of which has a URL
- **HTTP uses a client-server model** where clients request + render, servers send objects in response to requests.
- **HTTP uses TCP**. Clients connect to port 80 of host, messages are exchanged between brower + server.
- **HTTP is “stateless”**. Servers maintain no information about previous requests.
- **Non-Persistent HTTP** (HTTP/1.0): Connection closes after one response. What’s the total response time? **$2 \times$ RTT + file transmit time** per object.
- **Persistent HTTP without pipelining**: Connection stays open, but waits for one message response before the next one is sent. What’s the total response time?

Application Layer: HTTP

- Web pages consist of **objects** (HTML, images, javascript, etc.), each of which has a URL
- **HTTP uses a client-server model** where clients request + render, servers send objects in response to requests.
- **HTTP uses TCP**. Clients connect to port 80 of host, messages are exchanged between brower + server.
- **HTTP is "stateless"**. Servers maintain no information about previous requests.
- **Non-Persistent HTTP** (HTTP/1.0): Connection closes after one response. What's the total response time? **2 $\times$ RTT + file transmit time** per object.
- **Persistent HTTP without pipelining**: Connection stays open, but waits for one message response before the next one is sent. What's the total response time? **(# of referenced objects + 1) $\times$ RTT + data transmit time**

Application Layer: HTTP

- Web pages consist of **objects** (HTML, images, javascript, etc.), each of which has a URL
- **HTTP uses a client-server model** where clients request + render, servers send objects in response to requests.
- **HTTP uses TCP**. Clients connect to port 80 of host, messages are exchanged between brower + server.
- **HTTP is "stateless"**. Servers maintain no information about previous requests.
- **Non-Persistent HTTP** (HTTP/1.0): Connection closes after one response. What's the total response time? **2 $\times$ RTT + file transmit time** per object.
- **Persistent HTTP without pipelining**: Connection stays open, but waits for one message response before the next one is sent. What's the total response time? **(# of referenced objects + 1) $\times$ RTT + data transmit time**
- **Persistent HTTP *with* pipelining** (HTTP/1.1 default): Connection stays open, *and* client requests a file as soon as it's referenced. What's the min possible total response time?

Application Layer: HTTP

- Web pages consist of **objects** (HTML, images, javascript, etc.), each of which has a URL
- **HTTP uses a client-server model** where clients request + render, servers send objects in response to requests.
- **HTTP uses TCP**. Clients connect to port 80 of host, messages are exchanged between brower + server.
- **HTTP is "stateless"**. Servers maintain no information about previous requests.
- **Non-Persistent HTTP** (HTTP/1.0): Connection closes after one response. What's the total response time? **2 $\times$ RTT + file transmit time** per object.
- **Persistent HTTP without pipelining**: Connection stays open, but waits for one message response before the next one is sent. What's the total response time? **(# of referenced objects + 1) $\times$ RTT + data transmit time**
- **Persistent HTTP *with* pipelining** (HTTP/1.1 default): Connection stays open, *and* client requests a file as soon as it's referenced. What's the min possible total response time? **Setup + data transmit + 1 RTT for *all* objects**.

Application Layer: HTTP

- Two types of messages: **request**, **response**. Headers in ASCII (except for HTTP/2 or later versions).
  - Example Request:

```
GET / HTTP/1.1
Host: illinois.edu
User-Agent: curl/8.9.1
Accept: */*
```

Application Layer: HTTP

- Two types of messages: **request**, **response**. Headers in ASCII (except for HTTP/2 or later versions).
  - Example Request:

    ```
    GET / HTTP/1.1
    Host: illinois.edu
    User-Agent: curl/8.9.1
    Accept: */*
    ```

  - Method types: GET (gets requested file, can have information in URL parameters), POST (uploads body to server), HEAD (leaves everything but headers out), PUT (uploads file to path in URL field), DELETE (deletes file in URL field)

Application Layer: HTTP

- Two types of messages: **request**, **response**. Headers in ASCII (except for HTTP/2 or later versions).
  - Example Request:
    ```
    GET / HTTP/1.1
    Host: illinois.edu
    User-Agent: curl/8.9.1
    Accept: */*
    ```
  - Method types: GET (gets requested file, can have information in URL parameters), POST (uploads body to server), HEAD (leaves everything but headers out), PUT (uploads file to path in URL field), DELETE (deletes file in URL field)
  - Example Response:
    ```
    HTTP/1.1 200 OK
    Date: Mon, 21 Oct 2024 23:15:43 GMT
    Server: Apache/2.4.57 (Red Hat Enterprise Linux) OpenSSL/3.0.7
    Last-Modified: Mon, 23 Sep 2024 21:24:01 GMT
    ETag: "eac6-622d001ecb792"
    Accept-Ranges: bytes
    Content-Length: 60102
    Content-Type: text/html; charset=UTF-8
    ```

Application Layer: Caching

- **Goal**: Satisfy client request without involving origin server

Application Layer: Caching

- **Goal**: Satisfy client request without involving origin server
- Browser sends all requests to cache, which acts as both client and server. If cache has file, returns immediately, else requests from server and returns. **Which requests does this help?**

Application Layer: Caching

- **Goal**: Satisfy client request without involving origin server
- Browser sends all requests to cache, which acts as both client and server. If cache has file, returns immediately, else requests from server and returns. **Which requests does this help?**
- Can use **conditional GET requests**. Add `If-modified-since` field to headers; if not modified, return status 304, else return file. Ensures that requests are up-to-date while still saving bandwidth. **Why?**

Application Layer: SMTP

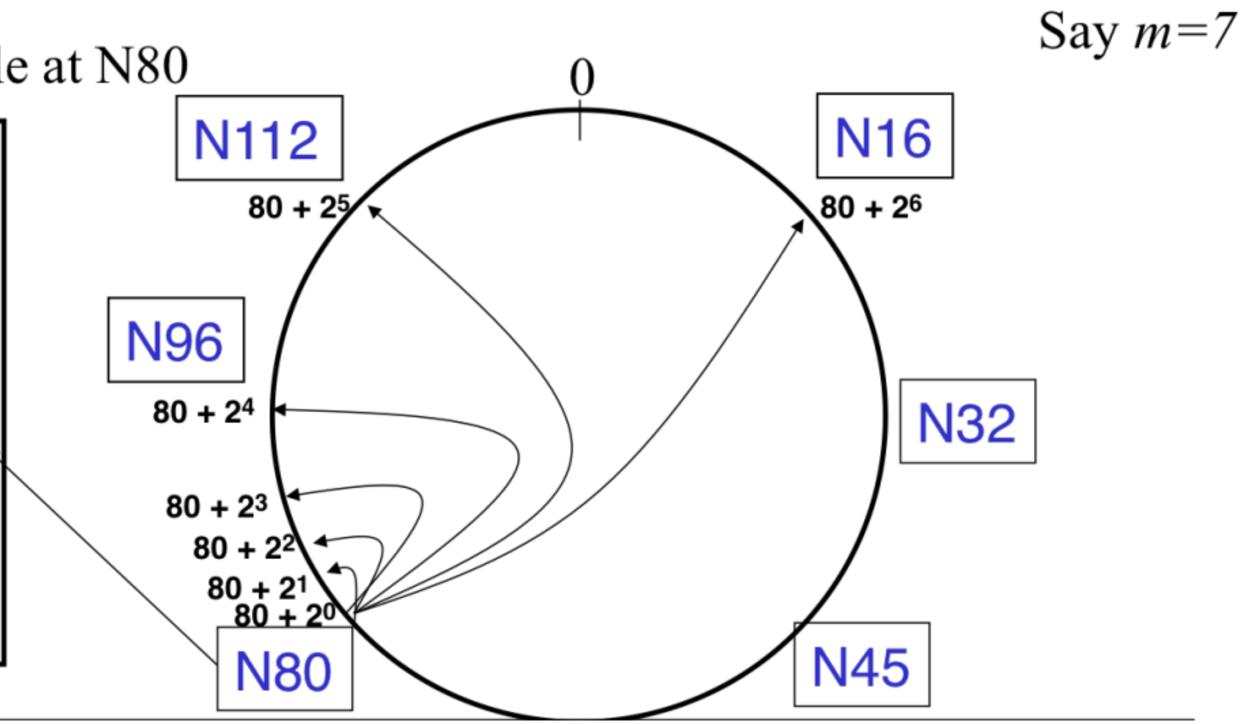
- Uses TCP on port 25 to send mail
- Sending mail server acts as "client", while recieving server acts as "server". This makes it a "push" protocol, rather than a "pull" protocol (like HTTP)
- Three phases of transfer: handshake, message transfer, closure. Commands in ASCII, response is status code + message.
- Users access email boxes via **user agents** (POP3/IMAP/webmail).
- Lots more details, but they're highly unlikely to come up on an exam.

Application Layer: CHORD

- Each file assigned a hash and assigned to the next highest node, each server knows a “finger table” of nodes exponentially far away from current id, recursive lookup structure.

Finger Table at N80

i	*ft[i]*
0	96
1	96
2	96
3	96
4	96
5	112
6	16



Transport Layer: Goals

- We need to provide a **reliable data stream** to the receiver's application from the sender's application. However, there's a lot that can go wrong:

Transport Layer: Goals

- We need to provide a **reliable data stream** to the receiver's application from the sender's application. However, there's a lot that can go wrong:
  - Packets can be lost

Transport Layer: Goals

- We need to provide a **reliable data stream** to the receiver's application from the sender's application. However, there's a lot that can go wrong:
  - Packets can be lost
  - Packets can arrive out-of-order

Transport Layer: Goals

- We need to provide a **reliable data stream** to the receiver's application from the sender's application. However, there's a lot that can go wrong:
  - Packets can be lost
  - Packets can arrive out-of-order
  - Packets can be delayed arbitrarily long

Transport Layer: Goals

- We need to provide a **reliable data stream** to the receiver's application from the sender's application. However, there's a lot that can go wrong:
  - Packets can be lost
  - Packets can arrive out-of-order
  - Packets can be delayed arbitrarily long
  - Packets can be duplicated

Transport Layer: Goals

- We need to provide a **reliable data stream** to the receiver's application from the sender's application. However, there's a lot that can go wrong:
  - Packets can be lost
  - Packets can arrive out-of-order
  - Packets can be delayed arbitrarily long
  - Packets can be duplicated
  - Packets can be corrupted

Transport Layer: Goals

- We need to provide a **reliable data stream** to the receiver's application from the sender's application. However, there's a lot that can go wrong:
  - Packets can be lost
  - Packets can arrive out-of-order
  - Packets can be delayed arbitrarily long
  - Packets can be duplicated
  - Packets can be corrupted
  - We can overwhelm a sending/receiving buffer (thereby dropping packets)

Transport Layer: Goals

- We need to provide a **reliable data stream** to the receiver's application from the sender's application. However, there's a lot that can go wrong:
    - Packets can be lost
    - Packets can arrive out-of-order
    - Packets can be delayed arbitrarily long
    - Packets can be duplicated
    - Packets can be corrupted
    - We can overwhelm a sending/receiving buffer (thereby dropping packets)
    - If we transmit too much, we can interfere with other communication going on

Transport Layer: Goals

- We need to provide a **reliable data stream** to the receiver's application from the sender's application. However, there's a lot that can go wrong:
  - Packets can be lost
  - Packets can arrive out-of-order
  - Packets can be delayed arbitrarily long
  - Packets can be duplicated
  - Packets can be corrupted
  - We can overwhelm a sending/receiving buffer (thereby dropping packets)
  - If we transmit too much, we can interfere with other communication going on
- We don't want to use any information about lower/higher layers

Transport Layer: Goals

- We need to provide a **reliable data stream** to the receiver's application from the sender's application. However, there's a lot that can go wrong:
  - Packets can be lost
  - Packets can arrive out-of-order
  - Packets can be delayed arbitrarily long
  - Packets can be duplicated
  - Packets can be corrupted
  - We can overwhelm a sending/receiving buffer (thereby dropping packets)
  - If we transmit too much, we can interfere with other communication going on
- We don't want to use any information about lower/higher layers
- Also, distributed consensus is *hard*. Some of what we want to do is the **Two Generals' Problem**: since message acknowledgments are as likely to be lost as messages, we'd potentially need infinite messages to come to consensus safely.

Transport Layer: Basic Validation

- Most protocols: to avoid out of sequence ordering, assign each packet a **sequence number**. Send an **acknowledgment** (ACK) if packet received.

Transport Layer: Basic Validation

- Most protocols: to avoid out of sequence ordering, assign each packet a **sequence number**. Send an **acknowledgment** (ACK) if packet received.
- Packets can be corrupted. Receiver can calculate **checksum** to verify validity, as well as run ECCs if protocol provides (Hamming codes, etc.) Some protocols send NACKs to indicate that packet has been rejected, but most just ignore packet.

Transport Layer: Basic Validation

- Most protocols: to avoid out of sequence ordering, assign each packet a **sequence number**. Send an **acknowledgment** (ACK) if packet received.
- Packets can be corrupted. Receiver can calculate **checksum** to verify validity, as well as run ECCs if protocol provides (Hamming codes, etc.) Some protocols send NACKs to indicate that packet has been rejected, but most just ignore packet.
- Idea: Instead of verifying message/ACK reception, have sender simply resend the packet if no ACK has been received after some time. If receiver receives duplicate packet (by sequence number), acknowledge but throw out. **How does this avoid two generals?**

Transport Layer: Basic Validation

- Most protocols: to avoid out of sequence ordering, assign each packet a **sequence number**. Send an **acknowledgment** (ACK) if packet received.
- Packets can be corrupted. Receiver can calculate **checksum** to verify validity, as well as run ECCs if protocol provides (Hamming codes, etc.) Some protocols send NACKs to indicate that packet has been rejected, but most just ignore packet.
- Idea: Instead of verifying message/ACK reception, have sender simply resend the packet if no ACK has been received after some time. If receiver receives duplicate packet (by sequence number), acknowledge but throw out. **How does this avoid two generals?** Receiver doesn't know (or care) *which* ACKs have been received, so no distributed consensus.

Transport Layer: Pipelined Protocols

- Validation + one-packet-at-a-time approach is correct, but is intolerably slow. **Solution:** allow *multiple* packets to be “in-flight” (forming a **context window**)

Transport Layer: Pipelined Protocols

- Validation + one-packet-at-a-time approach is correct, but is intolerably slow. **Solution:** allow *multiple* packets to be "in-flight" (forming a **context window**)
- **Go-Back-*N*** (GBN): Receiver keeps track of the first packet that has *not* been received (expected_seq_num).

```
procedure RECIEVE(k)
    if k = expected_seq_num then
        Send ACK(k); expected_seq_num ← expected_seq_num +1
    else
        Send ACK(expected_seq_num - 1)            ▷ "cumulative ACK"
```

  - On timeout: resend *all* packets in CW

Transport Layer: Pipelined Protocols

- Validation + one-packet-at-a-time approach is correct, but is intolerably slow. **Solution:** allow *multiple* packets to be "in-flight" (forming a **context window**)
- **Go-Back-*N*** (GBN): Receiver keeps track of the first packet that has *not* been received (expected_seq_num).

procedure RECIEVE(k)
    if k = expected_seq_num **then**
        Send ACK(k); expected_seq_num $\leftarrow$ expected_seq_num $+1$
    else
        Send ACK(expected_seq_num - 1) ▷ *"cumulative ACK"*

  - On timeout: resend *all* packets in CW
- **Selective ACK**: receiver individually acknowledges all correctly received packets (ACK(k)), buffers if needed for in-order delivery to application layer. Sender retransmits packets where no ACK received. If receiver receives a packet with the same sequence number as something in the buffer, throw out.

Transport Layer: Pipelined Protocols

- Validation + one-packet-at-a-time approach is correct, but is intolerably slow. **Solution:** allow *multiple* packets to be "in-flight" (forming a **context window**)
- **Go-Back-*N*** (GBN): Receiver keeps track of the first packet that has *not* been received (expected_seq_num).

  procedure RECIEVE(k)
      if k = expected_seq_num **then**
          Send ACK(k); expected_seq_num $\leftarrow$ expected_seq_num $+1$
      else
          Send ACK(expected_seq_num - 1) $\triangleright$ *"cumulative ACK"*

  - On timeout: resend *all* packets in CW
- **Selective ACK**: receiver individually acknowledges all correctly received packets (ACK(k)), buffers if needed for in-order delivery to application layer. Sender retransmits packets where no ACK received. If receiver receives a packet with the same sequence number as something in the buffer, throw out.
- TCP takes a **hybrid approach**, reports cumulative ACKs (lowest seq # not recieved - 1), but will accept out-of-order packets and reorder them.

Transport Layer: Pipelined Protocols

- Validation + one-packet-at-a-time approach is correct, but is intolerably slow. **Solution:** allow *multiple* packets to be "in-flight" (forming a **context window**)
- **Go-Back-*N*** (GBN): Receiver keeps track of the first packet that has *not* been received (expected_seq_num).

  procedure RECIEVE(k)
      if k = expected_seq_num **then**
          Send ACK(k); expected_seq_num $\leftarrow$ expected_seq_num $+1$
      else
          Send ACK(expected_seq_num - 1) $\triangleright$ *"cumulative ACK"*

  - On timeout: resend *all* packets in CW
- **Selective ACK**: receiver individually acknowledges all correctly received packets (ACK(k)), buffers if needed for in-order delivery to application layer. Sender retransmits packets where no ACK received. If receiver receives a packet with the same sequence number as something in the buffer, throw out.
- TCP takes a **hybrid approach**, reports cumulative ACKs (lowest seq # not recieved - 1), but will accept out-of-order packets and reorder them.
  - Sender considers multiple ACK(i)s as **dupACK**s, fresh i in ACK(i) **newACK**. Useful for estimating congestion.

Transport Layer: TCP Congestion Control

- **Main Problem**: How large do we make the context window?
  - If too large, then router queues fill up, dropped packets, no fair sharing.
  - If too small, then suboptimal performance.

Transport Layer: TCP Congestion Control

- **Main Problem**: How large do we make the context window?
  - If too large, then router queues fill up, dropped packets, no fair sharing.
  - If too small, then suboptimal performance.
- Initially, (after SYN/SYN-ACK/ACK), transmitter transmits small burst of packets, waits for ACK. Exponentially increases burst size (**slow start**).

Transport Layer: TCP Congestion Control

- **Main Problem**: How large do we make the context window?
  - If too large, then router queues fill up, dropped packets, no fair sharing.
  - If too small, then suboptimal performance.
- Initially, (after SYN/SYN-ACK/ACK), transmitter transmits small burst of packets, waits for ACK. Exponentially increases burst size (**slow start**).
- After some point (**SST**/**LIT**), switch from doubling the window size every burst to increasing it by 1 per burst.

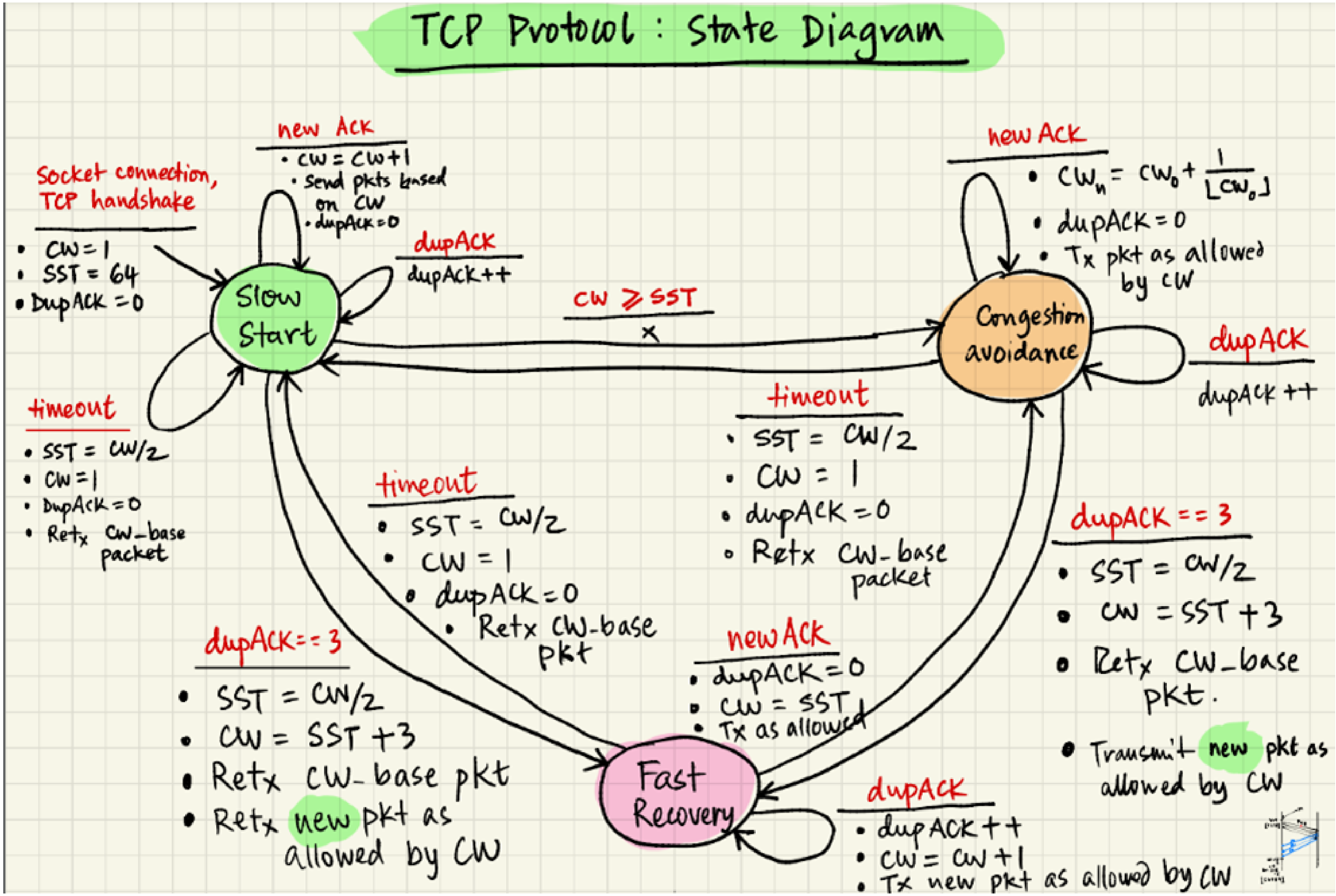
Transport Layer: TCP Congestion Control

- **Main Problem**: How large do we make the context window?
  - If too large, then router queues fill up, dropped packets, no fair sharing.
  - If too small, then suboptimal performance.
- Initially, (after SYN/SYN-ACK/ACK), transmitter transmits small burst of packets, waits for ACK. Exponentially increases burst size (**slow start**).
- After some point (**SST**/**LIT**), switch from doubling the window size every burst to increasing it by 1 per burst.
- Sender keeps a timer to interrupt for timeout. When CW shifted, increase the timer by the gap between packets. On timeout, drastically decrease CW, SST, resend packets.

Transport Layer: TCP Congestion Control

- **Main Problem**: How large do we make the context window?
  - If too large, then router queues fill up, dropped packets, no fair sharing.
  - If too small, then suboptimal performance.
- Initially, (after SYN/SYN-ACK/ACK), transmitter transmits small burst of packets, waits for ACK. Exponentially increases burst size (**slow start**).
- After some point (**SST**/**LIT**), switch from doubling the window size every burst to increasing it by 1 per burst.
- Sender keeps a timer to interrupt for timeout. When CW shifted, increase the timer by the gap between packets. On timeout, drastically decrease CW, SST, resend packets.
- DupACKs aren't necessarily a bad sign, but might be indicator of missed packets. If 3 dupACKs in a row, retransmit DupACK packet but don't reset SST, slightly cut CW (**fast recovery**).

Transport Layer: TCP State Machine



Transport Layer: TCP RTO Estimation

- How do we estimate how long timeout (RTO) should be?

Transport Layer: TCP RTO Estimation

- How do we estimate how long timeout (RTO) should be?
  - If too short, then premature timeout
  - If too long, then slow reactions to packet loss

Transport Layer: TCP RTO Estimation

- How do we estimate how long timeout (RTO) should be?
  - If too short, then premature timeout
  - If too long, then slow reactions to packet loss
- **Intuition**: should be AvgRTT + some “guard factor”.

Transport Layer: TCP RTO Estimation

- How do we estimate how long timeout (RTO) should be?
  - If too short, then premature timeout
  - If too long, then slow reactions to packet loss
- **Intuition**: should be AvgRTT + some "guard factor".
- AvgRTT estimated by rolling average: $RTT_{avg} \leftarrow (1-\alpha)RTT_{avg} + \alpha\, RTT_{packet}$

Transport Layer: TCP RTO Estimation

- How do we estimate how long timeout (RTO) should be?
  - If too short, then premature timeout
  - If too long, then slow reactions to packet loss
- **Intuition**: should be AvgRTT + some "guard factor".
- AvgRTT estimated by rolling average: $RTT_{avg} \leftarrow (1-\alpha)RTT_{avg} + \alpha\, RTT_{packet}$
- "guard factor" can be a deviation estimate:

$$devRTT_{avg} \leftarrow (1-\beta)\, devRTT_{avg} + \beta(|RTT_{packet} - RTT_{avg}|)$$
$$RTO \leftarrow RTT_{avg} + 4\, devRTT_{avg}$$

Transport Layer: Flow Control and Fairness

- **Problem**: Receiver has a limited buffer. If many nodes transmitting to same receiver, losses may happen at receiver

Transport Layer: Flow Control and Fairness

- **Problem**: Receiver has a limited buffer. If many nodes transmitting to same receiver, losses may happen at receiver
- **Solution**: Receiver reports how much space left to sender in ACKs. Sender will deliberately use a smaller congestion window (while calculating CW as normal).

Transport Layer: Flow Control and Fairness

- **Problem**: Receiver has a limited buffer. If many nodes transmitting to same receiver, losses may happen at receiver
- **Solution**: Receiver reports how much space left to sender in ACKs. Sender will deliberately use a smaller congestion window (while calculating CW as normal).

- TCP guarantees **max-min fairness** (in stable state): All flows requesting less than fair share get their request. Remaining flows divide equally.

Network layer (IP)

- **Goals**: Attempt to send data from one node to another. No guarantees on reliability or anything else.

Network layer (IP)

- **Goals**: Attempt to send data from one node to another. No guarantees on reliability or anything else.
- **Sender**: Recieves data from transport, encapsulated into datagrams, sends it on its way to reciever

Network layer (IP)

- **Goals**: Attempt to send data from one node to another. No guarantees on reliability or anything else.
- **Sender**: Recieves data from transport, encapsulated into datagrams, sends it on its way to reciever
- **Reciever**: Looks at incoming packets, transmit on to transport layer

Network layer (IP)

- **Goals**: Attempt to send data from one node to another. No guarantees on reliability or anything else.
- **Sender**: Recieves data from transport, encapsulated into datagrams, sends it on its way to reciever
- **Reciever**: Looks at incoming packets, transmit on to transport layer
- **Lots of locality**: the internet is **way** too big for everyone to know how best to send to everyone else, so within a network, our goal is usually just to deliver data to a place where someone else can get it even closer to our destination

Network layer (IP)

- **Goals**: Attempt to send data from one node to another. No guarantees on reliability or anything else.
- **Sender**: Recieves data from transport, encapsulated into datagrams, sends it on its way to reciever
- **Reciever**: Looks at incoming packets, transmit on to transport layer
- **Lots of locality**: the internet is **way** too big for everyone to know how best to send to everyone else, so within a network, our goal is usually just to deliver data to a place where someone else can get it even closer to our destination
- Routers examine headers of all IP packets, figure out where to pass it along

Routing vs Forwarding

- **Routing**: determine the path that packets should take from source to destination (Global)

Routing vs Forwarding

- **Routing**: determine the path that packets should take from source to destination (Global)
- **Forwarding**: move packets from input buffers to an appropriate output buffer (Local)

Routing vs Forwarding

- **Routing**: determine the path that packets should take from source to destination (Global)
- **Forwarding**: move packets from input buffers to an appropriate output buffer (Local)
- Routers run routing algorithms to determine **forwarding table**

Virtual Circuits (potentially skip)

- Modern networking is **connectionless** (each datagram routed individually) with packets being delivered at **best effort**, but what if we want additional guarantees? We can reserve a **flow/connection** from source to destination, with all packets routed along the flow

Virtual Circuits (potentially skip)

- Modern networking is **connectionless** (each datagram routed individually) with packets being delivered at **best effort**, but what if we want additional guarantees? We can reserve a **flow/connection** from source to destination, with all packets routed along the flow
- Virtual circuits issued numbers, routers remember which VC number connects to which interface, route all packets coming in with VC to corresponding interface until teardown. **Nothing** else can be sent to that interface.

Virtual Circuits (potentially skip)

- Modern networking is **connectionless** (each datagram routed individually) with packets being delivered at **best effort**, but what if we want additional guarantees? We can reserve a **flow/connection** from source to destination, with all packets routed along the flow
- Virtual circuits issued numbers, routers remember which VC number connects to which interface, route all packets coming in with VC to corresponding interface until teardown. **Nothing** else can be sent to that interface.
- Steps: (1) Sender initiates call, routers reserve interfaces, (2) receiver accepts incoming call, routers confirm reservation, (3) Sender receives setup confirmation, transmits data, (4) Once transmit done, teardown request sent, reservations released

Virtual Circuits (potentially skip)

- Modern networking is **connectionless** (each datagram routed individually) with packets being delivered at **best effort**, but what if we want additional guarantees? We can reserve a **flow/connection** from source to destination, with all packets routed along the flow
- Virtual circuits issued numbers, routers remember which VC number connects to which interface, route all packets coming in with VC to corresponding interface until teardown. **Nothing** else can be sent to that interface.
- Steps: (1) Sender initiates call, routers reserve interfaces, (2) receiver accepts incoming call, routers confirm reservation, (3) Sender receives setup confirmation, transmits data, (4) Once transmit done, teardown request sent, reservations released
- This sucks (**lots** of residual capacity along links left open), so it's not really used outside of old ATM networks.

Datagram Forwarding (Longest Prefix Matching)

- Goal: map destination IP to outgoing port

Datagram Forwarding (Longest Prefix Matching)

- Goal: map destination IP to outgoing port
- 2^{32} possible IP addresses (IPv4), so we can't keep a full table. Instead, map based on **prefix** (i.e. "All IPs of the form 105.76.x.x should go to interface 5"). **Intuition**: since IPs are hierarchical, nodes with similar IPs are probably pretty close, so the same interface should serve them well

Datagram Forwarding (Longest Prefix Matching)

- Goal: map destination IP to outgoing port
- 2^{32} possible IP addresses (IPv4), so we can't keep a full table. Instead, map based on **prefix** (i.e. "All IPs of the form 105.76.x.x should go to interface 5"). **Intuition**: since IPs are hierarchical, nodes with similar IPs are probably pretty close, so the same interface should serve them well
- To allow for more finegrained control, allow different prefixes, map packets to **longest** prefix that fully matches (ex: 240.128.x.x/5 is more specific than 128.x.x.x/1, so if both rules in table, choose first)

Routers: The Low-Level View

- Routers consist of processors (which control routing tables and act in millisecond time frames) and switching fabric (which just simply forwards packets along and acts in nanosecond time frames)
- On link-layer receive: using header values, lookup port to send to and queue for transmit. Traditionally, destination-based forwarding (only use destination IP address) is used, but **generalized forwarding** (based on any set of header values) can help at times.
- Switching fabric transfers packets between input + output links, goal is to have a high **switching rate** (rate at which inputs -> outputs can be transmit, measured as multiple of input/output line rate). Goal is with N inputs, switching rate of N times line rate

Switching Architectures

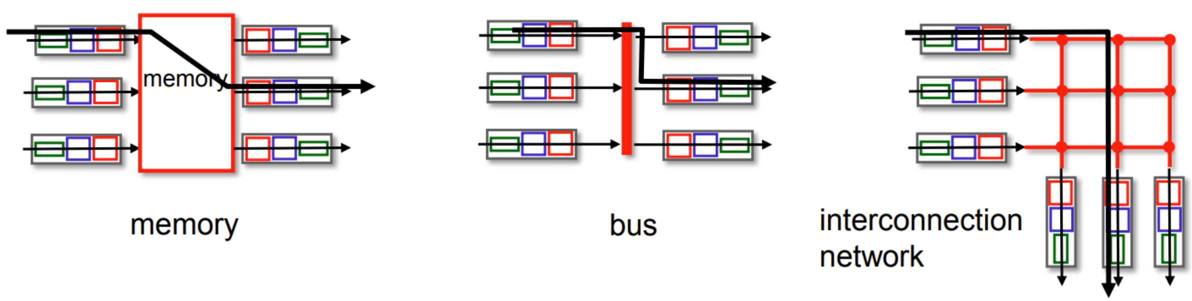
- **Memory**: Have input and output ports on a bus (single use). Load packets into memory, decide where to route, and then send. Super slow, as limited by memory bandwidth; 2 bus crossings per packet.

Switching Architectures

- **Memory**: Have input and output ports on a bus (single use). Load packets into memory, decide where to route, and then send. Super slow, as limited by memory bandwidth; 2 bus crossings per packet.
- **Bus**: One wire along which all inputs are connected to all outputs. Faster but still slow, because 1 packet out at a time.

Switching Architectures

- **Memory**: Have input and output ports on a bus (single use). Load packets into memory, decide where to route, and then send. Super slow, as limited by memory bandwidth; 2 bus crossings per packet.
- **Bus**: One wire along which all inputs are connected to all outputs. Faster but still slow, because 1 packet out at a time.
- **Others**: Crossbar (only fails when intersection between paths), multistaged switching (switch formed by switches), etc. We can also exploit parallelism by breaking packets up and reassembling them on transmit.



Forwarding and Buffering

- If switch fabric slower than input ports combined, queues may build up on input. Delay and loss due to buffer overflow possible.

Forwarding and Buffering

- If switch fabric slower than input ports combined, queues may build up on input. Delay and loss due to buffer overflow possible.
- Problem of **Head-of-the-Line** (HOL) blocking: even if we can send a later packet in our queue, we still have to wait for all earlier packets to be sent!

Forwarding and Buffering

- If switch fabric slower than input ports combined, queues may build up on input. Delay and loss due to buffer overflow possible.
- Problem of **Head-of-the-Line** (HOL) blocking: even if we can send a later packet in our queue, we still have to wait for all earlier packets to be sent!
- **Output port queuing**: If datagrams arive from fabric faster than they can be transmit (arrival rate via switching fabric > line speed), then we need a send buffer, and a **drop policy** to decide what to do when we have no space.

Forwarding and Buffering

- If switch fabric slower than input ports combined, queues may build up on input. Delay and loss due to buffer overflow possible.
- Problem of **Head-of-the-Line** (HOL) blocking: even if we can send a later packet in our queue, we still have to wait for all earlier packets to be sent!
- **Output port queuing**: If datagrams arive from fabric faster than they can be transmit (arrival rate via switching fabric > line speed), then we need a send buffer, and a **drop policy** to decide what to do when we have no space.
- **Scheduling discipline** chooses the packets to send first. We can force a priority scheduling scheme to ensure that certain services get best performance and are least affected by high load.

Forwarding and Buffering

- If switch fabric slower than input ports combined, queues may build up on input. Delay and loss due to buffer overflow possible.
- Problem of **Head-of-the-Line** (HOL) blocking: even if we can send a later packet in our queue, we still have to wait for all earlier packets to be sent!
- **Output port queuing**: If datagrams arive from fabric faster than they can be transmit (arrival rate via switching fabric > line speed), then we need a send buffer, and a **drop policy** to decide what to do when we have no space.
- **Scheduling discipline** chooses the packets to send first. We can force a priority scheduling scheme to ensure that certain services get best performance and are least affected by high load.
- **Buffer Management**: if new packet incoming and buffer full, either tail drop (drop arriving packet) or choose to drop packet on priority basis.

Packet Scheduling

- How to decide which packet to send on the link next

Packet Scheduling

- How to decide which packet to send on the link next
  - **FCFS**: transmit on the order of arrival to output port, can be done using FIFO queue

Packet Scheduling

- How to decide which packet to send on the link next
  - **FCFS**: transmit on the order of arrival to output port, can be done using FIFO queue
  - **Priority scheduling**: Keep separate queues for each priority, transmit from highest priority queue. FCFS within each priority class.

Packet Scheduling

- How to decide which packet to send on the link next
  - **FCFS**: transmit on the order of arrival to output port, can be done using FIFO queue
  - **Priority scheduling**: Keep separate queues for each priority, transmit from highest priority queue. FCFS within each priority class.
  - **Round Robin**: have multiple queues, but cyclically switch between classes to send from every time.

Packet Scheduling

- How to decide which packet to send on the link next
  - **FCFS**: transmit on the order of arrival to output port, can be done using FIFO queue
  - **Priority scheduling**: Keep separate queues for each priority, transmit from highest priority queue. FCFS within each priority class.
  - **Round Robin**: have multiple queues, but cyclically switch between classes to send from every time.
  - **Weighted Fair Queuing**: give each queue a weight, per cycle, spend $\frac{w_i}{\sum_j w_j}$ of the cycle on sending from this queue.

IP addressing and Subnets

- Each network interface (connection between host/router and physical link) given its own IP address (not true in IPv4 anymore but that was the goal)

IP addressing and Subnets

- Each network interface (connection between host/router and physical link) given its own IP address (not true in IPv4 anymore but that was the goal)
- **Subnets** are a set of devices connected without a router in between, share same IP prefix. Defined by any prefix length using **Classless InterDomain Routing** (CIDR)

IP addressing and Subnets

- Each network interface (connection between host/router and physical link) given its own IP address (not true in IPv4 anymore but that was the goal)
- **Subnets** are a set of devices connected without a router in between, share same IP prefix. Defined by any prefix length using **Classless InterDomain Routing** (CIDR)
- Subnet IP designationss of the form a.b.c.d/x where network mask x controls how many bits of prefix to check.

IP addressing and Subnets

- Each network interface (connection between host/router and physical link) given its own IP address (not true in IPv4 anymore but that was the goal)
- **Subnets** are a set of devices connected without a router in between, share same IP prefix. Defined by any prefix length using **Classless InterDomain Routing** (CIDR)
- Subnet IP designationss of the form a.b.c.d/x where network mask x controls how many bits of prefix to check.
- ISPs can seperate a large subnet allocation into a lot of smaller spaces with a wider mask- for example 200.100.16.0/20 can be split into 8 /23 subnets.

IP addressing and Subnets

- Each network interface (connection between host/router and physical link) given its own IP address (not true in IPv4 anymore but that was the goal)
- **Subnets** are a set of devices connected without a router in between, share same IP prefix. Defined by any prefix length using **Classless InterDomain Routing** (CIDR)
- Subnet IP designationss of the form a.b.c.d/x where network mask x controls how many bits of prefix to check.
- ISPs can seperate a large subnet allocation into a lot of smaller spaces with a wider mask- for example 200.100.16.0/20 can be split into 8 /23 subnets.
- IPs can either be allocated **statically** (manually configure on device) or **dynamically** (router assigns to next availiable IP in its subnet), through DHCP

IP addressing and Subnets

- Each network interface (connection between host/router and physical link) given its own IP address (not true in IPv4 anymore but that was the goal)
- **Subnets** are a set of devices connected without a router in between, share same IP prefix. Defined by any prefix length using **Classless InterDomain Routing** (CIDR)
- Subnet IP designationss of the form a.b.c.d/x where network mask x controls how many bits of prefix to check.
- ISPs can seperate a large subnet allocation into a lot of smaller spaces with a wider mask- for example 200.100.16.0/20 can be split into 8 /23 subnets.
- IPs can either be allocated **statically** (manually configure on device) or **dynamically** (router assigns to next availiable IP in its subnet), through DHCP
  - Client broadcasts request for DHCP server, which issues address. Client confirms use. Can also assign first-hop router, DNS server, network mask.

Network Address Translation (NAT)

- IPv4 not big enough to fit all of the devices that exist, so let’s instead have multiple devices on a subnet share an IP.

Network Address Translation (NAT)

- IPv4 not big enough to fit all of the devices that exist, so let's instead have multiple devices on a subnet share an IP.
- Uses transport-layer ports to seperate devices. Routers are given a NAT table to assign (public IP, port) to (private IP, port).

Network Address Translation (NAT)

- IPv4 not big enough to fit all of the devices that exist, so let’s instead have multiple devices on a subnet share an IP.
- Uses transport-layer ports to seperate devices. Routers are given a NAT table to assign (public IP, port) to (private IP, port).
- Routers translate (source, port) to (NAT, NAT port) on outgoing packets, and (NAT, NAT port) to (internal, port) on incoming packets.

Network Address Translation (NAT)

- IPv4 not big enough to fit all of the devices that exist, so let's instead have multiple devices on a subnet share an IP.
- Uses transport-layer ports to seperate devices. Routers are given a NAT table to assign (public IP, port) to (private IP, port).
- Routers translate (source, port) to (NAT, NAT port) on outgoing packets, and (NAT, NAT port) to (internal, port) on incoming packets.
- Controversial as this breaks layer separation, but extensively used as IPv4 addresses are expensive.

IPv6

- The “real” solution to IPv4 address exhaustion, 128 bit address but fewer additional headers, so 40-byte fixed length
- Not every device supports it, so sometimes we need to use IPv4 tunneling: simply encode your IPv6 packet with headers packet as a payload in an IPv4 packet. Set source to the node converting to IPv4, and destination to the node that is back on IPv6.

Routing Protocols

- **Main Goal**: determine “good” paths to send packets from sender to receiver through network of routers

Routing Protocols

- **Main Goal**: determine "good" paths to send packets from sender to receiver through network of routers
- Network can be visualized as a **graph** with nodes being routers and edges being links. Edge weights inversely related to bandwidth, related to congestion, etc. Optimal routing is the **least cost** path.

Routing Protocols

- **Main Goal**: determine “good” paths to send packets from sender to receiver through network of routers
- Network can be visualized as a **graph** with nodes being routers and edges being links. Edge weights inversely related to bandwidth, related to congestion, etc. Optimal routing is the **least cost** path.
- Algorithms can either be **static** (assume that network costs do not change much, so calculate costs once and leave alone) or **dynamic** (periodically update cost estimates/routes in response to link-cost changes).

Routing Protocols

- **Main Goal**: determine “good” paths to send packets from sender to receiver through network of routers
- Network can be visualized as a **graph** with nodes being routers and edges being links. Edge weights inversely related to bandwidth, related to congestion, etc. Optimal routing is the **least cost** path.
- Algorithms can either be **static** (assume that network costs do not change much, so calculate costs once and leave alone) or **dynamic** (periodically update cost estimates/routes in response to link-cost changes).
- Algorithms can either rely on **global** information (all routers have knowledge of network topology, even in areas that are far away) or be built to only use **decentralized** information (routers only know about their own physical connections, and gather everything else from each other)

Link state: Dijkstra's

Pseudocode

```
procedure LinkState(G, s)
    Initialize a table distances to all be ∞
    Initialize a priority queue pq
    for all s → t do
        Add ((s → t).wt, t, s) to pq
    while pq not empty do
        (dist, node, pred) ← pq.pop()
        Add (dist, node, pred) to distances
        for all node → v do
            if v not in distances then
                Add (dist + (node → v).wt, v, node) to pq (with a decrease-key
                operation if possible)
```

Link state: Dijkstra's

Pseudocode

```
procedure LinkState(G, s)
    Initialize a table distances to all be ∞
    Initialize a priority queue pq
    for all s → t do
        Add ((s → t).wt, t, s) to pq
    while pq not empty do
        (dist, node, pred) ← pq.pop()
        Add (dist, node, pred) to distances
        for all node → v do
            if v not in distances then
                Add (dist + (node → v).wt, v, node) to pq (with a decrease-key
                operation if possible)
```

- **Upsides**: gives you a full shortest path tree, total runtime fast in comparison to some distance vector algorithms, each node computes its own table so the worst error is a misestimation of link cost.

Link state: Dijkstra's

Pseudocode

procedure LinkState(G, s)
  Initialize a table `distances` to all be ∞
  Initialize a priority queue `pq`
  for all $s \to t$ **do**
    Add $((s \to t)$.`wt`, t, $s)$ to `pq`
  while `pq` not empty **do**
    (dist, node, pred)$\leftarrow$ `pq.pop`()
    Add (dist, node, pred) to `distances`
    for all `node` $\to v$ **do**
      if v not in `distances` **then**
        Add (dist $+$ (`node` $\to v$).`wt`, v, node) to `pq` (with a **decrease-key** operation if possible)

- **Upsides**: gives you a full shortest path tree, total runtime fast in comparison to some distance vector algorithms, each node computes its own table so the worst error is a misestimation of link cost.
- **Downsides**: needs global information, also oscillations are possible if two paths are nearly equal.

Distance Vector: Bellman-Ford

Update Rule

$$d(s,t) = \min_{s \to v}\left(d(v,t) + (s \to v).\texttt{wt}\right)$$

Distance Vector: Bellman-Ford

Update Rule

$$d(s,t) = \min_{s \to v}(d(v,t) + (s \to v).\texttt{wt})$$

- At each iteration, each node receives updates on connection advertisements from neighbors, then recomputes estimates and advertises new estimates to neighbors

Distance Vector: Bellman-Ford

Update Rule

$$d(s,t) = \min_{s \to v}(d(v,t) + (s \to v).\mathtt{wt})$$

- At each iteration, each node receives updates on connection advertisements from neighbors, then recomputes estimates and advertises new estimates to neighbors
- “Count to infinity” problem: *A* can give a lower estimate than expected given a path that uses *A*’s old link cost. Mitigated slightly by “poisoned reverse”: don’t advertise to node *X* paths that route to node *X*.

Distance Vector: Bellman-Ford

Update Rule

$$d(s,t) = \min_{s \to v}\left(d(v,t) + (s \to v).\texttt{wt}\right)$$

- At each iteration, each node receives updates on connection advertisements from neighbors, then recomputes estimates and advertises new estimates to neighbors
- “Count to infinity” problem: *A* can give a lower estimate than expected given a path that uses *A*’s old link cost. Mitigated slightly by “poisoned reverse”: don’t advertise to node *X* paths that route to node *X*.
- Errors propagate through network

Making Routing Scalable

- Too many nodes in the internet to calculate full tables

Making Routing Scalable

- Too many nodes in the internet to calculate full tables
- Split up routers into separate **autonomous systems** (UIUC is one)

Making Routing Scalable

- Too many nodes in the internet to calculate full tables
- Split up routers into separate **autonomous systems** (UIUC is one)
- Intra-AS routing can be chosen by the AS, as long as all routers in AS use same protocol.

Making Routing Scalable

- Too many nodes in the internet to calculate full tables
- Split up routers into separate **autonomous systems** (UIUC is one)
- Intra-AS routing can be chosen by the AS, as long as all routers in AS use same protocol.
- **Gateway routers** at the edge of an AS perform inter-domain routing.

Making Routing Scalable

- Too many nodes in the internet to calculate full tables
- Split up routers into separate **autonomous systems** (UIUC is one)
- Intra-AS routing can be chosen by the AS, as long as all routers in AS use same protocol.
- **Gateway routers** at the edge of an AS perform inter-domain routing.
- Forwarding tables calculated based on domain algorithm for intra-domain routing, both inter-domain and intra-domain for external destinations

Making Routing Scalable

- Too many nodes in the internet to calculate full tables
- Split up routers into separate **autonomous systems** (UIUC is one)
- Intra-AS routing can be chosen by the AS, as long as all routers in AS use same protocol.
- **Gateway routers** at the edge of an AS perform inter-domain routing.
- Forwarding tables calculated based on domain algorithm for intra-domain routing, both inter-domain and intra-domain for external destinations
- Intra-domain protocols/**Interior Gateway Protocols** (IGP) include **RIP** (distance vector), **OSPF** (Link state, with routers flooding entire AS with advertisements with message authentication, multicast, and hierarchical OSPF as “advanced” features), and whatever proprietary stuff Cisco does.

Routing between ASes: BGP

- **eBGP** obtains/shares subnet reachability information with other ASes, **iBGP** propagates internally

Routing between ASes: BGP

- **eBGP** obtains/shares subnet reachability information with other ASes, **iBGP** propagates internally
- eBGP gateways recieve advertisements of routes to other ASes of the form (AS-PATH, NEXT-HOP (specific router in the AS to send requests to)), takes shortest such path known.

Routing between ASes: BGP

- **eBGP** obtains/shares subnet reachability information with other ASes, **iBGP** propagates internally
- eBGP gateways recieve advertisements of routes to other ASes of the form (AS-PATH, NEXT-HOP (specific router in the AS to send requests to)), takes shortest such path known.
- Gateways propagate advertisements to internal nodes, other gateways in AS via iBGP, which then advertise to other ASes.

Routing between ASes: BGP

- **eBGP** obtains/shares subnet reachability information with other ASes, **iBGP** propagates internally
- eBGP gateways recieve advertisements of routes to other ASes of the form (AS-PATH, NEXT-HOP (specific router in the AS to send requests to)), takes shortest such path known.
- Gateways propagate advertisements to internal nodes, other gateways in AS via iBGP, which then advertise to other ASes.
- Routers can choose how to send to destination routes based on different criteria including shortest AS-PATH and “hot potato” routing (get it out of my AS as quickly as possible)

Link Layer: Introduction

- Goal is to transmit a packet over a medium. Each link may have its own protocol.

Link Layer: Introduction

- Goal is to transmit a packet over a medium. Each link may have its own protocol.
- Encapsulate datagram into **frames**, with **MAC address** used to identify source and destination of hop, provide reliable transmit between the two

Link Layer: Introduction

- Goal is to transmit a packet over a medium. Each link may have its own protocol.
- Encapsulate datagram into **frames**, with **MAC address** used to identify source and destination of hop, provide reliable transmit between the two
- Additional services: bit error correction, shared mediums, flow control (your 10Gbps ethernet card will need to send slower so that my 1Gbps ethernet adapter can recieve everything), half (only one of the two sides can transmit at a time) vs full-duplex (both sides can transmit at the same time)

Link Layer: Introduction

- Goal is to transmit a packet over a medium. Each link may have its own protocol.
- Encapsulate datagram into **frames**, with **MAC address** used to identify source and destination of hop, provide reliable transmit between the two
- Additional services: bit error correction, shared mediums, flow control (your 10Gbps ethernet card will need to send slower so that my 1Gbps ethernet adapter can recieve everything), half (only one of the two sides can transmit at a time) vs full-duplex (both sides can transmit at the same time)
- Bit error correction usually done via **pairity checking**: options include single bit, two dimensional pairity checking, Hamming coding, CRC. More reliable transmission mediums will use less complete correction means or will leave it out entirely.

Multiple Access Protocols

- Often, single shared broadcast channel (unswitched Ethernet, WiFi, etc.). If node receives two or more signals at a time, collision, data wasted, so we need a protocol to determine how we share a channel. Communication about channel sharing must be done via the channel itself.

Multiple Access Protocols

- Often, single shared broadcast channel (unswitched Ethernet, WiFi, etc.). If node receives two or more signals at a time, collision, data wasted, so we need a protocol to determine how we share a channel. Communication about channel sharing must be done via the channel itself.
- Goal is efficient sharing (if only one node wants to transmit, should use full channel rate, otherwise share at $\frac{R}{M}$ if M nodes transmitting), no central server to coordinate, and no synchronization.

Multiple Access Protocols

- Often, single shared broadcast channel (unswitched Ethernet, WiFi, etc.). If node receives two or more signals at a time, collision, data wasted, so we need a protocol to determine how we share a channel. Communication about channel sharing must be done via the channel itself.
- Goal is efficient sharing (if only one node wants to transmit, should use full channel rate, otherwise share at $\frac{R}{M}$ if M nodes transmitting), no central server to coordinate, and no synchronization.
- Three broad classes:
  - **Channel Partition**: Split channel up into smaller "pieces": TDMA for time division, FDMA for frequency division

Multiple Access Protocols

- Often, single shared broadcast channel (unswitched Ethernet, WiFi, etc.). If node receives two or more signals at a time, collision, data wasted, so we need a protocol to determine how we share a channel. Communication about channel sharing must be done via the channel itself.
- Goal is efficient sharing (if only one node wants to transmit, should use full channel rate, otherwise share at $\frac{R}{M}$ if M nodes transmitting), no central server to coordinate, and no synchronization.
- Three broad classes:
  - **Channel Partition**: Split channel up into smaller “pieces”: TDMA for time division, FDMA for frequency division
  - **Random Access**: When node has something to send, send it at full data rate. Protocol detects collisions and how to recover from them. Includes (slotted) ALOHA and CSMA/CD

Multiple Access Protocols

- Often, single shared broadcast channel (unswitched Ethernet, WiFi, etc.). If node receives two or more signals at a time, collision, data wasted, so we need a protocol to determine how we share a channel. Communication about channel sharing must be done via the channel itself.
- Goal is efficient sharing (if only one node wants to transmit, should use full channel rate, otherwise share at $\frac{R}{M}$ if M nodes transmitting), no central server to coordinate, and no synchronization.
- Three broad classes:
    - **Channel Partition**: Split channel up into smaller "pieces": TDMA for time division, FDMA for frequency division
    - **Random Access**: When node has something to send, send it at full data rate. Protocol detects collisions and how to recover from them. Includes (slotted) ALOHA and CSMA/CD
    - "Taking Turns": Nodes take turns but nodes that have more to send can take longer turns. Includes token-passing (control token giving the right to send passed between nodes)

(Slotted) ALOHA

- If all frames same size and time divided into equal sized slots, when node receives frame, transmit immediately. If collision, retransmit with probability p^k (with k consecutive failures) for each subsequent frame until success.

(Slotted) ALOHA

- If all frames same size and time divided into equal sized slots, when node receives frame, transmit immediately. If collision, retransmit with probability p^k (with k consecutive failures) for each subsequent frame until success.
- If N nodes active, the successful transmit probability is $Np(1-p)$, so for large N, successful transmit only happens 37% of the time!

(Slotted) ALOHA

- If all frames same size and time divided into equal sized slots, when node receives frame, transmit immediately. If collision, retransmit with probability p^k (with k consecutive failures) for each subsequent frame until success.
- If N nodes active, the successful transmit probability is $Np(1-p)$, so for large N, successful transmit only happens 37% of the time!
- Unslotted ALOHA doesn't even use slots, synchronization. Performance even worse: 18% expected success rate.

CSMA/CD

- Listen on channel before transmit. If channel busy, wait until free, then transmit full frame. On collision, abort and choose wait time at random between 0 and $2^m - 1$ for m consecutive collisions
- Efficiency trends to 1, so much better than ALOHA

MAC Addressing

- 48 bit address that's hardcoded, can move between LANs using same address. Each node in a LAN has an (IP address, MAC address, TTL) table. If MAC address not known, node broadcasts ARP query, requested device responds in unicast with its address
- Requests outside of LAN are targeted to router, which then will transmit to the outside

Ethernet

- Topologies include **bus** (all nodes in same collision domain) and **star** (switch in center, no collisions)
- Connectionless (no handshaking), unreliable (no acknowledgement), CSMA/CD with binary backoff for MAC
- Destination and source MAC addresses, higher layer protocol in header, CRC at end for bit error correction

Ethernet

- Topologies include **bus** (all nodes in same collision domain) and **star** (switch in center, no collisions)
- Connectionless (no handshaking), unreliable (no acknowledgement), CSMA/CD with binary backoff for MAC
- Destination and source MAC addresses, higher layer protocol in header, CRC at end for bit error correction
- **Switches** take an active role by selectively forwarding frame to outgoing links, using CSMA/CD to access segment. Hosts are unaware of the presence of switches
- Switches are **self learning**: they have a switching table as to what MAC addresses are accessible from which interface. On request coming from node, add its address/interface to table If request designated for known node, send on corresponding interface, else flood.

Wireless

- A bunch of new considerations that don't exist on wired media: can share frequencies of channel, may want to swap between channnels
- Larger issue: signal decay. On wired, you can usually detect all activity over the medium, but on wireless there might be a "hidden terminal" that you cannot detect but can interfere with reciever's ability to recieve your message. Also "exposed terminal" that you can detect but wouldn't interfere with your transmission because the reciever is out of their range.
- Solution: CSMA/CA: Send "Requests to Send" (RTS) messages when wanting to send to someone else, wait for "Clear to Send" (CTS) from reciever. Nodes only need to not send when they hear a CTS. Still possibility of collisions with data and RTS/CTS packets, but those are much shorter so probability lower.

Code Multiplexing and CDMA

- All users share same frequency, but each user has own “chipping” sequence (i.e., code) to encode data. This allows multiple users to “coexist” and transmit simultaneously with minimal interference (if codes are “orthogonal”)
- **Encoding**: data $\times$ chipping sequence
- **Decoding**: encoded data $\cdot$ chipping sequence

Code Multiplexing and CDMA

- All users share same frequency, but each user has own "chipping" sequence (i.e., code) to encode data. This allows multiple users to "coexist" and transmit simultaneously with minimal interference (if codes are "orthogonal")
- **Encoding**: data $\times$ chipping sequence
- **Decoding**: encoded data $\cdot$ chipping sequence
- **Advantages**: Shares bandwidth without any synchronization needed. Protected well against interference
- **Disadvantages**: You need to represent vectors in $\mathbb{R}^k$ for k orthogonal vectors, so limited user data rates.

Security

- **Threat Model**: Attacker has access somewhere along the channel that sender and receiver both have. We want to make sure that attacker can't eavesdrop on messages, modify message content, impersonate sender, etc.

Security

- **Threat Model**: Attacker has access somewhere along the channel that sender and receiver both have. We want to make sure that attacker can't eavesdrop on messages, modify message content, impersonate sender, etc.
- More formally, we want to maintain **confidentiality** (only sender and reciever should know what's in the message), **integrity** (no one can modify the message without being detected), and **accessibility** (transmitter should always be able to send to reciever) of data. Also, we want to ensure authentication (receiver should know that transmitter actually sent the data and not someone else).

Security

- **Threat Model**: Attacker has access somewhere along the channel that sender and receiver both have. We want to make sure that attacker can't eavesdrop on messages, modify message content, impersonate sender, etc.
- More formally, we want to maintain **confidentiality** (only sender and reciever should know what's in the message), **integrity** (no one can modify the message without being detected), and **accessibility** (transmitter should always be able to send to reciever) of data. Also, we want to ensure authentication (receiver should know that transmitter actually sent the data and not someone else).
- We'll achieve it through processing data through an **encryption algorithm**. Given some message m, we want there to be a K_e, K_d s.t. $K_d(K_e(m)) = m$, but it's hard to recover m without some information internal to K_d

Symmetric-Key Encryption: AES

- In a **symmetric-key** scheme, $K_e = K_d$ (or really that K_e and K_d use the same internal information, even if their implementation details are different).

Symmetric-Key Encryption: AES

- In a **symmetric-key** scheme, $K_e = K_d$ (or really that K_e and K_d use the same internal information, even if their implementation details are different).
- Main implementation is **AES**, which sequentially processes 128 bit "blocks" with some incredibly complicated algorithm that is really hard to reverse without knowing the key. Take ECE 407 if you want to know (some of) the details, but *how* it works won't be tested at all. We haven't found a proof that AES is computationally costly to decrypt without a key, but it's survived 20 years of attacks, so we're pretty sure.

Symmetric-Key Encryption: AES

- In a **symmetric-key** scheme, $K_e = K_d$ (or really that K_e and K_d use the same internal information, even if their implementation details are different).
- Main implementation is **AES**, which sequentially processes 128 bit "blocks" with some incredibly complicated algorithm that is really hard to reverse without knowing the key. Take ECE 407 if you want to know (some of) the details, but *how* it works won't be tested at all. We haven't found a proof that AES is computationally costly to decrypt without a key, but it's survived 20 years of attacks, so we're pretty sure.
- Unfortunately, this isn't enough on its own. We don't have a way to transmit decryption keys over an insecure channel ("mailman"/"padlock" problem), and the keys are shared, so this won't work for authentication.

Public-Key Encryption: RSA

- Goal is to have two keys K_- and K_+, such that $K_+(K_-(m)) = K_-(K_+(m)) = m$. We can obtain K_+ from K_- easily, but not the other way.

Public-Key Encryption: RSA

- Goal is to have two keys K_- and K_+, such that $K_+(K_-(m)) = K_-(K_+(m)) = m$. We can obtain K_+ from K_- easily, but not the other way.
- Anyone can send publicly undecipherable messages to the private key-holder by encoding them in K_+, and the private key holder can send authenticated (but public) messages by encoding them in K_-

Public-Key Encryption: RSA

- Goal is to have two keys K_{-} and K_{+}, such that $K_{+}(K_{-}(m)) = K_{-}(K_{+}(m)) = m$. We can obtain K_{+} from K_{-} easily, but not the other way.
- Anyone can send publicly undecipherable messages to the private key-holder by encoding them in K_{+}, and the private key holder can send authenticated (but public) messages by encoding them in K_{-}
- **Main Implementations**: ElGamal, RSA. How RSA works relies on a lot of number theory and the fact that prime factorization is hard. I doubt that questions would be asked on an exam about them (take ECE 407 to learn them).

Public-Key Encryption: RSA

- Goal is to have two keys K_- and K_+, such that $K_+(K_-(m)) = K_-(K_+(m)) = m$. We can obtain K_+ from K_- easily, but not the other way.
- Anyone can send publicly undecipherable messages to the private key-holder by encoding them in K_+, and the private key holder can send authenticated (but public) messages by encoding them in K_-
- **Main Implementations**: ElGamal, RSA. How RSA works relies on a lot of number theory and the fact that prime factorization is hard. I doubt that questions would be asked on an exam about them (take ECE 407 to learn them).
- Public-key encryption is slow in comparison to symmetric-key encryption, so we often only send AES encryption keys and hashes (one-way function that returns a fixed-length bitstring) of messages under RSA

Public-Key Encryption: RSA

- Goal is to have two keys K_- and K_+, such that $K_+(K_-(m)) = K_-(K_+(m)) = m$. We can obtain K_+ from K_- easily, but not the other way.
- Anyone can send publicly undecipherable messages to the private key-holder by encoding them in K_+, and the private key holder can send authenticated (but public) messages by encoding them in K_-
- **Main Implementations**: ElGamal, RSA. How RSA works relies on a lot of number theory and the fact that prime factorization is hard. I doubt that questions would be asked on an exam about them (take ECE 407 to learn them).
- Public-key encryption is slow in comparison to symmetric-key encryption, so we often only send AES encryption keys and hashes (one-way function that returns a fixed-length bitstring) of messages under RSA
- Still can't tell who someone is unless you know for a fact that their public key is something specific.Have a trusted source (**Certificate Authority**, CA) store this information.

Practice Questions I

1. Suppose I implement CDMA/CD, but instead of picking a random pause, I deterministically pause for 2^i frames. Will I run into problems if a single machine uses my implementation? What about if many machines are using my implementation on the same ethernet link?

Practice Questions I

1. Suppose I implement CDMA/CD, but instead of picking a random pause, I deterministically pause for 2^i frames. Will I run into problems if a single machine uses my implementation? What about if many machines are using my implementation on the same ethernet link?
2. Argue for or against: In a public-key encryption scheme, any message sent that guarantees the authenticity of the sender also guarantees its integrity.

Practice Questions I

1. Suppose I implement CDMA/CD, but instead of picking a random pause, I deterministically pause for 2^i frames. Will I run into problems if a single machine uses my implementation? What about if many machines are using my implementation on the same ethernet link?
2. Argue for or against: In a public-key encryption scheme, any message sent that guarantees the authenticity of the sender also guarantees its integrity.
3. Give an example of a graph with edge costs where there is still a count-to-infinity issue, even with poisoned reverse, after an edge is removed.

Practice Questions I

1. Suppose I implement CDMA/CD, but instead of picking a random pause, I deterministically pause for 2^i frames. Will I run into problems if a single machine uses my implementation? What about if many machines are using my implementation on the same ethernet link?
2. Argue for or against: In a public-key encryption scheme, any message sent that guarantees the authenticity of the sender also guarantees its integrity.
3. Give an example of a graph with edge costs where there is still a count-to-infinity issue, even with poisoned reverse, after an edge is removed.
4. I don't want the public internet to know that I'm sending data to a particular port on a machine, so I make a new transport-layer protocol which is exactly the same as UDP but encrypts some fields of transport-layer headers, including the port number using RSA. Is my new transport protocol compatible with the modern internet?

Practice Questions I

1. Suppose I implement CDMA/CD, but instead of picking a random pause, I deterministically pause for 2^i frames. Will I run into problems if a single machine uses my implementation? What about if many machines are using my implementation on the same ethernet link?
2. Argue for or against: In a public-key encryption scheme, any message sent that guarantees the authenticity of the sender also guarantees its integrity.
3. Give an example of a graph with edge costs where there is still a count-to-infinity issue, even with poisoned reverse, after an edge is removed.
4. I don't want the public internet to know that I'm sending data to a particular port on a machine, so I make a new transport-layer protocol which is exactly the same as UDP but encrypts some fields of transport-layer headers, including the port number using RSA. Is my new transport protocol compatible with the modern internet?
5. I want to add a link to my network graph that has a negative weight. My AS runs a distance vector protocol. Will this cause problems for the routers? If not, how quickly will distances using my negative link update on other servers?

Practice Questions II

6. If an attacker placed a malicious router in a datacenter, could they do more damage if the datacenter was running a distance vector or link-state protocol?

Practice Questions II

6. If an attacker placed a malicious router in a datacenter, could they do more damage if the datacenter was running a distance vector or link-state protocol?
7. Suppose I have a router with a bus switching fabric that can have packets cross the switching fabric twice as fast as any input port or output port. If I have 3 input ports and 3 output ports, under what load scenarios would I drop packets?

Practice Questions II

6. If an attacker placed a malicious router in a datacenter, could they do more damage if the datacenter was running a distance vector or link-state protocol?
7. Suppose I have a router with a bus switching fabric that can have packets cross the switching fabric twice as fast as any input port or output port. If I have 3 input ports and 3 output ports, under what load scenarios would I drop packets?
8. Suppose I have N devices on an wired link using slotted ALOHA that always want to transmit data. Will my link utilization be higher with a lower N?

Practice Questions II

6. If an attacker placed a malicious router in a datacenter, could they do more damage if the datacenter was running a distance vector or link-state protocol?
7. Suppose I have a router with a bus switching fabric that can have packets cross the switching fabric twice as fast as any input port or output port. If I have 3 input ports and 3 output ports, under what load scenarios would I drop packets?
8. Suppose I have N devices on an wired link using slotted ALOHA that always want to transmit data. Will my link utilization be higher with a lower N?
9. Why do you only need to stop transmission on a channel upon recieving a CTS and not also an RTS?

Practice Questions II

6. If an attacker placed a malicious router in a datacenter, could they do more damage if the datacenter was running a distance vector or link-state protocol?
7. Suppose I have a router with a bus switching fabric that can have packets cross the switching fabric twice as fast as any input port or output port. If I have 3 input ports and 3 output ports, under what load scenarios would I drop packets?
8. Suppose I have *N* devices on an wired link using slotted ALOHA that always want to transmit data. Will my link utilization be higher with a lower *N*?
9. Why do you only need to stop transmission on a channel upon recieving a CTS and not also an RTS?
10. Explain why a network with a lot of servers but whose activity is mostly transmitting small “keep alive” packets would achieve better performance on CSMA/CD over CSMA/CA

Practice Questions II

6. If an attacker placed a malicious router in a datacenter, could they do more damage if the datacenter was running a distance vector or link-state protocol?
7. Suppose I have a router with a bus switching fabric that can have packets cross the switching fabric twice as fast as any input port or output port. If I have 3 input ports and 3 output ports, under what load scenarios would I drop packets?
8. Suppose I have N devices on an wired link using slotted ALOHA that always want to transmit data. Will my link utilization be higher with a lower N?
9. Why do you only need to stop transmission on a channel upon recieving a CTS and not also an RTS?
10. Explain why a network with a lot of servers but whose activity is mostly transmitting small "keep alive" packets would achieve better performance on CSMA/CD over CSMA/CA
11. If I design a router with a total input capacity equal to the switching speed equal to the total output capacity, do I need to worry about buffers overflowing?

Practice Questions II

6. If an attacker placed a malicious router in a datacenter, could they do more damage if the datacenter was running a distance vector or link-state protocol?
7. Suppose I have a router with a bus switching fabric that can have packets cross the switching fabric twice as fast as any input port or output port. If I have 3 input ports and 3 output ports, under what load scenarios would I drop packets?
8. Suppose I have *N* devices on an wired link using slotted ALOHA that always want to transmit data. Will my link utilization be higher with a lower *N*?
9. Why do you only need to stop transmission on a channel upon recieving a CTS and not also an RTS?
10. Explain why a network with a lot of servers but whose activity is mostly transmitting small “keep alive” packets would achieve better performance on CSMA/CD over CSMA/CA
11. If I design a router with a total input capacity equal to the switching speed equal to the total output capacity, do I need to worry about buffers overflowing?
12. Why do we need IP addresses? Why can’t we just route to MAC addresses?

Feedback

http://go.acm.illinois.edu/cs438_final_feedback