University of Houston

Introduction to Computer Networks

COSC 6377

Midterm Review

Author
K.M. Hourani

Based on Notes By
Dr. Omprakash Gnawali

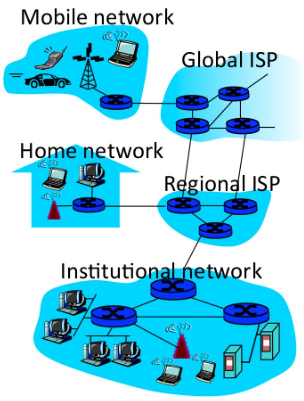
March 10, 2021

Contents

1 Intro

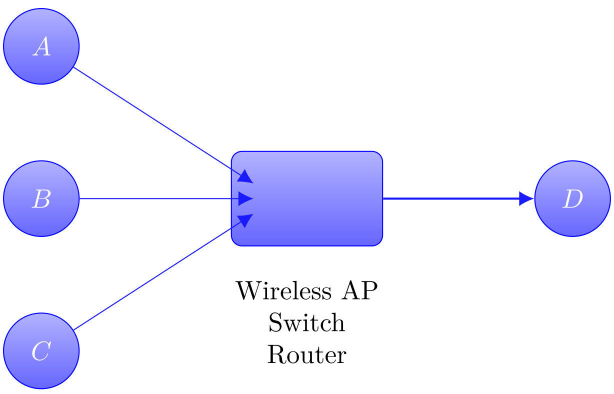
1.1 The Internet

- Collection of nodes, wired and wireless technology connecting these nodes, applications and services
- Types of nodes
  - Desktops and Laptop
  - Servers
  - TV/Refrigerator
  - Cellphones



- Goal: Connect all the nodes to each other
- Solutions
  - $\binom{n}{2} = \mathcal{O}(n^2)$ cables
  - Sharing the links
    - Circuit Switching
    - Packet Switching
- Packet
  - Collection of bits to transfer across a network
  - Think: envelope and its contents
- Circuit
  - Pre-allocated path/resource

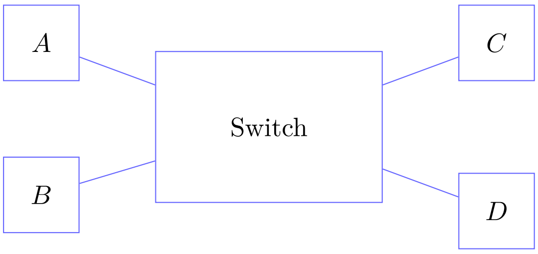
1.2 Packet vs. Circuit Switching



1.3 Circuit Switching

- Setup the connection or resource
  - Schedule (e.g., TDMA)

- State in the network



Time	Circuit
$T, 3T, 5T, \ldots$	$A - D$
$2T, 4T, 6T, \ldots$	$B - C$

- Natural for predictable data races
- Can guarantee certain level of services
- Can be inefficient for many applications

1.4 Some Circuit Switching Techniques

- Time
  - Reserve to use the link at a given schedule
  - Read: https://en.wikipedia.org/wiki/Time-division_multiplexing
- Frequency
  - Reserve to use certain frequencies (channel)
  - Read: https://en.wikipedia.org/wiki/Frequency-division_multiplexing

1.5 Packet Switching

- Wire is selected for each packet
- No network **state**
- Supports unpredictable/bursty traffic pattern
- Higher link utilization
- No guarantees but good enough for most applications

https://en.wikipedia.org/wiki/Packet_switching

1.6 Summary

- Packet Switching
  - Plus: more sharing (more efficient)
  - Minus: no service guarantee
- Circuit Switching
  - Plus: service guarantee
  - Minus: less sharing (less efficient)
- Every day examples
  - Road network

1.7 Describing a Network

- How to describe how well a network is working?
  - Metrics
- Performance metrics
  - Throughput
  - Latency
  - Reliability

1.8 Throughput

- How many bytes can we send through in a given time?
  - Bytes per second
  - How many bits/s in kbps?

    - Read: https://en.wikipedia.org/wiki/Data-rate_units
- Useful bytes transferred vs. overhead
    - Goodput
    - Everyday example: car vs. passenger

https://en.wikipedia.org/wiki/Throughput

1.9 Latency

- How long does it take for one bit to travel from one end to the other end?
    - ms, s, minutes, etc.
- Typical latencies
    - Speed of light
    - Why is web browsing latency in seconds?

1.10 Relation between Latency and Throughput

- Characterize the latency and throughput of
    - Oil Tanker –
    - Aircraft –
    - Car –
    - Tractor Trailer –
- Which metrics matter most for these applications?
    - Netflix
    - Skype
    - Amazon
    - Facebook

1.11 Reliability

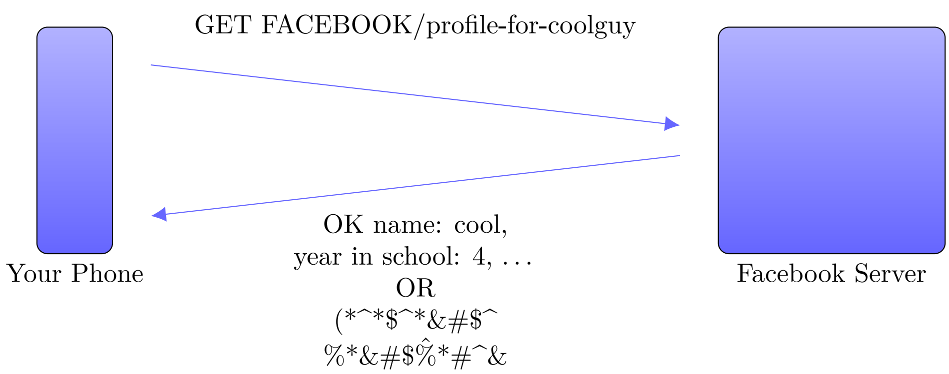
- How often does a network fail?
- How often do packets drop?
    - Damage (corruption)
    - Drops in the queues
- How persistent are failures?
- Typical metrics
    - uptime percentage
    - packet or bit loss rate

1.12 Protocols

- Agreed-upon rules, format, and meaning for message exchange
- Let's examine this sequence:
    - Hellow
    - How are you?
    - Fine.

https://en.wikipedia.org/wiki/Communication_protocol

1.13 Network Protocols



What are the rules, format, and meaning in this message exchange?

1.14 Protocols and Standards

- How can your phone (HTC running Android) access Facebook (runs on UNIX-like OS on big servers)?
- Using standard protocol enables interoperation
- Who standardizes the protocols?

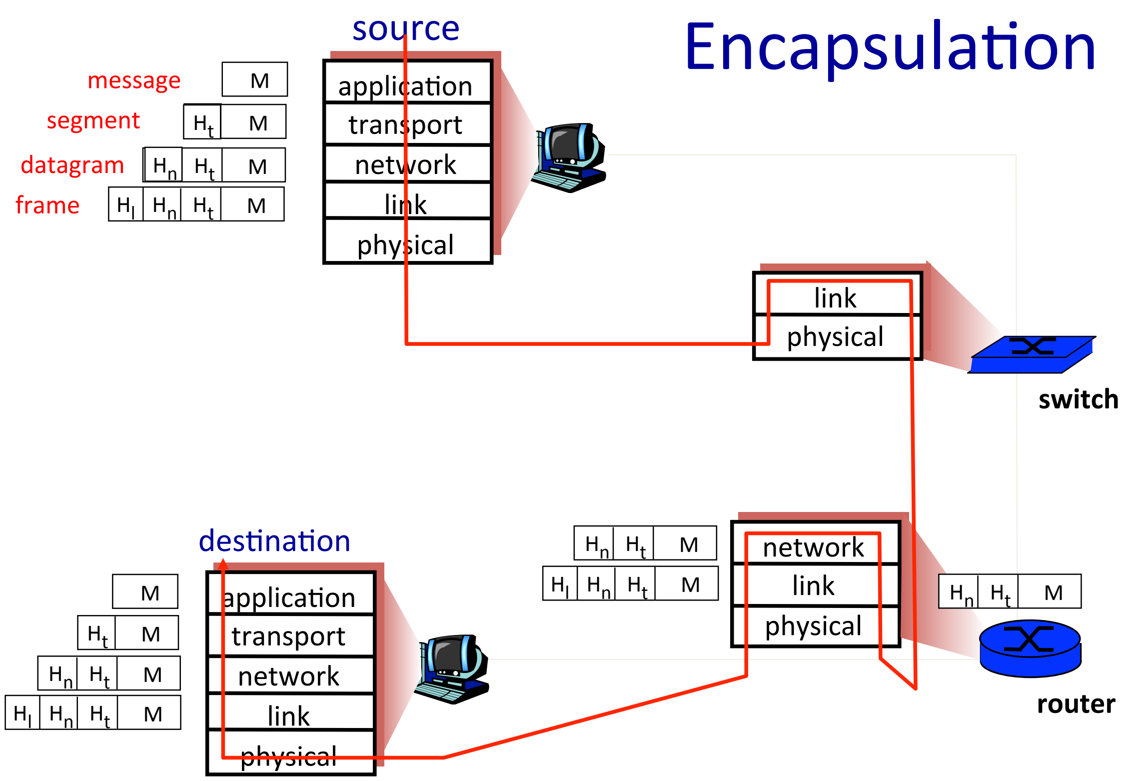
1.15 Protocol Layers

- Lower level to higher level message exchange
  - Organize the functionalities
  - Abstractions in services used and provided
- 5-7 layers depending on who you talk to
  - Physical, Link, Network, Transport, Application
- Should a smartphone app developer worry about
  - Voltages being applied on the wire
  - If the underlying media uses packet or circuit switching

https://en.wikipedia.org/wiki/Protocol_stack

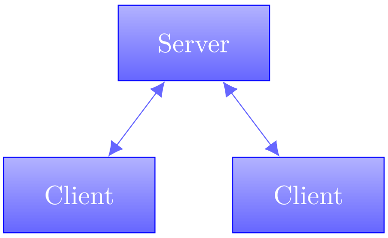
1.16 Encapsulation

- Think of how paperwork is processed in a university
  - Each person processes and adds some information to it and passes it along
- On the transmitter, the lower layers include the message from upper layers, add their own information, and send it along
- On the receiver: reverse

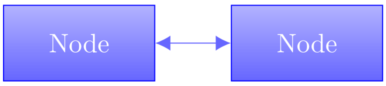


2 Network Applications and Socket Programming

2.1 Network Applications



Client-Server



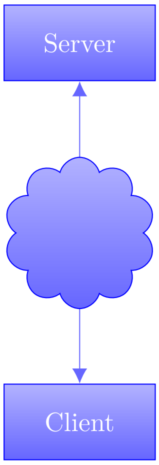
Peer-to-Peer

Hybrid

https://en.wikipedia.org/wiki/Peer-to-peer

2.2 Inter-Application Communication

- Need a way to send and receive messages
- Inter-process communication
- Need naming, routing, transport
- Transport using TCP and UDP
  - On top of IP

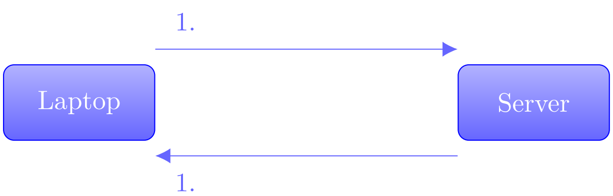


2.3 Application Protocols

- Messages between processes, typically encapsulated within TCP or UDP
- Need agreement between
  - Sending process
  - Receiving process

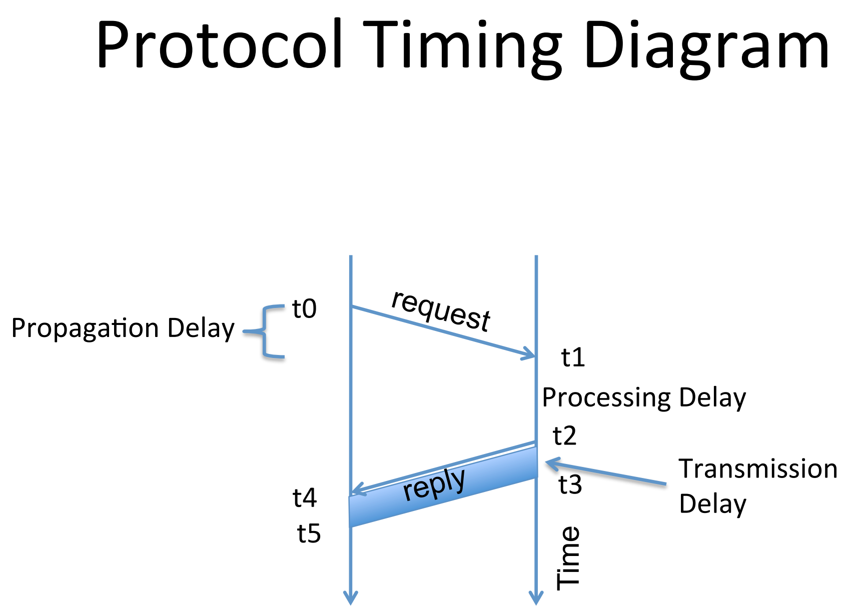
2.4 Network Time Service

Client-server or peer-to-peer?



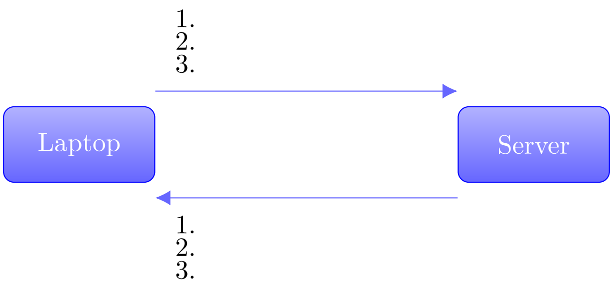
Atomic clock
facility

2.5 Protocol Timing Diagram



2.6 Cloud-based File Backup Application

- Client-server or peer-to-peer?
- Where do the applications run?
- Who/how to run these applications?
- What messages are exchanged?

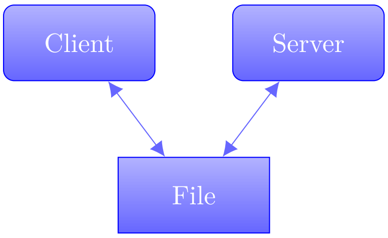


2.7 Socket Programming

2.8 Using TCP/IP

- How can applications use the network?
- Sockets API
  - Originall from BS, widely implemented (*BSD, Linux, Mac OS X, Windows, . . .)
  - Higher-level APIs build on them
- After basic setup, much like files

One could test network protocols with read/write on a file



2.9 System Calls

- Problem: how to access resources other than the CPU
  - Disk, netowrk, terminal, other processes
  - CPU prohibits instructions that would access devices
  - Only privileged OS kernel can access devices
- Kernel supplies well-defined system call interface
  - Applications request I/O oeprations through syscalls
  - Set up syscall arguments and trap to kernel
  - Kernel performs operation and returns result
- Higher-level functions built on syscall interface
  - `printf`, `scanf`, `gets`, all user-level code

2.10 File Descriptors

- Most I/O in Unix done through **file descriptors**
  - Integer **handles** to per-process table in kernel
- `int open(char *path, int flags, ...);`
- Returns file descriptor, used for all I/O to file

https://en.wikipedia.org/wiki/File_descriptor

2.11 Error Returns

- What if `open` fails? Return `-1` (invalid file descriptor)
- Most system calls return `-1` on failure
  - Specific type of error in gobal `int errno`
- `#include <sys/errno.h>` for possible values
  - `2` = `ENOENT` "no such file or directory"
  - `13` = `EACCES` "permission denied"

2.12 Some operations on File Descriptors

- `ssize_t read(int fd, void* buf, int nbytes);`
  - Returns number of bytes read
  - Returns `0` bytes at end of file, or `-1` on error
- `ssize_t write(int fd, void* buf, int nbytes);`
  - Returns number of bytes written, `-1` on error
- `off_t lseek(int fd, off_t offset, int whences);`
  - `whence`: `SEEK_SET`, `SEEK_CUR`, `SEEK_END`
  - returns new offset, or `-1` on error
- `int close(int fd);`

2.13 Sockets: Communincation Between Machines

- Network sockets are file descriptors too
- Datagram sockets: unreliable message delivery
  - With IP, gives you UDP
  - Send atomic messages, which may be reordered or lost
  - Special system calls to read/write: `send`/`recv`
- Stream sockets: bi-directional pipes
  - With IP, gives you TCP
  - Bytes written on one end read on another
  - Reads may not return full amount requested, must reread

2.14 System calls for using TCP

	Client	Server
1.		`socket` – make socket
2.		`bind` – assign address, port
3.		`listen` – listen for clients
4.	`socket` – make socket	
5.	`bind` – assign address[1]	
6.	`connect` – connect to listening socket	
7.		`accept` – accept connection

2.15 Socket Naming

- Naming of TCP and UDP communication endpoints
  - IP address specifies host (129.7.240.18)
  - 16-bit port number demultiplexes within host
  - Well-known services listen on standard ports (e.g. ssh – 22, http – 8, see `/etc/services` for list)
  - Clients connect from arbitrary ports to well-known ports
- A connection is named by 5 components
  - Protocol, local IP, local port, remote IP, remote port
  - TCP requires connected sockets, but not UDP

2.16 Socket Address Structures

- Socket interface supports multiple network types
- Most calls take a generic `sockaddr`:

```
struct sockaddr {
    uint16_t sa_family;   /* address family */
    char     sa_data[14]; /* protocol-specific addr */
};
```

- e.g. `int connect(int s, struct sockaddr* srv, socklen_t addrlen);`
- Cast `sockaddr*` from protocol-specific struct, e.g.

```
struct addr_in {
    short   sin_family;        /* = AF_INET */
    u_short sin_port;          /* = htons (PORT) */
    struct  in_addr sin_addr; /*32-bit IPV4 addr */
    char   in_zero[8];
};
```

2.17 Dealing with Address Types

- All values in network byte order (Big Endian)
  - `htonl()`, `htons()`: host to network, 32 and 16 bits
  - `ntohl()`, `ntohs()`: network to host, 32 and 16 bits
  - Remember to always convert!
- All address types begin with family
  - `sa_family` in `sockaddr` tells you the actual type
- Not all addresses are the same size
  - e.g. `struct sockaddr_in6` is typically 28 bytes, yet generic `struct sockaddr` is only 16 bytes
  - so most calls require passing around socket length
  - new `sockaddr_storage` is big enough

2.18 Client Skeleton (IPv4)

```
struct sockaddr_in {
    short   sin_family; /* = AF_INET */
```

[1] This call to bind is optional, connect can choose address and port

```
        u_short sin_port;   /* = htons (PORT) */
        struct  in_addr sin_addr;
        char    sin_zero[8];
    } sin;

    int s = socket (AF_INET, SOCK_STREAM, 0);
    memset(&sin, sizeof(sin), 0);
    sin.sin_family = AF_INET;
    sin.sin_port = htons(13); /* daytime port */
    sin.sin_addr.s_addr = htonl(IP_ADDRESS);
    connect(s, (sockaddr*)&sin, sizeof(sin));
    while ((n = read(s, buf, sizeof(buf))) > 0) {
        write(1, buf, n);
    }
```

2.19 Server Skeleton (IPv4)

```
    int s = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in sin;
    memset(&sin, sizeof(sin), 0);
    sin.sin_family = AF_INET;
    sin.sin_port = htons(9999);
    sin.sin_addr.s_addr = htonl(INADDR_ANY);
    bind(s, (struct sockaddr*)&sin, sizeof(sin));
    listen(s, 5);
    while (true) {
        socklen_t len = sizeof (sin);
        int cfd = accept(s, (struct sockaddr*)&sin, &len);
        /* cfd is new connection; you never read/write s */
        do_something_with(cfd);
        close(cfd);
    }
```

2.20 Looking up socket address with `getaddrinfo`

```
struct addrinfo hints, *ai;
int err;
memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;     /* or AF_INET or AF_INET6 */
hints.ai_socktype = SOCK_STREAM; /* or SOCK_DGRAM for UDP */

err = getaddrinfo("www.brown.edu", "http", &hints, &ai);
if (err) {
    fprintf (stderr, "%s\n", gia_strerror (err));
} else {
    /* ai->ai_family = address type (AF_INET or AF_INET6) */
    /* ai->ai_addr = actual address cast to (sockaddr *) */
    /* ai->ai_addrlen = length of actual address */
    freeaddrinfo (ai); /* must free when done! */
}
```

2.21 `getaddrinfo()` [RFC3493]

- Protocol-independent node name to address translation
  - Can specify port as a service name or number
  - May return multiple addresses
  - You must free the structure with `freeaddrinfo`

- Other useful functions to know about
  - `getnameinfo` – lookup hostname based on address
  - `inet_ntop` – convert IPv4 or 6 address to printable
  - `inet_prton` – convert string to IPv4 or 6 address

2.22 EOF in more detail

- What happens at the end of store?
  - Server receives EOF, renames file, responds OK
  - Client reads OK, **after** sending EOF: didn't close `fd`
- `int shutdown(int fd, int how);`
  - Shuts down a socket without closing the file descriptor
  - how: `0` = read, `1` = write, `2` = both
  - Note 1: applies to **socket**, not descriptor, so copies of descriptor (through fork or dup) affected
  - Note 2: with TCP, can't detect if other side shuts down for reading

2.23 Using UDP

- Call socket with `SOCK_DGRAM`, bind as before
- New calls for sending/receiving individual packets
  - ```
    sendto(int s, const void* msg, int len, int flags,
              const struct sockaddr* to, socklen_t tolen);
    ```
  - ```
    recvfrom(int s, void* buf, int len, int flags,
               struct sockaddr *from, socklen_t* fromlen);
    ```
  - Must send/get peer address with each packet
- Can use UDP in connected mode (why?)
  - connect assigns remote address
  - `send`/`recv` syscalls, like `sendto`/`recvfrom`, without last two arguments

2.24 Serving Multiple Clients

- A server may block when talking to a client
  - Read or write of a socket connected to a slow client can block
  - Server may be busy with CPU
  - Server might be blocked waiting for disk I/O
- Concurrency through multiple processes
  - Accept, fork, close in parent; child services request
- Advantages of one process per client
  - Doesn't block on slow clients
  - May use multiple cores
  - Can keep disk queues full for disk-heavy workloads

2.25 Threads

- One process per client has disadvantages:
  - High overhead – fork + exit $\approx 100\mu$sec
  - Hard to share state across clients
  - Maximum number of processes limited
- Can use threads for concurrency
  - Data races and deadlocks make programming tricky
  - Must allocate one stack per request
  - Many thread implementations block on some I/O or have heavy thread-switch overhead

Rough equivalents to `fork()`, `waitpid()`, `exit()`, `kill()`, plus locking primitives.

2.26 Non-blocking I/O

- `fcntl` sets `O_NONBLOCK` flag on descriptor

```
int n;
if ((n = fcntl(s, F_GETFL)) >= 0) {
    fcntl(s, F_SETFL, n | O_NONBLOCK);
}
```

- Non-blocking semantics of system calls:

  - read immediately returns -1 with errno EAGAIN if no data
  - write may not write all data, or may return EAGAIN
  - connect may fail with EINPROGRESS (or may succeed, or may fail with a real error like ECONNREFUSED)
  - accept may fail with EAGAIN or EWOULDBLOCK if no connections present to be accepted

2.27 How do you know when to read/write?

```
struct timeval {
    long tv_sec;  /* seconds */
    long tv_usec; /* and microseconds */
};
int select(int nfds, fd_set* readfds, fd_set* writefds,
           fd_set* exceptfds, struct timeval* timeout);
FD_SET(fd, &fdset);
FD_CLR(fd, &fdset);
FD_ISSET(fd, &fdset);
FD_ZERO(&fdset);
```

- Entire program runs in an **event loop**

2.28 Event-driven servers

- Quite different from processes/threads
  - Race conditions, deadlocks rare
  - Often more efficient
- But...
  - Unusual programming model
  - Sometimes difficult to avoid blocking
  - Scaling to more CPUs is more complex

3 HTTP and the Web

3.1 Precursors

- 1945, Vannevar Bush, Memex:
  - "*a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility*"
- Precursors to hypertext
  - "*The human mind [...] operates by association. WIth one item in its grasp, it snaps instantly to the next that is suggested by the association of thoughts, in accordance with some intricate web of trails carried by the cells of the brain*"
- Read his 1945 essay, "As we may think"
  - https://www.theatlantic.com/magazine/archive/1945/07/as-we-may-think/303881/

3.2 Tim Berners-Lee

- Physicist at CERN, trying to solve real problem
  - Distributed access to data
- WWW: distributed database of pages linked thrugh the Hypertext Transfer Protocol
  - First HTTP implementation: 1990
  - HTTP/0.9 – 1991
    * Simple GET command
  - HTTP/1.0 – 1992
    * Client/server information, simple caching
  - HTTP/1.1 – 1996
    * Extensive caching support
    * Host identification
    * Pipelined, persistent connections, ...

3.3 Components

- Content

    - Objects (may be static or dynamically generated)
- Clients
    - Send requests / receive responses
- Servers
    - Receive requests / send responses
    - Store or generate content
- Proxies
    - Placed between clients and servers
    - Provide extra functions
        * Caching, anonymization, logging, transcoding, filtering access
    - Explicit or transparent

3.4 Ingredients

- HTTP
    - Hypertext Transfer Protocol
- HTML
    - Language for description of content
- Names (mostly URLs)

3.5 URLs

`protocol://[name@]hostname[:port]/directory/resource?k1=v1&k2=v2#tag`

- **Name** is for possible client identification
- **Hostname** could be an IP address
- **Port** defaults to protocol default (e.g. 80)
- **Directory** is a path to the resource
- **Resource** is the name of the object
- **?parameters** are passed to the server for execution
- **#tag** allows jumps to named tags within document

3.6 Examples of URLs

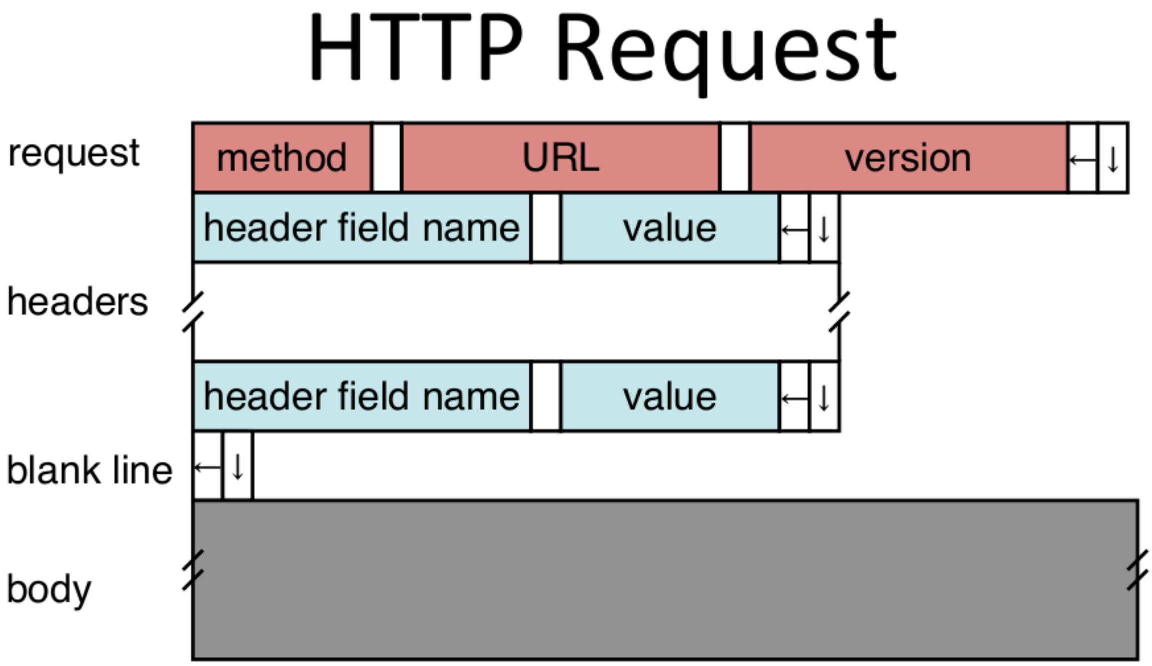
- `http://www2.cs.uh.edu/~gnawali/courses/cosc4377-s12/schedule.html`
- `http://en.wikipedia.org/wiki/Domain_name#Top-level_domains`
- `http://www.uh.edu/search/?q=computer+science&x=0&y=0`

3.7 HTTP

- Important properties
    - Client-server protocol
    - Protocol (but not data) in ASCII
    - Stateles
    - Extensible (header fields)
- Server typically listens on port 80
- Server sends response, may close connection (client may ask it to stay open)
- Version 1.1 in use by less than 45% of websites, version 2 in use by over 45% of websites, version 3 in use by 5.8% of websites

3.8 Steps in HTTP Request

- Open TCP connection to server
- Send request
- Receive response
- TCP connection terminates
    - How many RTTs for a single request?
- You may also need to do a DNS lookup first!



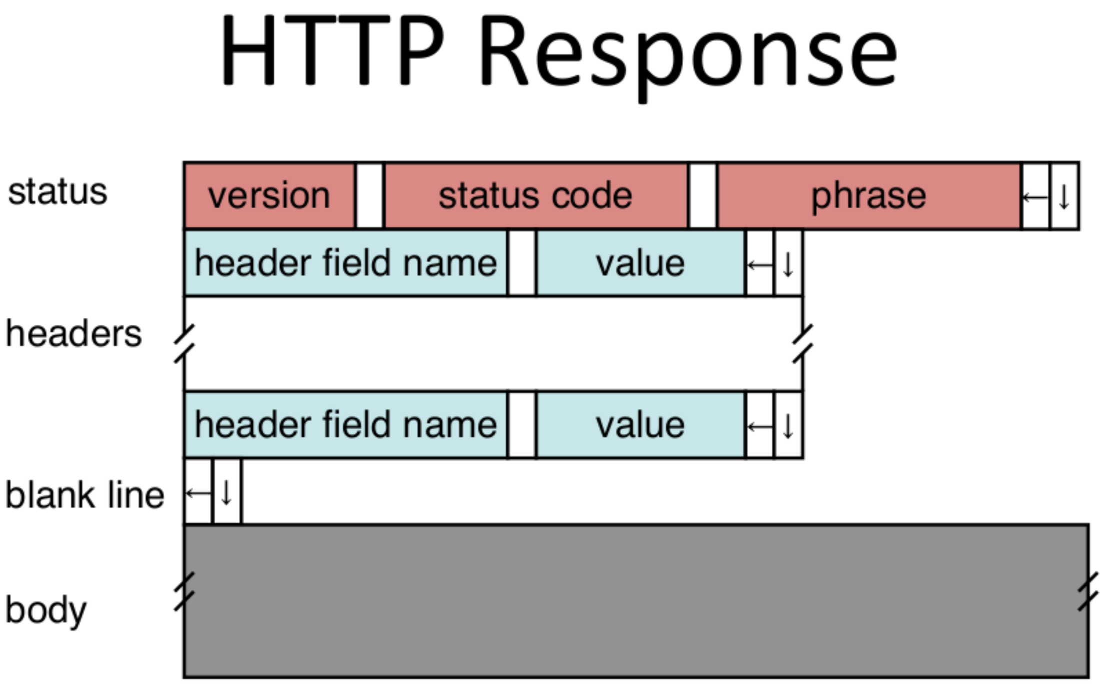
- Method:
  - GET: current value of resource, run program
  - HEAD: return metadata assocated with a resource
  - POST: update a resource, provide input for a program
- Headers: useful info for proxies or the server
  - e.g. desired language

3.9 Sample Browser Request

```
GET / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macinto ...
Accept: text/xml,application/xm ...
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
(empty line)
```

3.10 Sample HTTP Response

```
HTTP/1.0 200 OK
Date: Wed, 25 Jan 2012 08:11:09 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
Set-Cookie: PREF=ID....
P3P: CP="This is not a P3P policy! See http://
www.google.com/support/accounts/bin/answer.py?
hl=en&answer=151657 for more info."
Server: gws
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
<!doctype html><html><head><meta http-equiv="content-type"
content="text/html; charset=ISO-8859-1"><meta...>
```



- Status Codes:
  - 1xx: Information, e.g. 100 Continue
  - 2xx: Success, e.g. 200 OK
  - 3xx: Redirection, e.g. 302 Found (elsewhere)
  - 4xx: Client Error, e.g. 404 Not Found
  - 5xx: Server Error, e.g. 503 Service Unavailable

3.11 HTTP is Stateless

- Each request/response treated independently
- Servers not required to maintain state
- This is good!
  - Improves server scalability
- This is also bad...
  - Some applications need persistent state
  - Need to uniquely identify user to customize content
  - e.g. shopping cart, web-mail, usage tracking, (most sites today!)

3.12 HTTP Cookies

- Client-side state maintenance
  - Client stores small state on behalf of server
  - Sends request in future requests to the server
  - Cookie value is meaningful to the server (e.g. session ID)
- Can provide authentication
- `https://en.wikipedia.org/wiki/HTTP_cookie`

Where to find official HTTP specificaton?

`www.w3.org`

3.13 Anatomy of a Web Page

- HTML content
- A number of additional resources
  - Images
  - Scripts
  - Frames
- Browser makes one HTTP request for each object

  - Course web page: 4 objects
  - My facebook page this morning: 100 objects

3.14 AJAX

- **Asynchronous JavaScript and HTML**
- Based on XMLHttpRequest object in browsers, which allow code in the page to:
  - Issue a new, non-blocking request to the server, without leaving the current page
  - Receive the content
  - Process the content
- Used to add interactivity to web pages
  - XML not always used, HTML fragments, JSON, and plain text also popular

3.15 HTTP Performance

- What matters for performance?
- Depends on type of request
  - Lots of small requests (objects in a page)
  - Some big requests (large download or video)

3.16 Small Requests

- Latency matters
- RTT dominates
- Two major causes:
  - Opening a TCP connection
  - Actually sending the request and receiving response
  - And a third one: DNS lookup!
- Mitigate the first one with persistent connections (HTTP/1.1)
  - Which also means you don't have to "open" the connection each time

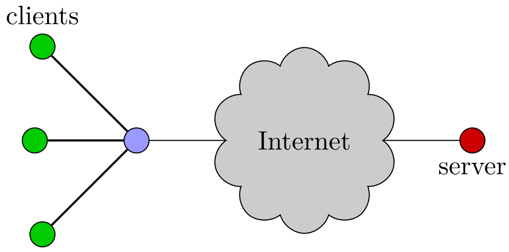
Browser Request

```
GET / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macinto ...
Accept: text/xml,application/xm ...
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

- Second problem is that requests are serialized
  - Similar to stop-and-wait protocols!
- Two solutions
  - Pipelined requests (similar to sliding windows)
  - Parallel Connections
    - HTTP standard says no more than 2 concurrent connections per host name
    - Most browsers use more (up to 8 per host, $approx$35 total)
  - How are these two approaches different?
  - https://en.wikipedia.org/wiki/HTTP_pipelining

3.17 Larger Objects

- Problem is throughput in bottleneck link
- Solution: HTTP Proxy Caching
  - Also improves latency and reduces server load



4 Domain Name System

4.1 Host names and IP Addresses

- Host names
  - Mnemonics appreciated by humans
  - Variable length, ASCII characters
  - Provide little (if any) information about location
  - Examples: www.facebook.com, bbc.co.uh
- IP Addresses
  - Numerical address appreciated by routers
  - Fixed length, binary numbers
  - Hierarchical, related to host location (in the network)
  - Examples: 69.171.228.14, 212.58.241.131

4.2 Separating Naming and Addressing

- Names are easier to remember
  - www.cnn.com vs. 157.166.244.26
- Addresses can change underneath
  - e.g. renumbering when changing providers
- Name could map to multiple addresses
  - www.cnn.com maps to at least 6 IP addresses
  - Enables
    * Load balancing
    * Latency reduction
    * Tailoring request based on requester's location/device/identity
  - Multiple names for the same address
    * Aliases: www.cs.brown.edu and cs.brown.edu
    * Multiple servers in the same node (e.g. apache virtual servers)

4.3 Scalable Address $\leftrightarrow$ Name Mappings

- Original kept in a local file, `hosts.txt`
  - Flat namespace
  - Central administrator kept master copy (for the internet)
  - To add a host, emailed admin
  - Downloaded file regularly
- Completely impractical today
  - File would be huge (gigabytes)
  - Traffic implosion (lookups and updates)
    * Some names change mappings every few days (dynamic IP)
  - Single point of failure
  - Impractical politics (repeated names, ownership, etc.)

4.4 Goals for an Internet-scale name system

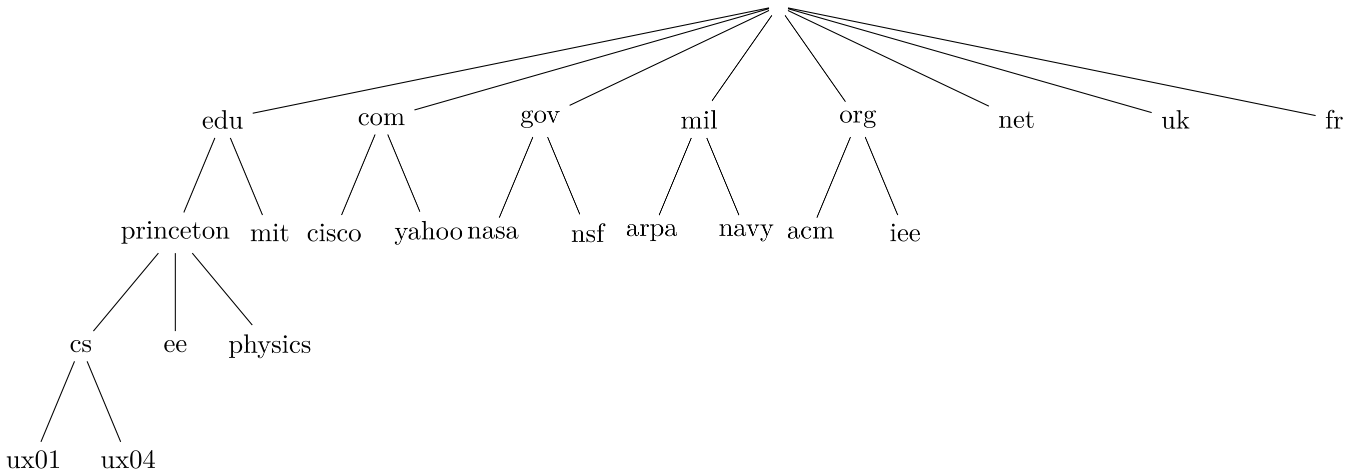
- Scalability
  - Must handle a huge number of records
    * With some software synthesizing names on the fly
  - Must sustain update and lookup load

- Distributed Control
  - Let people control their own names
- Fault tolerance
  - Minimize lookup failures in face of other network problems

4.5 The Good News

- Properties that make these goals easier to achieve
  1. Read-mostly database
     - Lookups **much** more frequent than updates
  2. Loose consistency
     - When adding a machine, not end of the world if it takes minutes or hours to propagate
  3. These suggest aggressive caching
     - Once you've looked up a hostname, remember
     - Don't have to look again in the near future

4.6 Domain Name System (DNS)



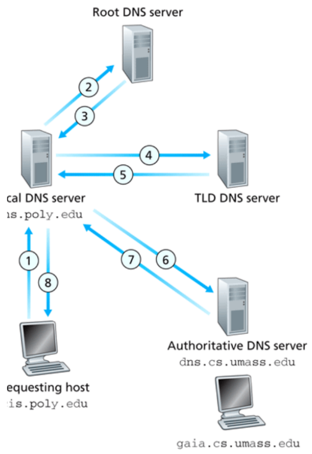
- Hierarchical namespace broken into **zones**
  - root (.), edu., princeton.edu, cs.princeton.edu,
  - Zones separately administred :: delegation
  - Parent zone tells you how to find servers for subdomains
- Each zone served from multiple replicated servers

4.7 DNS Architecture

- Hierarchy of DNS Servers
  - Root servers
  - Top-level domain (TLD) servers
  - Authoritative DNS servers
- Performing the translation
  - Local DNS servers
  - Resolver software

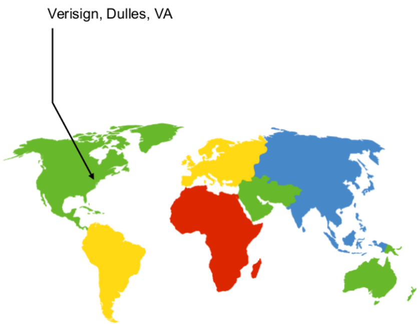
4.8 Resolver Operation

- Apps make recursive queries to local DNS server
  - Ask server to get answer for you
- Server makes iterative queries to remote servers
  - Ask servers who to ask next
  - Cache results aggresively



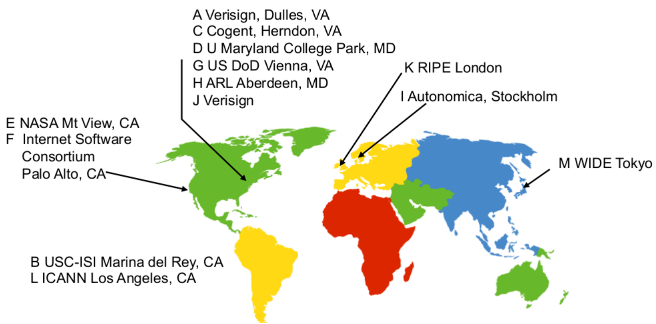
4.9 DNS Root Server

- Located in Virginia, USA
- How do we make the root scale?

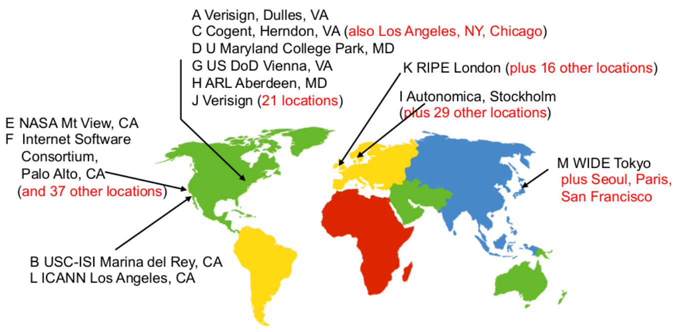


4.10 DNS Root Servers

- 13 root servers (www.root-servers.org)
  - Labeled A through M (e.g. A.ROOT-SERVERS.NET)
- Does this scale?



- Replication via **anycasting**



4.11 TLD and Authoritative DNS Servers

- Top Level Domain (TLD) servers
  - Generic domains (e.g. com, org, edu)
  - Country domains (e.g. uk, br, tv, in, ly)
  - Special domains (e.g. arpa)
  - Typically managed professionally
- Authoritative DNS servers
  - Provides public records for hosts at an organization
    - e.g. for the organization's own servers (www, mail, etc)
  - Can be maintained locally or by a service provider

4.12 Reverse Mapping

- How do we get the other direction, IP address to name?
- Addresses have a hierarchy:
  - 128.148.34.7
- But, most significant element comes first
- Idea: reverse the numbers, 7.34.148.128...
  - And look that up in DNS
- Under what TLD?
  - Convention: in-addr.arpa
  - Lookup 7.34.148.128.in-addr.arpa
  - in6.arpa for IPv6

https://en.wikipedia.org/wiki/Reverse_DNS_lookup

4.13 DNS Caching

- All these queries take a long time!
  - And could impose tremendous load on root servers
  - This latency happens before any real communication, such as downloading your web page
- Caching greatly reduces overhead
  - Top level servers very rarely change
  - Popular sites visited often
  - Local DNS server caches information from many users
- How long do you store a cached response?
  - Original server tells you: TTL entry
  - Server delete entry after TTL expires

4.14 Negative Caching

- Remember things that don't work:
  - Misspellings like www.cnn.comm, ww.cnn.com
- These can take a long time to fail for the first time
  - Good to cache negative results so it will fail faster next time
- But negative caching is optional and not widely implemented

4.15 DNS Protocol

- TCP/UDP port 53

- Most traffic uses UDP
  - Lightweight protocol has 512 byte message limit
  - Retry using TCP if UDP fails (e.g. reply truncated)
- TCP requires message boundaries
  - Prefix all messages with 16-bit length
- Bit in query determines if query is recursive

4.16 Resource Records

- All DNS info represented as resource records (RR)

  name [ttl] [class] type rdata

  - name: domain name
  - TTL: time to live in seconds
  - class: for extensibility, normally IN (1) "Internet"
  - type: type for the record
  - rdata: resource data dependent on the type
- Two import RR types
  - A – Internet Address (IPv4)
  - NS – name server
- Example RRs

```
bayou.cs.uh.edu. 3600 IN A 129.7.240.18
cs.uh.edu. 3600 IN NS ns2.uh.edu.
cs.uh.edu. 3600 IN NS dns.cs.uh.edu.
```

4.17 Some important details

- How do local servers find root servers?
  - DNS lookup on a.root-servers.net?
  - Servers configured with **root cache** file
  - ftp://ftp.rs.internic.net/domain/db.cache
  - Contains root name servers and their addresses
- How do you get addresses of other name servers?
  - To obtain the address of www.cs.brown.edu, ask a.edu-servers.net, says a.root.servers.net
  - How do you find a.edu-servers.net?
  - Glue records: A records in parent zone.

5 DNS and P2P

5.1 DNS

5.2 Structure of a DNS Message

Header	
Question	the question for the name server
Answer	RRs answering the question
Authority	RRs pointing toward an authority
Additional	RRs holding additional information

- Same format for queries and replies
  - Query has 0 RRs in Answer/Authority/Additional
  - Reply includes question, plus has RRs
- Authority allows for delegation
- Additional for glue, other RRs client might need

5.3 Header Format

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ID															
QR	Opcode				AA	TC	RD	RA	Z			RCode			
QDCount															
ANCount															
NSCount															
ARCount															

- ID: match response to query; QR: 0 query/1 response
- RCode: error code
- AA: authoritative answer, TC: truncated
- RD: recursion desired, RA: recursion available

5.4 Other RR Types

- CNAME (canonical name): specifies an alias

```
www.google.com.      446199 in CNAME    www.l.google.com
www.l.google.com.  300 IN  A    72.14.204.147
```

- MX record: specifies servers to handle mail for a domain (the part after the @ in email address)
- SOA (start of authority)
  - Information about a DNS zone and the server responsible for the zone
- PTR (reverse lookup)
  `18.240.7.129.in-addr.arpa. 3600 IN PTR bayou.cs.uh.edu.`
  https://en.wikipedia.org/wiki/List_of_DNS_record_types

5.5 Inserting a Record in DNS

- Your new startup httpserver.com
- Get a block of addresses from ISP
  - say 212.44.9.128/25
- Register helpme.com at GoDaddy.com (for example)
  - Provide name and address of your authoritative name server (primary and secondary)
  - Registrar inserts RR pair into the com TLD server:
    * helpme.com NS dns1.httpserver.com
    * dns1.helpme.com A 212.44.9.129
- Configure your authoritative server (dns1.helpme.com)
  - Tyep A record for www.httpserver.com
  - Type MX record for httpserver.com
- Need to provide reverse PTR bindings
  - e.g. 212.44.9.129 → dns1.httpserver.com
- Normally, these would go into 9.44.212.in-addr.arpa zone
- Problem: you can't run the name server for that domain. Why not?
  - Your block is 212.44.9.128/25, not 212.44.9.0/24
  - Whoever has 212.44.9.0/24 would not be happy with you setting their PTR records
- Solution: [RFC2317, Classless Delegation]
  - Install CNAME records in parent zone, e.g. 129.9.44.212.in--addr.arpa CNAME 129.ptr.httpserver.com

5.6 DNS Security

- You go to Starbucks, how does your browser find www.google.com?
  - ask local name server, obtained from DHCP
  - you implicitly trust this server
  - can return any answer for google.com, including a malicious IP that poses as a man in the middle
- How can you know you are getting correct data?

  - today, you can't
  - HTTPS can help
  - DNSSEC extension will allow you to verify

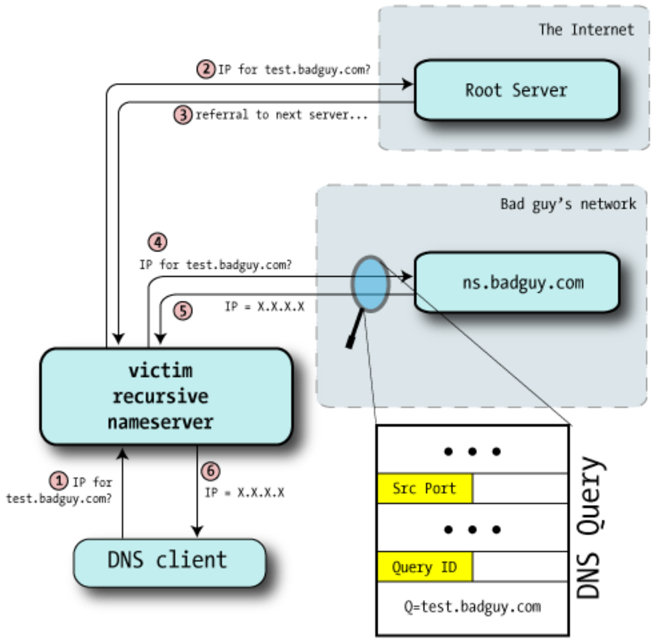
5.7 Cache Poisoning

- Suppose you can tronl evil.com. You receive a query for www.evil.com and reply

```
;; QUESTION SECTION:
;www.evil.com.                  IN      A

;; ANSWER SECTION:
www.evil.com.           300     IN      A       212.44.9.144

;; AUTHORITY SECTION:
evil.com.               600     IN      NS      dns1.evil.com.
evil.com.               600     IN      NS      google.com.

;; ADDITIONAL SECTION:
google.com.               5     IN      A      212.44.9.155
```

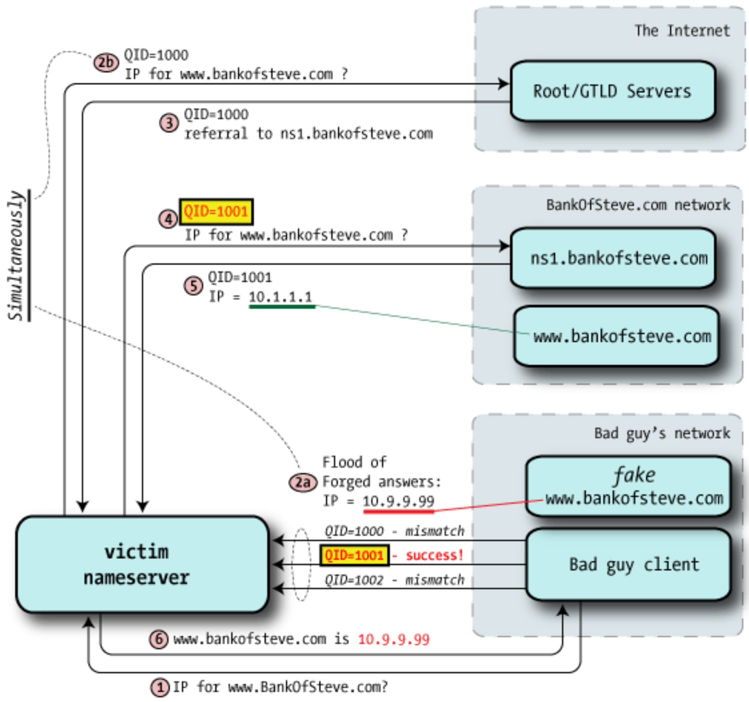
- Glue record pointing to your IP, not Google's
- Gets cached!
- But how do you get a vimctim to look up evil.com?
- You might connect to their mail server and send
  - HELO www.evil.com
  - Which their mail server then looks up to see if it corresponds to your IP address (SPAM filtering)
- Mitigation (bailiwick checking)
  - Only accepts glue records from the domain you asked for
- Bad guy at Starbucks can sniff or **guess** the ID field the local server will use
  - Not hard if DNS server generates ID numbers sequentially
  - Can be done if you force the DNS server to look up something in **your** name server
  - Guess has 1 in 65535 chance (or does it?)
- Now:
  - Ask the local server to lookup google.com
  - Spoof the response from google.com using the correct ID
  - Bogus response arrives before legit one (maybe)
- Local server caches first response it receives
  - Attacker can set a long TTL

5.8 Guessing Query ID



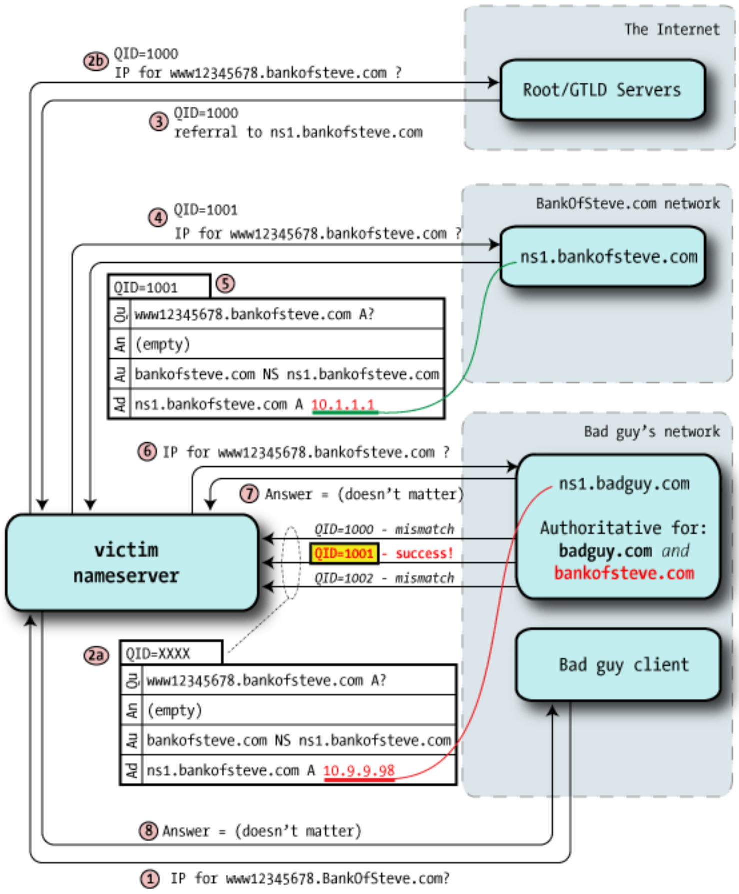
http://www.unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html

5.9 Cache Poisoning



http://www.unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html

5.10 Hijacking Authority Record



http://www.unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html

5.11 Kaminsky Exploit

- If good guy wins the race, you have to wait until the TTL to race again
- But...
  - What if you start a new race for AAAA.google.com, AAAB.google.com, ...?
  - Forge CNAME responses for each
  - Circumvents bailiwick checking

5.12 Countermeasures

- Randomize ID
  - Used to be sequential
- Randomize source port number
  - Used to be the same for all requests from the server
- Offers some protection, but attack still possible

5.13 Load Balancing using DNS

- Return multiple IP addresses ("A" records) for a name
- Benefits
  - Spread the load evenly across the IP addresses
- Problems
  - Caching, no standard on which address to use, ...
- How to solve these problems?
  - Poll load to compute return list
  - https://en.wikipedia.org/wiki/Round-robin_DNS

5.14 Peer-to-Peer

5.15 Client-Server Bottlenecks

- Download time can scale linearly ($\mathcal{O}(n)$ with n clients)
- Scaling up server bandwidth can be expensive
- Too expensive to provision for flash crowds

5.16 Peer-to-Peer Systems

- How did it start?
  - A killer application: file distribution
  - Free music over the internet (not exactly legal…)
- Key idea: share storage, content, and bandwidth of individual users
  - Lots of them
- Big challenge: coordinate all of these users
  - In a scalable way (not $n \times n = n^2$)
  - With changing population (aka **churn**)
  - With no central administration
  - With no trust
  - With large heterogeneity (content, storage, bandwidth, …)

5.17 3 Key Requirements

- P2P Systems do Three things:
  1. Help users **determine what they want**
     - Some form of search
     - P2P version of Google
  2. **Locate** that content
     - Which node(s) hold the content?
     - P2P version of DNS (map name to location)
  3. **Download** the content
     - Should be efficient
     - P2P form of Akamai

5.18 Napster

- Search & Location: central server
- Download: contact a peer, transfer directly
- Advantages:
  - Simple, advanced search possible
- Disadvantages:
  - Single point of failure (technical and … legal!)
  - The latter is what got Napster killed

5.19 Gnutella: Flooding on Overlays (2000)

- Search & Location: flooding (with TTL)
- Download: direct

5.20 BitTorrent

- One big problem with previous approaches
  - Asymmetric bandwidth
- BitTorrent
  - Search: independent search engines (e.g. PirateBay, isoHunt)
    * Maps keywords → .torrent file
  - Location: centralized **tracker** node per file
  - Download: chunked
    * File split into many pieces
    * Can download from many peers
- How does it work?
  - Split files into large pieces (245KB - 1MB)
  - Split pieces into subpieces

    - Get peers from tracker, exchange info on pieces
- Three phases in download
    - Start: get a piece as soon as possible (random)
    - Middle: spread pieces fast (rarest piece)
    - End: don't get stuck (parallel downloads of last pieces)

5.21 BitTorrent Tracker Files

- Torrent file (.torrent) describes files to download
    - Names tracker, server tracking who is participating
    - File length, piece length, SHA1 hash of pieces
    - Additional metadata
- Client contacts tracker, starts communicating with peers

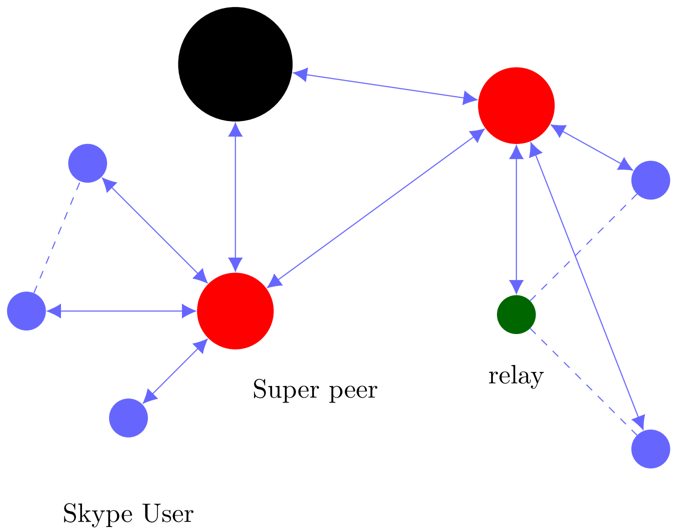
```
d8:announce39:http://torrent.ubuntu.com:6969/announce13:announce-
listll39:http://torrent.ubuntu.com:6969/announceel44:http://
ipv6.torrent.ubuntu.com:6969/announceee7:comment29:Ubuntu CD
releases.ubuntu.com13:creation
datei1272557944e4:infod6:lengthi733837312e4:name29:ubuntu-10.04-
netbook-i386.iso12:piece lengthi524288e6:pieces28000:...
```

  Example tracker from ubuntu.com

- Self-scaling: incentivize sharing
    - If people upload as much as they download, system scales with number of users (no free-loading)
- Uses tit-for-tat: only upload to those who give you data
    - **Choke** most of your peers (don't upload to them)
    - Order peers by download rate, choke all but P best
    - Occasionally unchoke a random peer (might become a nice uploader)

5.22 Skype

- Real-time communication
- Two major challenges:
    - Finding what host a user is on
    - Being able to communicate with those hosts
- Uses Superpeers for registering presence, searching for where you are
    - Need bootstrap super-peers
- Those Superpeers organize index of users
- Making a call
    - Many nodes don't allow incoming connections
    - Uses regular nodes, outside of NATs, as decentralized relays



6 Structured P2P and the Transport Layer

6.1 Structured P2P Systems

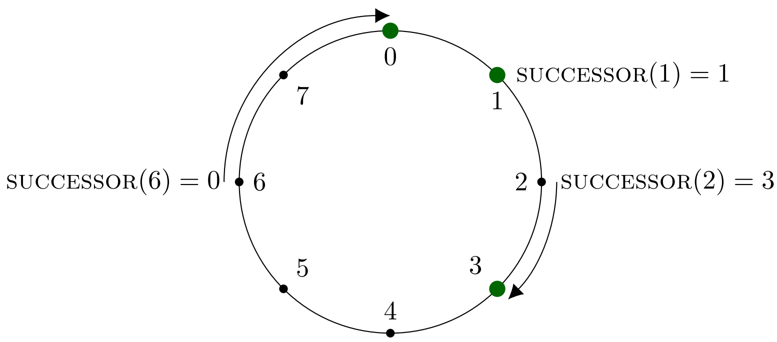
- Distributed Hash Table (DHT)
  - Efficient (`Key`, `Value`) storage
  - Approach: map the ID to a host
- Challenges
  - Scale to millions of nodes
  - Churn
  - Heterogeneity

6.2 DHTs

- IDs from a **flat** namespace
  - Contrast with hierarchical IP, DNS
- Metaphor: hash table, but distributed
- Interface
  - GET(`key`)
  - PUT(`key`, `value`)
- How?
  - Every node supports a single operation:
    Given a `key`, route messages to node holding `key`

6.3 Consistent Hashing

- Map keys to nodes
- `nodeID` = HASH(`nodeIP`)
- k mapped to SUCCESSOR(k)
- SUCCESSOR(k) is the first active node beginning at k



6.4 Consistent Hashing Properties

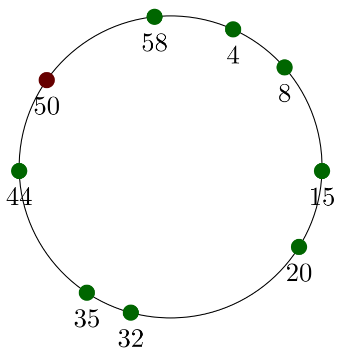
- Designed for node join/leave with minimal churn in key mapping
- k/n keys per node
- k/n keys change hands during join/leave

6.5 Lookup

- Each node maintains its successor
- Route packet (ID, data) to the node responsible for ID using successor pointers

6.6 Joining

- Node with ID 50 joins the ring
- Node 50 needs to know at least one node already in the system
  - Assume known node is 15
- Node 50: send JOIN(50) to node 15
- Node 44: returns node 58
- Node 50: updates its successor to 58
- Node 50: send stabilize to node 58
- Node 58:
  - update predecessor to 50
  - send NOTIFY() back
- Node 44 sends a stabilize message to its successor, node 58
- Node 58 replies with a notify message
- Node 44 updates it successor to 50
- Node 44 sends a stabilize message to its new succcesor, node 50
- Node 50 sets its predecessor to node 44



6.7 Transport Layer

6.8 Network Applications

- Centralized and Peer-to-peer arhictectures
- How to design and write network applications
- Case studies
  - HTTP
  - DNS
  - P2P applications
- These applications need a **reliable method to send information across the network**
- Transport Layer provides that service

6.9 Transport Layer

- Transport protocols sit on top of network layer and provide
  - Application-level multiplexing (“ports”)
  - Error detection, reliability, etc.

6.10 Error Detection

- Idea: add redundant information to catch errors in packet
- Three examples

  - Parity
  - Internet Checksum
  - CRC

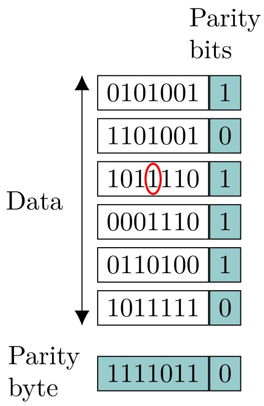
6.11 Parity Bit

- Can detect odd number of bit errors
- No correction

Data	1101101
Parity	1
Transmit	11011011

https://en.wikipedia.org/wiki/Parity_bit

6.12 2-D Parity



- Add 1 parity bit for each 7 bits
- Add 1 parity bit for each bit position across the frame
  - Can correct single-bit errors
  - Can detect 2- and 3-bit errors, most 4-bit errors

6.13 Checksum

- Algorithm
  - Set `checksum` field to 0
  - Sum all 16-bit words, adding any carry bits to the LSB (one's complement sum)
  - Flip bits to get checksum (one's complement)
- Transmit: data + checksum
- To check: sum whole packet, including sum, should get `0xffff`

https://tools.ietf.org/html/rfc1071

6.14 How good is it?

- 16 bits is not very long
  - Probability 1-bit error not detected?
- Checksum does catch any 1-bit error
- But not any 2-bit error
  - e.g. increment word ending 0, decrement one ending in 1

6.15 CRC – Error Detection with Polynomials

- Consider message to be a polynomial in $\mathbb{Z}_2[x]$
  - Each bit is one coefficient
  - e.g. message 10101001 $\rightarrow m(x) = x^7 + x^5 + x^3 + 1$
- Can reduce one polynomial modulo another
  - Select a degree k **irreducible** polynomial $C(x)$ in $\mathbb{Z}_2[x]$
  - Let $n(x) = m(x) \cdot x^k$
  - Compute $r(x) = n(x) \bmod C(x)$

- Compute $n(x) - r(x)$

- Checking CRC is easy
  - Reduce message by $C(x)$, make sure remainder is 0

6.16 Reliable Delivery

- Error detection can discard bad packets
- Problem: if bad packets are lost, how can we ensure reliable delivery?
  - Exactly-once semantics = at least once + at most once

6.17 At Least Once Semantics

- How can the sender know the packet arrived **at least once**?
  - Acknowledgements + Timeout
- Stop and Wait Protocol
  - S: Sent packet, wait
  - R: Receive packet, send ACK
  - S: Receive ACK, send next packet
  - S: No ACK, timeout and retransmit

6.18 Stop and Wait Problems

- Duplicate Data
- Duplicate ACKs
- Can't fill pipe
- Difficult to set the timeout value

6.19 At Most Once Semantics

- How to avoid duplicates?
  - Uniquely identify each packet
  - Have receiver and sender remember
- Stop and wait: add 1 bit to the header
  - Why is it enough?

6.20 Sliding Window Protocol

- Still have the problem of keeping pipe full.
  - Generalize approach > 1-bit counter
  - Allow multiple outstanding (unACKed) frames
  - Upper bound on unACKed frames, called **window**

6.21 Sliding Window Sender

- Assign sequence number (`SeqNum`) to each frame
- Maintain three state variables
  - send window size (`SWS`)
  - last acknowledgement received (`LAR`)
  - last frame send (`LFS`)
- Maintain invariant: $\texttt{LFS} - \texttt{LAR} \leq \texttt{SWS}$
- Advance `LAR` when ACK arrives
- Buffer up to `SWS` frames

6.22 Sliding Window Receiver

- Maintain three state variables
  - receive window size (`RWS`)
  - largest acceptable frame (`LAF`)
  - last frame received (`LFR`)
- Maintain invariant: $\texttt{LAF} - \texttt{LFR} \leq \texttt{RWS}$
- Frame `SeqNum` arrives:
  - if $\texttt{LFR} < \texttt{SeqNum} \leq \texttt{LAF}$, accept
  - if $\texttt{SeqNum} \leq \texttt{LFR}$ or $\texttt{SeqNum} > \texttt{LAF}$, discard
- Send **cumulative** ACKs

7 Transport Protocols

7.1 UDP – User Datagram Protocol

- Unreliable, unordered datagram service
- Adds multiplexing checksum
- End points identified by **ports**
  - Scope is an IP address (interface)
- Checksum aids in error detection

https://en.wikipedia.org/wiki/User_Datagram_Protocol

7.2 UDP Header

0	16 31
SrcPort	DstPort
Length	Checksum
Data	

7.3 UDP Checksum

- Uses the same algorithm as the IP checksum
  - Set checksum field to 0
  - Sum all 16-bit words, adding any carry bits to the LSB
  - Flip bits to get checksum (except `0xffff` → `0xffff`)
  - To check: sum whole packet, including sum, should get `0xffff`
- How many errors?
  - Catches any 1-bit error
  - Not all 2-bit errors
- Optional in IPv4: not checked if value is 0

7.4 Pseudo Header

0 7	8 15	16 23	24 31
source address			
destination address			
zero	protocol	UDP length	

- UDP Checksum is computed over **pseudo-header** prepended to the UDP header
  - For IPv4: IP Source, IP Dest, Protocol (=17), plus UDP length
- Benefits? Problems?
  - Is UDP a layer on top of IP?

http://www.postel.org/pipermail/end2end-interest/2005-February/004616.html

7.5 Next Problem: Reliability

Problem	Mechanism
Dropped Packets	Acknowledgements + Timeout
Duplicate Packets	Sequene Numbers
Packets out of Order	Receiver Window
Keeping the pipe full	Sliding Window (Pipelining)

7.6 Transport Layer Reliability

- Extra difficulties
  - Multiple hosts
  - Multiple hops
  - Multiple potential paths
- Need for connection establishments, tear down
  - Analogy: dialing a number versus a direct line
- Varring RTTs
  - Both across connections and **during** a connection
  - Why do they vary? What do they influence?
- Out of order packets
  - Not only because of drops/retransmissions
  - Can get very old packets (up to 120s), must not get confused
- Unknown resources at other end
  - Must be able to discover receiver buffer: flow control
- Unknown resources in the network
  - Should not overload the network
  - But should use as much as safely possible
  - Congestion Control

7.7 TCP – Transmission Control Protocol

- Service model: "reliable, connection oriented, full duplex byte stream"
  - Endpoints: <IP Address, Port>
- Flow control
  - If one end stops reading, writes at other eventually stop/fail
- Congestion control
  - Keeps sender from overloading the network

7.8 TCP

- Specification
  - RFC 793 (1981), RFC 1222 (1989, some corrections), RFC 5681 (2009, congestion control), ...
- Was born coupled with IP, later factored out
- End-to-end protocol
  - Minimal assumptions on the network
  - All mechanisms run on the end points
- Alternative idea:
  - Provide reliability, flow control, etc, link-by-link
  - Does it work?

7.9 TCP Header

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td colspan="16">Source Port</td><td colspan="16">Destination Port</td></tr>
<tr><td colspan="32">Sequence Number</td></tr>
<tr><td colspan="32">Acknowledgement Number</td></tr>
<tr><td colspan="4">Data
Offset</td><td colspan="3">Reserved</td><td>N
S</td><td>C
W
R</td><td>E
C
E</td><td>U
R
G</td><td>A
C
K</td><td>P
S
H</td><td>R
S
T</td><td>S
Y
N</td><td>F
I
N</td><td colspan="16">Window</td></tr>
<tr><td colspan="24">Options</td><td colspan="8">Padding</td></tr>
<tr><td colspan="32">data</td></tr>
</table>

7.10 Header Fields

- Ports: multiplexing

- Sequence number
  - Correspond to **bytes**, not packets!
- Acknowledgement Number
  - Next expected sequence number
- Window: willing to receive
  - Lets receiver limit SWS (even to 0) for flow control
- Data Offset: number of 4 byte header + option bytes
- Flags, Checksum, Urgen Pointer

7.11 Header Flags

- URG: whether there is urgent data
- ACK: ack no. valid (all but first segment)
- PSH: push data to the application immediately
- RST: reset connection
- SYN: synchronize, establishes connection
- FIN: close connection

7.12 Establishing a Connection

- Three-way handshake
- Two sides agree on respective initial sequence nums
- If no one is listening on port: server sends RST
- If server is overloaded: ignore SYN
- If no SYN-ACK: retry, timeout

7.13 Connection Termination

- FIN bit says no more data to send
  - Caused by close or shutdown
  - Both sides must send FIN to close a connection
- Typical close

7.14 TIME_WAIT

- Why do yo have to wait for 2MSL in TIME_WAIT?
  - What if last ACK is severely delayed, **and**
  - Same port pair is immediately reused for a new connection?
- Solution: active closer goes into TIME_WAIT
  - Waits for 2MSL (Maximum Segment Lifetime)
- Can be problematic for active servers
  - OS has too many sockets in TIME_WAIT, can accept fewer connections
    * Hack: send RST and dlete socket, SO_LINGER = 0
  - OS won't let you restart server because port in use
    * SO_REUSEADDR lets you rebind

7.15 Reliable Delivery

- TCP retransmits if data corrupted or dropped
  - Also retransmit if ACK lost
- When should TCP retransmit?
- Challenges in estimating RTT
  - Dynamic
  - No additional traffic

7.16 Smoothing RTT

- RTT measurement can have large variation
- Need to smooth the samples
  - One RTT measurement = one sample
- Some ways to smooth the sample
  - Average of the whole sequence
  - Windowed Mean
- Problems?

7.17 EWMA

- EWMA: Exponentially Weighted Moving Average
- Give greater weight to recent samples.
  - Why?

  https://en.wikipedia.org/wiki/Moving_average#Exponential_moving_average
- Estimate RTT
- $\mathrm{RTT}(t) = \alpha \mathrm{RTT}(t-1) + (1-\alpha)\mathtt{newEst}$
- More generally, for a dataset $Y = Y_1, Y_2, \ldots$

$$S(t) = \begin{cases} Y_1 & t = 1 \\ \alpha S(t-1) + (1-\alpha)Y_t & t > 1 \end{cases}$$

8 Flow and Congestion Control

8.1 Flow Control

8.2 First Goal

- We should not send more data than the receiver can take: **flow control**
- Data is sent in MSS-sized segments
  - Chosen to avoid fragmentation
- Sender can delay sends to get larger segments
- When to send data?
- How much data to send?

8.3 Flow Control

- Part of TCP specification (even before 1988)
- Goal: don't send more data than the receiver can handle
- Sliding window protocol
- Receiver uses window header field to tell sender how much space it has
- Receiver:

$$\mathtt{AdvertisedWindow} = \mathtt{MaxRcvBuffer} - ((\mathtt{NextByteExpected} - 1) - \mathtt{LastByteRead})$$

- Sender:

$$\mathtt{LastByteSent} - \mathtt{LastByteAcked} \leq \mathtt{AdvertisedWindow}$$

$$\mathtt{EffectiveWindow} = \mathtt{AdvertisedWindow} - \mathtt{BytesInFlight}$$

$$\mathtt{LastByteWritten} - \mathtt{LastByteAcked} \leq \mathtt{MaxSendBuffer}$$

- Advertised window can fall to 0
  - How?
  - Sender eventually stops sending, blocks application
- Sender keeps sendnig 1-byte segments until windows comes back > 0
- 50 students have ssh window open to bayou and are typing 1 character per second
- How many packets are read and written by bayou per second?
  - Consider minimum frame size

8.4 When to Transmit?

Algorithm ?? – reduce the overhead of small packets

```
1: if available data and window ≥ MSS:
2:     send an MSS segment
3: else:
4:     if there is unAcked data in flight:
5:         buffer the new data until ACK arrives
6:     else:
7:         send all new data now
```

- Receiver should avoid advertising a window $\leq$ `MSS` after advertising a window of 0

https://tools.ietf.org/html/rfc896

8.5 Delayed Acknowledgements

- Goal: piggy-back ACKs on data
  - Delay ACK for 200ms in case application sends data
  - If more data received, immediately ACK second segment
  - Note: never delay duplicate ACKs (if missing a segment)
- Warning: can interact **very** badly with Nagle
  - Temporary deadlock
  - Can disable Nagle with TCP_NODELAY
  - Application can also avoid many small writes

https://en.wikipedia.org/wiki/TCP_delayed_acknowledgment
https://developers.slashdot.org/comments.pl?sid=174457&cid=14515105

8.6 Turning off Nagle's Algorithm

"In general, since Nagle's algorithm is only a defense against careless applications, disabling Nagle's algorithm will not benefit most carefully written applications that take proper care of buffering. Disabling Nagle's algorithm will enable the application to have many small packets in flight on the network at once, instead of a smaller number of large packets, which may increase load on the network, and may or may not benefit the application performance."

- Who wants to turn the algorithm off?
  - Search on Google and find out.

https://en.wikipedia.org/wiki/Nagle's_algorithm

8.7 Limitations of Flow Control

- Network may be the bottleneck
- Signal from receiver not enough!
- Sending too fast will cause queue overflows, heavy packet loss
- Flow control provides **correctness**
- Need more for performance: congestion control

8.8 A Short History of TCP

- 1974: 3-way handshake
- 1978: IP and TCP split
- 1983: January 1st, ARPAnet switches to TCP/IP
- 1984: Nagle predicts congestion collapses
- 1986: Internet begins to suffer congestion collapses
  - LBL to Berkeley drops from 32Kbps to 40bps
- 1987/8: Van Jacobsen fixes TCP, publishes sminal paper: (TCP Tahoe)
- 1990: Fast transmit and fast recovery added (TCP Reno)

8.9 Second Goal

- We should not send more data than the network can take: **congestion control**

8.10 TCP Congestion Control

- 3 Key Challenges
  - Determining the available capacity in the first place
  - Adjusting to changes in the available capacity
  - Sharing capacity between flows
- Idea
  - Each source determines network capacity for itself
  - Rate is determined by window size
  - Uses implicit feedback (drops, delay)
  - ACKs page transmission (self-clocking)

8.11 Dealing with Congestion

- TCP keeps congestion and flow control windows
  - Max packets in flight is lesser of two
- Sending rate: $\approx$ `Window/RTT`

- The key here is how to set the congestion window to respond to congestion signals

8.12 Starting Up

- Before TCP Tahoe
  - On connection, nodes send full (rcv) window of packets
  - Retransmit packet immediately after its timer expires
- Result: window-sized bursts of packets in network

8.13 Determining Initial Capacity

- Question: how to set w initially?
  - Should start at 1MSS (to avoid overloading the network)
  - Could increase additively until we hit congestion
  - May be too slow on fast network
- Start by doubling with each RTT
  - Then will dump at most one extra window int network
  - This is called **slow start**
- **Slow start**, this sounds quite fast!
  - In contrast to initial algorithm: sender would dump entire **control flow** window at once

9 Flow and Congestion Control (continued)

9.1 Congestion Control

9.2 Slow Start Implementation

- Let w be the size of the window in **bytes**
  - We have $w/\texttt{MSS}$ segments per RTT
- We are doubling w after each RTT
  - We receive $w/\texttt{MSS}$ ACKs each RTT
  - So we can set $w = w + \texttt{MSS}$ on every ACK
- At some point, we hit the network limit
  - Experience loss
  - We are at most one window size above the limit
  - Remember this: `ssthresh` and reduce window

9.3 Slow Start

- We double `cwnd` every round trip
- We are still sending $\min(\texttt{cwnd}, \texttt{rcvwnd})$ packets
- Continue until `ssthresh` estimate or packet drop

9.4 Dealing with Congestion

- Assume losses are due to congestion
- After a loss, reduce congestion window
  - **How much to reduce?**
- Idea: conservation of packets at equilibrium
  - Want to keep roughly the same number of packets network
  - Analogy with water in fixed-size pipe
  - Put new packet into network when one exits

9.5 How much to reduce window?

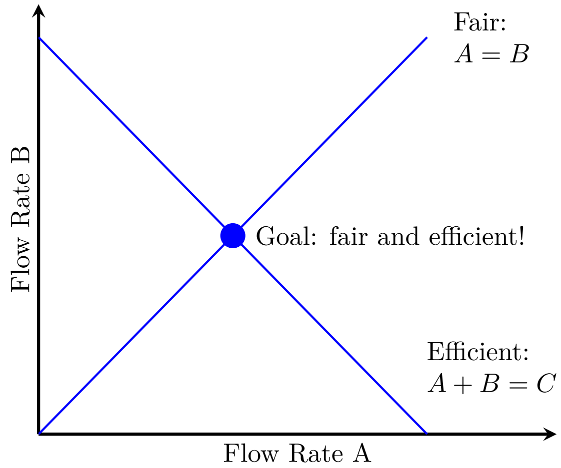
- What happens under congestion?
  - Exponential increase in congestion
- Sources must decrease offered rate exponentially
  - i.e., multiplicative decrease in window size
  - TCP chooses to cut window in half

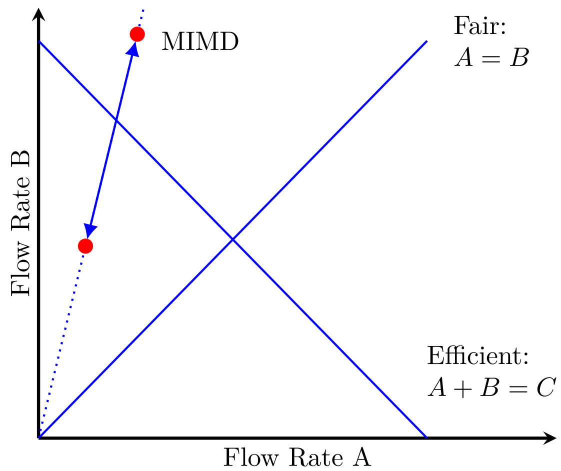
9.6 How to use extra capacity?

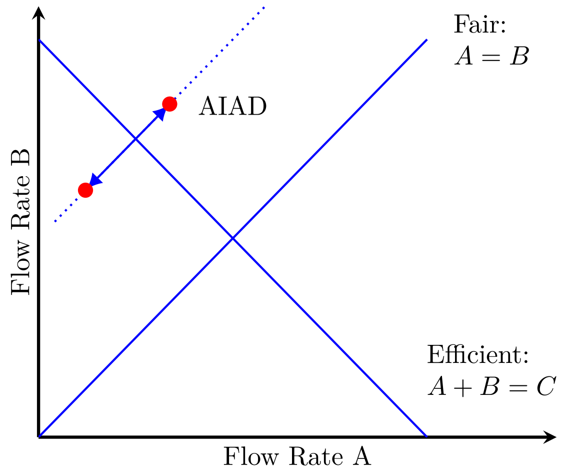
- Network signals congestion, but says nothing of underitilization
  - Senders constantly try to send faster, see if it works
  - So, increase window if no losses... By how much?
- Multiplicative increase?

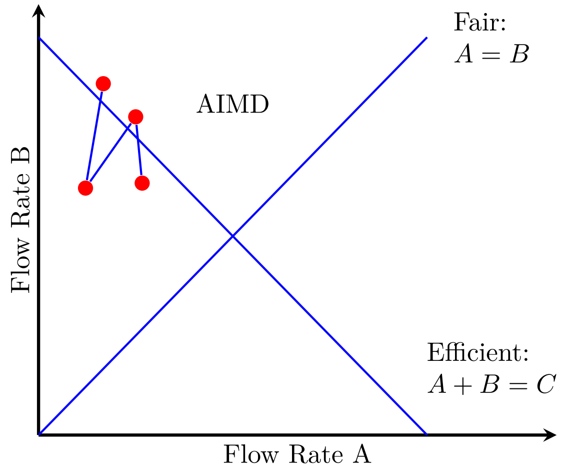
  - Easier to saturate the network than to recover
  - Too fast, will lead to saturation, wild fluctuations
- Additive Increase?
  - Won't saturate the network

9.7 Chiu Jain Phase Plots







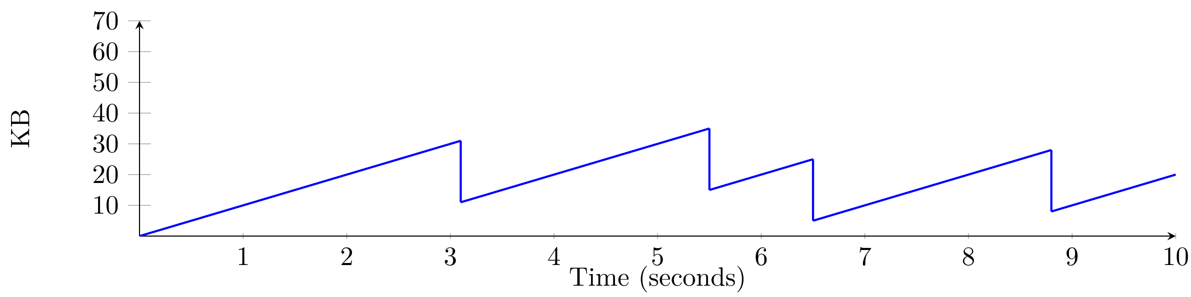


9.8 AIMD Implementation

- In practice, send `MSS`-sized segments
  - Let window size in bytes be w (a multiple of `MSS`)
- Increase:
  - After w bytes ACKed, could set $w = w + \texttt{MSS}$
  - Smoother to increment on each ACK
    - $w = w + \texttt{MSS} \times \texttt{MSS}/w$
    - (receive $w/\texttt{MSS}$ ACKs per RTT, increase by $\frac{\texttt{MSS}}{w/\texttt{MSS}}$ for each)
  - Decrease:
    - After a packet loss, $w = w/2$
    - But dont want $w < \texttt{MSS}$
    - So react differently to multiple consecutive losses
    - Back off exponentially (pause with no packets in flight)

9.9 AIMD Trace

- AIMD produces sawtooth pattern of window size
  - Always probing available bandwidth



9.10 Putting it Together

- TCP has two states: Slow Start (SS) and Congestion Avoidance (CA)
- A window size threshold governs the state transition
  - Window $\leq$ threshold: SS
  - Window $>$ threshold: CA
- States differ in how they respond to ACK
  - Slow start $w = w + \texttt{MSS}$
  - Congestion Avoidance: $w = w + \texttt{MSS}^2/w$ (1 `MSS` per RTT)
- On loss event: set $w = 1$, slow start

9.11 How to Detect Loss

- Timeout

- Any other way?
  - Gap in sequence numbers at receiver
  - Receiver uses cumulative ACKs: drops $\Rightarrow$ duplicate ACKs
- 3 duplicate ACKs considered loss

9.12 RTT

- We want an estimate of RTT so we can know a packet was likely lost, and not just delayed
- **Key for correct operation**
- Challenge: RTT can be highly variable
  - Both at long and short time-scales!
- Both average and variance increase a lot with load
- Solution
  - Use exponentially weighted moving average (EWMA)
  - Estimate deviation as well as expected value
  - Assume packet is lost when time is well beyond reasonable deviation

9.13 Originally

- $\texttt{EstRTT} = (1-\alpha) \times \texttt{EstRTT} + \alpha\texttt{SampleRTT}$
- $\texttt{Timeout} = 2 \times \texttt{EstRTT}$
- Problem 1:
  - in case of retransmission, ACK corresponds to which send?
  - Solution: only sample for segments with no retransmission
- Problem 2:
  - does not take variance into account: too aggressive when there is more load!

9.14 Jacobson/Karels Algorithm (Taho)

- $\texttt{EstRTT} = (1-\alpha) \times \texttt{EstRTT} + \alpha \times \texttt{SampleRTT}$
  - Recommended α is 0.125
- $\texttt{DevRTT} = (1-\beta) \times \texttt{DevRTT} + \beta|\texttt{SampleRTT} - \texttt{EstRTT}|$
  - Recomended β is 0.25
- $\texttt{Timeout} = \texttt{EstRTT} + 4 \cdot \texttt{DevRTT}$
- For successive retransmissions: use exponential backoff

9.15 Slow start every time?!

- Losses have large effect on throughput
- Fast Recovery (TCP Reno)
  - Same as TCP Tahoe on Timeout: $w = 1$, slow start
  - On triple duplicate ACKs: $w = w/2$
  - Retransmit missing segment (fast retransmit)
  - Stay in Congestion Avoidance mode

9.16 3 Challenges Revisited

- Determining the available capacity in the first place
  - Exponential increase in congestion window
- Adjusting to changes in the available capacity
  - Slow probing, AIMD
- Sharing capacity between flows
  - AIMD
- Detecting Congestion
  - Timeout based on RTT
  - Triple duplicate acknowledgements
- Fast retransmit/Fast recovery
  - Reduces slow starts, timeouts

10 TCP Friendliness and Getting Help from the Network

10.1 TCP Friendliness

- Can other protocols co-exist with TCP?

    - e.g. if you want to write a video streaming app using UDP, how to do congestion control?
- Equation-based Congestion Control
    - Instead of implementing TCP's CC, estimate the rate at which TCP would send. Function of what?
    - RTT, MSS, Loss
- Measure RTT, Loss, send at that rate!

10.2 TCP Throughput

- Assume a TCP connection of window W, rount-trip time of `RTT`, segment size of `MSS`
    - Sending Rate $S = W \times \texttt{MSS}/\texttt{RTT}$ (1)
- Drop $W = W/2$
    - grows by $\texttt{MSS}W/2$ RTTs, until another drop at $W \approx W$
- Average window then $0.75 \times S$
    - From (1), $S = 0.75W\texttt{MSS}/\texttt{RTT}$ (2)
- Loss rate is 1 in number of packets between losses:

$$\begin{aligned}\texttt{Loss} &= \frac{1}{1 + (W/2 + W/2 + 1 + W/2 + 2 + \cdots + W)} \\ &= \frac{1}{3/8W^2} \quad (3)\end{aligned}$$

- $\texttt{Loss} = 8/(3W^2) \implies W = \sqrt{\frac{8}{3 \cdot \texttt{Loss}}}$ (4)
- Substituting (4) in (2), $S = 0.75W\texttt{MSS}/\texttt{RTT}$

$$\texttt{Throughput} \approx 1.22 \times \frac{\texttt{MSS}}{\texttt{RTT} \cdot \sqrt{\texttt{Loss}}}$$

- Equation-based rate control can be TCP friendly and have better properties, e.g., small jitter, fast ramp-up...

$$W = \sqrt{\frac{8}{3p}}$$

Substitute W into the bandwidth equation below:

$$\texttt{BW} = \frac{\text{data per cycle}}{\text{time per cycle}} = \frac{\texttt{MSS} \cdot \frac{3}{8}W^2}{\texttt{RTT} \cdot \frac{W}{2}} = \frac{\texttt{MSS}/p}{\texttt{RTT}\sqrt{\frac{2}{3p}}}$$

Collect the constants in one term, $C = \sqrt{3/2}$, then we arrive at

$$\texttt{BW} = \frac{\texttt{MSS}}{\texttt{RTT}} \frac{C}{\sqrt{p}}$$

10.3 What happens when Link is Lossy

- Throughput $= 1/\sqrt{\text{Loss}}$

10.4 What can we do about it?

- Two types of losses: congestion and corrupt
- One option: mask corruption losses from TCP
    - Retranmissions at the link layer
    - e.g. snoop TCP: intercept duplicate acknowledgments, retransmit locally, filter them from the sender
- Another option:
    - Tell the sender about the cause for the drop
    - Requires modification of the TCP endpoints

10.5 Congestion Avoidance

- TCP creates congestion to then back off
    - Queues at bottleneck link are often full: increased delay
    - Sawtooth pattern: jitter
- Alternative strategy
    - Predict when congestion is about to happen
    - Reduce rate early
- Two approaches

  - Host centric: TCP vegas
  - Router-centric: RED, DECBit

10.6 TCP Vegas

- Idea: source watches for sign that router's queue is building up (e.g. sending rate flattens)
- Compare Actual Rate (A) with Expected Rate (E)
  - If $E - A > \beta$, decrease `cwnd` linearly: A isn't responding
  - If $E - A < \alpha$, increase `cwnd` lienarly: room for A to grow

10.7 Vegas

- Shorter router queues
- Lower jitter
- Problem:
  - Doesn't compete well with Reno. Why?
  - Reacts earlier, Reno is more aggressive, ends up with higher bandwidth...

10.8 Help from the network

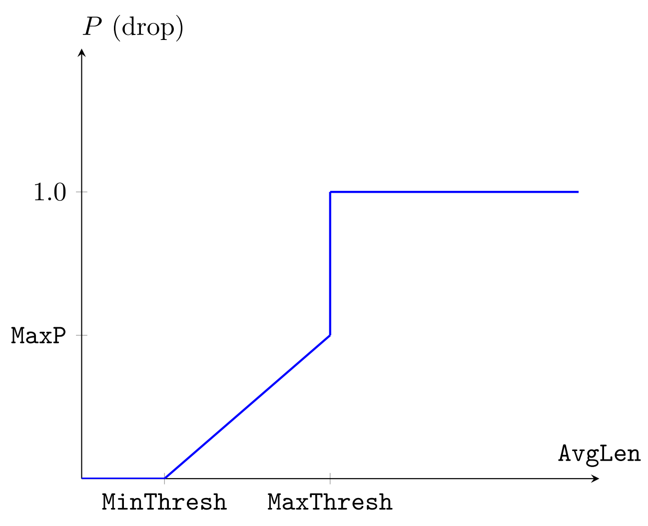
- What if routers could **tell** TCP that congestion is happening?
  - Congestion causes queues to grow: rate mismatch
- TCP responds to drops
  - Idea: Random Early Drop (RED)
    * Rather than wait for queue to become full, drop packet with some probability that increases with queue length
    * TCP will react by reducing `cwnd`
    * Could also mark instead of dropping: ECN

10.9 RED Details

- Computer average queue length (EWMA)
  - Don't want to react to very quick fluctuations

10.10 RED Drop Probability

- Define two thresholsd: `MinThresh`, `MaxThresh`
- Drop probability:



- Improvements to spread drops (see book)

10.11 RED Avantages

- Probability of dropping a packet of a particular flow is roughly proportional to the share of the bandwidth that flow is currently getting
- Higher network utilization with low delays
- Average queue length small, but can absorb bursts
- ECN

- Similar to RED, but router sends bit in the packet
- Must be supported by both ends
- Avoids retransmissions optionally dropped packets

10.12 More help from the network

- Problem: still vulnerable to malicious flows!
  - RED will drop packets from large flows preferentially, but they don't have to respond appropriately
- Idea: Multiple Queues (one per flow)
  - Serve queues in Round-Robin
  - Nagle (1987)
  - Good: protects against misbehaving flows
  - Disadvantage?
  - **Flows with larger packets get higher bandwidth**

11 TCP Friendliness and Getting Help from the Network (Continued)

11.1 Help from the network

- Problem: still vulnerable to malicious flows!
  - RED will drop packets from large flows preferentially, but they don't have to respond appropriately
- Idea: Multiple Queues (one per flow)
  - Serve queues in Round-Robin
  - Nagle (1987)
  - Good: protects against misbehaving flows
  - Disadvantage?
  - **Flows with larger packets get higher bandwidth**

11.2 Solution

- Bit-by-bit round robin
- Can we do this?
  - No, packets cannot be preempted!
- we can only approximate it. . .

11.3 Fair Queueing

- Define a **fluid flow** system as one where flows are served bit-by-bit
- Simulate **ff** and serve packets in the order in which they would finish in the **ff** system
- Each flow will receive exactly its fair share

11.4 Implementing Fair Queueing

- Suppose clock ticks with each bit transmitted
  - (RR, among all active flows)
- P_i is the length of the packet
- S_i is packet i's start of transmission time
- F_i is packet i's end of transmission time
- $F_i = S_i + P_i$
- Across all flows
  - Calculate F_i for each packet that arrives on each flow
  - Next packet to transmit is that with the lowest F_i
  - Clock rate depends on the number of flows
- Advantages
  - Achieves **max-min fairness**, independent of sources
  - Work conserving
- Disadvantages
  - Requires non-trivial support from routers
  - Requires reliable identification of flows

- Not perfect: can't preempt packets

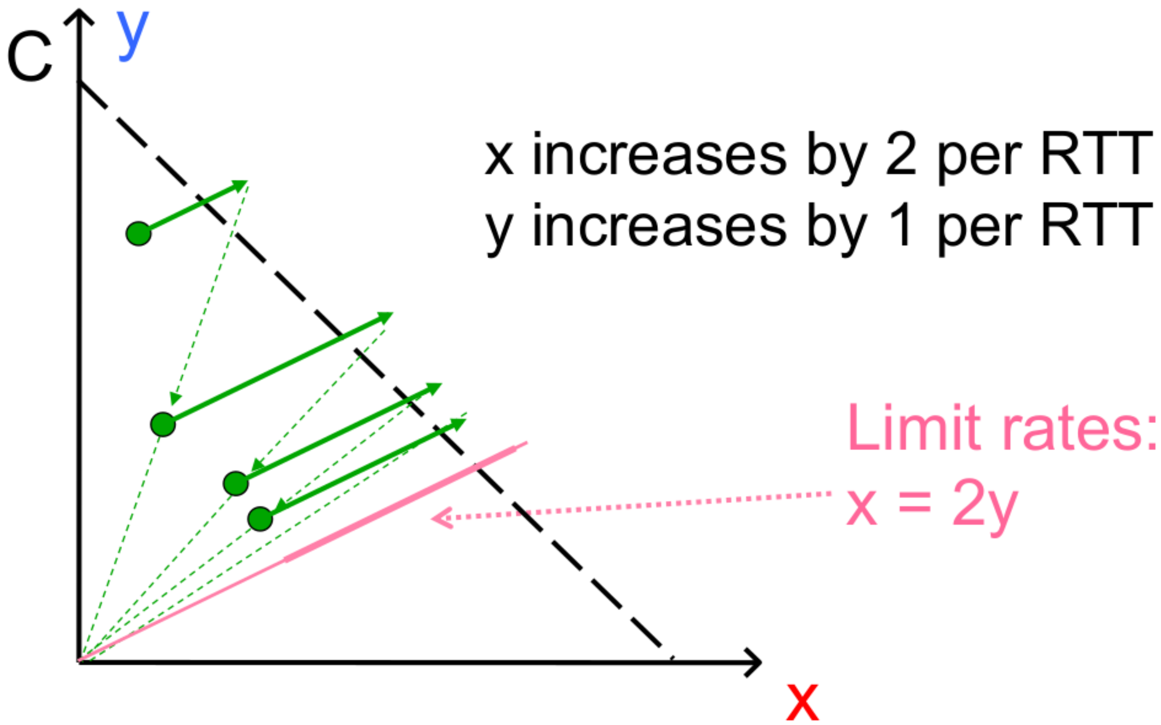
11.5 Big Picture

- Fair Queueing doesn't eliminate congestion: just manages it
- You need both, ideally:
  - End-host congestion control to adapt
  - Router congestion control to provide isolation

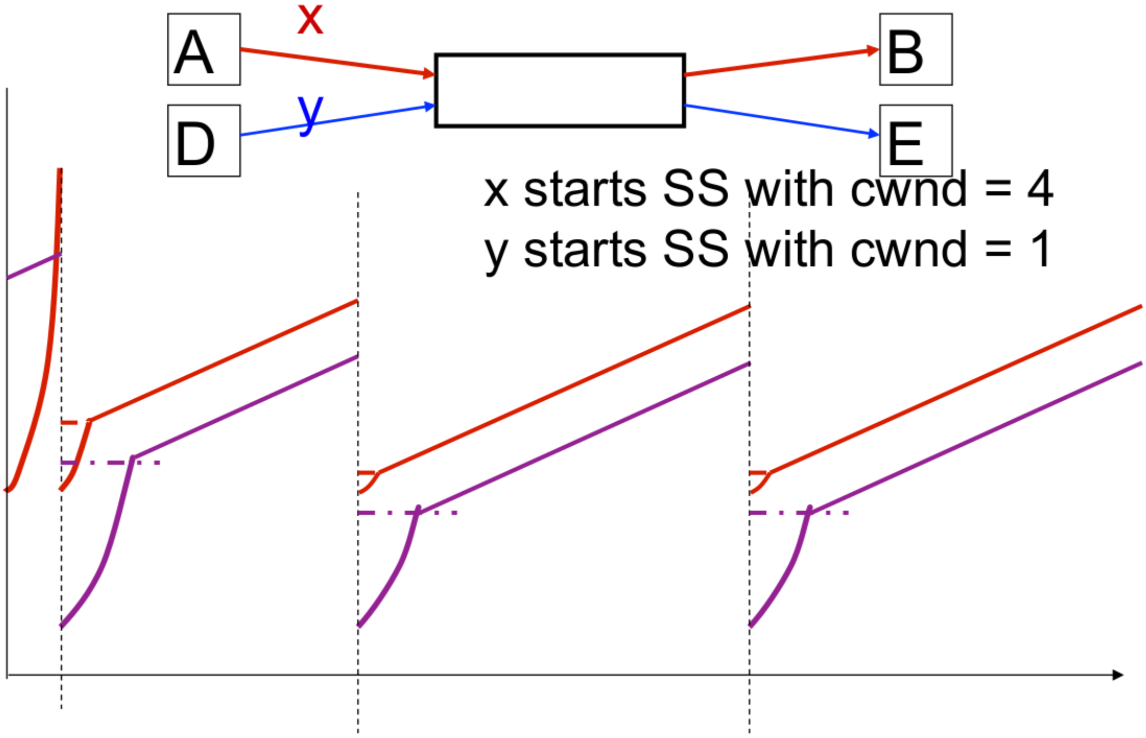
11.6 Cheating TCP

- Three possible ways to cheat
  - Increase `cwnd` faster
  - Large initial `cwnd`
  - Opening many connections
  - ACK Division Attack

11.7 Increasing `cwnd` Faster

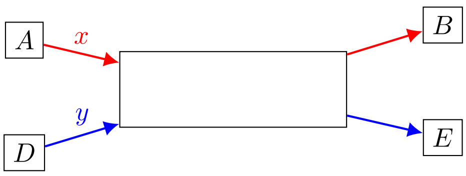


11.8 Larger Initial Window



11.9 Open Many Connections

- Web Browser: has to download k objects for a page
  - Open many connections or download sequentially?



  - Assume:
    - A opens 10 connections to B
    - B opens 1 connection to E
  - TCP is fair among connections
    - A gets 10 times more bandwidth than B

11.10 Exploiting Implicit Assumptions

- Savage, et al., CCR 1999
  - TCP Congestion Control with a Misbehaving Receiver
  - https://cseweb.ucsd.edu/~savage/papers/CCR99.pdf
- Exploits ambiguity of meaning of ACK
  - ACKs can specify any byte range for error control
  - Congestion control assumes ACKs cover entire sent segments

11.11 ACK Division Attack

- Receiver: *"upon receiving a segment with N bytes, divide the bytes into M groups and ackowledge each group separately"*
- Sender will grow M times faster
- Could cause growth to 4GB in 4 RTTs!
  - $M = N = 1460$

11.12 Defense

- Appropriate Byte Counting
  - [RFC 3465 (2003), RFC 5681 (2009)]
  - In slow start, $\texttt{cwnd}\mathrel{+}= \min(N, \texttt{MSS})$ where N is the number of newly acknowledged bytes in the received ACK

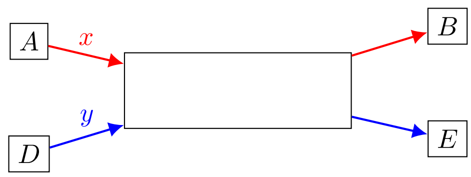
11.13 DupACK Spoofing

- Receiver: *"Upon receiving a data segment, the receiver sends a long stream of acknowledgments for the last sequence number received"*
- Sender sends at a rate proportional to the ACK rate

11.14 Optimistic ACKing

- Receiver: *"Upon receiving a data segment, the receiver sends a stream of acknowledgments anticipating data that will be sent by the sender"*

11.15 Cheating TCP and Game Theory



A ↓ / D →	Increases by 1	Increases by 5
Increases by 1	22, 22	10, 35
Increases by 5	35, 10	15, 15

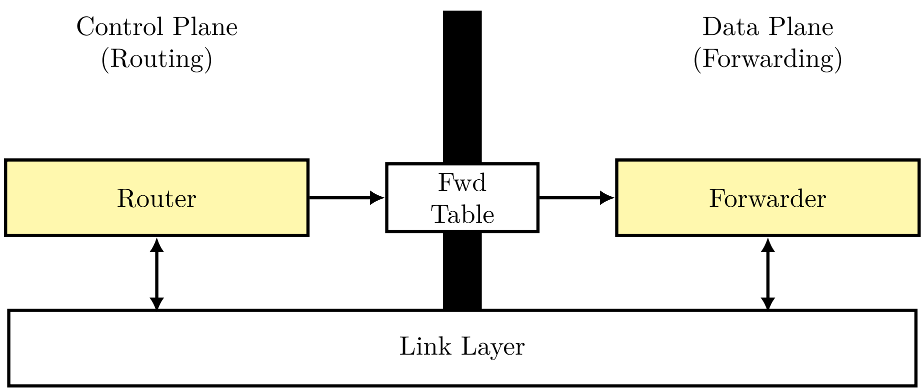
← Too aggressive
→Losses
→Throughput falls

Individual incentives: cheating pays

Social incentives: better off without cheating

12 Overview of Routing

12.1 Router Architecture



12.2 Routing

- Routing is the process of updating forwarding tables
  - Routers exchange messages about routers or networks they can reach
  - Goal: find optimal route for every destination
  - ...or maybe a good route, or **any** route (depending on scale)
- Challenges

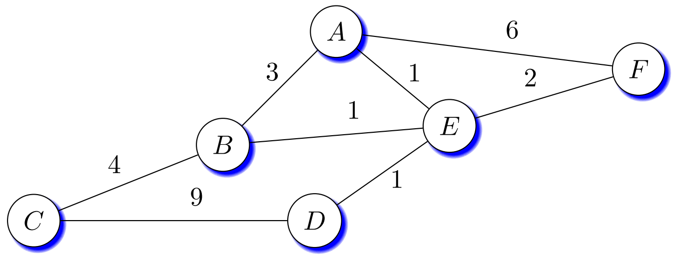
- Dynamic topology
- Decentralized
- Scale

13 Routing and Distance Vector Routing

13.1 Inter and Intra-domain routing

- Routing organized in two levels
- Intra-domain routing
  - Complete knowledge, strive for **optimal** paths
  - Scale to $\approx$ 100 networks.
- Inter-domain routing
  - Aggregated knowledge, scale to internet
  - Dominated by **policy**

13.2 Network as a Graph

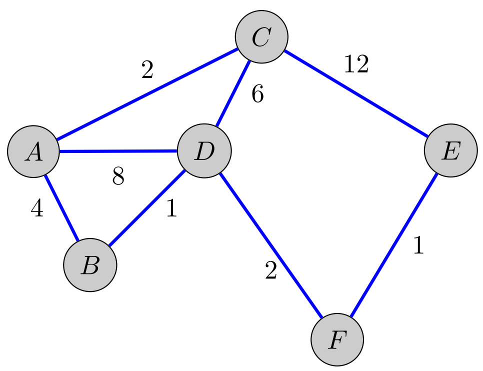


- Nodes are routers
- Assign **cost** to each edge
  - Can be based on latency, bandwidth, queue length, ...
- Problem: find lowest-cost path between nodes
  - Each node individually computes route

13.3 Basic Algorithms

- Two classes of intra-domain routing algorithms
- Distance Vector
  - Requires only local state
  - Harder to debug
  - Can suffer from loops
- Link State
  - Each node has global view of the network
  - Simpler to debug
  - Requires global state

13.4 Shortest Path Example



		Shortest Path					
		End					
		A	B	C	D	E	F
Start	A		4	2	5	8	7
	B	4		6	1	4	3
	C	2	6		6	9	8
	D	5	1	6		3	2
	E	8	4	9	3		1
	F	7	3	8	2	1	

13.5 Distance Vector

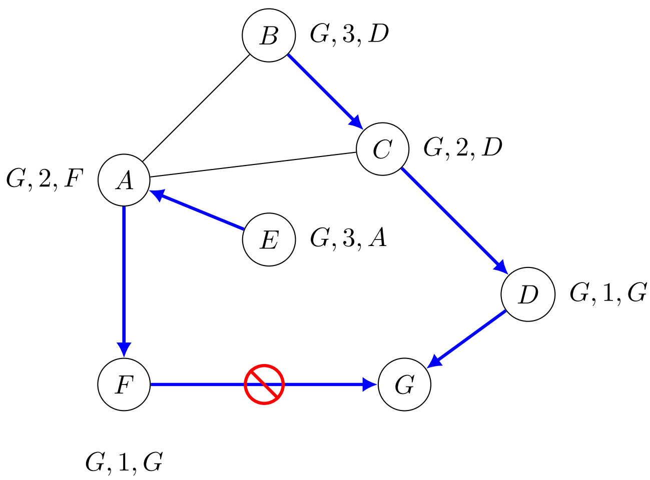
- Local routing algorithm
- Each node maintains a set of triples
  - $\langle$`Destination`, `Cost`, `NextHop`$\rangle$
- Exchange updates with neighbors
  - Periodically (seconds to minutes)
  - Whenever table changes (**triggered** update)
- Each update is a list of pairs
  - $\langle$`Destinaton`, `Cost`$\rangle$
- Update local table if receive a "beter" route
  - Smaller cost
- Refresh existing routes, delete if time out

13.6 Calculating the best path

- Bellman-Ford Equation
- Let:
  - $D_a(b)$ denote the current best distance from a to b
  - $c(a, b)$ denote the cost of a link from a to b
- Then $D_x(y) = \min_z(c(x, z) + D_z(y))$
- Routing messages contain D
- D is any additive metric
  - e.g. number of hops, queue length, delay
  - log can convert multiplicative metric into an additive one (e.g. probability of failure)

https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm

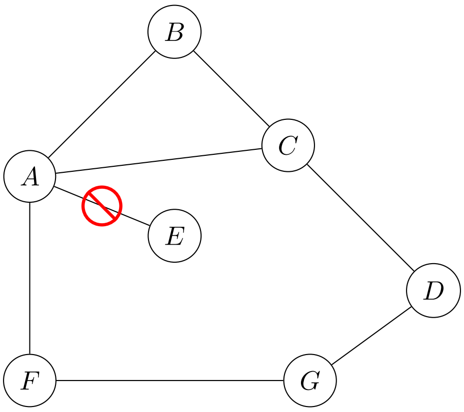
13.7 Adapting to Failures



- $F - G$ fails

- F sets distance to G to ∞, propagates
- A sets distance to G to ∞
- A receives periodic update from C with 2-hop path to G
- A sets distance to G to 3 and propagates
- F sets distance to G to 4, through A

13.8 Count-to-Infinity

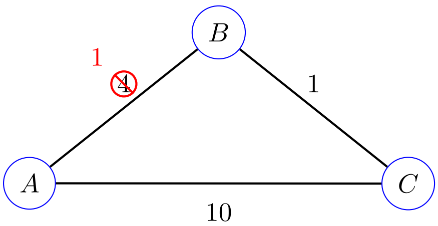


- Link from A to E fails
- A advertises distance of infinity to E
- B and C advertise a distance of 2 to E
- B decides it can reach E in 3 hops through C
- A decides it can reach E in 4 hops through B
- C decides it can reach E in 5 hops through A, ...
- **When does this stop?**

14 Distance Vector, Link State, and Inter-AS Routing

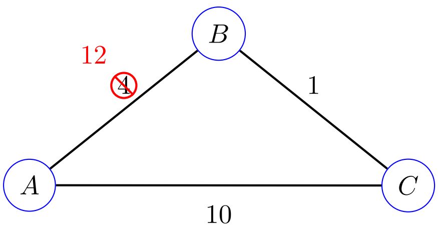
14.1 Routing

14.2 Good news travels fast



- A decrease in link cost has to be fresh information
- Network converges in $\mathcal{O}(d)$ steps (d is the diameter)

14.3 Bad news travels slowly



- An increase in cost may cause confusion with old information
- May form loops

14.4 How to avoid loops

- IP TTL field prevents a packet from living forever
  - Does **not** repair a loop
- Simple approach: consider a small cost n (e.g. 16) to be infinity
  - After n rounds decide node is unavailable
  - But rounds can be long, this takes time

14.5 Better loop avoidance

- Split Horizon
  - When sending updates to node A, don't include routes you learned from A
  - Prevents B and C from sending cost 2 to A
- Split Horizon with Poison Reverse
  - Rather than advertising routes learned from A, explicitly include cost of ∞
  - Faster to break out of loops, but increases advertisement sizes

14.6 Warning

- Split Horizon/Split Horizon with Poison Reverse only helps between nodes
  - Can still get a loop with three nodes involved
  - Might need to delay advertising routes after changes, but affects convergence time

14.7 Link State Routing

- Strategy
  - send to all nodes information about directly connected neighbors
- Link State Packet (LSP)
  - ID of the node that created the LSP
  - Cost of link to each directly connected neighbor
  - Sequence number (`SEQNO`)
  - TTL

14.8 Reliable Flooding

- Store most recent LSP from each node
  - Ignore earlier versions of the same LSP
- Forward LSP to all nodes but the one that sent it
- Generate new LSP periodically
  - Increment `SEQNO`
- Start at $\texttt{SEQNO} = 0$ when reboot
  - If you hear your own packet with $\texttt{SEQNO} = n$, set your next `SEQNO` to $n + 1$
- Decrement TTL of each stored LSP
  - Discard when $\texttt{TTL} = 0$

14.9 Calculating best path

?? computes the shortest path from node s ("yourself") to every other node in the graph. Let `Nodes` denote the set of nodes in the graph, $\textsc{Weight}(i, j)$ the weight of the edge between i and j (∞ if there is no edge), $\textsc{Cost}(n)$ the cost of the path from s to n, and $\textsc{Route}(n)$ the next node to visit in the path from s to n.

Algorithm ??

```
unvisited ← Nodes − {s}
for each n ∈ unvisited:
    Cost(n) ← Weight(s, n)
    if Weight(s, n) < ∞:
        Route(n) ← n
    else:
        Route(n) ← Null
while there are nodes in unvisited:
    let w be the node in unvisited with lowest value Cost(w)
    unvisited.remove(w)
    for each node n ∈ unvisited:
        if Cost(w) + Weight(w, n) < Cost(n):
            Cost(n) ← Cost(w) + Weight(w, n)
            Route(n) ← Route(w)
```

14.10 Distance Vector vs. Link State

- Number of messages (per node v with degree d)
  - DV: $\mathcal{O}(d)$
  - LS: $\mathcal{O}(nd)$ for n nodes in the system
- Computation
  - DV: convergence time varies (e.g. count-to-infinity)
  - LS: $\mathcal{O}(n^2)$ with $\mathcal{O}(nd)$ messages
- Robustness: what happens with malfunctioning router?
  - DV:
    - Nodes can advertise incorrect **path** cost
    - Others can use the cost, propagates through the network
  - LS:
    - Nodes can advertise incorrect **link** cost

14.11 Examples

- RIPv2
  - Fairly simple implementation of DV
  - RFC 2453 (38 pages)
- OSPF (Open Shortest Path First)
  - More complex link-state protocol
  - Adds notion of **areas** for scalability
  - RFC 2328 (244 pages)

14.12 RIPv2

- Runs on UDP port 520
- Link cost = 1
- Periodic updates every 30 seconds, plus triggered updates
- Relies on count-to-infinity to resolve loops
  - Maximum diameter 15 ($\infty = 16$)
  - Supports Split Horizon, Poison Reverse

14.13 Packet Format

0–7	8–15	16–31
command (1)	version (1)	must be zero (2)
RIP Entry (20)		

14.14 RIPv2 Entry

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Address Family Identifier (2)	Route Tag (2)
IP Address (4)	
Subnet Mask (4)	
Next Hop (4)	
Metric (4)	

14.15 Next Hop Field

- Allows one router to advertise routes for multiple routers on the same subnet
- Suppose only `XR1` talks RIPv2

```
-----   -----   -----           -----   -----   -----
|IR1|   |IR2|   |IR3|           |XR1|   |XR2|   |XR3|
--+--   --+--   --+--           --+--   --+--   --+--
  |       |       |               |       |       |
--+-------+-------+---------------+-------+-------+--
  <-------------RIP-2------------->
```

14.16 OSPFv2

- Link state protocol
- Runs directly over IP (protocol 89)
  - Has to provide its own reliability
- All exchanges are authenticated
- Adds notion of **areas** for scalability

14.17 Inter-Domain Routing

14.18 Why Inter vs. Intra

- Why not just use OSPF everywhere?
  - e.g. hierarchies of OSPF areas?
  - Hint: scaling is not the only limitation
- BGP is a policy control and information hiding protocol
  - intra = trusted, inter = untrusted
  - Different policies by different ASes
  - Diffrent costs by different ASes

15 Inter-Domain Routing

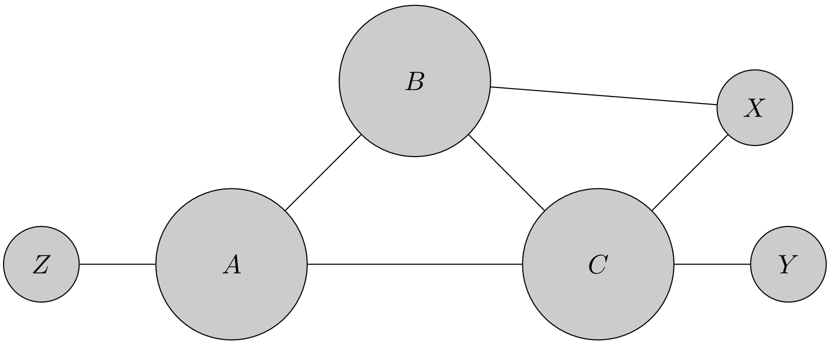
15.1 Why Inter vs. Intra

- Trust
- Policy
- Scale
- Performance

15.2 Types of ASes

- Local Traffic – source or destination in local AS
- Transit Traffic – passes through AS
- Stub AS
  - Connects to only a single other AS
- Multihomed AS
  - Connects to multiple ASes
  - Carries no transit traffic
- Transit AS
  - Connects to multiple ASes and carries transit traffic

15.3 AS Relationships



- How to prevent X from forwarding transit between B and C?
- How to avoid transmit between CBA?
  - $B : BAZ \to X$
  - $B : BAZ \to C$? ($\implies Y : CBAZ$ and $Y : CAZ$)

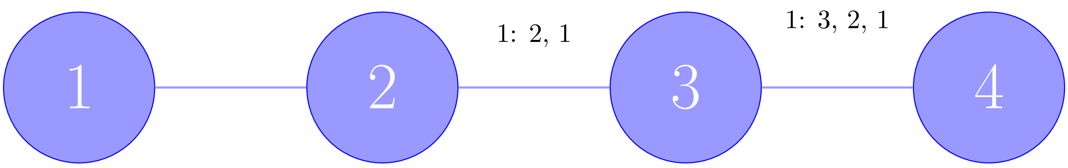
15.4 Autonomous System

- Group of routers/prefixes typically under the control of a single operation
- Example: University of Houston
- Here is one list
  - https://bgp.potaroo.net/cidr/autnums.html

https://en.wikipedia.org/wiki/Autonomous_system_(Internet)

15.5 Path Vector Protocol

- Distance vector algorithm with extra information
  - For each route, store the complete path (ASes)
  - No extra computation, just extra storage (and traffic)
- Advantages
  - Can make policy choices based on set of ASes
  - Can easily avoid loops



15.6 BGP = High Level

- Abstract each AS to a single node
- Destinations are CIDR prefixes
- Exchange prefix **reachability** with all neighbors
  - e.g. "I can reach prefix 128.148.0.0/16 through ASes 44444 3356 14325 11078"
- Select a single path by routing **policy**
- Critical: learn many paths, propagate one
  - Add your ASN to advertised path

15.7 Why study BGP?

- Critical protocol: makes the Internet run
  - Only widely deployed EGP
- Active area of problems!
  - Efficiency
  - Cogent vs. Level3: Internet Partition
  - Spammers use prefix hijacking

- Pakistan accidentally took down YouTube
- Egypt disconnected for 5 days

15.8 BGP Protocol Details

- Separate roles of **speakers** and **gateways**
  - Speakers talk BGP with other AS
  - Gateways are routers that border other AS
  - Can have more gateways than speakers
  - Speakers know how to reach gateways
- Speakers connect over TCP on port 179
  - Bidirectional exchange over long-lived connection

15.9 BGP Implications

- Explicit AS Path = Loop free
  - Except under churn, IGP/EGP mismatch
- Reachability not guaranteed
  - Decentralized combination of policies
- Not all ASes know all paths
- AS abstract → loss off efficiency
- scaling
  - 37 ASes
  - 350K+ prefixes
  - ASes with one prefix: 15664
  - Most prefixes by one AS: 3686 (AS6389, BellSouth)

15.10 BGP and Policy

- BGP provides capability for enforcing various policies
- Policies are not part of BGP: they are provided to BGP as configuration information
- BGP enforces policies by choosing paths from multiple alternatives and controlling advertisement to other ASes

15.11 BGP Path Selection

- Policies determined by path selection
- Information based on path attributes
- Attributes + external (policy) information

15.12 Route Selection

- **More specific prefix?**
- Next-hop reachable?
- Prefer highest weight
  - Computed using some AS-specific local policy
- Prefer highest local-pref
- Prefer locally originated routes
- Prefer routes with shortest AS path length
- Prefer eBGP over iBGP
- Prefer routes with lowest cost to egress point
  - Hot-potato routing
- Tie-breaking rules
  - e.g. oldest route, lowest router-id

15.13 Customer/Provider AS relationships

- Customer pays for connectivity
  - e.g. University of Houston contracts with AboveNet and TW Telecom
  - Customer is stub, provider is a transit
- Many customers are multi-homed
  - e.g. AboveNet connects to Level3, Cogent,...
- Typical policies:
  - Provider tells all neighbors how to reach customer

– Provider prefers routes from customers ($$)
– Customer does not provide transit service

15.14 Peer Relationships

- ASes agree to exchange traffic for free
  – Penalties/Renegotiate if imbalance
- Tier 1 ISPs have no default route: all peer with each other
- You are Tier $i + 1$ if you have a default route to Tier i
- Typical policies
  – AS only exports customer routes to peer
  – AS exports a peer's routes only to its customers
  – Goal: avoid being transit when no gain

15.15 Peering Drama

- Cogent vs. Level3 were peers
- In 2003, Level3 decided to start charging Cogent
- Cogent said no
- **Internet partition**: Cogent's customers couldn't get to Level3's customers and vice-versa
  – Other ISPs were affected as well
- Took 3 weeks to reach an undisclosed agreement

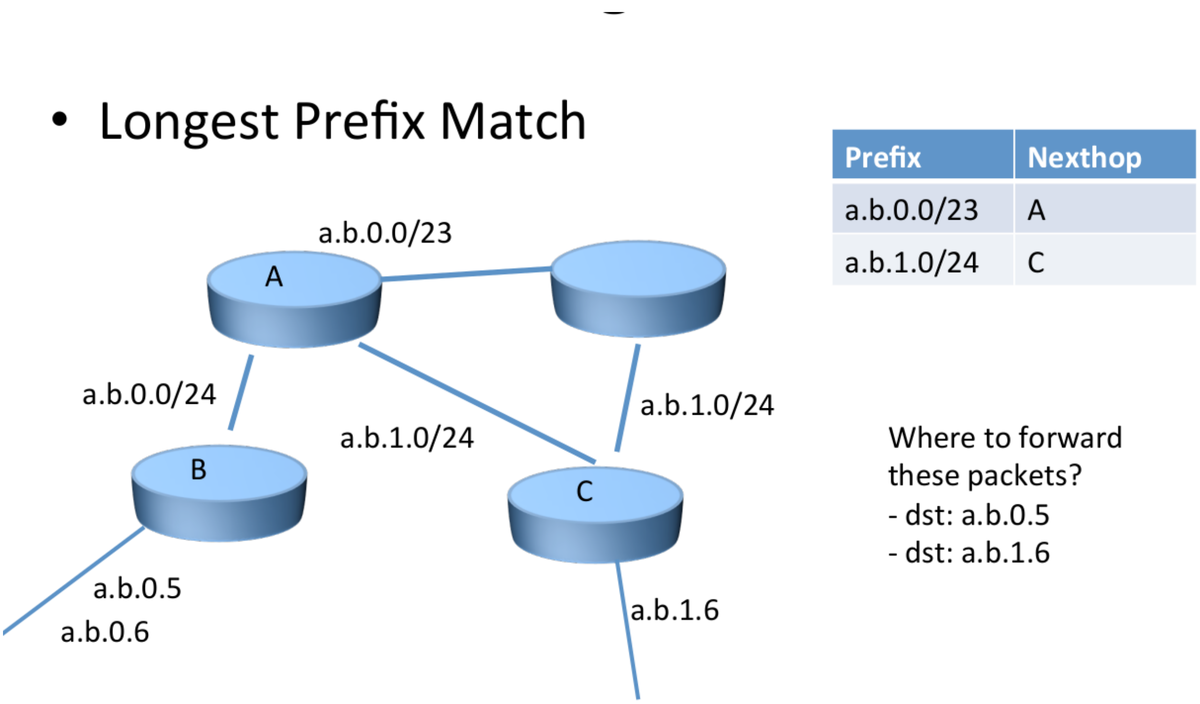
15.16 "Shutting Off" the Internet

- Starting from January 27, 2011, Egypt was disconnected from the Internet
  – 2769/2903 networks withdrawn from BGP (95%!)

16 BGP

16.1 Forwarding with CIDR

- Longest Prefix Match



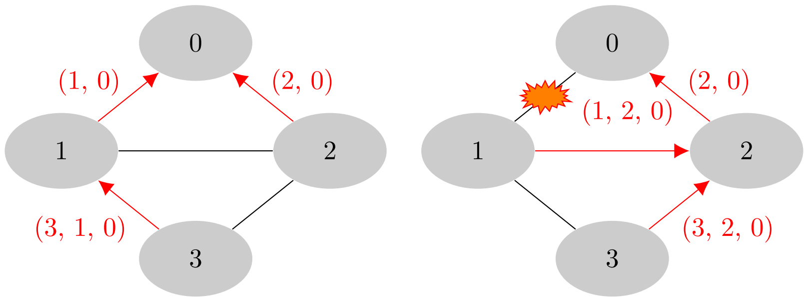
16.2 Some BGP Challenges

- Convergence
- Traffic engineering
  – How to assure certain routes are selected
- Scaling (route reflectors)
- Security

16.3 Convergence

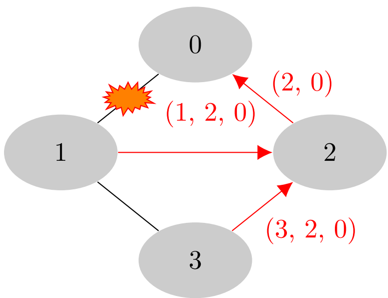
- Given a change, how long until the network re-stabilizes?
  - Depends on change: sometimes never
  - Open research problem: "tweak and pray"
  - Distributed setting is challenging
- Some reasons for change
  - Topology changes
  - BGP session failures
  - Changes in policy
  - Conflicts between policies can cause oscillation

16.4 Routing Change: Before and After



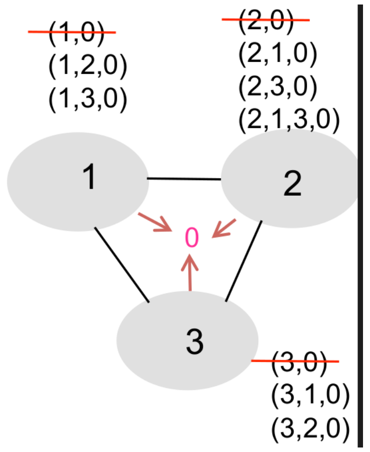
16.5 Routing Change: Path Exploration

- AS 1
  - Delete the route (1, 0)
  - Switch to next route (1, 2, 0)
  - Send route (1, 2, 0) to AS 3
- AS 3
  - Sees (1, 2, 0) replace (1, 0)
  - Compares to route (2, 0)
  - Switches to using AS 2



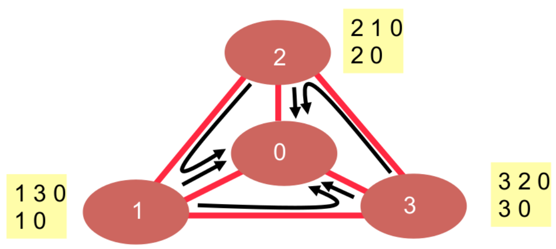
- Initial Situation
  - Destination 0 is alive
  - All ASes use direct path
- When destination dies
  - All ASes lose direct path
  - All switch to longer paths
  - Eventually withdrawn
- e.g. AS 2
  - (2, 0) $\rightarrow$ (2, 1, 0)
  - (2, 1, 0) $\rightarrow$ (2, 3, 0)
  - (2, 3, 0) $\rightarrow$ (2, 1, 3, 0)
  - (2, 1, 3, 0) $\rightarrow$ `Null`

- **Convergence may be slow!**



16.6 Unstable Configurations

- Due to policy conflicts



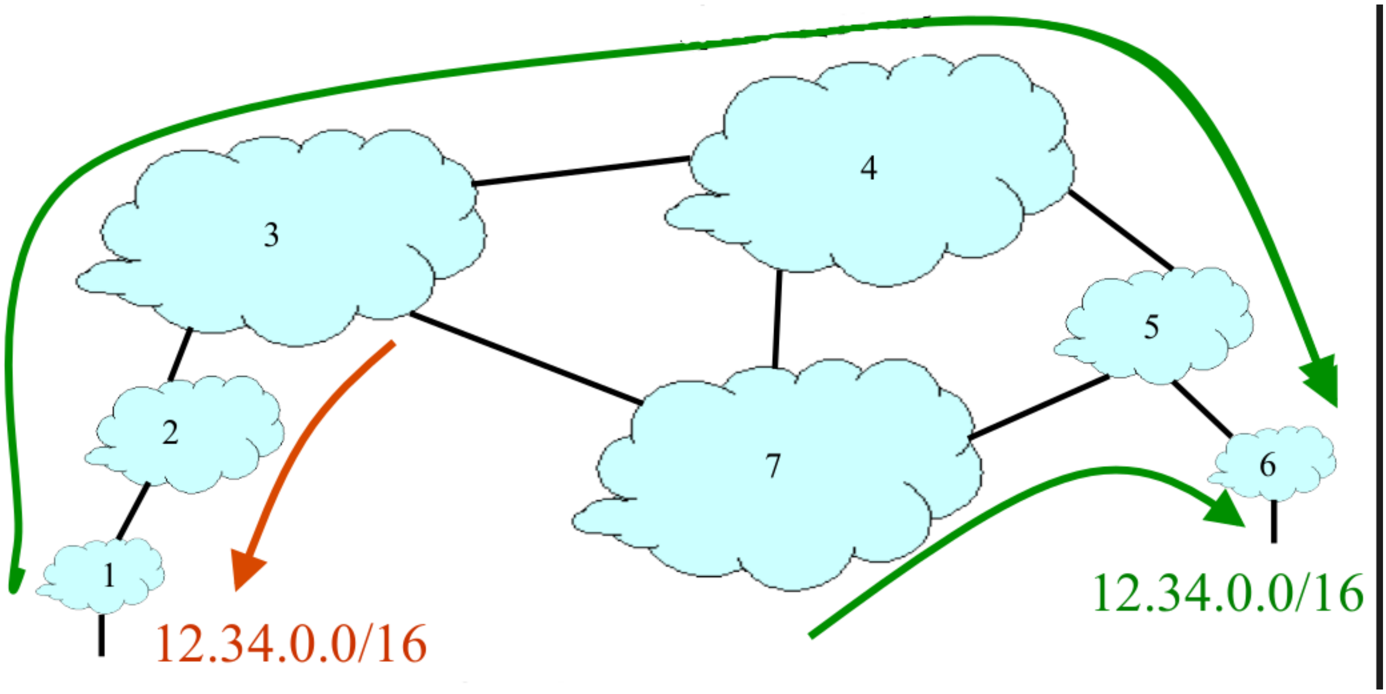
16.7 BGP Security Goals

- Confidential messages exchange between neighbors
- **Validity of routing information**
  - Origin, Path, Policy
- Correspondence to the data path

16.8 Origin: IP Address Ownership and Hijacking

- IP address block assignment
  - Regional Internet Registries (ARIN, RIPE, APNIC)
  - Internet Service Providers
- Proper Origination of a prefix into BGP
  - By the AS who owns the prefix
  - . . . or, by its upstream provider(s) on its behalf
- However, what's to stop someone else?
  - Prefix hijacking: another AS originates the prefix
  - BGP does not verify that the AS is authorized
  - Registries of prefix ownership are inaccurate

16.9 Prefix Hijacking

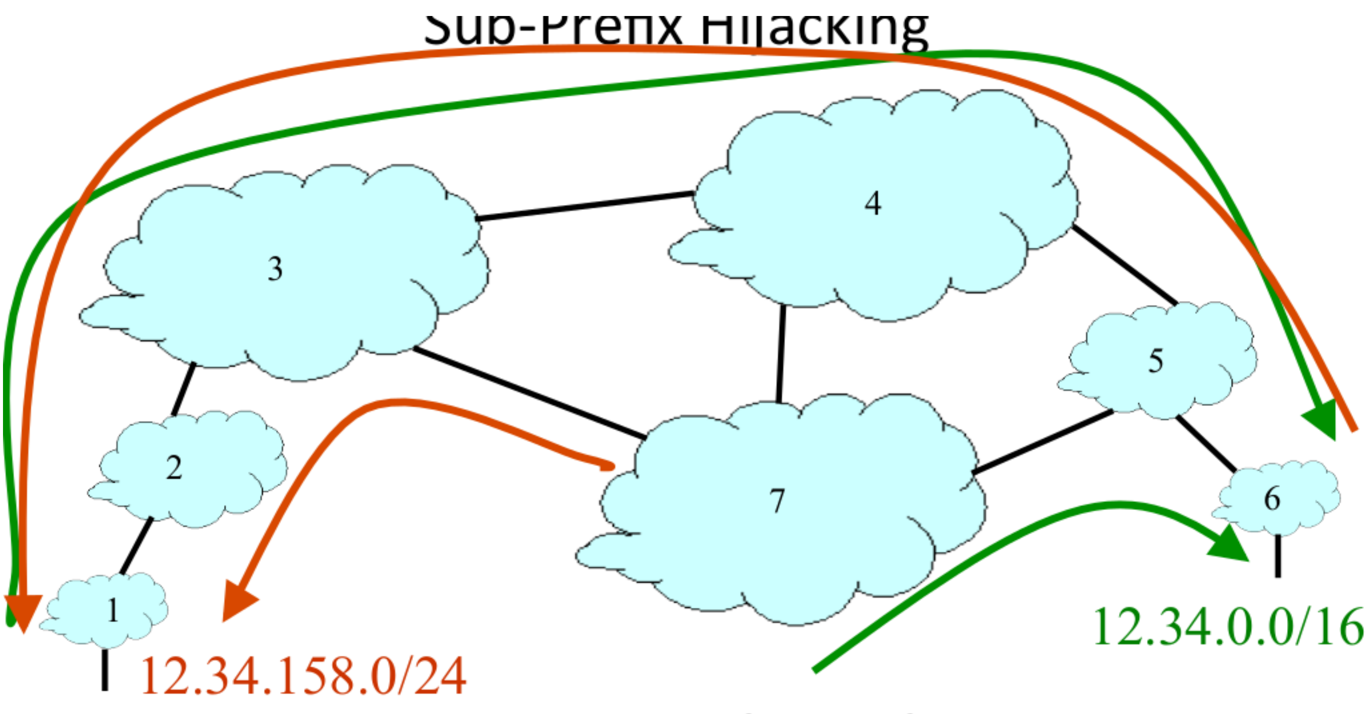


- Consequences for the affected ASes
  - Blackhole: data traffic is discarded
  - Snooping: data traffic is inspected and then redirected
  - Impersonation: data traffic is sent to bogus destinations

16.10 Hijacking is Hard to Debug

- Real origin AS doesn't see the problem
  - Picks its own route
  - Might not even learn the bogus route
- May not cause loss of connectivity
  - e.g. if the bogus AS snoops and redirects
  - . . . may only cause performance degradation
- Or, loss of connectivity is isolated
  - e.g. only for sources in parts of the internet
- Diagnosing prefix hijacking
  - Analyzing updates from many vantage points
  - Launching traceroute from many vantage points

16.11 Sub-Prefix Hijacking



- Originating in a more-specific prefix
  - Every AS picks the bogus route for that prefix
  - Traffic follows the longest matching prefix

16.12 How to Hijack a Prefix

- The hijacking AS has
  - Router with eBGP sessions(s)
  - Configured to originate the prefix
- Getting access to the router
  - Network operator makes configuration mistake
  - Disgruntled operator launches an attack
  - Outsider breaks in to the router and reconfigures
- Getting other ASes to believe bogus route
  - Neighbor ASes not filtering the routes
  - e.g. by allowing only expected prefixes
  - But, specifying filters on **peering** links is hard

16.13 Pakistan YouTube Incident

- YouTube has prefix 208/65/152.0/22
- Pakistan's government order YouTube blocked
- Pakistan Telecom (AS 17557) announces 208.65.153.0/24 in the wrong direction (outwards!)
- Longest prefix match caused worldwide outage
- https://www.youtube.com/watch?v=IzLPKuAOe50

16.14 Many other incidents

- Spammers steal unused IP space to hide
  - Announce very short prefixes (e.g. /8). Why?
  - For a short amount of time
- China incident, April 8, 2010
  - China's Telecom AS23724 generally announces 40 prefixes
  - On April 8, announced ≈ 37,000 prefixes
  - About 10% leaked outside of China

    - Suddenly, going to www.dell.com might have you routing through AS23724!

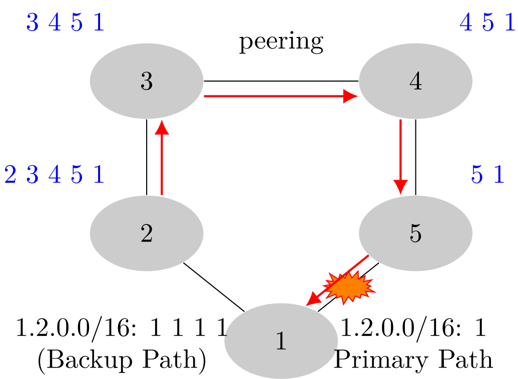
16.15 Attacks on BGP Paths

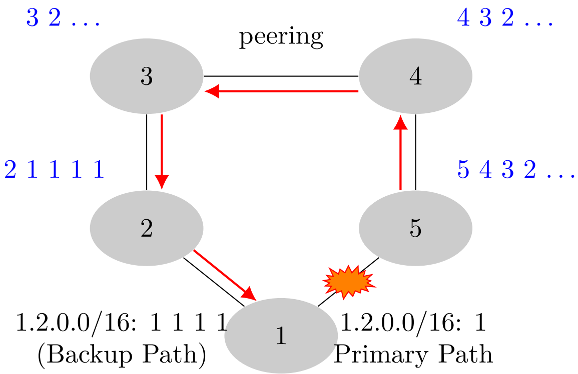
- Remove an AS from the path
    - e.g. 701 3715 88 → 701 88
- Why?
    - Attract sources that would normally avoid AS 3715
    - Make AS 88 look like it is closer to the core
    - Can fool loop detection!
- May be hard to tell whether this is a lie
    - 88 could indeed connect directly to 701!
- Adding ASes to the path
    - e.g. 701 88 → 701 3715 88
- Why?
    - Trigger lookup detection in AS 3715
        * This would block unwanted traffic from AS 3715!
    - Make your AS look more connected
- Who can tell this is a lie?
    - AS 3715 could, if it could see the route
    - AS 88 could, but would it really care?
- Adding ASes at the end of the path
    - e.g. 701 88 → 701 88 3
- Why?
    - Evade detection for a bogus route (if added AS is legitimate owner of a prefix)
- Hard to tell that the path is bogus!

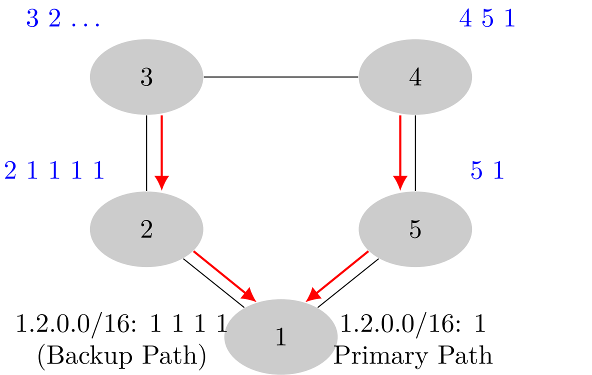
17 BGP Wedgies and IP

17.1 Multiple Stable Configurations BGP Wedgies [RFC 4264]

- Typical Policy
    - Prefer routes from customers
    - Then prefer shortest paths







17.2 BGP Security Goals

- Confidential message exchange between neighbors
- **Validity of routing information**
  - **Origin, Path, Policy**
- Correspondence to the data path

17.3 Proposed Solution: S-BGP

- Based on public key infrastructure
- Address attestations
  - Claims the right to originate a prefix
  - Signed and distributed out of band
  - Checked through delegation chain from ICANN
- Route attestations
  - Attribute in BGP update message
  - Signed by each AS as route along path
- S-BGP can avoid
  - Prefix-hijacking
  - Addition, removal, or reordering of intermediate ASes

17.4 S-BGP Deployment

- Very challenging
  - PKI
  - Accurate address registries
  - Need to perform cryptographic operations on all path operations
  - Flag day almost impossible
  - Incremental deployment offers little incentive
- But there is hope! [Goldberg et al, 2011]

- Road to incremental deployment
- Change rules to break ties for secure paths
- If a few top tier-1 ISPs plus their respective stub clients
- deploy simplified version (just sign, not validate)
- Gains in traffic $\implies$ \$ $\implies$ adoption!

17.5 Data Plane Attacks

- Routers/ASes can advertise one route, but not necessarily follow it!
- May drop packets
  - Or a fraction of packets
  - What if you just slow down some traffic?
- Can send packets in a different direction
  - Impersonation attack
  - Snooping attack
- How to detect?
  - Congestion or an attack?
  - Can let ping/traceroute packets go through
  - End-to-end checks?
- Harder to pull off, as you need control of a router

17.6 IP Protocol

- Provides addressing and **forwarding**
  - Addressing is a set of conventions for naming nodes in an IP network
  - Forwarding is a local action by a router: passing a packet from input to output port
- IP forwarding finds output port based on destination address
  - Also defines certain conventions on how to handle packets (e.g. fragmentation, time to live)
- Contrast with **routing**
  - Routing is the process of determining how to map packets to output ports (topic of next two lectures)

17.7 Service Model

- Connectionless (datagram-based)
- Best-effort delivery (unreliable service)
  - packets may be lost
  - packets may be delivered out of order
  - duplicate copies of packets may be delivered
  - packets may be delayed for a long time
- It's the lowest common denominator
  - A network that delivers no packets fits the bill!
  - All these can be dealt with above IP (if probability of delivery is non-zero...)

17.8 IPv4 Packet Format

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td colspan="4">Version</td><td colspan="4">IHL</td><td colspan="8">Type of Service</td><td colspan="16">Total Length</td></tr>
<tr><td colspan="16">Identification</td><td colspan="3">Flags</td><td colspan="13">Fragment Offset</td></tr>
<tr><td colspan="8">Time to Live</td><td colspan="8">Protocol</td><td colspan="16">Header Checksum</td></tr>
<tr><td colspan="32">Source Address</td></tr>
<tr><td colspan="32">Destination Address</td></tr>
<tr><td colspan="24">Options</td><td colspan="8">Padding</td></tr>
<tr><td colspan="32">data</td></tr>
</table>

0	1	2	
0	DF	MF	} Flags

17.9 IP Header Details

- Forwarding based on destination address
- TTL (time-to-live) decremented at each hop
  - Originally was in seconds (no longer)
  - Mostly prevents forwarding loops
  - Other cool uses...
- Fragmentation possible for large packets
  - Fragmented in network if crossing link with small frame
  - MF: more fragments for this IP packet
  - DF: don't fragment (returns error to sender)
- Following IP header is "payload" data
  - Typically beginning with TCP or UDP header

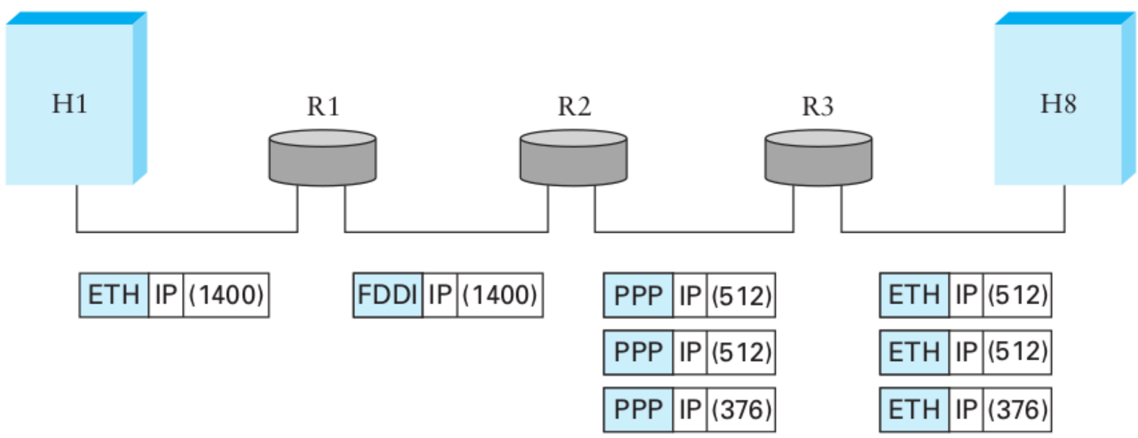
17.10 Other fields

- Version: 4 (IPv4) for most packets, there is also 6
- IHL: Internet Header Length: in 32-bit units (>5 implies options)
- Type of Service (won't go into this)
- Protocol Identifier (TCP: 6, UDP: 17, ICMP: 1, ...)
- Checksum over the header

17.11 Fragmentation and Reassembly

- Each network has maximum transmission unit (MTU)
- Strategy
  - Fragment when necessary (MTU < size of datagram)
  - Source tries to avoid fragmentation (why?)
  - Re-fragmentation is possible
  - Fragments are self-contained datagrams
  - Delay reassembly until destination host
  - No recovery of lost fragments

17.12 Fragmentation Example



- Ethernet MTU is 1,500 bytes
- PPP MTU is 576 bytes
  - R2 must fragment IP packets to forward them
- IP addresses plus identification field identify fragments of same packet
- MF (more fragments bit) is 1 in all but last fragment
- Fragment offset multiple of 8 bytes

- Multiply offset by 8 for fragment position original packet

17.13 Internet Control Message Protocol (ICMP)

- Echo (ping)
- Redirect
- Destination unreachable (protocol, port, or host)
- TTL exceeded
- Checksum failed
- Reassembly failed
- Can't fragment
- Many ICMP messages include part of packet that triggered them
- see https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml

17.14 ICMP message format

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

20-byte IP header (protocol=1-ICMP)		
Type	Code	Checksum
Depend on type/code		

17.15 Example: Time Exceeded

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

20-byte IP header (protocol=1-ICMP)		
Type=11	Code	Checksum
Unused		
IP header + first 8 payload bytes of packet that caused ICMP to be generated		

17.16 Translating IP to lower level addresses

- Map IP addresses into physical addresses
  - e.g. ethernet address of destination host
  - or ethernet address of next hop router
- Techniques
  - Encode physical address in host part of IP address (IPv6)
  - Each network node maintains lookup table (IP→physical)

17.17 ARP - *address resolution protocol*

- Dynamically builds table of IP to physical address bindings
- Broadcast request if IP address not in table
- All learn IP address of requesting node (broadcast)
- Target machine responds with its physical address
- Table entries are discarded if not refreshed

17.18 ARP Ethernet frame format

<table>
<tr><th>0</th><th>8</th><th>16</th><th>31</th></tr>
<tr><td colspan="2">`Hardware Type` = 1</td><td colspan="2">`ProtocolType` = `0x0800`</td></tr>
<tr><td>`HLen` = 48</td><td>`PLen` = 32</td><td colspan="2">Operation</td></tr>
<tr><td colspan="4">`SourceHardwareAddr` (bytes 0 - 3)</td></tr>
<tr><td colspan="2">`SourceHardwareAddr` (bytes 4 - 5)</td><td colspan="2">`SourceProtocolAddr` (bytes 0 - 1)</td></tr>
<tr><td colspan="2">`SourceProtocolAddr` (bytes 2 - 3)</td><td colspan="2">`TargetHardwareAddr` (bytes 0 - 1)</td></tr>
<tr><td colspan="4">`TargetHardwareAddr` (bytes 2 - 5)</td></tr>
<tr><td colspan="4">`TargetProtocolAddr` (bytes 0 - 3)</td></tr>
</table>

17.19 Format of IP Addresses

- Globally unique (or made to seem that way)
  - 32-bit integers, read in groups of 8-bits: 128.148.32.110
- Hierarchical: network + host
- Originally, a routing prefix embedded in address

0	1 … 7	8 … 31
0	Network	Host

0	1	2 … 15	16 … 31
1	0	Network	Host

0	1	2	3 … 23	24 … 31
1	1	0	Network	Host

  - Class A (8-bit prefix), B (16-bit), C (24-bit)
  - Routers need only know route for each network

17.20 Forwarding Tables

- Exploit hierarchical structure of addresses: need to know how to reach **networks**, not hosts

Network	Next Address
212.31.32.*	0.0.0.0
18.*.*.*	212.31.32.5
128.148.*.*	212.31.32.4
Default	212.31.32.1

- Keyed by network portion, not entire address
- Next address should be local

17.21 Classed Addresses

- Hierarchical: network + host
  - Saves memory in backbone routers (no default routers)
  - Originally, routing prefix embedded in address
  - Routers in same network must share network part
- Inefficient use of address space
  - Class C with 2 hosts (2/255 = 0.78% efficient)
  - Class B with 256 hosts (256/65535 = 0.39% efficient)
  - Shortage of IP addresses
  - Makes address authorities reluctant to give out class B's
- Still too many networks
  - Routing table does not scale

- Routing protocols do not scale

17.22 Subnetting

Network Number	Host Number

Class B Address

111111111111111111111111	00000000

Subnet Mask (255.255.255.0)

Network Number	Subnet ID	Host ID

Subnetted Address

- Add another level to address/routing hierarchy
- **Subnet mask** defines variable portion of host part
- Subnets visible only within site
- Better use of address space

17.23 Supernetting

- Assign blocks of contiguous networks to nearby networks
- Called CIDR: Classless Inter-Domain Routing
- Represent blocks with a single pair
  - (first network address, count)
- Restrict block sizes to powers of 2
- Use a bit mask (CIDR mask) to identify block size
- Address aggregation: reduce routing tables

17.24 CIDR Forwarding Table

Network	Next Address
212.31.32/24	0.0.0.0
18/18	212.31.32.5
128.148/16	212.31.32.4
128.148.128/17	212.31.32.8
0/0	212.31.32.1

17.25 Obtaining IP Addresses

- Blocks of IP addresses allocated hierarchically
  - ISP obtains an address block, may subdivide

	IP	Binary
ISP	128.35.16/20	10000000 00100011 00010000 00000000
Client 1	128.35.16/22	10000000 00100011 00010000 00000000
Client 2	128.35.20/22	10000000 00100011 00010100 00000000
Client 3	128.35.24/21	10000000 00100011 00011000 00000000

- Global allocation: ICANN, /8's (**ran out!**)
- Regional registries: ARIN, RIPE, APNIC, LACNIC, AFRINIC

18 NAT and Link Layer

18.1 Obtaining Host IP Addresses – DHCP

- Networks are free to assign addresses within block to hosts
- Tedious and error-prone, e.g. laptop moving between buildings
- Solution: Dynamic Host Configuration Protocol
  - Client: DHCP Discover to 255.255.255.255 (broadcast)
  - Server(s): DHCP Offer to 255.255.255.255

- Client: choose offer, DHCP Request
- DHCP ACK

- Result: address, gateway, netmask, DNS Server

18.2 We're running out of internet addresses

We're running out of internet addresses

Don't panic, but we're running out of internet addresses.
Not domain names – those website names that you see at the top of this page and which always start with some semblance of "http://" and "www."
We've got plenty of those.
But, according to statements from prominent internet thinkers this week, we may run out of internet protocol – or IP – addresses in less than a year.
IP addresses are numbers assigned to all of the devices – computers, phones, cars, wireless sensors, etc. – that log on to the internet.
According to the blog ReadWriteWeb, the internet is changing and evolving so quickly – with so many new types of devices connecting – that we're running out of numbers to assign to all of these Web-enabled electronics.
https://www.cnn.com/2010/TECH/innovation/07/23/internet.addresses/index.html

18.3 The internet has (kind of) run out of space

The internet has (kind of) run out of space

On Thursday, the internet as we know it ran out of space.
The nonprofit group that assigns addresses to service providers announced that, on Thursday morning, it allocated the last free internet addresses available from the current pool used for most of the internet's history.
"This is an historic day in the history of the internet, and one we have been anticipating for quite some time," said Raul Echeberria, chairman of the Number Resource Organization.
But fear not. The group has seen this coming for more than a decade and is ready with a new pool of addresses that it expects to last, well, forever.
John Curran, CEO of the American Registry for Internet Numbers, said the old pool of Internet Protocol addresses had about 4.3 billion addresses.
"A billion sounds like a lot," Curran said Thursday morning. "But when you think that there's nearly 7 billion people on the planet, and you're talking about two, three, four, five addresses per person (for some Web users), obviously 4.3 billion isn't enough."
http://edition.cnn.com/2011/TECH/web/02/03/internet.addresses.gone/index.html

18.4 The Last 5 Allocations

```
102/8   AfriNIC    2011-02   whois.afrinic.net   ALLOCATED!
103/8   APNIC      2011-02   whois.apnic.net     ALLOCATED!
104/8   ARIN       2011-02   whois.arin.net      ALLOCATED!
179/8   LACNIC     2011-02   whois.lacnic.net    ALLOCATED!
185/8   RIPE NCC   2011-02   whois.ripe.net      ALLOCATED!
```

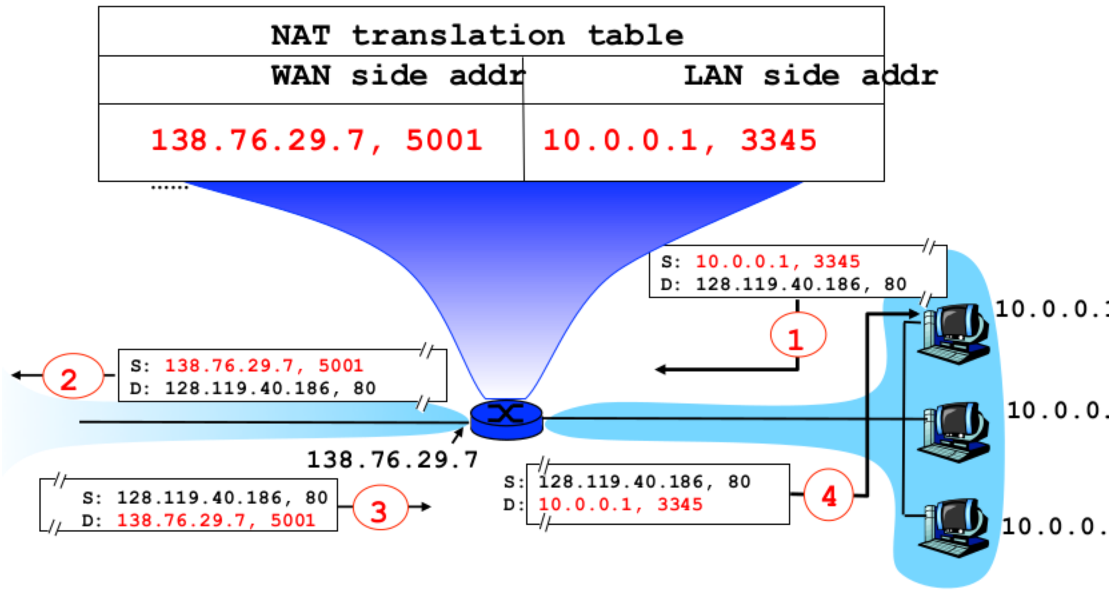
- IP addresses: 2^32 is only 4 billion
- How do we connect devices if we run out of IP addresses?
  - IPv6
  - Other solutions?

18.5 Port-Translating NAT

- Two hosts communicate with the same destination
  - Destination needs to differentiate between the two
- Map outgoing packets
  - Change source address and source port

- Maintain a translation table
  - Map of (`src addr`, `port #`) to (`NAT addr`, `new port #`)
- Map incoming packets
  - Map the destination address/port to the local host

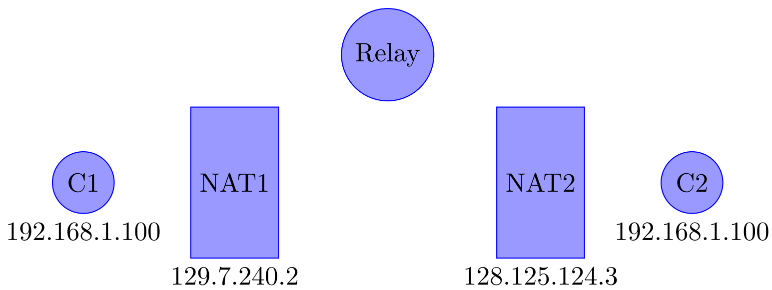
18.6 Network Address Translation Example



18.7 Maintaining the Mapping Table

- Create an entry upon seeing an outgoing packet
  - Packet with new (`src addr`, `src port`) pair
- Eventually, need to delete entries to free up #'s
  - When? If not pockets arrive before a timeout
  - (At risk of disrupting a temporarily idle connection)
- An example of **soft state**
  - i.e., removing state if not refreshed for a while

18.8 P2P Connections across NAT



18.9 NAT Traversal

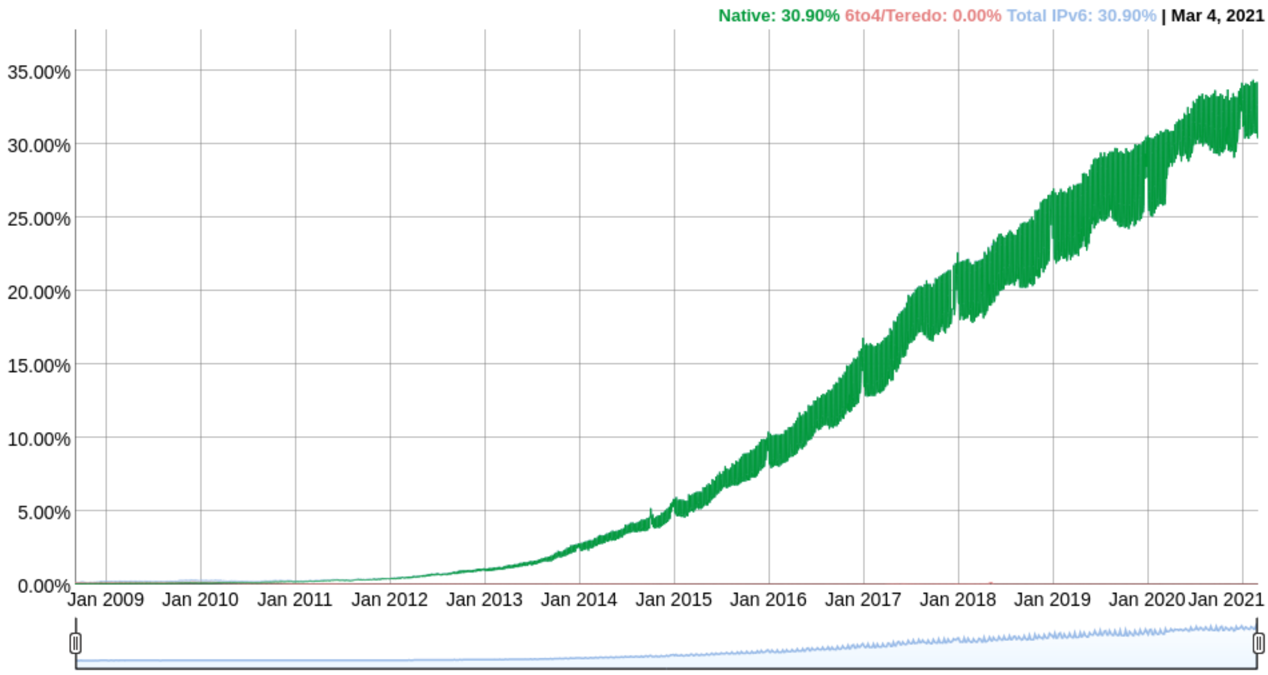
- How do we connect to "servers" behind a NAT?

https://en.wikipedia.org/wiki/NAT_traversal

18.10 IPv6

- Address space 128 bits
- Other features
  - Multicast
  - Stateless addressing

18.11 IPv6 Adoption



18.12 Link Layer

- Error Detection
- Reliability
- Media Access
- Ethernet

18.13 Error Detection

- Idea: add redundant information to catch errors in packet
- Used in multiple layers
- Three examples:
  - Parity
  - Internet Checksum
  - CRC

18.14 Simplest Schemes

- Repeat Frame
  - High overhead
  - Can't correct error
- Parity
  - Can detect odd number of bit errors
  - No correction

18.15 Reliable Delivery

- Error detection can discard bad packets
- Problem: if bad packets are lost, how can we ensure reliable delivery?
  - Exactly-once semantics = at least once + at most once

18.16 At Least Once Semantics

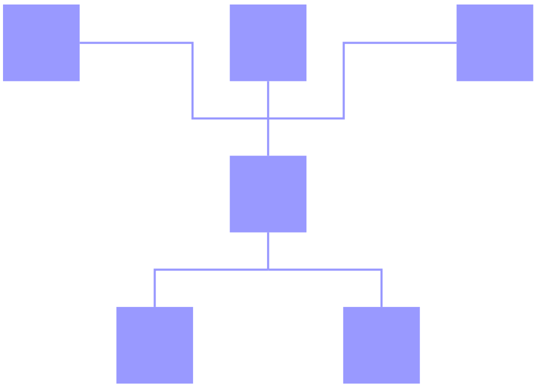
- How can the sender know the packet arrived **at least once**?
  - Acknowledgements + Timeout
- Stop and Wait Protocol
  - S: Sent packet, wait
  - R: Receive packet, send ACK
  - S: Receive ACK, send next packet
  - S: No ACK, timeout and retransmit

18.17 Stop and Wait Problems

- Duplicate Data
- Duplicate ACKs
- Can't fill pipe (remember bandwidth-delay product)
- Difficult to set the timeout value

19 Link Layer – Media Access and Switching

19.1 Wired Media Access



19.2 Link Layer

- Single-hop addressing
- Media Access
- Single-hop reliability

19.3 Media Access Control

- Control access to shared physical medium
  - e.g. who can talk when?
  - If everyone talks at once, no one hears anything
  - Job of the Link Layer
- Two conflicting goals
  - Maximize utilization when one node sending
  - Aproach $1/n$ when n nodes sending

19.4 Different Approaches

- Partitioned Access
  - Time Division Multiple Access (TDMA)
  - Frequency Division Multiple Access (FDMA)
  - Code Division Multiple Access (CDMA)
- Random Access
  - ALOHA/Slotted ALOHA
  - Carrier Sense Multiple Access/Collision Detection (CSMA/CD)
  - Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA)
  - RTS/CTS (Request to Send/Clear to Send)
  - Token-based

19.5 Collision Detection

- Without minimum frame length, might not detect collision

19.6 Case Study: Ethernet (802.3)

- Dominant wired LAN technology
- Both Physical and Link Layer specification
- CSMA/CD
  - Carrier Sense / Multiple Access / Collision Detection
- Frame Format (Manchester Encoding):

64	48	48	16		32
preamble	Dest Addr	Src Addr	Type	Body	CRC

19.7 Ethernet MAC

- Problem: shared medium
  - 10Mbps: 2500m, with 4 repeaters at 500m
- Transmit algorithm
  - If line is idle, transmit immediately
  - Upper bound message size of 1500 bytes
  - Must wait 9.6µs between back-to-back frames
  - If line is busy: wait until idle and transmit immediately

19.8 Handling Collisions

- Collision detection (10Base2 Ethernet)
  - Uses Manchester encoding
  - Constant average voltage unless multiple transmitters
- If collision
  - Jam for 32 bits, then stop transmitting frame
- Collision detection constrains protocol
  - Imposes minimum packet size (64 bytes or 512 bits)
  - Imposes maximum network diameter (2500m)
  - Ensure transmission time $\geq 2\times$ propagation delay (why?)

19.9 When to transmit again?

- Delay and try again: exponential backoff
- nth time: $k \times 51.2$µs, for $k \in \left\{0, 1, \ldots, 2^{\min(n,10)} - 1\right\}$
  - 1st time: 0 or 51.2µs
  - 2nd time: 0, 51.2, 102.4, or 153.6µs
- Give up after several times (usually 16)

19.10 Capture Effect

- Exponential backoff leads to self-adaptive use of channel
- A and B are trying to transmit, and collide
- Both will back off either 0 or 51.2µs
- Say A wins
- Next time, collide again
  - A will wait between 0 or 1 slots
  - B will wait between 0, 1, 2, or 3 slots
- ...

19.11 Ethernet Addressing

- Globally unique, 48-bit unicast address per adapter
  - Example: 00:1c:43:00:3d:09 (Samsung adapter)
  - 24 msb: organization
  - `http://standards-oui.ieee.org/oui/oui.txt`
- Broadcast address: all 1
- Multicast address: first bit 1
- Adapter can work in **promiscuous** mode

19.12 Ethernet Efficient

$$\texttt{Efficienty} = \frac{1}{1 + \frac{5 d_{\text{prop}}}{d_{\text{trans}}}}$$

19.13 Scaling the Network

- Problem: all the nodes on the same wire
  - Collision
  - Capacity
- Solution
  - Group the nodes into separate networks connected by hubs or switches
  - Consequences?
- Very little Ethernet today is shared

19.14 Switching

- Switches must be able to, given a packet, determine the outgoing port
- 3 ways to do this:
  - Datagram Switching
  - Virtual Circuit Switching
  - Source Routing

19.15 Datagram Switching

- Each packet carries destination address
- Switches maintain address-based tables
  - Maps [destination address] : [out-port]
- Also called **connectionless** model

19.16 Datagram Switching

- No delay for connection setup
- Source can't know if network can deliver a packet
- Possible to route around failures
- Higher overhead per-packet
- Potentially larger tables at switches

19.17 Learning Bridges

- Idea: don't forward a packet where it isn't needed
  - If you know recipient is not on that port
- Learn hosts' locations based on source addresses
  - Build a table as you receive packets
- Table says when **not** to forward a packet
  - Doesn't need to be complete for **correctness**

19.18 Dealing with Loops

- Problem:people may create loops in LAN!
  - Accidentically, or to prevent redundancy
  - Don't want to forward packets indefinitely

19.19 Spanning Tree

- Need to disable ports, so that no loops in the network
- Like creating a spanning tree in a graph
  - View switches and networks as nodes, ports as edges

20 Swiching and Physical Layer

20.1 Spanning Tree

Spanning tree

In the mathematical field of graph theory, a spanning tree T of an undirected graph G is a subgraph that is a tree which includes all of the vertices of G. In general, a graph may have several spanning trees, but a graph that is not connected will not contain a spanning tree (see spanning forests below). If all of the edges of G are also edges of a spanning tree T of G, then G is a tree and is identical to T (that is, a tree has a unique spanning tree and it is itself).

https://en.wikipedia.org/wiki/Spanning_tree

20.2 Spanning Tree Algorithms

- Graph search algorithms
- Dijkstra's algorithm
- Minimum-spanning Tree Algorithms

20.3 Spanning Tree

- Need to disable ports, so that no loops in the network
- Like creating a spanning tree in a graph
  - View switches and networks as nodes, ports as edges

20.4 Distributed Spanning Tree Algorithm

- Every bridge has a unique ID (Ethernet Address)
- Goal:
  - Bridge with the smallest ID is the root
  - Each segment has one designated bridge, responsible for forwarding its packets towards the root
    * Bridge closest to root is designated bridge
    * If there is a tie, bridge with lowest ID wins

20.5 Spanning Tree Protocol

- Spanning Tree messages contain:
  - ID of bridge sending the message
  - ID sender believes to be the root
  - Distance (in hops) from sender to root
- Bridges remember best config msg on each port
- Send message when you think you are the root
- Otherwise, forward messages from best known root
  - Add one to distance before forwarding
  - Don't forward if you know you aren't dedicated bridge

20.6 Limitations of Bridges

- Scaling
  - Spanning tree algorithm doesn't scale
  - Broadcast does not scale
  - No way to route around congested links, **even if path exists**
- May violate assumptions
  - Could confuse some applications that assume single segment
  - Much more likely to drop packets
  - Makes latency between nodes non-uniform
  - Beware of transparency

20.7 Local Area Network

20.8 Physical Layer

- Responsible for specifying the physical medium
  - Type of cable, fiber, wireless frequency
- Responsible for specifying the signal (modulation)
  - Transmitter varies **something** (amplitude, frequency, phase)
  - Receiver samples, recovers signal
- Responsible for specifying the bits (encoding)
  - Bits above physical layer $\to$ **chips**

20.9 Specifying the signal

- Chips vs. bits
  - Chips: data (in bits) at the physical layer
  - Bits: data above the physical layer
- Physical layer specifies Analog signal $\Leftrightarrow$ chip mapping
  - On-ioff keying (OOK): voltage of 0 is 0, $+V$ is 1
  - PAM-5: 000 is 0, 001 is $+1$, 010 is -1, 011 is -2, 100 is $+2$

- Frequency shift keying (FSK)
- Phase shift keying (PSK)

20.10 Modulation

- Specifies mapping between digital signal and some variation in analog signal
- Why not just a square wave ($1V = 1$, $0V = 0$)?
  - Not square when bandwidth limited
- Bandwidth – frequencies that a channel propagates well
  - Signals consist of many frequency components
  - Attenuation and delay frequency-dependent

20.11 Use Carriers

- Idea: use only frequencies that transmit well
- **Modulate** the signal to encode bits

20.12 How Fast can you Really Send?

- Depends on frequency and signal/noise ratio
- Shannong $C = B\log_2(1 + S/N)$
  - C is the channel capacity in bits/second
  - B is the bandwidth of the channel in Hz
  - S and N are average signal and noise power
- Example: Telephone Line
  - 3KHz bandwith, 30dB$S/N = 10^{30/10} = 1000$
  - $C \approx 30$Kbps

20.13 Encoding

- Now assume that we can somehow modulate a signal: receiver can decode our binary stream
- How od we encode binary data onto signals?
- One approach: Non-return to Zero (NRZ)
  - Transmit 0 as low, 1 as high!

20.14 Drawbacks of NRZ

- No signal could be interpreted as 0 (or vice-versa)
- Consecutive 1s or 0s are problematic
- Baseline wander problem
  - How do you set the threshold?
  - Could compare to average, but average may drift
- Clock recovery problem
  - For long runs of no change, could miscount periods

20.15 Alternative Encodings

- Non-return to Zero Inverted (NRZI)
  - Encode 1 with transition from current signal
  - Encode 0 by staying at the same level
  - At least solves problem of consecutive 1s

20.16 Manchester

- Map $0 \rightarrow$ chips 01; $1 \rightarrow$ chips 10
  - Transmission rate now 1 bit per two clock cycles
- Solves clock recovery, baseline wander
- But cuts transmission rate in half

20.17 4B/5B

- Can we have a more efficient encoding?
- Every 4 bits encoded as 4 **chips**
- Need 16 5-bit codes:
  - selected to have no more than one leading 0 and no more than two trailing 0s
  - Never get more than 3 consecutive 0s
- Transmit chips using NRZI

- Other codes used for other purposes
  - e.g. 11111: line idle; 00100: half
- Achieves 80% efficiency

20.18 Encoding Goals

- DC Balancing (same number of 0 and 1 chips)
- Clock synchronization
- Can recover some chip errors
- Constrain analog signal patterns to make signal more robust
- Want near channel capacity with negligible errors
  - Shannon says it's possible, doesn't tell us how
  - Codes can get computationally expensive
- In practice
  - More complex encoding: fewer bps, more robust
  - Less complex encoding: more bps, less robust

20.19 802.15.4

- Standard for low-power, low-rate wireless PANs
  - Must tolerate high chip error rates
- Uses a 4B/32B bit-to-chip encoding

Symbols

Bits	Chips
0000	11011001110000110101001000101110
0001	11101101100111000011010100100010
0010	00101110110110011100001101010010
0011	00100010111011011001110000110101
0100	01010010001011101101100111000011
0101	00110101001000101110110110011100
0110	11000011010100100010111011011001
0111	10011100001101010010001011101101
1000	10001100100101100000011101111011
1001	10111000110010010110000001110111
1010	01111011100011001001011000000111
1011	01110111101110001100100101100000
1100	00000111011110111000110010010110
1101	01100000011101111011100011001001
1110	10010110000001110111101110001100
1111	11001001011000000111011110111000

20.20 Framing

- Given a stream of bits, how can we represent boundaries?
- Break sequence of bits into a frame
- Typically done by a network adaptor

20.21 Representing Boundaries

- Sentinels
- Length counts
- Clock-based

20.22 Bit-Oriented Protocols

- View message as a stream of bits, not bytes
- Can use sentinel approach as well, (e.g. HDLC)
  - HDLC begin/end sequence 01111110
- Use **bit stuffing** to escape 01111110
  - Always append 0 after five consecutive 1s in data

    - After five 1s, receiver uses next two bits to decide if stuffed, end of frame, or error

20.23 Length-based Framing

- Drawback of sentinel techniques
    - Length of frame depends on data
- Alternative: put length in header (e.g. DDCMP)
- Dancer: Framing Errors
    - What if high bit counter gets corrupted?
    - Adds 8K to length of frame, may lose many frames
    - CRC checksum helps detect error

21 Wireless Networks

21.1 Wireless

- Today: wireless networking truly ubiquitous
    - 802.11, 3G, 4G, 5G, WiMAX, Bluetooth, RFID, ...
    - Sensor networks, Internet of Things (IoT)
    - Some new computers have no **wired** networking
- What's behind the scenes?

21.2 Wireless is different

- Signals sent by the sender don't always reach the receiver intact
    - Varies with **space**: attenuation, multipath
    - Varies with **time**: conditions change, intereference, mobility
- **Distributed**: sender doesn't know what happens at receiver
- Wireless medium is inherently **shared**
    - No easy way out with switches

21.3 Implications

- Different mechanisms needed
- Physical layer
    - Different knobs: antennas, transmission power, encodings
- Link Layer
    - Distributed medium access protocols
    - Topology awareness
- Network, Transport Layers
    - Routing, forwarding
- Most advances **do not** abstract away the physical and link layers

21.4 Interference

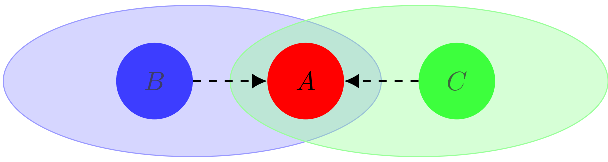
- External sources
    - e.g. 2.4GHz unlicensed ISM band
    - 802.11
    - 802.15.4 (ZigBee), 802.15.1 (Bluetooth)
    - 2.4GHz phones
    - Microwave ovens
- Internal sources
    - Nodes in the same network/protocol can (and do) interfere
- Multipath
    - Self-interference (destructive)

21.5 Link Layer

- Medium Access Control
    - Should give 100% if one user
    - Should be efficient and fair if more users
- Ethernet uses CSMA/CD
    - Can we use CD here?
- No! Collision happens at the receiver

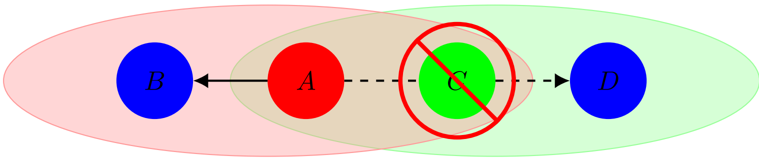
- Protocls try to **avoid** collision in the first place

21.6 Hidden Terminals



- A can hear B and C
- B and C can't hear each other
- They both interfere at A
- B is a **hidden terminal** to C, and vice-versa.
- **Carrier sense at sender is useless**

21.7 Exposed Terminals



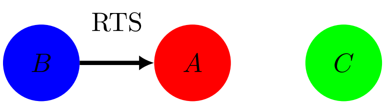
- A transmits to B
- C hears the transmission, backs off, even though D would hear C
- They both interfere at A
- C is an **exposed terminal** to A's transmission

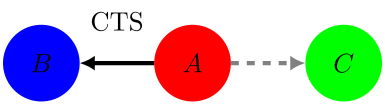
21.8 Key points

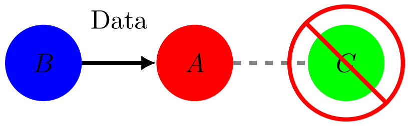
- No global view of collision
  - Different receivers hear different senders
  - Different senders reach different receivers
- Collisions happen at the **receiver**
- Goals of a MAC protocol
  - Detect if receiver can hear sender
  - Tell senders who might interfere with receiver to shut up

21.9 RTS/CTS

- Idea: transmitter can check availabilty of channel at receiver
- Before every transmission
  - Sender sends an RTS (Request-to-Send)
  - Contains length of data (in **time** units)
  - Receiver sends a CTS (Clear-to-Send)
  - Sender sends data
  - Receiver sends ACK after transmission
- If you don't hear a CTS, assume collision
- If you hear a CTS for someone else, shut up







21.10 Benefits of RTS/CTS

- Solves hidden terminal problem
- Does it?
  - Control frames can still collide
  - e.g. can cause CTS to be lost
  - In practice: reduces hidden terminal problem on data packets

21.11 Drawbacks of RTS/CTS

- Overhead is too large for small packets
  - 3 packets per packet: RTS/CTS/Data (4-22% for 802.11b)
- RTS still goes through CSMA: can be lost
- 33% of IP packets are TCP ACKs
- **In practice, WiFi doesn't use RTS/CTS**

21.12 Other MAC Strategies

- Time Division Multiplexing (TDMA)
  - Central controller allocates a time slot for each sender
  - May be inefficient when not everyone is sending
- Frequency Division
  - Multiplexing two networks on same space
  - Nodes with two radios (think graph coloring)
  - Different frquency for upload and download

21.13 Network Layer

- What about the network topology?
- Almost everything you use is **single hop**!
  - 802.11 in infrastructure mode
  - Bluetooh
  - Cellular networks
  - WiMax (some 4G Networks)
- Why?
  - Really hard to make multihop wireless efficient

21.14 WiFi Distribution System

- 802.11 typically works in **infrastructure mode**
  - Access points – fixed noeds on a wired network
- Distributed system connects APs
  - Typically connect to the same Ethernet, use learning bridge to route to nodes' MAC addresses
- Association
  - Node negotiates with AP to get access
  - Security negotiated as well (WEP, WPA, etc)
  - Passive or Active

21.15 Wireless Multi-Hop Networks

- Some networks are multihop, though!
  - Ad-hoc networks for emergency areas
  - Vehicular Networks
  - Sensor Networks
    - e.g. infrastructure monitoring
  - Multihop networking to share Internet access

21.16 What can happen to signals?

- Attenuation

- Signal power attenuates by $\approx r^2$ factor for omni-directional antennas in a free-space
- Exponent depends on type and placement of antennas
  - < 2 for directional antennas
  - > 2 if antennas are close to the ground

21.17 Interference

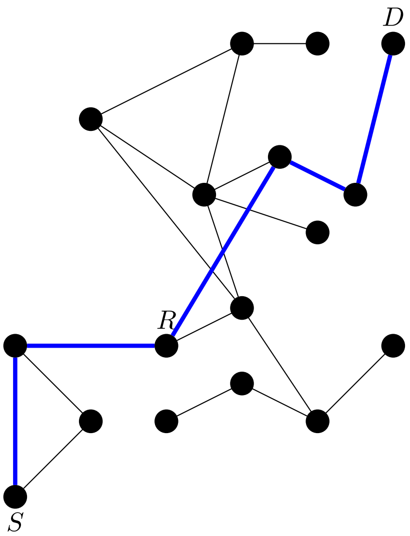
21.18 Multipath

- May cause attenuation, destructive intereference

21.19 Many Challenges

- Routing
  - Link estimation
- Multihop throughput dropoff

21.20 The Routing Problem



- Find a route from S to D
- Topology can be very dynamic

21.21 Routing

- Routing in ad-hoc networks has had a lot of research
  - General problem: any-to-any routing
  - Simplified versions: any-to-one (base station), one-to-any (dissemination)
- DV too brittle: inconsistencies can cause loops
- DSDV
  - Destination Sequenced Distance Vector

21.22 DSDV

- Charles Perkins (1994)
- Avoid loops by using sequence numbers
  - Each destination increments own sequence
    - Only use **even** numbers
  - A node selects a new parent if
    - Newer sequence number or
    - Same sequence number and **better** route
- If disconnected, a node increments destination sequence number to next **odd** number!
- No loops (only transient loops)
- Slow: on some changes, need to wait for root

21.23 Many Others

- DSR, AODV: on-demand

- Geographic routing: use nodes' physical location and do greedy routing
- Virtual coordinates: derive coordinates from topology, use greedy routing
- Tree-based routing with on-demand shortcuts
- ...

22 Multihop Wireless Networks, Security, and RPL

22.1 Routing Metrics

- How to choose between two routes?
- Hopcount is a poor metric!
  - Paths with few hops may use long, marginal links
  - Must find balance
- All links do **local retransmissions**

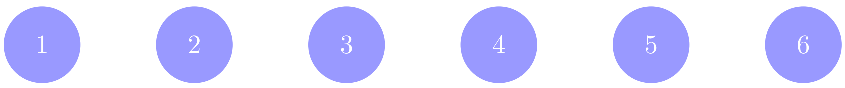
22.2 Link Quality Estimation

$$\texttt{ETX}(L) = \frac{1}{\text{PRR}(f) \cdot \text{PRR}(b)}$$

22.3 Routing Metrics

- Idea: use expected transmissions over a link as its cost!
  - $\texttt{ETX} = 1/\texttt{PRR}$ (Packet Reception Rate)
  - Variation: `ETT`, takes data rate into account

22.4 Multihop Throughput



- Only every third node can transmit!
  - Assuming a node can talk to its immediate neighbors
    (1) Nodes can't send and receive at the same time
    (2) Third hop transmission prevents second hop from receiving
    (3) Worse if you are doing link-local ACKs
- In TCP, problem is worse: data and ACK

22.5 Sometimes you can't (or shouldn't) hide that you are on wireless!

22.6 TCP over wireless

- How to handle
  - Link losses
  - Hop-by-hop retransmissions
  - Congestion vs. lossy links

22.7 Security

22.8 Basic Requirements for Secure Communication

- **Availabilty:** Will the network deliver data?
  - Infrastructure compromise, DDoS
- **Authentication:** Who is this actor?
  - Spoofing, phishing
- **Integrity:** Do messages arrive in original form?
- **Confidentiality:** Can adversary read the data?
  - Sniffing, man-in-the-middle
- **Provenance:** Who is responsible for this data?
  - Forging responses, denying responsibility
  - Not who sent the data, but who created it

22.9 Other Desirable Security Properties

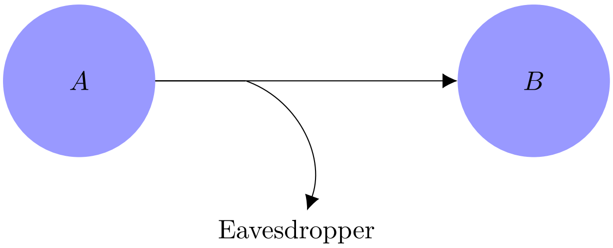
- **Authorization:** is actor allowed to do this action?
  - Access controls
- **Accountability/Attribution:** who did this activity?
- **Audit/Forensics:** what occurred in the past?
  - A broader notion of accountability/attribution
- **Appropriate use:** is action consistent with policy?
  - e.g. no spam, no games during business hours
- **Freedom from traffic analysis:** can someone tell when I am sending and to whom?
- **Anonymity:** can someone tell I sent this packet?

22.10 Internet's Design: Insecure

- Designed for simplicity in a naïve era
- "On by default" design
- Readily available zombie machines
- Attacks look like normal traffic
- Internet's federated operation obstructs cooperation for diagnosis/mitigation

22.11 Eavesdropping – Message Interception (Attack on Confidentiality)

- Unauthorized access to information
- Packet sniffers and wiretappers
- Illicit copying of files and programs

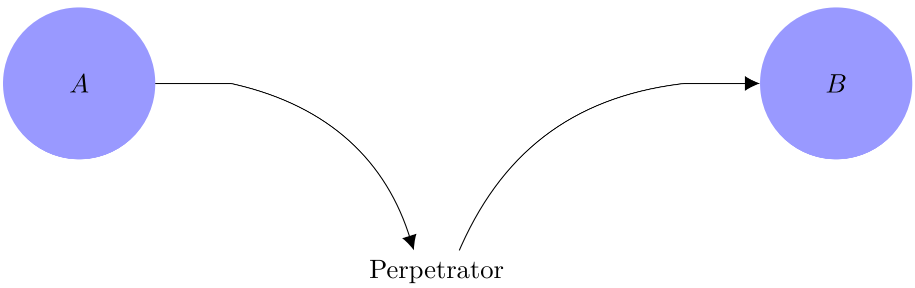


22.12 Eavesdropping Attack: Example

- tcpdump with promiscuous network
  - On a switched network, what can you see?
- What might the following traffic types reveal about communications?
  - DNS lookups (and replies)
  - IP packets without payloads (headers only)
  - Payloads

22.13 Integrity Attacks – Tampering

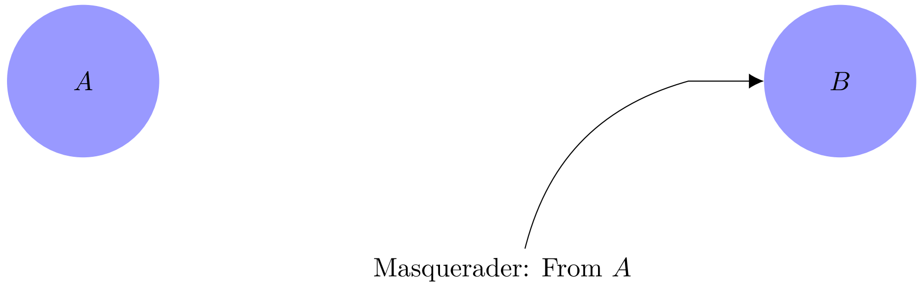
- Stop the flow of the message
- Delay and optionally modify the emssage
- Release the message again



22.14 Authenticity Attack – Fabrication

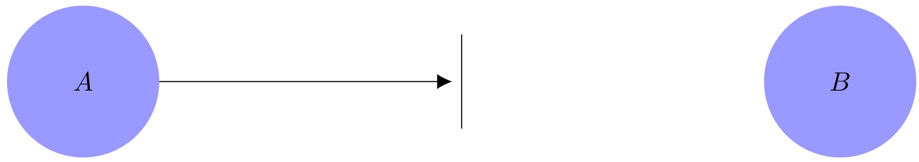
- Unauthorized assumption of other's identitiy

- Generate and distribute objects under this identity



22.15 Attack on Availability

- Destroy hardware (cutting fiber) or software
- Modify software in a subtle way
- Corrupt packets in transit



- Blatant **denial of service** (DoS):
  - Crashing the server
  - Overwhelm the server (use up its resource)

23 Security – Encryption, Integrity, Authentication, Certificate, HTTPS, and Pharming

23.1 Confidentality through Cryptography

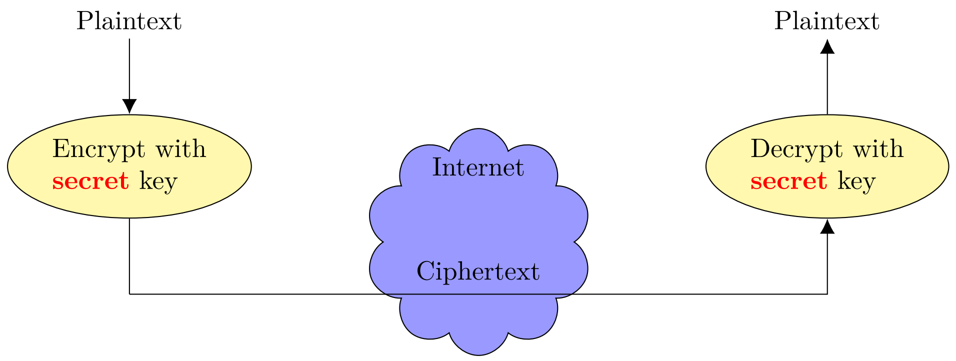
- Cryptography **communication over insecure channel in the presence of adversaries**
- Central goal: how to encode information so that an adversary can't extract it, ... but a friend can
- General premise: a **key** is required for decoding
  - Give it to friends, keep it away from attackers
- Two different categories of encryption
  - Symmetric: efficient, requires key distribution
  - Asymmetric (Public Key): computationally expensive, but no key distribution problem

23.2 Symmetric Key Encryption

- Same key for encryption and decryption
  - Both sender and receiver know key
  - But adversary does not know key
- For communication, problem is **key distribution**
  - How do the parties (securely) agree on the key?
- What can you do with a huge key? One-time pad
  - Huge key of random bits
- To encrypt/decrypt, just `XOR` with the key!
  - **Provably secure!** ... provided:
    - You never reuse the key, ... and it really is random/unpredictable
  - Spies actually use these

23.3 Using Symmetric Keys

- Both the sender and the receiver use the same symmetric key

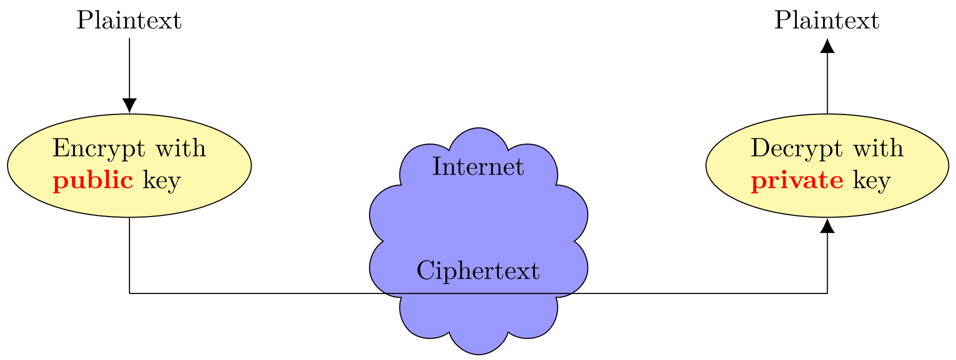


23.4 Asymmetric Encryption (Public Key)

- Idea: use two **different** keys, one to encrypt (**e**) and one to decrypt (**d**)
  - A **key pair**
- Crucial property: knowing **e** does not give away **d**
- Therefore **e** can be public: everyone knows it!
- If Alice wants to send to Bob, she fetche's Bob's public key (say, from Bob's home page) and encrypts with it
  - **Alice** can't decrypt what she is sending to Bob...
    - ...but then, **neither can anyone else** (except Bob)

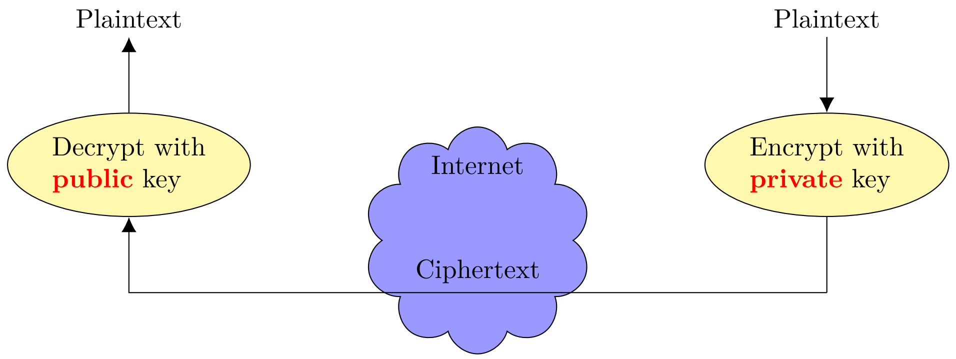
23.5 Public Key/Asymmetric Encryption

- Sender uses receiver's **public** key
  - Adverised to everyone
- Receiver uses complementary **private** key
  - Must be kept secret



23.6 Works in Reverse Direction Too!

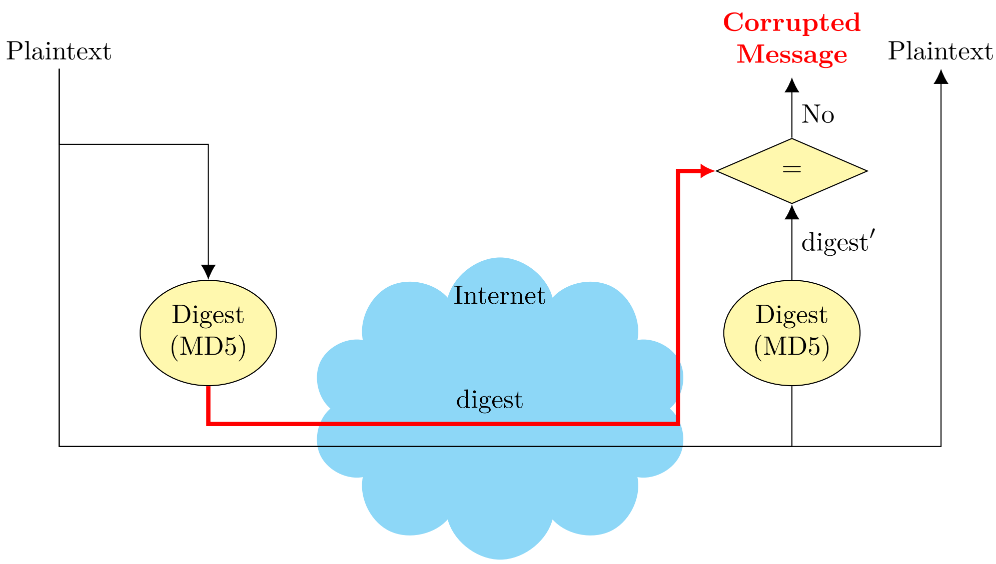
- Sender uses own **private** key
- Receiver uses complementary **public** key
- Allows sender to prove he knows private key



23.7 Integrity: Cryptographic Hashes

- Sender computes a **digest** of messages **m**, i.e., $H(m)$
  - H is a publicy known **hash function**
- Send m in any manner
- Send digest $\mathbf{d} = H(\mathbf{m})$ to receiver in a secure way:
  - Using another physical channel
  - Using encryption (why does this help?)
- Upon receiving **m** and **d**, receiver recomputs $H(\mathbf{m})$ to see whether result agrees with **d**.

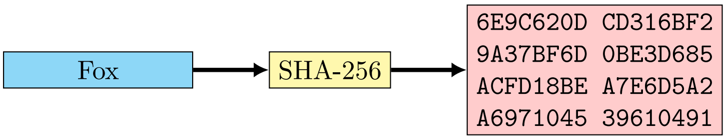
23.8 Operation of Hashing for Integrity

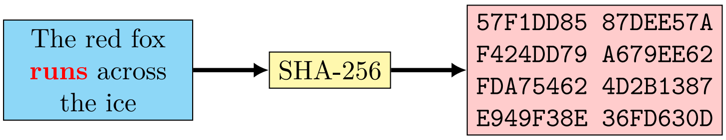


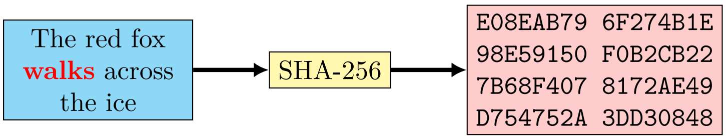
23.9 Cryptographically Strong Hashes

- Hard to find **collisions**
  - Adversary can't find two inputs that produce same hash
  - Someone cannot alter a message without modifying digest
  - Can succinctly refer to large objects
- Hard to **invert**
  - Given hash, adversary can't find input that produces it
  - Can refer obliquely to private objects (e.g. passwords)
    - Send hash of object rather than object itself

23.10 Effects of Cryptographic Hashing

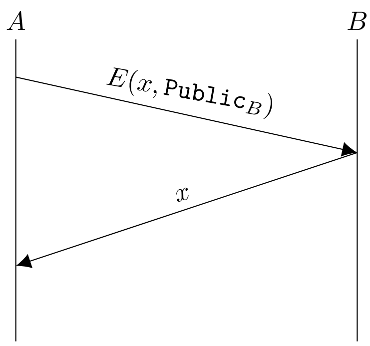






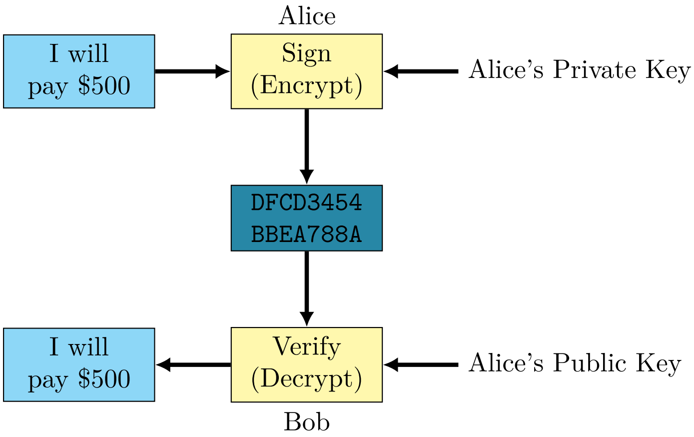
23.11 Public Key Authentication

- Each side only needs to know the other side's public key
  - No secret key needs to be shared
- A encrypts a `nonce` (random number) x using B's public key
- B proves it can recover x
- A can authenticate itself to B in the same way



23.12 Digital Signatures

- Suppose Alice has published public key K_E
- If she wishes to prove who she is, she can send a message x encrypted with her **private** key K_D
  - Therefore: anyone with public key K_E can recover x, verify that Alice must have sent the message
  - It provides a **digital signature**
  - Alice can't later deny it $\Longrightarrow$ **non-repudiation**



23.13 Public Key Infrastructure (PKI)

- Public key crypto is **very** powerful...
- ...by the **realities** of tying public keys to real world identities turn out to be quite hard
- PKI: **Trust distribution** mechanism
  - Authentication via **Digital Certificates**
- Trust doesn't mean someone is honest, just that they are who they say they are...

23.14 Managing Trust

- The most solid level of trust is rooted in our direct personal experience
  - e.g. Alice's trust that Bob is who they say they are
  - Clearly doesn't scale to a global network!
- In its absence, we rely on **delegation**
  - Alice trusts Bob's identity because Charlie attests to it...
  - ...and Alice trusts Charlie
- Trust is not particularly transitive
  - Should Alice trust Bob because she trusts Charlie...
  - ...and Charlie vouches for Donna ...
  - ...and Donna says Eve is trustworthy...
  - ...and Eve vouces for Bob's identity?
- Two models of delegating trust
  - Rely on your set of friends and their friends
    * "Web of Trust" – e.g. PGP
  - Rely on trust, well known authorities (**and their minions**)
    * "Trusted root" – e.g. HTTPS

23.15 PKI Conceptual Framework

- Trusted-Root PKI
  - Basis: well-known public key serves as **root** of a hierarchy
  - Managed by the Certificate Authority (CA)
- To publish a public key, ask the CA to digitally sign a statement indicating that they agree ("certify") that it is indeed your key
  - This is a **certificate** for your key (certificate = bunch of bits)
    * Includes both your public key and the signed statement
  - Anyone can verify the signature
- Delegation of trust to the CA
  - They'd better not screw up (duped into signing bogus key)
  - They'd better have procedures for dealing with stolen keys
  - Note: can build up a **hierarchy** of signing

23.16 HTTPS

- Steps after clicking `https://www.amazon.com`
- https = "use HTTP over SSL/TLS"

  - SSL = Secure Socket Layer
  - TLS = Transport Layer Security
    * Successor to SSL, and compatible with it
  - RFC 4346
- Provides security layer (authentication, encryption) on top of TCP
  - Fairly transparent to the app

23.16.1 HTTPS Connection via SSL/TLS

- Browser (client) connects via TCP to Amazon's `HTTPS` server
- Client sends over a list of crypto protocols it supports
- Server picks protocols to use for this session
- Server sends over its cerificate
- (all of this is in the clear)

23.17 Inside the Server's Certificate

- **Name** associated with the cert (e.g. Amazon)
- Amazon's **public key**
- A bunch of auxhiliary info (physical address, type of cert, expiration time)
- URL to **revocation center** to check for revoked keys
- Name of certificate's **signatory** (who signed it)
- A public-key signature of a has (**MD5**) of all this
  - Constructed using the signatory's private RSA key

23.18 Validating Amazon's Identity

- Example: certificate of entity Amazon

$$\texttt{Cert} = E(\{\text{Amazon}, \text{KAmazon}_{\text{public}}\}, \text{KCA}_{\text{private}})$$

- Browser retrieves cert belonging to the **signatory**
  - These are **hardwired into the browser**
- If it can't find the cert, it warns the user that the site has not been verified
  - And may ask whether to continue
  - Note, can still proceed, just **without authentication**
- Browser uses public key in signatory's cert to decrypt signature
- Assuming signature matches, not have high confidence it is indeed Amazon
  - ...assuming signatory is trustworth

23.19 HTTPS Connection (SSL/TLS)

- Browser constructs a random **session key** K
- Browser encrypts K using Amazon's public key
- Browser sends $E(K, KA_{\text{public}})$ to server
- Browser displays lock
- All subsequent communication encrypted with symmetric cipher using key K
  - e.g. client can authentication using a password

23.20 Pharming

- How can we get web clients to redirect to malicious sites?
- Name resolution
  - Send a query to a DNS
  - Trust the IP address returned by the DNS
  - Other ways to go from name to IP?

24 Policy

24.1 Internet and Policies Governing it

- Access
- Domain Names
- IP Addresses
- Government Control
- Privacy

- Neutrality
- Wireless Spectrum

24.2 Access

- Who determines what networks are accessible?
  - Websites
- Technology
  - Enforcement/Circumvention

24.3 Domain Names

- Who should determine what names are allowed?
  - Organization
  - TLDs
- Technology
  - DNS
  - Alternative DNS roots

24.4 IP Address

- Is IP Address your "property" once allocated?
  - Organization
- Technology
  - What prevents use of someone else's IP?

24.5 Government Control

- Should government "take over" critical network systems when under threat?
  - Legislation introduced
- Technology
  - What does "take over" mean?

24.6 Privacy

- Can we expect communication over the Internet to remain private?
  - Law enforcement (wiretapping)
    * Can your ISP turn over your emails to the government?
  - Company – employee/customers
- Technology
  - Encryption
  - Spyware

24.7 Net Neutrality

- Are ISPs required to be neutral to different services?
  - Comcast case
- Technology
  - How to detect if ISPs are neutral?

24.8 Google/Verizon Policy Recommendations

- Google and Verizon released their take on net neutrality debate in August 2010
- Support openness, neutrality, transparency in network access to third party and consumers
- ... but wireless is different

Facebook post by Reed Hastings, CEO of Netflix

Comcast no longer following net neutrality principles.
Comcast should apply caps equally, or not at all.
I spent the weekend enjoying four good internet video apps on my Xbox: Netflix, HBO GO, Xfinity, and Hulu.
When I watch video on my Xbox from three of these four apps, it counts against my Comcast internet cap. When I watch through Comcast's Xfinity app, however, it does not count against my Comcast internet cap.
For example, if I watch last night's SNL episode on my Xbox through the Hulu app, it eats up about one gigabyte of my cap, but if I watch that same episode through the Xfinity Xbox app, it doesn't use up my cap at all.
The same device, the same IP address, the same wifi, the same internet connection, but totally different cap treatment.
In what way is this neutral?

24.9 Wireless Spectrum

- Spectrum allocation
  - Auction
  - Alternatives?
- Should we be able to use any device on wireless networks?
  - Companies pay money to get the spectrum