Announcements / Pending Questions

- Homework #3 out - due Nov 6
  - Note, due date a week later than originally announced
  - Homework #4 to be out Nov 6, tentatively due Nov 27

- Additional office hours
  - Sukun this week: Friday 4-5PM
  - Me next week: Monday 1:30-3:30PM

- Physical-layer design issues: EE 121 (Spring ’07)

- Identifier for electing a Spanning Tree root
  - Switch’s MAC address + configurable priority knobs
  - (thanks to Gene Zhang & Wikipedia & Mike Bennett)

- How does switch know when spanning tree done?
  - It doesn’t, it just uses what it currently has
**Robust Spanning Tree Algorithm**

- Algorithm must react to failures
  - Failure of the root node
    - Need to elect a new root, with the next lowest identifier
  - Failure of other switches and links
    - Need to recompute the spanning tree
- Root switch continues sending messages
  - Periodically reannouncing itself as the root (1, 0, 1)
  - Other switches continue forwarding messages
- Detecting failures through timeout *(soft state)*
  - Switch waits to hear from others
  - Eventually times out and claims to be the root

*See Section 3.2.2 in the textbook for details and another example*

**Moving From Switches to Routers**

- Advantages of switches over routers
  - Plug-and-play
  - Fast filtering and forwarding of frames
- Disadvantages of switches over routers
  - Topology restricted to a spanning tree
  - Large networks require large ARP tables
  - Broadcast storms can cause the network to collapse
  - Can’t accommodate non-Ethernet segments (why not?)
Comparing Hubs, Switches & Routers

<table>
<thead>
<tr>
<th></th>
<th>hubs</th>
<th>switches</th>
<th>routers</th>
</tr>
</thead>
<tbody>
<tr>
<td>traffic isolation</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>plug &amp; play</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>optimized routing</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>cut-through</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>

Midterm Review

- In-class next Monday
- Closed book
- You can have one regular-sized (8.5”x11”) sheet of paper with notes on both sides
- No PDAs, calculators, electronic/Internet gadgets, smart cell phones, jeweler’s loupes, etc.
- No Blue Books - all answers on exam sheets
- Ensure legibility (pencil + eraser)
Fundamental Challenges for Networking

- Speed-of-light
- Desiring a pervasive global network
- Need for it to work efficiently/cheaply
- Failure of components
- Enormous dynamic range
  - “no such thing as typical”
- Disparate parties must work together
- Rapid growth/evolution
- Crooks & other bad guys

Taxonomy of Communication Networks

- Communication networks can be classified based on the way in which the nodes exchange information:
Circuit Switching (e.g., Phone Network)

- Establish: source creates circuit to destination
  - Node along the path store connection info
  - Nodes generally reserve resources for the connection
  - If circuit not available: “Busy signal”

- Transfer: source sends data over the circuit
  - No destination address, since nodes know path

- Teardown: source tears down circuit when done

Timing in Circuit Switching
Time-Division Multiplexing/Demultiplexing

- Time divided into frames; frames into slots
- Relative slot position inside a frame determines to which conversation data belongs
  - E.g., slot 0 belongs to orange conversation
- Requires synchronization between sender and receiver—surprisingly difficult!
- In case of non-permanent conversations
  - Need to dynamically bind a slot to a conversation
  - How to do this?
- If a conversation does not use its circuit the capacity is lost!

Timing of Datagram Packet Switching

transmission time of Packet 1 at Host 1

propagation delay between Host 1 and Node 1

processing delay of Packet 1 at Node 2

Packet 1
Packet 2
Packet 3
Packet 1
Packet 2
Packet 3
Packet 1
Packet 2
Packet 3
Packet-Switching vs. Circuit-Switching

• Critical advantage of packet-switching over circuit switching: \textit{Exploitation of statistical multiplexing}

• Another: since routers don’t know about individual conversations, when a router or link fails, it’s easy to fail over to a different path

• A third: easier for different parties to link their networks together because they’re not promising to reserve resources for one another

• However, packet-switching must handle congestion:
  – More complex routers
  – Harder to provide good network services (e.g., delay and bandwidth guarantees)

• In practice, sometimes combined, e.g., IP over SONET

Protocol Standardization

• Ensure communicating hosts speak the same protocol
  – Standardization to enable multiple implementations
  – Or, the same folks have to write all the software

• Standardization: Internet Engineering Task Force
  – Based on working groups that focus on specific issues
  – Produces “Request For Comments” (RFCs)
    • Promoted to standards via rough consensus and running code
  – IETF Web site is \url{http://www.ietf.org}
  – RFCs archived at \url{http://www.rfc-editor.org} (per Homework #1)

• De facto standards: same folks writing the code
  – P2P file sharing, Skype, <your protocol here>…
Layering: A Modular Approach

- Paritition the system
  - Each layer solely relies on services from layer below
  - Each layer solely exports services to layer above

- Interface between layers defines interaction
  - Hides implementation details
  - Layers can change without disturbing other layers

![Diagram of layering](image)

Drawbacks of Layering

- Layer N may duplicate lower level functionality
  - E.g., error recovery to retransmit lost data

- Layers may need same information
  - E.g., timestamps, maximum transmission unit size

- Strict adherence to layering may hurt performance
  - E.g., hiding details about what is really going on

- Some layers are not always cleanly separated
  - Inter-layer dependencies for performance reasons
  - Some dependencies in standards (header checknums)

- Headers start to get really big
  - Sometimes header bytes >> actual content
Layer Violations

- Sometimes the gains from not respecting layer boundaries too great to resist
- Can occur with higher-layer entity inspecting lower-layer information:
  - E.g., TCP-over-wireless system that monitors wireless link-layer information to try to determine whether packet loss due to congestion or corruption
- Can occur with lower-layer entity inspecting higher-layer information
  - E.g., firewalls, NATs (network address translators), “transparent proxies”
- Just as with in-line assembly code, can be messy and paint yourself into a corner (you know too much)

Layer Encapsulation

Common case: 20 bytes TCP header + 20 bytes IP header + 14 bytes Ethernet header = 54 bytes overhead
The Internet *Hourglass*

There is just one network-layer protocol, IP. The “narrow waist” facilitates interoperability.

**End-to-End Principle (Moderate Interpretation)**

- Think twice before implementing functionality in the network
- If hosts can implement functionality correctly, implement it in a lower layer only as a performance enhancement
- But do so only if it does not impose burden on applications that do not require that functionality
Using Ports to Identify Services

Client host

Service request for 128.2.194.242:80 (i.e., the Web server)

OS

Web server (port 80)

Echo server (port 7)

Service request for 128.2.194.242:7 (i.e., the echo server)

OS

Web server (port 80)

Echo server (port 7)

Recovering message boundaries

• Stream socket data separation:
  – Use records (data structures) to partition data stream
  – How do we implement variable length records?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed length record</td>
<td>fixed length record</td>
<td>variable length record</td>
<td></td>
</tr>
</tbody>
</table>

• What if field containing record size gets corrupted?
  • Not possible! Why?
Putting it All Together

Server

- socket()
- bind()
- listen()
- accept()
- read()
- write()
- block
- process request
- write()

Client

- socket()
- connect()
- send request
- write()
- establish connection
- send request
- read()

IP Service: “Best Effort” Delivery

- No error detection or correction
  - Higher-level protocol can provide error checking
- Successive packets may not follow the same path
  - Not a problem as long as packets reach the destination
- Packets can be delivered out-of-order
  - Receiver can put packets back in order (if necessary)
- Packets may be lost or arbitrarily delayed
  - Sender can send the packets again (if desired)
- No network congestion control (beyond “drop”)
  - Sender can slow down in response to loss or delay
**IP Packet Structure**

<table>
<thead>
<tr>
<th>4-bit Version</th>
<th>4-bit Header Length</th>
<th>8-bit Type of Service (TOS)</th>
<th>16-bit Total Length (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-bit Identification</td>
<td>3-bit Flags</td>
<td>13-bit Fragment Offset</td>
<td></td>
</tr>
<tr>
<td>8-bit Time to Live (TTL)</td>
<td>8-bit Protocol</td>
<td>16-bit Header Checksum</td>
<td></td>
</tr>
</tbody>
</table>

**32-bit Source IP Address**

**32-bit Destination IP Address**

Options (if any)

Payload

---

**Fragmentation**

- **Identifier (16 bits):** used to tell which fragments belong together
- **Flags (3 bits):**
  - **Reserved (RF):** unused bit (why “reserved”?)
  - **Don't Fragment (DF):** instruct routers to *not* fragment the packet even if it won't fit
    - Instead, they drop the packet and send back a “Too Large” ICMP control message
    - Forms the basis for “Path MTU Discovery”, covered later
  - **More (MF):** this fragment is not the last one
- **Offset (13 bits):** what part of datagram this fragment covers in 8-byte units
- **Thus, a fragment has either MF set or Offset > 0**
Security Implications of IP’s Design

<table>
<thead>
<tr>
<th>4-bit Version</th>
<th>4-bit Header Length</th>
<th>8-bit Type of Service (TOS)</th>
<th>16-bit Total Length (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>16-bit Identification</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3-bit Flags</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>13-bit Fragment Offset</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8-bit Time to Live (TTL)</th>
<th>8-bit Protocol</th>
<th>16-bit Header Checksum</th>
</tr>
</thead>
</table>

32-bit Source IP Address

32-bit Destination IP Address

Options (if any)

Payload

Classless Inter-Domain Routing (CIDR)

Use two 32-bit numbers to represent a network.
Network number = IP address + Mask

IP Address : 12.4.0.0       IP Mask: 255.254.0.0

Address: 00001100 00000100 00000000 00000000

Mask: 11111111 11111110 00000000 00000000

Network Prefix for hosts

Written as 12.4.0.0/15
But, Aggregation Not Always Possible

Multi-homed customer with 201.10.6.0/23 has two providers. Other parts of the Internet need to know how to reach these destinations through both providers.

Obtaining a Block of Addresses

- Separation of control
  - Prefix: assigned to an institution
  - Addresses: assigned by the institution to their nodes

- Who assigns prefixes?
  - Internet Corporation for Assigned Names and Numbers
    - Allocates large address blocks to Regional Internet Registries
    - ICANN is politically charged
  - Regional Internet Registries (RIRs)
    - E.g., ARIN (American Registry for Internet Numbers)
    - Allocates address blocks within their regions
    - Allocated to Internet Service Providers and large institutions
  - Internet Service Providers (ISPs)
    - Allocate address blocks to their customers (could be recursive)
Longest-Prefix-Match Forwarding

- Router needs to identify longest-matching prefix

```
forwarding table

destination  201.10.6.17

4.0.0.0/8
4.83.128.0/17
201.10.0.0/21
201.10.6.0/23
126.255.103.0/24

outgoing link  Serial0/0.1
```

- Algorithmic problem: how do we do this fast?

Why Would Anyone Use UDP?

- Finer control over what data is sent and when
  - As soon as an application process writes into the socket
  - … UDP will package the data and send the packet

- No delay for connection establishment
  - UDP just blasts away without any formal preliminaries
  - … which avoids introducing any unnecessary delays

- No connection state
  - No allocation of buffers, sequence #s, timers …
  - … making it easier to handle many active clients at once

- Small packet header overhead
  - UDP header is only 8 bytes
Transmission Control Protocol (TCP)

• Connection oriented
  – Explicit set-up and tear-down of TCP session

• Stream-of-bytes service
  – Sends and receives a stream of bytes, not messages

• Congestion control
  – Dynamic adaptation to network path’s capacity

• Reliable, in-order delivery
  – TCP tries very hard to ensure byte stream (eventually) arrives intact
    • In the presence of corruption and loss

• Flow control
  – Ensure that sender doesn’t overwhelm receiver

Automatic Repeat reQuest (ARQ)

• Automatic Repeat Request
  – Receiver sends acknowledgment (ACK) when it receives packet
  – Sender waits for ACK and times out if does not arrive within some time period

• Simplest ARQ protocol
  – Stop and Wait
  – Send a packet, stop and wait until ACK arrives
How Fast Can Stop-and-Wait Go?

- Suppose we’re sending from UCB to New York:
  - Bandwidth = 1 Mbps (megabits/sec)
  - RTT = 100 msec
  - Maximum Transmission Unit (MTU) = 1500 B = 12,000 b
  - No other load on the path and no packet loss

- What (approximately) is the fastest we can transmit using Stop-and-Wait?

- How about if Bandwidth = 1 Gbps?

---

Sliding Window

- Allow a larger amount of data “in flight”
  - Allow sender to get ahead of the receiver
  - … though not too far ahead
Performance with Sliding Window

• Given previous UCB ↔ New York 1 Mbps path
  + Sender window = 100 Kb = 12.5 KB

• How fast can we transmit?

• What about with 12.5 KB window & 1 Gbps path?

• Window required to fully utilize path:
  • Bandwidth-delay product (or “delay-bandwidth product”)
  • 1 Gbps * 100 msec = 100 Mb = 12.5 MB
  • Can picture as path’s storage capacity
  • Note: large window = many packets in flight

Distributed Hierarchical Database

[Diagram of distributed hierarchical database]
DNS Root Servers

• 13 root servers (see http://www.root-servers.org/)
  – Labeled A through M
• Replication via any-casting (localized routing for addresses)

Recursive vs. Iterative Queries

• Recursive query
  – Ask server to get answer for you
  – E.g., request 1 and response 8
• Iterative query
  – Ask server who to ask next
  – E.g., all other request-response pairs
Reverse Mapping (Address → Host)

• How do we go the other direction, from an IP address to the corresponding hostname?
• Addresses already have natural “quad” hierarchy:
  – 12.34.56.78
• But: quad notation has most-sig. hierarchy element on left, while www.cnn.com has it on the right
• Idea: reverse the quads = 78.56.34.12 …
  – … and look that up in the DNS
• Under what TLD?
  – Convention: in-addr.arpa
  – So lookup is for 78.56.34.12.in-addr.arpa

DNS Caching

• Performing all these queries takes time
  – And all this before actual communication takes place
  – E.g., 1-second latency before starting Web download
• Caching can greatly reduce overhead
  – The top-level servers very rarely change
  – Popular sites (e.g., www.cnn.com) visited often
  – Local DNS server often has the information cached
• How DNS caching works
  – DNS servers cache responses to queries
  – Responses include a “time to live” (TTL) field
  – Server deletes cached entry after TTL expires
DNS Resource Records

DNS: distributed DB storing resource records (RR)

RR format: (name, value, type, ttl)

- **Type=A**
  - name is hostname
  - value is IP address

- **Type=NS**
  - name is domain (e.g. foo.com)
  - value is hostname of authoritative name server for this domain

- **Type=PTR**
  - name is reversed IP quads
    - E.g. 78.56.34.12.in-addr.arpa
  - value is corresponding hostname

- **Type=CNAME**
  - name is alias name for some "canonical" name
    - E.g., www.cs.mit.edu is really eecsweb.mit.edu
  - value is canonical name

- **Type=MX**
  - value is name of mailserver associated with name
    - Also includes a weight/priority

```
unix> dig +nocrude @a.root-servers.net www.cnn.com

; <<>> DiG 9.2.2 <<>> +nocrude @a.root-servers.net www.cnn.com
 ;; global options: printcmd
 ;; Got answer:
 ;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 21041
 ;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 14

 ;; QUESTION SECTION:
 www.cnn.com.    IN      A

 ;; AUTHORITY SECTION:
 com. 172800 IN  NS  A.GTLD-SERVERS.NET.
 com. 172800 IN  NS  G.GTLD-SERVERS.NET.
 com. 172800 IN  NS  H.GTLD-SERVERS.NET.
 com. 172800 IN  NS  C.GTLD-SERVERS.NET.
 com. 172800 IN  NS  I.GTLD-SERVERS.NET.
 com. 172800 IN  NS  B.GTLD-SERVERS.NET.
 com. 172800 IN  NS  D.GTLD-SERVERS.NET.
 com. 172800 IN  NS  L.GTLD-SERVERS.NET.
 com. 172800 IN  NS  F.GTLD-SERVERS.NET.
 com. 172800 IN  NS  J.GTLD-SERVERS.NET.
 com. 172800 IN  NS  K.GTLD-SERVERS.NET.
 com. 172800 IN  NS  E.GTLD-SERVERS.NET.
 com. 172800 IN  NS  M.GTLD-SERVERS.NET.
```
Cache Poisoning

• Suppose you are a Bad Guy and you control the name server for foobar.com. You receive a request to resolve www.foobar.com and reply:

```plaintext
;; QUESTION SECTION:
;www.foobar.com. IN A

;; ANSWER SECTION:
www.foobar.com. 300 IN A 212.212.212.44

;; AUTHORITY SECTION:
foobar.com. 600 IN NS google.com.

;; ADDITIONAL SECTION:
google.com. 600 IN A 212.212.212.55
```

A foobar.com machine, not google.com

---

Example: E-Mail Message Using MIME

```plaintext
From: jrex@cs.princeton.edu
To: feamster@cc.gatech.edu
Subject: picture of my cat
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

Base64 encoded data ....
JVBERi0xLjMNJeLjz9MNMSAwI

..........................
......base64 encoded data
```
SMTP *Store-and-Forward* Protocol

- Messages sent through a series of servers
  - A server stores incoming messages in a queue
  - … to await attempts to transmit them to the next hop
- If the next hop is not reachable
  - The server stores the message and tries again later
- Each hop adds its identity to the message
  - By adding a “Received” header with its identity
  - Helpful for diagnosing problems with e-mail

---

**Example With Received Header**

```
Return-Path: <casado@cs.stanford.edu>
Received: from ribavirin.CS.Princeton.EDU (ribavirin.CS.Princeton.EDU [128.112.136.44])
  by newark.CS.Princeton.EDU (8.12.11/8.12.11) with SMTP id k04M5RYY023164
  for <jrex@newark.CS.Princeton.EDU>; Wed, 4 Jan 2006 17:05:37 -0500 (EST)
Received: from bluebox.CS.Princeton.EDU (SMSSMTP 4.1.0.19) with SMTP id M2006010417053607946
  for <jrex@newark.CS.Princeton.EDU>; Wed, 04 Jan 2006 17:05:36 -0500
Received: from smtp-roam.Stanford.EDU (smtp-roam.Stanford.EDU [171.64.10.152])
  by bluebox.CS.Princeton.EDU (8.12.11/8.12.11) with ESMTP id k04M5XNQ005204
  for <jrex@cs.princeton.edu>; Wed, 4 Jan 2006 17:05:35 -0500 (EST)
Received: from [192.168.1.101] (authenticated bits=0)
  (version=TLSv1/SSLv3 cipher=DHE-RSA-AES256-SHA bits=256 verify=NOT);
  Wed, 4 Jan 2006 14:05:32 -0800
Message-ID: <43BC46AF.3030306@cs.stanford.edu>
Date: Wed, 04 Jan 2006 14:05:35 -0800
From: Martin Casado <casado@cs.stanford.edu>
User-Agent: Mozilla Thunderbird 1.0 (Windows/20041206)
MIME-Version: 1.0
To: jrex@CS.Princeton.EDU
CC: Martin Casado <casado@cs.stanford.edu>
Subject: Using VNS in Class
Content-Type: text/plain; charset=ISO-8859-1; format=flowed
Content-Transfer-Encoding: 7bit
```
Sample SMTP interaction

S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: From: alice@crepes.fr
C: To: hamburger-list@burger-king.com
C: Subject: Do you like ketchup?
C: How about pickles?
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection

URL Syntax

protocol: //hostname[:port]/directorypath/resource

- Protocol might be http, ftp, https, smtp, rtsp, ...
- In practice, hostname can instead be an IP address
  - What does your browser (maybe) show for http://2850372702/?
- Port defaults to the standard port associated w/ protocol
  - E.g., 80/tcp for http, 443/tcp for https
- Directory path is hierarchical, often reflecting file system
- Can extend resource to program executions as well...
  - http://us.f413.mail.yahoo.com/ym/ShowLetter?box=%40B%40Bulk&MsgId=2604_1744106_29699_1123_1261_0_28917_3552_1289957100&Search=&Nhead=f&YY=31454&order=down&sort=date&pos=0&view=a&head=b
HTTP Request Message

- Request message sent by a client
  - Request line: method, resource, and protocol version
  - Request headers: provide information or modify request
  - Body: optional data (e.g., to "POST" data to the server)

GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr

HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 2006 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 2006 ...
Content-Length: 6821
Content-Type: text/html

HTTP Response Message

- Response message sent by a server
  - Status line: protocol version, status code, status phrase
  - Response headers: provide information
  - Body: optional data

HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 2006 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 2006 ...
Content-Length: 6821
Content-Type: text/html

Data data data data data data ...
Stateless Operation

• Stateless protocol
  – Each request-response exchange treated independently
  – Clients and servers not required to retain state

• Statelessness improves scalability
  – Avoid need for server to retain info across requests
  – Enable server to handle a higher rate of requests

• However, some applications need persistent state
  – To uniquely identify the user or store temporary info
  – E.g., personalize a Web page, compute profiles or access statistics by user, track a shopping cart …
  – Done using “cookies”

Cookies

• Cookie
  – Small(?) state stored by client on behalf of server
  – Included in future requests to the server

Request

Response

Set-Cookie: XYZ

Request

Cookie: XYZ
Pipelined Requests/Responses

- Batch requests and responses to reduce the number of packets
- Multiple requests can be contained in one TCP segment
- Small items (common) can also share segments
- Note: maintains order of responses
  - Item 1 always arrives before item 2
- HTTP 1.1 feature (not in 1.0)

Concurrent Requests/Responses

- Use multiple connections to issue requests and responses **in parallel**
- Does **not** necessarily maintain order of responses
- Raises question of **fairness**
  - Set of \( N \) parallel connections “grabs” bandwidth \( N \) times more aggressively than just one
  - What’s a reasonable/fair limit as traffic competes with that of other users?
### Persistent Connections

- Handle multiple transfers per connection
  - Including transfers subsequent to imaging current page
  - Maintain TCP connection across multiple requests
  - Either client or server can tear down the connection

- Performance advantages
  - Avoid overhead of connection set-up and tear-down
  - Allow TCP to learn more accurate RTT estimate
  - Allow the TCP congestion window to increase
    - I.e, leverage previously discovered bandwidth

### Forward & Reverse Proxies

- Cache documents close to clients → reduce network traffic and decrease latency
- Typically done by ISPs or corporate LANs
Caching vs. Replication (CDNs)

- Motivations for moving content close to users
  - Reduce latency for the user
  - Reduce load on the network and the server

- Caching
  - Replicating the content “on demand” after a request
  - Storing the response message locally for future use
  - May need to verify if the response has changed
  - … and some responses are not cacheable

- Replication
  - Planned replication of the content in multiple locations
  - Updating of resources handled outside of HTTP
  - Can replicate scripts that create dynamic responses

Message, Segment, Packet, and Frame
Link-Layer Services

- **Encoding**
  - Representing the 0s and 1s

- **Framing**
  - Encapsulating packet into frame, adding header, trailer
  - Using MAC addresses rather than IP addresses

- **Error detection**
  - Errors caused by signal attenuation, noise
  - Receiver detects presence, may ask for repeat (ARQ)

- **Resolving contention**
  - Deciding who gets to transmit when multiple senders want to use a shared media

- **Flow control** (pacing between sender & receiver)

Non-Return to Zero (NRZ)

- 1 → high signal; 0 → low signal
- (Actual signals are of course not so sharp)

![NRZ Signal Diagram](image)

- Receiver reads the signal on the clock’s **leading edge**
Manchester Encoding

- 1 → high-to-low transition; 0 → low-to-high transition
- Addresses clock recovery and “baseline wander” problems
- Disadvantage: clock must be twice as fast
  - Efficiency of 50%

4-bit/5-bit (100Mb/s Ethernet)

- Goal: address inefficiency of Manchester encoding, while avoiding long periods of low signals
- Solution:
  - Use 5 bits to encode every sequence of four bits such that
    - No 5 bit code has more than one leading 0 or two trailing 0’s
  - Use NRZI to then encode the 5 bit codes
  - Efficiency is 80%
Simple Approach to Framing: Counting

- Sender: begin frame with byte(s) giving length
- Receiver: extract this length and count

<table>
<thead>
<tr>
<th>53</th>
<th>Frame contents</th>
<th>21</th>
<th>Frame contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>53 bytes of data</td>
<td>21 bytes of data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- How can this go wrong?
- On occasion, the count gets corrupted

<table>
<thead>
<tr>
<th>58</th>
<th>Frame contents</th>
<th>21</th>
<th>Frame contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>58 bytes of data misdelivered</td>
<td>Bogus new frame length; desynchronization</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Framing: Sentinels

- Delineate frame with special pattern
  - e.g., 01111110

| 01111100 | Frame contents | 01111100 |

- Problem: what if sentinel occurs within frame?
- Solution: escape the special characters
  - E.g., sender always inserts a 0 after five 1s
  - … receiver always removes a 0 appearing after five 1s
- Similar to escaping special characters in C programs
Error Detection

• Errors are unavoidable
  – Electrical interference, thermal noise, etc.

• Error detection
  – Transmit extra (redundant) information
  – Use redundant information to detect errors
  – Extreme case: send two copies of the data
  – Trade-off: accuracy vs. overhead

• Techniques for detecting errors
  – Parity checking
  – Checksum
  – Cyclic Redundancy Check (CRC)

Channel Partitioning: TDMA & FDMA

TDMA: time division multiple access

• Access to channel in "rounds"
  – Each station gets fixed length slot in each round

• Time-slot length is packet transmission time
  – Unused slots go idle

• Example: 6-station LAN with slots 0, 3, and 4

FDMA: frequency division multiple access

  – Each station assigned fixed frequency band
“Taking Turns” MAC protocols

**Polling**
- Master node “invites” slave nodes to transmit in turn
- Concerns:
  - Polling overhead
  - Latency
  - Single point of failure (master)

**Token passing**
- Control token passed from one node to next sequentially
- Node must have token to send
- Concerns:
  - Token overhead
  - Latency
  - Single point of failure (token)

---

Key Ideas of Random Access

- **Carrier sense**
  - *Listen before speaking, and don’t interrupt*
  - Checking if someone else is already sending data
  - … and waiting till the other node is done

- **Collision detection**
  - *If someone else starts talking at the same time, stop*
  - Realizing when two nodes are transmitting at once
  - … by detecting that the data on the wire is garbled

- **Randomness**
  - *Don’t start talking again right away*
  - Waiting for a random time before trying again
Slotted ALOHA

**Assumptions**
- All frames same size
- Time divided into equal slots (time to transmit a frame)
- Nodes are synchronized
- Nodes begin to transmit frames only at start of slots — **Not** clock-based framing
- If two or more nodes transmit, all nodes detect collision

**Operation**
- When node obtains fresh frame, transmits in next slot
- No collision: node can send new frame in next slot
- Collision: node retransmits frame in each subsequent slot with probability $p$ until success

CSMA/CD Collision Detection

[Diagram showing time and space with nodes A, B, C, D and time intervals $t_0$, $t_1$, collision detection/abort time]
Limitations on Ethernet Length

- A needs to wait for time $2d$ to detect collision
  - So, A should keep transmitting during this period
  - … and keep an eye out for a possible collision

- Imposes restrictions on Ethernet. For 10 Mbps:
  - Maximum length of the wire: 2,500 meters
  - Minimum length of the packet: 512 bits (64 bytes)
    - $512 \text{ bits} = 51.2 \mu\text{sec (at 10 Mbit/sec)}$
    - For light in vacuum, $51.2 \mu\text{sec} \approx 15,000$ meters
      vs. 5,000 meters “round trip” to wait for collision

Ethernet Frame Structure

- Sending adapter encapsulates packet in frame

- **Preamble:** synchronization
  - Seven bytes with pattern 10101010, followed by one byte with pattern 10101011
  - Used to synchronize receiver & sender clock rates

- **Type:** indicates the higher layer protocol
  - Usually IP (but also Novell IPX, AppleTalk, …)

- **CRC:** cyclic redundancy check
  - Receiver checks & simply drops frames with errors
Ethernet Frame Structure (Continued)

• **Addresses:** 48-bit source and destination MAC addresses
  - Receiver’s adaptor passes frame to network-level protocol
    • If destination address matches the adaptor’s
    • Or the destination address is the broadcast address (ff:ff:ff:ff:ff:ff)
    • Or the destination address is a multicast group receiver belongs to
    • Or the adaptor is in promiscuous mode
  - Addresses are **globally unique**
    • Assigned by NIC vendors (top three octets specify vendor)
    • During any given week, > 500 vendor codes seen at LBNL

• **Data:**
  - Maximum: 1,500 bytes
  - Minimum: 46 bytes (+14 bytes header + 4 byte trailer = 512 bits)

Physical Layer: Repeaters & Hubs

• **Distance limitation in local-area networks**
  - Electrical signal becomes weaker as it travels
  - Imposes a limit on the length of a LAN
    • In addition to limit imposed by collision detection

• **Repeaters & Hubs join LANs together**
  - Analog electronic device
  - Continuously monitors electrical signals on each LAN
  - Repeater transmits an amplified copy
**Link Layer: Bridges & Switches**

- Connects two or more LANs at the link layer
  - Extracts destination address from the frame
  - Looks up the destination in a table
  - Forwards the frame to the appropriate LAN segment

- Each segment is its own collision domain

---

**Advantages Over Hubs/Repeaters**

- Only forwards frames as needed
  - Filters frames to avoid unnecessary load on segments
  - Sends frames only to segments that need to see them

- Extends the geographic span of the network
  - Separate collision domains allow longer distances

- Improves privacy by limiting scope of frames
  - Hosts can “snoop” the traffic traversing their segment
  - … but not all the rest of the traffic

- If needed, applies carrier sense & collision detection
  - Does not transmit when the link is busy
  - Applies exponential back-off after a collision

- Joins segments using different technologies
Disadvantages Over Hubs/Repeaters

• Delay in forwarding frames
  – Bridge/switch must receive and parse the frame
  – … and perform a look-up to decide where to forward
  – Introduces store-and-forward delay
  – Solution: cut-through switching

• Need to learn where to forward frames
  – Bridge/switch needs to construct a forwarding table
  – Ideally, without intervention from network administrators
  – Solution: self-learning

• Higher cost
  – More complicated devices that cost more money

Cut-Through Switching

• Buffering a frame takes time
  – If \( L \) is length of the frame, \( R \) is the transmission rate …
  – … then receiving the frame takes \( \frac{L}{R} \) time units
  – When will this be significant?

• Cut-Through: Begin sending as soon as possible
  – Inspect frame header & look-up destination
  – If outgoing link idle, start forwarding
  – Can transmit head of packet while still receiving tail
Self Learning: Building the Table

- When a frame arrives
  - Inspect source MAC address
  - Associate address with the incoming interface
  - Store mapping in the switch table
  - Use time-to-live field to eventually forget the mapping
    - Soft state

Avoiding Loops: Spanning Trees

- Ensure the forwarding topology has no loops
  - Avoid using some of the links when flooding
  - … to prevent loop from forming

- Spanning tree
  - Sub-graph that covers all vertices but contains no cycles
  - Links not in the spanning tree do not forward frames
<table>
<thead>
<tr>
<th></th>
<th>hubs</th>
<th>switches</th>
<th>routers</th>
</tr>
</thead>
<tbody>
<tr>
<td>traffic isolation</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>plug &amp; play</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>optimized routing</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>cut-through</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>