11. Data Link Layer

Paul Krzyzanowski
Rutgers University
Spring 2016
Data Link Layer

• Transport Layer (4)
  – Logical connection between processes
  – Transport layer multiplexing & demultiplexing

• Network Layer (3)
  – End-to-end communication between hosts
  – Possibly through multiple networks via routers

• Data Link Layer (2)
  – Deals with individual communication links
Link Layer

- Data is encapsulated in a link-level frame
- **MAC** = Medium Access Control
  - Protocol for transmitting and receiving frames at the link layer
- **Error detection & correction**
  - Detect (and possibly correct) errors in the frame
- **MAC Address**
  - Link-layer address
Error Detection & Correction
Error Detection & Correction Goals

Why do we want this at the link layer?

– **Drop a bad frame at the receiver**
  - If the link layer detects it, no overhead checking at the network/transport layers
  - No need to forward the packet (avoid wasting network bandwidth)
  - Avoid end-to-end delay of having the receiver detect & sender retransmission

– **Attempt to correct errors**
  - Avoid the need to reject bad packets & retransmit
Parity

• Simplest form of error detection: add one bit (parity bit)
  – Even parity
    • Set the parity bit such that there is an even number of 1 bits
      01110000 ⇒ 01110001
  – Odd parity
    • Set the parity bit such that there is an odd number of 1 bits
      01110000 ⇒ 011100000

• An even number of bit errors will be undetected
• In real life, bit errors typically occur in bursts
  – Multiple consecutive bits get corrupted
Two-Dimensional Parity

- Break up $d$ bits into $i$ rows and $j$ columns
- Generate a **parity bit per row and per column**
  - For a single bit error, we can identify the row & column of the bit

Example: $1011\ 0001\ 1100\ 1110$ with even parity:

\[
\begin{array}{cccc|c}
1 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1
\end{array}
\]

We can transmit: $1011\ 0001\ 1100\ 1110\ 1101\ 1000\ 1$
Two-Dimensional Parity

For a single bit error, we can identify the row & column of the corrupted bit

We sent: 1011 0001 1100 1110 1101 1000 1

They got: 1011 0011 1100 1110 1101 1000 1

Place this back into the grid:

Here’s the bad bit

Bad parity

Here’s the bad bit

Bad parity

By identifying the row & column, we can identify the bad bit
Error Correction

- Two-dimensional parity
  - Simple example of an error correcting code (ECC)

- Error correcting codes
  - Invented by Richard Hamming in 1950
  - Common types of ECCs
    - Reed-Solomon codes (used in CDs, DVDs, disk drives)
    - Hamming codes (ECC memory)
    - Low-density parity-check, LDPC (802.11n, 10G Ethernet)
    - Viterbi codes (cellular LTE)

- Forward Error Correction (FEC)
  - Data transmission that uses ECC in the message
  - The receiver can correct some errors without the need for retransmission
Checksums

- **Checksum** = treat the bits of a packet as a set of integers
  - Perform operations on those integers

- **Internet checksum**
  - We saw this in IP, UDP, TCP, ICMP, OSPF, and IGMP headers
    - Treat data as 16-bit chunks
    - Sum it up (add one for each carry)
    - Take a 1s complement of the result
  - Simple, easy to compute efficiently (important!)
  - BUT very weak protection against errors

- **Cyclic Redundancy Check (CRC)**
  - Much more robust checksum
  - More compute intensive (hence not appealing at higher layers)
  - Done with dedicated hardware at the transceiver
Cyclic Redundancy Check

• Polynomial code

• Works well for detecting burst errors: a sequence of bad bits

• $n$-bit CRC code will usually detect an error burst up to $n$ bits
  – Will detect longer bursts with a probability of $1-2^{-n}$
  – Example: Ethernet uses a 32-bit CRC
    • Detects up to 32 consecutive bad bits
    • Detects longer streams of bad bits 99.99999997671% of the time
    • That is, there’s a $2.329 \times 10^{-10}$ chance that the CRC will not detect bit errors >32 bits
How is a CRC calculated?

- CRC performed by division: all subtractions replaced with XOR
  \[ a \oplus b = a + b = a - b \] if we ignore carries and borrows

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a \oplus b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- To send a message \( D \) with \( d \) data bits
  - Compute CRC code \( R \) with \( r \) bits
  - Transmit \( D, R \)

- Receiver and transmitter agree upon a Generator, \( G \)
  - \( G \) has \( r+1 \) bits; starts with 1
  - CRC = \( R \) = remainder of \( \frac{D \times 2^r}{G} \)

\( D \times 2^r \) is \( D \) left-shifted by \( r \) bits
CRC calculation example

• We want to send $D = 01110000011$
• Assume the generator bits are 10111 ($r = 4$; $G$ has $r + 1$ bits)
• Perform a division (but with xor instead of subtraction with borrowing)

\[
\begin{array}{c|cccccc}
\multicolumn{2}{c}{1} & 01110000011 & 0000 \\
10111 & \\
\hline
10111 & 01110000011 & 0000 \\
10111 & \\
\hline
010110 & \\
\end{array}
\]

shift $D$ by $r$ (4) bits

April 22, 2016
CRC calculation example

- We want to send $D = 01110000011$
- Assume the generator bits are $10111$ ($r = 4; G$ has $r + 1$ bits)
- Perform a division (but with xor instead of subtraction with borrowing)

\[
\begin{array}{c|c}
11 & 011100000110000 \\
\hline
10111 & 011100000110000 \\
\downarrow & \downarrow \\
10111 & 010110 \\
\downarrow & \downarrow \\
010110 & 10111 \\
\downarrow & \downarrow \\
00010011 & 00010011 \\
\end{array}
\]

shift $D$ by $r$ (4) bits
CRC calculation example

- We want to send $D = 01110000011$
- Assume the generator bits are 10111 ($r = 4$; $G$ has $r + 1$ bits)
- Perform a division (but with xor instead of subtraction with borrowing)

\[
\begin{array}{c|cccccccc}
\multicolumn{1}{c|}{1100010101} & 011100000110000 \\
10111 & 011100000110000 \\
\multicolumn{1}{c|}{010110} & 00100000100110111 \\
\multicolumn{1}{c|}{} & 00100000100110111 \\
\end{array}
\]

CRC = 1011
Transmit $(D, R) = 011100000111011$
CRC verification example

- We received \( D = 01110000111011 \)
- Same Generator, \( G = 10111 \) (\( r = 4 \); \( G \) has \( r+1 \) bits)
- Perform the same division (no shift; we have 4 CRC bits at the right)

If the remainder = 0 then no error detected

R = 0 Correct!

No need to shift
We have our CRC bits
CRC Generators

- Ethernet uses a 32-bit CRC generator (CRC-32)
  - 0x04C11DB7
  - Also used by FDDI, ZIP, and PNG
Multiple Access Protocols
Categories of link layer access protocols

• Types of links
  – Point-to-point links connect one sender with one receiver
    • No conflict for access
  – Broadcast links have multiple nodes connected to the same channel

• Broadcast links have a multiple access problem
  – How do you coordinate multiple senders?
  – Collision: when two nodes transmit at the same time
    • Signals from both get damaged

• Three categories of multiple access protocols
  1. Channel partitioning
  2. Random access
  3. Taking turns
Channel Partitioning Protocols

1. **Time division multiplexing** (TDM)
   - Divide a channel into time slots
   - A node can transmit only during its allocated time slot

2. **Frequency division multiplexing** (FDM)
   - Divide a channel into frequency bands

• If a channel has a bandwidth \( R \) and there are \( N \) nodes
  - Both TDM and FDM are fair: each node gets bandwidth \( = \frac{R}{N} \)
  - BUT a node gets \( \frac{R}{N} \) even if no other node needs to transmit!
TDM vs. FDM

FDM: Frequency Division Multiplexing

TDM: Time Division Multiplexing
Random Access Protocols

- Node has full use of the channel
- No scheduled time slots as in TDM
- If there is a collision
  - Colliding nodes wait a random time & retransmit
  - The nodes will (usually) pick different intervals & not collide next time
Slotted ALOHA

• One of the oldest random access protocols
• Not used anymore but useful to study
• Environment
  – All frames $L$ bits
  – Time divided into 1-frame slots of $L/R$ seconds ($R$=bandwidth)
  – Nodes are synchronized and transmit at the start of a slot
• If there’s a collision
  – All transmitting nodes detect it during transmission
  – Retransmit on the next slot with probability $p$
  – Otherwise skip the slot and try again: retransmit with probability $p$
Slotted ALOHA

• Efficiency
  – Time slots with collisions: wasted
  – Time slots with no transmissions: also wasted

• \( P(\text{success for 1 node}) = P(\text{one node transmits}) \times P(\text{N-1 do not}) \)
  \[ = p \times (1 - p)^{N-1} \]

• \( P(\text{success for all nodes}) = Np(1 - p)^{N-1} \)

• Maximum efficiency
  – Find \( p \) that maximizes the expression
  – Take limit of \( N \rightarrow \infty \)
  – This is \( 1/e \approx 0.37 \)
    • 37% slots have useful data; 37% are empty; 26% have collisions
  – A 1 Gbps link will behave like a 370 Mbps link!
Carrier Sense Multiple Access with Collision Detection

• **Carrier Sensing**
  – Listen first
  – If the channel has communications, wait until it is clear

• **Collision Detection**
  – If you are transmitting but detect a collision, stop transmitting
  – Wait a random time interval and try again (sense & transmit)
How do collisions occur?

• Node A senses quiet & transmits
  • Remember propagation delay?
    – It takes time for the signal to reach other nodes
    – $\sim 2 \times 10^8 \text{ m/s} = 5 \text{ nanoseconds per meter}$
How do collisions occur?

• Node A senses quiet & transmits
• A short while later…
  – Node B senses quiet because the signal from A didn’t reach it
  – Node B transmits
How do collisions occur?

• Node A senses quiet & transmits

• A short while later…
  – Node B senses quiet because the signal from A didn’t reach it
  – Node B transmits

Node A

Node B

Node C

Node A transmits

A detects a collision

B detects a collision

time
Collision Detection

• A node listens while it is transmitting

• As soon as it detects a collision
  – Stop transmitting
  – Wait a random interval

  – We’d like a possibly long interval if there are many nodes sending
  – We’d like a short interval if there are few transmitters
  – BUT … we don’t know what’s going on!
If a frame experienced $b$ collisions ($b =$ backoff count)

Choose a delay $W$ with equal probability from $0 \ldots 2^{b-1}$

- 1st time \{0, 1\}  
- 2nd time \{0 \ldots 3\} 
- 3rd time \{0 \ldots 7\}  
- 4th time \{0 \ldots 15\} 
- 5th time \{0 \ldots 31\}  
- 6th time \{0 \ldots 63\} 

Ethernet: a delay is $W \times 512$ bit-times

- 512 bit-times = time to send 512 bits = 5.12 $\mu$s for 100 Mbps
- Backoff count limit (maximum $b$) = 10
- 10 or more collisions: choose a delay \{0 \ldots 1023\}

Status

- CSMA/CD is not needed with switched Ethernet
- Binary Exponential Backoff also used in DOCSIS cable modems
Multiple Access via Taking Turns

• Goal: ensure that each node can get a fair throughput
  – Close to $R/N$ bps for bandwidth $R$ and $N$ nodes

• Polling protocol (used by Bluetooth)
  – Master polls each of the nodes to see if they want to transmit
  – No collisions or empty slots
  – But: polling delay & chance of master dying

• Token passing protocol
  – Special frame, a token, is passed around nodes in some sequence
  – If a node has it, it can transmit & then forward the token
  – Decentralized & efficient
  – But: failure of a node can stop the network
Ethernet
Ethernet technology

- **Mid-1970s**: created at Xerox by Bob Metcalfe
  - 2.74 Mbps Ethernet over 9.5mm thick coax
- **1980s**
  - Standardized in 1985 as IEEE 802.3
  - & 10BASE-5 (9.5mm coax) & 10BASE-2 (5mm coax)
- **1990**:
  - 10BASE-T over twisted pair wiring @ 10 Mbps
  - Category 3 UTP (unshielded twisted pair) wiring with RJ45 connectors
- **1995**: Fast Ethernet: 100BASE-TX over cat 5 UTP
- **1999**: Gigabit Ethernet: 1000BASE-T over cat 5e
- **2006**: 10 Gb Ethernet: 10GBASE-T over cat 6a
- **2010**: 100GbE / 40GbE 40GBASE-T over cat 8
Ethernet Frame

- 8 bytes: preamble & start-of-frame delimiter
- Variable size data: 42-1500 bytes
  - No length field: the transceiver grabs the entire frame
- Interframe gap: at least 96 bit wait time

- Jumbo frames: maximum size 8000 bytes
- Super Jumbo frames (SJF): maximum size > 8000 bytes

MAC = Media Access Control = link-layer address

See https://en.wikipedia.org/wiki/EtherType for protocol IDs
Link Layer Addressing
Link-Layer Addressing

• Each NIC has a unique link-layer address
  – MAC address – unrelated to IP address

• LAN communication at layer 2 needs MAC addresses
  – An Ethernet transceiver cannot send a frame to an IP address!

• E.g., Ethernet uses a **EUI-48** address
  – **EUI** = *Extended Unique Identifier*, managed by IEEE
  – Used in Ethernet, 802.11, Bluetooth, and a few other networks
  – 48-bit address (6 bytes long)
  – E.g., c8:2a:14:3f:92:d1 (my iMac)
  – **Globally unique address**
    • First three bytes: identify manufacturer
    • Next three bytes: assigned by manufacturer
    • Flat address space
Find MAC address given an IP address

• We need to send a datagram to an IP address
• It is encapsulated in an Ethernet frame and a MAC address

• How do we know what MAC address to use?
Address Resolution Protocol (ARP)

- **ARP table**
  - Kernel table mapping IP addresses & corresponding MAC addresses
  - OS uses this to fill in the MAC header given an IP destination address
  - What if the IP address we want is not in the cache?

- **ARP Messages**
  - A host creates an ARP query packet & broadcasts it on the LAN
    - Ethernet broadcast MAC address: `ff:ff:ff:ff:ff:ff`
  - All adapters receive it
    - If an adapter’s IP address matches the address in the query, it responds
    - Response is sent to the MAC address of the sender

- ARP packet structure

<table>
<thead>
<tr>
<th>HW Protocol (ethernet)</th>
<th>Protocol type (e.g., IPv4)</th>
<th>MAC addr length</th>
<th>query/ response</th>
<th>sender MAC addr</th>
<th>sender IP addr</th>
<th>target MAC addr</th>
<th>target IP addr</th>
</tr>
</thead>
</table>

  see the `arp` command on Linux/BSD/Windows/OS X
My ARP cache

- **Timeout on Linux systems:** `/proc/sys/net/ipv4/neigh/eth0/gc_stale_time`
  - Default = 60 seconds

- **Windows (Vista & Later)**
  - Timeout = random value between 15 and 45 seconds
  - But remains cached longer if used during that time

```
# arp -a
crapper.pk.org (192.168.60.129) at f0:f7:55:bb:17:26 [ether] on eth0
air.pk.org (192.168.60.143) at 28:37:37:19:65:96 [ether] on eth0
? (192.168.60.182) at d0:23:db:77:ff:5a [ether] on eth0
? (192.168.60.174) at a4:67:06:65:21:f8 [ether] on eth0
? (192.168.60.169) at 68:96:7b:09:bc:2a [ether] on eth0
nas.pk.org (192.168.60.136) at 00:0d:a2:01:84:79 [ether] on eth0
? (192.168.60.179) at f8:1e:df:d7:4a:1b [ether] on eth0
? (192.168.60.176) at 18:b4:30:0a:c7:d7 [ether] on eth0
? (192.168.60.181) at e8:06:88:90:2d:1e [ether] on eth0
? (192.168.60.186) at e4:8b:7f:ac:5b:10 [ether] on eth0
pk-imac.pk.org(192.168.60.153) at c8:2a:14:3f:92:d1 [ether] on eth0
tc.pk.org (192.168.60.138) at 00:1e:52:f5:b5:e3 [ether] on eth0
net.pk.org (192.168.60.131) at d0:67:e5:01:ec:5b [ether] on eth0
box.pk.org (192.168.60.132) at 00:1f:16:f7:92:67 [ether] on eth0
tc.pk.org (192.168.60.137) at 00:1e:52:f5:b5:e3 [ether] on eth0
```
IPv6: Neighbor Discovery

- IPv6 does not support ARP
  - **Neighbor Discovery** accomplishes the same thing as ARP
    - Extends ICMP (ICMPv6) with new commands
    - Neighbor Advertisement (NA) and Neighbor Solicitation (NS) commands

- Host A wants to contact Host B
  - ICMPv6 Type 135 (**Neighbor Solicitation**) message
    - Host A’s source address
    - **Solicited-Node Multicast destination address**
      - IPv6 prefix of ff02::0:0:0:1::ff00
      - IPv6 address suffix of the last 24 bits of Host B’s IP address
    - Data: Host A’s MAC address
    - Link Layer address: multicast mapping of IPv6 multicast address

- Host B responds
  - ICMPv6 Type 136 (**Neighbor Advertisement**) message
  - Datagram addressed to Node A

Every IPv6 host must listen on its *solicited-node* multicast address.
Transmitting a datagram

Three possibilities

1. We need to send to a host on our subnet (LAN)
   – We can do this at the link layer
   – We just need to find the MAC address that corresponds to the destination’s IP address

2. We need to send to a host outside of our subnet
   – We need to get the datagram to a connected router
   – The datagram may pass through multiple routers

3. We need to send a multicast datagram
   – Convert it to link-layer multicast
What if we need to send outside our LAN?

We need to get the datagram to a router

- Each router has an IP address (and a MAC address) for each interface
- Find the MAC address for the IP address of the router interface

<table>
<thead>
<tr>
<th>IP Address</th>
<th>MAC Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.11.11.10</td>
<td>20:c9:d0:72:cb:be</td>
</tr>
<tr>
<td>11.11.11.11</td>
<td>d0:23:db:77:ff:5a</td>
</tr>
<tr>
<td>22.22.22.20</td>
<td>a4:67:06:65:21:f8</td>
</tr>
<tr>
<td>22.22.22.21</td>
<td>68:96:7b:09:bc:2a</td>
</tr>
<tr>
<td>33.33.33.30</td>
<td>00:0d:a2:01:84:79</td>
</tr>
<tr>
<td>33.33.33.33</td>
<td>f8:1e:df:d7:4a:1b</td>
</tr>
</tbody>
</table>
What if we send outside our LAN?

IP datagram, source=11.11.11.11 destination=33.33.33.33

1. H1 looks up the route to H2: needs to send to router R1
   - Looks up MAC address for 11.11.11.10; sends frame to 68:96:7b:09:bc:2a

2. Router R1 needs to route to R2
   - Forwards to interface with IP addr 22.22.22.20
   - Looks up MAC address for 22.22.22.21; sends IP datagram to 68:96:7b:09:bc:2a

3. Router R2 forwards to interface with IP addr 33.33.33.30
   - Looks up MAC address for destination 33.33.33.33

Routers also use ARP
Link-Layer (Ethernet) multicasting

- Ethernet supports multicast in one (or both) of two ways:
  - Packets filtered based on hash(multicast_address)
    - Some unwanted packets may pass through
    - Simplified circuitry
  - Exact match on small number of addresses
    - If host needs more, put LAN card in multicast promiscuous mode
      - Receive all hardware multicast packets

- In either case:
  - Link-layer driver must check to see if the packet is really targeted to the system
Intel 82546EB
- Dual Port Gigabit Ethernet Controller
- 10/100/1000 BaseT Ethernet

Supports:
- 16 exact MAC address matches
- 4096-bit hash filter for multicast frames
- promiscuous unicast & promiscuous multicast transfer modes

Broadcom BCM57762
- 10/100/1000BASE-T Ethernet PCIe Controller
- Used in Apple’s Thunderbolt Ethernet adapter

Supports:
- 1 exact MAC address match (may be reprogrammed up to 4 times)
- Hash filter for multicast frames
  - 128-bit 7-bit CRC hash
  - or 256-bit 8-bit CRC hash
- promiscuous mode (accept all frames)
IP multicast on a LAN

- IP driver must translate 28-bit **IP multicast group** to a multicast Ethernet address
  - IANA allocated range of Ethernet MAC addresses for multicast
  - Copy least significant 23 bits of IP address to MAC address
    - 01:00:5e:xx:xx:xx
- Send out multicast Ethernet packet
  - Payload contains multicast IP packet
- Notice something?
  - The IP layer needs to filter out addresses that it is not subscribed to
IPv6 multicast on a LAN

- IPv6 multicast addresses have a 112-bit group ID and start with ff00
- IP driver must translate 128-bit IP multicast address to a multicast Ethernet address
  - Copy least significant 32 bits of IPv6 address to MAC address

IP addr: <ignore top 96 bits> dddddddd dddddddd dddddddd dddddddd
MAC addr: 00110011 00110011 dddddddd dddddddd dddddddd dddddddd

Switched LANs
Ethernet Evolution

• Ethernet started as a broadcast LAN with a shared **bus topology**
  - All packets were visible by all adapters
  - This is why we needed CSMA/CD

• Coax gave way to twisted pair
  - Category 5 (Cat 5) cable
  - Star topology
  - Dedicated cable for each adapter
  - Cables plugged into a **hub**

• **Ethernet hub**
  - Simulates a bus-based LAN
  - Every bit received on an interface is transmitted onto every other interface
Switched Ethernet

- Hubs gave way to switches in the mid-1990s

- Same star topology ... but smarter
  - Like a hub, transparent to hosts
  - **Full duplex**: separate receive vs. transmit wires
  - Forwards received frames to the right interface(s)

- Works *sort of* like a router
  - Link layer forwarding
  - But
    - Invisible – **frames are never addressed to the switch**
    - Self-learning: it learns what address is at which interface
Cisco Nexus 9516 Switch
• 1/10/40 GbE
• 21-rack-unit chassis
• Up to 576 1/10 Gb ports

TP-Link Switch
• 8 1-GbE ports
Inside an Ethernet Switch

Switch table (also known as MAC address table)

- Contains entries for known MAC addresses & their interface

- **Forwarding & filtering**: a frame arrives for some destination address $D$
  - Look up $D$ in the switch table to find the interface
  - If found & the interface is the same as the one the frame arrived on
    - Discard the frame *(filter)*
  - If found & a different interface
    - **Forward** the frame to that interface: queue if necessary
  - If not found
    - Forward to **ALL** interfaces
Building the switch table

A switch is self-learning

• **Switch table** (MAC address → interface): initially empty
• Whenever a frame is received, associate the interface with the source MAC address in the frame
• Delete switch table entries if they have not been used for some time

• **What about multicast?**
  – Treat it like broadcast (simplest)
  – Some switches can snoop on IGMP join/leave messages
  – Some switches (Cisco) support downloading a local multicast table from the local router

• **What about promiscuous mode?**
  – Need a managed switch – configure port for `monitor` mode or `port mirroring`
Building the switch table

• A switch interface can be associated with multiple MAC addresses
  – Cascaded links
  – Multicast addresses (if supported)
## Example Ethernet Switch

### Intel FM2112 Ethernet Switch
- 24 ports
- 1G / 10G links
- Crossbar switch built with shared memory and a crossbar
- 750 Gb/s bandwidth
- 16 banks of 64KB memory for packet payload; headers queued & scheduled separately (1 MB total)
- Switch element scheduler manages frame data & forwarding
  - Up to 4096 packets can be in the switch at one time
- Multicast/broadcast replication
- 16K (16,384) entry MAC address table
  - Binary (0/1) age: “new” refreshed whenever the entry is accessed
  - An age clock periodically purges “old” (non-refreshed) entries

### ASIX AX88655
- 5 port
- 10/100/1000 Mbps
- 4K (4096) MAC address table
- 128K byte SRAM packet buffer
- Multicast/broadcast replication

Switching

• Huge benefit: no collisions
  – No need for CSMA/CD

• Support heterogeneous links
  – 1 Gbps, 100 Mbps, fiber links, etc.

• Management
  – Disable ports
  – Prioritize ports
  – Collect statistics
  – Enable port monitoring (mirroring)
Virtual Local Area Networks (VLANs)
VLANs

• A switch + cables creates a local area network (LAN)

• We use LANs to
  – Isolate broadcast traffic from other groups of systems
  – Isolate users into groups
  – What if users move? What if switches are inefficiently used?

• Virtual Local Area Networks (VLANs)
  – Create multiple virtual LANs over one physical switch infrastructure
  – Network manager can assign a switch’s ports to a specific VLAN
  – Each VLAN is a separate broadcast domain
If we have multiple VLANs, how do we route between them?
   - As with physical LANs, connect a port from each one to a router

VLAN switches often integrate a router in them to make this easy
VLAN Trunking

• How about extending VLANs to multiple locations?
  – **VLAN Trunking**: a single connection between two VLAN-enabled switches carries all traffic for all VLANs
  – How does the switch do multiplexing/demultiplexing of traffic to the correct VLAN?
VLAN Trunking

• Extended Ethernet frame format
  – 802.1Q for frames on an Ethernet trunk

• 4-byte VLAN tag added to the frame
  – 2-byte Tag Protocol ID
  – 2-byte Tag Control Information: 12-bit VLAN ID, 3-bit priority field

• Switch adds VLAN tag for traffic on the trunk

• Switch removes VLAN tag upon receipt
  – Traffic in the trunk is sent to the appropriate VLAN based on VLAN ID
The end