Network Layer (The Data Plane): Recap

- Network Layer Overview
- Router Architecture
- Network Layer Functions and Service Models
  - Network Layer Functions
  - IP Addressing
  - DHCP
  - Network Service Models: Virtual Circuit vs. Datagram
- IP Forwarding and IP Protocol
  - IP Datagram Forwarding Model
  - IP and ICMP: Datagram Format, IP Fragmentation
- NAT, IPv6 and IPv6 transition (over IPv4)
Data Link Layer: Part I

- **Data Link Layer Functions**
  - deliver frames over a single link
  - framing, media access, error checking (error correction), ...
    - Cyclic Redundancy Code for error detection

- **Local Area Networks (LANs) and MAC Addresses**
  - MAC addresses (vs. IP address)
  - point-to-point vs. shared access
  - IP Address Resolution Protocol (ARP) and IP datagram forwarding (revisited!)

- **Extending LANs & Switched LANs:**
  - Self-learning
  - Spanning tree algorithms

- **Readings:** Textbook, Chapter 6
Some terminology:

- hosts and routers are **nodes** (bridges and switches too)
- communication channels that connect adjacent nodes along communication path are **links**
  - wired links
  - wireless links
  - **LANs** (local area networks)
- layer 2 PDU ("packet") referred to as **frame**, which encapsulates a layer-3 packet, e.g., an **IP datagram**
What Does Data Link Layer Do?

Data link layer has responsibility of transferring frames from one node to adjacent node over a single link

• An IP packet from host A to host B may traverses different links using different data link protocols
  - e.g., Ethernet on first link, frame relay on intermediate links, 802.11 on last link
• Each link protocol provides different services
  - e.g., may or may not provide reliable data delivery
• Different link protocols are not inter-operable!
  - IP packets are encapsulated/decapsulated with appropriate data link protocol header over each link
  - IP protocol and IP routers glue the links (“physical networks”) together and provide end-to-end data delivery!
Data Link Layer Functions

• Framing
  - sender (transmitter): encapsulate datagram into frame, adding header, trailer, transmit frame
  - receiver: detect beginning of frames, receive frame, decapsulate frame, stripping off header, trailer

• Link Access (Media Access Control)
  - determine whether it’s Okay to transmit over the link
    • particularly important when link shared by many nodes
      - also an issue over “half-duplex” point-to-point link (why?)
    • need media access control (MAC)
  - “physical addresses” identify sender/receiver on a link!
    • particularly important when link shared by many nodes, while over point-to-point link, not necessary
    • “physical addresses” often referred to as “MAC” addresses
      - different from IP addresses (which are logical & global)!
Other Data Link Layer Functions

• Error Detection (commonly implemented)
  - errors caused by signal attenuation, noise, etc.
  - sender computes “checksum”, attaches to frame
  - receiver detects presence of errors by verifying “checksum”
    • drops corrupted frame, may ask sender for retransmission
  - Commonly used “checksum”: cyclic redundancy code (CRC)

• Reliable delivery between adjacent nodes (optional)
  - using, e.g., go-back-N or selective repeat protocol
    • seldom used on low bit error link (fiber, some twisted pair)
    • wireless links: high error rates
    • Q: why both link-level and end-end reliability?

• Error Correction (optional)
  - receiver identifies and corrects bit error(s) without resorting to retransmission, using forward error correction (FEC) codes

• Flow Control (optional)
  - negotiating transmission rates between two nodes
Where is the link layer implemented?

- in each and every host
- link layer implemented in “adaptor” (aka *network interface card* NIC) or on a chip
  - Ethernet card, 802.11 card; Ethernet chipset
  - implements link, physical layer
- attaches into host’s system buses
- combination of hardware, software, firmware
Adaptors Communicating

- **sending side:**
  - encapsulates datagram in frame
  - adds error checking bits, rdt, flow control, etc.

- **receiving side**
  - looks for errors, rdt, flow control, etc.
  - extracts datagram, passes to upper layer at receiving side
Error Detection

EDC = Error Detection and Correction bits (redundancy)
D = Data protected by error checking, may include header fields

- Error detection not 100% reliable!
  - protocol may miss some errors, but rarely
  - larger EDC field yields better detection and correction

CSci4211: Data Link Layer: Part 1
Parity Checking

**Single Bit Parity:**
Detect single bit errors

- d data bits
- parity bit

**Two Dimensional Bit Parity:**
Detect and correct single bit errors

```
\[
\begin{array}{cccc}
  d_{1,1} & \cdots & d_{1,j} \\
  d_{2,1} & \cdots & d_{2,j} \\
  \vdots & \cdots & \vdots \\
  d_{i,1} & \cdots & d_{i,j} \\
\end{array}
\]
```

- row parity
- column parity

```
\[
\begin{array}{cccc}
  d_{1,1} & \cdots & d_{1,j} & d_{1, j+1} \\
  d_{2,1} & \cdots & d_{2,j} & d_{2, j+1} \\
  \vdots & \cdots & \vdots & \vdots \\
  d_{i,1} & \cdots & d_{i,j} & d_{i, j+1} \\
  d_{i+1,1} & \cdots & d_{i+1,j} & d_{i+1, j+1} \\
\end{array}
\]
```

```
\[
\begin{array}{cc}
  101011 \\
  111100 \\
  011101 \\
  0.010100 \\
\end{array}
\]

\[\text{no errors}\]
```

```
\[
\begin{array}{cc}
  101011 \\
  101100 \\
  011101 \\
  0.010100 \\
\end{array}
\]

\[\text{parity error}\]
```

```
\[
\begin{array}{cc}
  101011 \\
  101100 \\
  011101 \\
  0.010100 \\
\end{array}
\]

\[\text{correctable single bit error}\]
```
Internet Checksum (Review)

**Goal:** detect “errors” (e.g., flipped bits) in transmitted segment (note: used at transport layer only)

**Sender:**
- treat segment contents as sequence of 16-bit integers
- checksum: addition (1’s complement sum) of segment contents
- sender puts checksum value into UDP checksum field

**Receiver:**
- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. But maybe errors nonetheless? More later ....
Checksumming: Cyclic Redundancy Check

- view data bits, $D$, as a binary number
- choose $r+1$ bit pattern (generator), $G$
- goal: choose $r$ CRC bits, $R$, such that
  - $<D,R>$ exactly divisible by $G$ (modulo 2)
  - receiver knows $G$, divides $<D,R>$ by $G$. If non-zero remainder: error detected!
  - can detect all burst errors less than $r+1$ bits
- widely used in practice (Ethernet, 802.11 WiFi, ATM)

\[ D \times 2^r \text{ XOR } R \]

\begin{align*}
D \text{: data bits to be sent} & & R \text{: CRC bits}
\end{align*}
CRC Example

Want:
\[ D \cdot 2^r \text{ XOR } R = nG \]
equivalently:
\[ D \cdot 2^r = nG \text{ XOR } R \]
equivalently:
if we divide \( D \cdot 2^r \) by \( G \), want remainder \( R \)

\[ R = \text{remainder} \left[ \frac{D \cdot 2^r}{G} \right] \]
Multiple Access Links and LANs

Two types of “links”:
• point-to-point, e.g.,
  – PPP for dial-up access,
  – point-to-point link between Ethernet switch, host
• broadcast (shared wire or medium), e.g.
  – traditional Ethernet
  – 802.11 wireless LAN
LAN: Issues & Technologies

• Issues:
  - addressing: physical (or MAC) addresses
  - media access control (MAC) for broadcast LANs
  - expanding LANs: connecting multiple LAN segments

• Various commonly used LAN technologies
  - Ethernet
  - 802.11(WiFi)
  - PPP
MAC (Physical, or LAN) Addresses

- used to get frames from one interface to another physically-connected interface (same physical network, i.e., p2p or LAN)
- 48 bit MAC address (for most LANs)
  - fixed for each adaptor, burned in the adapter ROM
  - MAC address allocation administered by IEEE
    - 1st bit: 0 unicast, 1 multicast.
    - all 1’s : broadcast
- MAC flat address -> portability
  - can move LAN card from one LAN to another
- MAC addressing operations on a LAN:
  - each adaptor on the LAN “sees” all frames
  - accept a frame if dest. MAC address matches its own MAC address
  - accept all broadcast (MAC= all1’ s) frames
  - accept all frames if set in “promiscuous” mode
  - can configure to accept certain multicast addresses (first bit = 1)
MAC vs. IP Addresses

32-bit IP address:

- *network-layer address*, logical
  - i.e., not bound to any physical device, can be re-assigned
- IP hierarchical address **NOT** portable
  - depends on IP network to which an interface is attached
  - when move to another IP network, IP address re-assigned
- used to get IP packets to destination IP network
  - Recall how IP datagram forwarding is performed
- IP network is “virtual,” actually packet delivery done by the underlying physical networks
  - from source host to destination host, hop-by-hop via IP routers
  - over each link, different link layer protocol used, with its own frame headers, and source and destination MAC addresses
    - Underlying physical networks do not understand IP protocol and datagram format!
LAN Addresses and ARP

each adapter on LAN has unique **LAN** address

LAN (wired or wireless)

- 1A-2F-BB-76-09-AD
- 71-65-F7-2B-08-53
- 58-23-D7-FA-20-B0
- 0C-C4-11-6F-E3-98

adapter
Recall: IP Datagram Forwarding

Starting at A, given IP datagram addressed to B:

- look up net. address of B, find B on same net. as A
- link layer send datagram to B inside link-layer frame

<table>
<thead>
<tr>
<th>frame source, dest address</th>
<th>datagram source, dest address</th>
</tr>
</thead>
<tbody>
<tr>
<td>B’s MAC addr</td>
<td>A’s MAC addr</td>
</tr>
<tr>
<td>A’s IP addr</td>
<td>B’s IP addr</td>
</tr>
<tr>
<td>IP payload</td>
<td></td>
</tr>
</tbody>
</table>

CSci4211: Data Link Layer: Part 1
ARP: Address Resolution Protocol

Question: how to determine MAC address of B knowing B’s IP address?

- Each IP node (host, router) on LAN has **ARP table**
- ARP Table: IP/MAC address mappings for some LAN nodes
  
  `< IP address; MAC address; timer>`

  - timer: time after which address mapping will be forgotten (typically 20 min)

- try out “arp -a” command

LAN

- 137.196.7.23
- 137.196.7.78
- 1A-2F-BB-76-09-AD
- 137.196.7.14
- 58-23-D7-FA-20-B0
- 0C-C4-11-6F-E3-98
- 137.196.7.88
- 71-65-F7-2B-08-53
- 137.196.7.88
ARP Protocol

- A wants to send datagram to B, and A knows B’s IP address.
- A looks up B’s MAC address in its ARP table.
- Suppose B’s MAC address is not in A’s ARP table.
- A broadcasts (why?) ARP query packet, containing B’s IP address.
  - destination MAC address = FF-FF-FF-FF-FF-FF
  - all machines on LAN receive ARP query
- B receives ARP packet, replies to A with its (B’s) MAC address.
  - frame sent to A’s MAC address (unicast)
- A caches (saves) IP-to-MAC address pair in its ARP table until information becomes old (times out).
  - soft state: information that times out (goes away) unless refreshed
- ARP is “plug-and-play”:
  - nodes create their ARP tables without intervention from net administrator
**ARP Messages**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HARDWARE ADDRESS TYPE</td>
<td>protocol address Type: e.g., IP</td>
</tr>
<tr>
<td>HADDR LEN</td>
<td>HARDWARE ADDRESS TYPE</td>
</tr>
<tr>
<td>PAADDR LEN</td>
<td>PROTOCOL ADDRESS TYPE</td>
</tr>
<tr>
<td>SENDER HADDR (first 4 octets)</td>
<td>HARDWARE ADDRESS TYPE</td>
</tr>
<tr>
<td>SENDER HADDR (last 2 octets)</td>
<td>PROTOCOL ADDRESS TYPE</td>
</tr>
<tr>
<td>SENDER PAADDR (last 2 octets)</td>
<td>HARDWARE ADDRESS TYPE</td>
</tr>
<tr>
<td>TARGET HADDR (first 2 octets)</td>
<td>PROTOCOL ADDRESS TYPE</td>
</tr>
<tr>
<td>TARGET HADDR (last 4 octets)</td>
<td>HARDWARE ADDRESS TYPE</td>
</tr>
<tr>
<td>TARGET PAADDR (all 4 octets)</td>
<td>PROTOCOL ADDRESS TYPE</td>
</tr>
</tbody>
</table>

- Hardware Address Type: e.g., Ethernet
- Protocol address Type: e.g., IP
- Operation: ARP request or ARP response
ARP Request & Response Processing

• The requester broadcasts ARP request
• The target node unicasts (why?) ARP reply to requester
  - With its physical address
  - Adds the requester into its ARP table (why?)
• On receiving the response, requester
  - updates its table, sets timer
• Other nodes upon receiving the ARP request
  - Refresh the requester entry if already there
  - No action otherwise
• Some questions to think about:
  - Shall requester buffer IP datagram while performing ARP?
  - What shall requester do if never receive any ARP response?
ARP Operation Illustration

[Diagram of ARP operation with arrows and nodes labeled V, W, X, Y, Z]
forwarding to another LAN

walkthrough: send datagram from A to B via R

- focus on addressing - at IP (datagram) and MAC layer (frame)
- assume A knows B’s IP address
- assume A knows IP address of first hop router, R (how?)
- assume A knows R’s MAC address (how?)
Forwarding to Another LAN

- A creates IP datagram with IP source A, destination B
- A creates link-layer frame with R's MAC address as destination address, frame contains A-to-B IP datagram
Forwarding to Another LAN

- frame sent from A to R
- frame received at R, datagram removed, passed up to IP
Forwarding to Another LAN

- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as destination address, frame contains A-to-B IP datagram
Forwarding to Another LAN

- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as destination address, frame contains A-to-B IP datagram

IP src: 111.111.111.111
IP dest: 222.222.222.222
MAC src: 1A-23-F9-CD-06-9B
MAC dest: 49-BD-D2-C7-56-2A

IP
Eth
Phy

A
111.111.111.111
74-29-C-E8-FF-55
111.111.111.112
CC-49-DE-D0-AB-7D

222.222.222.220
1A-23-F9-CD-06-9B

B
222.222.222.221
88-B2-2F-54-1A-0F
222.222.222.222
49-BD-D2-C7-56-2A
Forwarding to Another LAN

- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as dest, frame contains A-to-B IP datagram

```
| IP src: 111.111.111.111 | IP dest: 222.222.222.222 |
| MAC src: 1A-23-F9-CD-06-9B | MAC dest: 49-BD-D2-C7-56-2A |
```

CSci4211: Data Link Layer: Part 1
Ethernet Frame Structure

sending adapter encapsulates IP datagram (or other network layer protocol packet) in Ethernet frame

<table>
<thead>
<tr>
<th>preamble</th>
<th>dest. address</th>
<th>source address</th>
<th>data (payload)</th>
<th>CRC</th>
</tr>
</thead>
</table>

**preamble:**
- 7 bytes with pattern 10101010 followed by one byte with pattern 10101011
- used to synchronize receiver, sender clock rates
Ethernet Frame Structure (More)

- **addresses**: 6 byte source, destination MAC addresses
  - if adapter receives frame with matching destination address, or with broadcast address (e.g. ARP packet), it passes data in frame to network layer protocol
  - otherwise, adapter discards frame
- **type**: indicates higher layer protocol (mostly IP but others possible, e.g., Novell IPX, AppleTalk)
- **CRC**: cyclic redundancy check at receiver
  - error detected: frame is dropped
Ethernet: Unreliable, Connectionless

- **connectionless**: no handshaking between sending and receiving NICs
- **unreliable**: receiving NIC doesn't send acks or nacks to sending NIC
  - data in dropped frames recovered only if initial sender uses higher layer rdt (e.g., TCP), otherwise dropped data lost
- Ethernet’s MAC protocol: unslotted **CSMA/CD** with binary backoff
Ethernet Switch

• link-layer device: takes an active role
  - store, forward Ethernet frames
  - examine incoming frame’s MAC address, selectively forward frame to one-or-more outgoing links when frame is to be forwarded on segment, uses CSMA/CD to access segment

• transparent
  - hosts are unaware of presence of switches

• plug-and-play, self-learning
  - switches do not need to be configured
Switch: Multiple Simultaneous Transmissions

- hosts have dedicated, direct connection to switch
- switches buffer packets
- Ethernet protocol used on each incoming link, but no collisions; full duplex
  - each link is its own collision domain
- **switching**: A-to-A’ and B-to-B’ can transmit simultaneously, without collisions

Switch with six interfaces (1,2,3,4,5,6)
Switch Forwarding Table

Q: how does switch know A’ reachable via interface 4, B’ reachable via interface 5?

- A: each switch has a switch table, each entry:
  - (MAC address of host, interface to reach host, time stamp)
  - looks like a routing table!

Q: how are entries created, maintained in switch table?

- something like a routing protocol?
Self Learning

- A bridge/switch has a **forwarding (or switch) table**
- entry in forwarding table:
  - (MAC Address, Interface, Time Stamp)
  - stale entries in table dropped (TTL can be 60 min)
- Bridge/switch *learns* which hosts can be reached through which interfaces
  - when frame received, switch “learns” location of sender: incoming LAN segment
  - records sender/location pair in forwarding table
Self-learning, forwarding: example

- switch *learns* which hosts can be reached through which interfaces
  - when frame received, switch “learns” location of sender: incoming LAN segment
  - records sender/location pair in switch table

<table>
<thead>
<tr>
<th>MAC addr</th>
<th>interface</th>
<th>TTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>60</td>
</tr>
</tbody>
</table>

Switch table (initially empty)
Filtering/Forwarding

when frame received at switch:

1. record incoming link, MAC address of sending host
2. index switch table using MAC destination address
3. if entry found for destination
   then {
     if destination on segment from which frame arrived
       then drop frame
     else forward frame on interface indicated by entry
   }
else flood /* forward on all interfaces except arriving interface */
Self-learning, forwarding: example

- frame destination, A’, location unknown: *flood*
- destination A location known: *selectively send on just one link*

<table>
<thead>
<tr>
<th>MAC addr</th>
<th>interface</th>
<th>TTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>60</td>
</tr>
<tr>
<td>A’</td>
<td>4</td>
<td>60</td>
</tr>
</tbody>
</table>

switch table (initially empty)
Interconnecting switches

self-learning switches can be connected together:

Q: sending from A to G - how does S₁ know to forward frame destined to G via S₄ and S₃?

- A: self learning! (works exactly the same as in single-switch case!)
Self-learning multi-switch example

Suppose $C$ sends frame to $I$, $I$ responds to $C$

- **Q:** show switch tables and packet forwarding in $S_1$, $S_2$, $S_3$, $S_4$
Spanning Tree Protocol

- for increased reliability, desirable to have redundant, alternative paths from source to destination
- with multiple paths, cycles result - switches may multiply and forward frame forever
- solution: organize switches in a spanning tree by disabling subset of interfaces
Switch Spanning Tree Algorithm:
Algorhyme

I think that I shall never see
A graph more lovely than a tree.
A tree whose crucial property
Is loop-free connectivity.
A tree that must be sure to span
So packets can reach every LAN.
First, the root must be selected.
By ID, it is elected.
Least cost paths from root are traced.
In the tree, these paths are placed.
A mesh is made by folks like me,
Then bridges find a spanning tree
-- Radia Perlman
Some Switch Features

• Isolates collision domains resulting in higher total max throughput
• Limitless number of nodes and geographical coverage
• Can connect different Ethernet types
• Transparent ("plug-and-play"): no configuration necessary
Institutional Network

to external network

router

mail server

web server

IP subnet
Switches vs. Routers

both are store-and-forward:

- **routers**: network-layer devices (examine network-layer headers)
- **switches**: link-layer devices (examine link-layer headers)

both have forwarding tables:

- **routers**: compute tables using routing algorithms, IP addresses
- **switches**: learn forwarding table using flooding, learning, MAC addresses
Routers vs. Switches

Switches+ and -
+ Switch operation is simpler requiring less packet processing
+ Switch tables are self learning
- All traffic confined to spanning tree, even when alternative bandwidth is available
- Switches do not offer protection from broadcast storms
Routers vs. Switches

Routers + and -

+ arbitrary topologies can be supported, cycling is limited by TTL counters (and good routing protocols)
+ provide protection against broadcast storms
- require IP address configuration (not plug and play)
- require higher packet processing

• switches do well in small (few hundred hosts) while routers used in large networks (thousands of hosts)
A day in the life: scenario

Comcast network
68.80.0.0/13

Google’s network
64.233.160.0/19

Browser

School network
68.80.2.0/24

Web page

Web server
64.233.169.105
A day in the life… connecting to the Internet

- connecting laptop needs to get its own IP address, addr of first-hop router, addr of DNS server: use **DHCP**
  - DHCP request encapsulated in **UDP**, encapsulated in **IP**, encapsulated in **802.3 Ethernet**
  - Ethernet frame broadcast (dest: $\text{FFFFFF}^{12}$) on LAN, received at router running **DHCP** server
  - Ethernet demuxed to **IP** demuxed, **UDP** demuxed to **DHCP**
A day in the life… connecting to the Internet

- DHCP server formulates **DHCP ACK** containing client’s IP address, IP address of first-hop router for client, name & IP address of DNS server
  - encapsulation at DHCP server, frame forwarded (**switch learning**) through LAN, demultiplexing at client
  - DHCP client receives DHCP ACK reply

**Client now has IP address, knows name & addr of DNS server, IP address of its first-hop router**
A day in the life… ARP (before DNS, before HTTP)

- before sending *HTTP* request, need IP address of www.google.com: *DNS*
  - DNS query created, encapsulated in UDP, encapsulated in IP, encapsulated in Eth. To send frame to router, need MAC address of router interface: *ARP*
  - *ARP* query broadcast, received by router, which replies with *ARP reply* giving MAC address of router interface
  - client now knows MAC address of first hop router, so can now send frame containing DNS query
A day in the life… using DNS

- IP datagram containing DNS query forwarded via LAN switch from client to 1st hop router
- IP datagram forwarded from campus network into Comcast network, routed (tables created by RIP, OSPF, IS-IS and/or BGP routing protocols) to DNS server
- demuxed to DNS server
- DNS server replies to client with IP address of www.google.com
A day in the life…TCP connection carrying HTTP

- to send HTTP request, client first opens TCP socket to web server
- TCP SYN segment (step 1 in 3-way handshake) inter-domain routed to web server
- web server responds with TCP SYNACK (step 2 in 3-way handshake)

TCP connection established!
A day in the life… HTTP request/reply

- web page finally (!!!) displayed

- HTTP request sent into TCP socket
- IP datagram containing HTTP request routed to www.google.com
- web server responds with HTTP reply (containing web page)
- IP datagram containing HTTP reply routed back to client