Forwarding and Addressing in the Link Layer

3035/GZ01 Networked Systems
Kyle Jamieson
Lecture 4

Department of Computer Science
University College London
The link layer

The link layer (L2) has the job of transferring a datagram **point-to-point** over a link.

Unit of data is called a **frame** at the link layer.
Where is the link layer implemented?

- In each and every host
- **Network adapter**
  - Physical card
  - Chip in a device
  - Attaches to host’s system bus
- Implemented both in hardware and software
  - Software: device driver running on CPU
  - Hardware: FPGA, ASIC in network adapter
Hardware or software?

- Software is more flexible, unless you need:

  1. Time-critical functionality
     - Backoff transmission timing in Ethernet
  2. Highly parallel processing
     - Dot product computations for CDMA, filtering for the PHY
Today

1. Framing
   - Byte-oriented framing
   - Bit-oriented framing
2. Link layer addressing
3. Link layer switching and forwarding
Framing: The problem

- We have seen how to encode **bits** on a link
  - Ethernet: Manchester encoding, synchronization on edges
  - Result in general: An infinite stream of bits
- Problem: where does each frame begin and end?
 Byte-oriented framing

• Treat each frame as a collection of bytes rather than bits
• Byte-oriented framing link layer protocols:
  – PPP (point-to-point protocol), used to carry Internet Protocol (IP) packets over dialup lines, Virtual Private Networks
  – BISYNC (IBM, 60’s), Digital Data Communication Message Protocol (DDCMP), Digital Equipment Corporation’s DECNET
• BISYNC uses *sentinel* characters to frame:

  ![Diagram of BISYNC framing](image)

  • SYN: Beginning of frame
  • SOH: Start of header
  • STX: Start of text (packet data)
  • ETX: End of text
  • *What if sentinel occurs in data?* **Character stuffing:** insert escape character DLE before sentinel
  • *What if escape character occurs in data?* Escape the escape character
PPP byte stuffing

• Like BISYNC, PPP takes a sentinel-based approach

• At the sender:
  – Stuff escape byte before each flag byte in payload
  – Stuff escape byte before each escape byte in payload

• At the receiver:
  – Escape, flag byte → discard escape byte, keep flag byte, continue data reception
  – Single flag byte: interpret as the flag byte
  – Two escape bytes → discard one
Byte-counting approach: DDCMP

- **Byte-counting framing**: Include number of bytes in header (Count field):

- **What if Count field is corrupted?**
  - Will frame the wrong bytes
  - This is called a *framing error*
  - With high probability, CRC will detect the framing error and discard the frame

- **Do we still need to escape the SYNs?**
Bit-oriented framing

• Main idea: View link as stream of bits instead of bytes
• Example: High-Level Data Link Control (HDLC), used in Cisco routers and the V.42 standard for telephone modems
• Choose some pattern of bits as a sentinel e.g.: \[\underbrace{111111}_{\text{Seven 1s}}\]

• **Bit stuffing algorithm:** Modify the outgoing data stream as follows:
  • At sender: \[\underbrace{111111}_{\text{Six 1s}} \rightarrow \underbrace{111110}_{\text{Six 1s}}\]
  • At receiver: \[\underbrace{111110}_{\text{Six 1s}} \rightarrow \underbrace{111111}_{\text{Seven 1s}} \rightarrow \underbrace{11111}_{\text{end of frame}}\]

• Size of frame depends on data it contains!
• Not guaranteed free of framing errors
Today

1. Framing
2. Link layer addressing
3. Link layer switching and forwarding
4. Virtualizing links
Comparing addressing schemes

• Network layer address (IP address)
  – Function: move datagram to destination network
  – 32-bit address, dotted quad notation $a.b.c.d$ where each component is an eight-bit unsigned integer
  – Hierarchical address space

• Link layer address (MAC address, Ethernet address):
  – Function: move frame from one point to another point on the same network
  – 48-bit address (in most LANs)
  – Burned in NIC ROM, also sometimes software settable
  – Usually a flat address space
Link-layer addressing

- Each adapter on LAN has unique link layer address, my_station_id
- Special **broadcast** address broadcast_id

```
+-----------------+  +-----------------+  +-----------------+  +-----------------+  +-----------------+
| 17  | 24  | 12  | 05  | 19  |
   |     |     |     |     |     |
Station Identifier (Ethernet Address)```

- Each L3 protocol registers itself in net_handler array
- L2 address-related functionality for inbound frames:
  - Test received_frame.destination against my_station_id or broadcast_id, drop if neither match
  - handler ← net_handler
    ```python
    [received_frame.net_protocol]
    ```
  - call handler(received_frame.payload)
Mapping network to link addresses

- Goal: Translate from
  \texttt{network\_send(data, length, IP, N)} to
  \texttt{link\_send(data, length, IP, enet/18)}

- Name-mapping table: example of soft state
Link layer addresses aren’t routable

- Network layer *routes* from one network to another
The Address Resolution Protocol (ARP)

L sending a datagram to N:
L: network_send(data, length, IP, N)
L: link_send(“where is N?”, length, ARP, broadcast_id)
N: link_send(“N is at enet/18”, length, ARP, enet/17)
L: link_send(data, length, IP, enet/18)
ARP’s role in L3 routing

• Walkthrough: send IP data from L to E via router K
• Router K
  • Two ARP tables: one for each subnet
  • Two interfaces (interface ids 6, 4): one for each subnet, with two link-layer addresses (19, 27)
Example: routing between subnets

L sending a datagram to E:
L: network_send(data, length, IP, E)
L: link_send(data, length, IP, enet/28)
L: ARPs for K’s link layer address (enet/19)
L: link_send(data, length, IP, enet/19)
K: ARPs for E’s link layer address (enet/28) on link/4
K: link_send(link/4, data, length, IP, enet/28)
Hubs

• Replicating device: bits coming in go out all ports
  – Operates at the physical layer (L1)
• Doesn’t run CSMA/CD: immediately replicates bits
• Collisions possible between all hosts
• No frame buffering at the hub
• Physical limitations: can’t grow collision domain too big
Today

1. Framing
2. Link layer addressing
3. Link layer switching and forwarding
   - Learning switches
   - Spanning Tree Protocol
   - Virtual LANs
Ethernet nowadays: Switched

- Hosts have dedicated, direct connection to switch
- Switches buffer frames
- Ethernet protocol used on each incoming link, but no collisions
  - Each link in own collision domain
- Switching: Allows two conversations $A \rightarrow A'$ and $B \rightarrow B'$ simultaneously, without collisions
The switch table

• How to tell which port to forward packet?

• The switch table:
  LAN addr  Port  Time-to-live

• Switch table initially empty

• How to build up switch table entries?
Building the switch table: Self-learning

- **Switch table**: entries of (LAN address, port id, time-to-live)

- **Learning algorithm**: on receiving a packet from LAN address S to LAN address D:
  1. Add S to switch table with incident port id
  2. If D is broadcast id: forward out all ports
  3. If D is in switch table, lookup entry and forward to resulting port id. **Otherwise**, forward out all ports

- Periodically flush link table entries based on time-to-live
Building up the switch table: example

1. A → S: [A (enet/21) → D (enet/24)]
   S → (all ports)
2. B → S: [B (enet/22) → C (enet/23)]
   S → (all ports)
3. D → S: [D (enet/24) → B (enet/22)]
   S → port 2 only

Switch table:

<table>
<thead>
<tr>
<th>LAN addr</th>
<th>Port</th>
<th>TTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>enet/21</td>
<td>1</td>
<td>20 min</td>
</tr>
<tr>
<td>enet/22</td>
<td>2</td>
<td>20 min</td>
</tr>
<tr>
<td>enet/24</td>
<td>4</td>
<td>20 min</td>
</tr>
</tbody>
</table>
### Aside: What is a *hub*?

<table>
<thead>
<tr>
<th><strong>Hub</strong></th>
<th><strong>Switch</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Differences</strong></td>
<td><strong>Commonalities</strong></td>
</tr>
<tr>
<td>– Physical layer (L1) device</td>
<td>– Hosts are unaware of presence of hubs</td>
</tr>
<tr>
<td>– Replicating device: bits in one port go out all ports</td>
<td>– No configuration needed</td>
</tr>
<tr>
<td>– No medium access control</td>
<td>– Hosts are unaware of presence of switches</td>
</tr>
<tr>
<td>– No frame buffering</td>
<td>– No configuration needed</td>
</tr>
</tbody>
</table>

- **Switch**
  - Link-layer (L2) device
  - Selectively forward frame to one or more outgoing links
  - CSMA/CD MAC
  - Buffers frames
Interconnecting LANs

- Switches can connect LANs as well as hosts
- Sometimes called **bridges** in this context
- The entirety is called an **extended LAN**
Interconnecting LANs: example

Suppose C sends frame to F, F responds to C

Link layer addresses

```
<table>
<thead>
<tr>
<th>Address</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>4</td>
</tr>
</tbody>
</table>
```
Interconnecting LANs: example

Suppose C sends frame to F, F responds to C
Interconnecting LANs: example

Suppose C sends frame to F, F responds to C
Interconnecting LANs: example

Suppose C sends frame to F, F responds to C
Problem: Forwarding loops
Problem: Forwarding loops
Problem: Forwarding loops

• Can’t learn the direction of a source if it’s in more than one direction, so bridge learning algorithm breaks

• Why might loops form?
  – Inadvertently: many people responsible for network, one person adds a bridge

• Intentionally: more connections between bridges increases redundancy, helping to cope with failure
  – So we need to revise the bridge learning algorithm
The spanning tree protocol (STP)

• Manager at DEC asked Radia Perlman to build a switch (*bridge*) to connect two Ethernets

• **Perlman’s idea:** Switches agree on a loop-free and connected *spanning tree*

• Implementers at DEC resisted (wanted simplest possible design), first customer site connected bridge to one Ethernet twice, generating a “broadcast storm”

• Once the spanning tree is formed:
  – Switches block some ports from sending or receiving data
  – Switches continue using the learning switch algorithm to forward over the spanning tree
Spanning tree protocol (STP): Outline

1. Elect one **root switch** (switch with the lowest ID)

2. Compute shortest paths tree from root to each switch $S$
   - Note which port of $S$ is on path to root: **root port** ($R$)

3. All switches connected to a LAN choose a **designated switch** to forward frames to root (switch on path to root from LAN)
   - *e.g.* switch 2 is designated for the LAN below
   - *i.e.*, each port decides if it is a **designated port** ($D$)

4. Block all other ports: **blocked port** ($B$)
STP: Messages and switch state

• All switches exchange **configuration messages**
  switch $X$ sends: (Root identifier, distance to root, $X$)
  – Configuration messages are never blocked
  – Switch $X$ generates configuration messages periodically with a
    “clock tick” timer, initially sending ($X$, 0, $X$)

• State at each switch $X$:
  1. Root identifier (initially $X$)
  2. Configuration message to send (initially ($X$, 0, $X$))
  3. State at each port:
     a) **Forwarding** data traffic or **blocking** data traffic
        (initially forwarding)
     b) “Best” configuration message + *age* of that message
        (initially empty)
Electing a per-LAN designated switch

- **Designated port rule:** At a switch, for each port $p$
  - Consider all configuration messages received on port $p$ and the configuration message the switch would send
  - If receive “better” configuration message on a port $p$, don’t send configuration messages on port $p$, otherwise $p$ is designated: send your configuration message on $p$

- **Rule for comparing configuration messages:**
  $(R_1, d_1, X_1)$ better than $(R_2, d_2, X_2)$ if $R_1 < R_2$ or
  $(R_1 = R_2 \text{ and } d_1 < d_2)$ or
  $(R_1 = R_2 \text{ and } d_1 = d_2 \text{ and } X_1 < X_2)$
Comparing configuration messages

• Example: Switch 2 sends (2, 0, 2); 3 sends (3, 0, 3)
• Message (2, 0, 2) is better than (3, 0, 3)
  – Subsequently, switch 2, port 1 sends configuration messages periodically
  – Subsequently, switch 3, port 1 does not send configuration messages
STP: initial startup phase

- **Initially**, X generates configuration messages periodically, sending \((X, 0, X)\)
- All ports designated and forwarding
- Result: For each LAN, one attached switch (the designated switch) transmits configuration messages

<table>
<thead>
<tr>
<th>Switch 5</th>
<th>Root</th>
<th>Dist</th>
<th>Transmitter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port 1</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Port 2</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Port 3</td>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>
Calculating root ID and cost to root

• Switches continuously take the following steps:

1. **Root ID rule:** Root ID $r$ at switch $X$ is the minimum of $X$ and root IDs received at all ports

2. Calculate distance to root $d$ at switch $X$:
   – If $X$ is the root ($X = r$), $d = 0$
   – Otherwise, $X$ is not the root ($X \neq r$):
     • $d = \text{one plus the minimum cost from configuration messages received on all ports (transmitter field breaks ties).}$ Suppose this comes from port $p$.
     • **Root port rule:** Designate port $p$ as a *root port*

3. Switch $X$’s configuration message is now $(r, d, X)$. Reapply designated port rule on all ports

4. **Blocked port rule:** Don’t forward data to or from a port if it is not a designated port or a root port
STP: building the tree

- On LANs A & B, 5 hears (2, 0, 2)
- 5’s root ID $r$: 2, $d = 1$ (port breaks tie)
- 5’s new message: (2, 1, 5)
STP: building the tree

- On LAN A, 66 hears (2, 0, 2)
- 66’s root ID r: 2, d = 1
- 66’s new message: (2, 1, 66)
STP: building the tree

- On LAN C, **101** hears (2, 1, 5)
- 101’s root ID r: 2, d = 2
- 101’s new message: (2, 2, 101)
STP: building the tree

- On LAN A, 90 hears (2, 0, 2)
- 90’s root ID: r = 2, d = 1
- 90’s new message: (2, 1, 90)

<table>
<thead>
<tr>
<th>Switch 90</th>
<th>Root</th>
<th>Dist</th>
<th>Transmitter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port 1</td>
<td>2</td>
<td>1</td>
<td>66</td>
</tr>
<tr>
<td>Port 2</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Port 3</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Port 4</td>
<td>2</td>
<td>1</td>
<td>90</td>
</tr>
<tr>
<td>Port 5</td>
<td>2</td>
<td>1</td>
<td>90</td>
</tr>
</tbody>
</table>
On LAN F, 87 hears (2, 1, 90)

87’s root ID $r$: 2, $d = 2$

87’s new message: (2, 2, 87)
STP: Handling topology changes

• Configuration messages also contain age: \((r, d, X, \text{age})\)
  – age = 0 when sending configuration message

• Best configuration message for each port contains age
  – Age incremented each unit of time
  – If age reaches some threshold (max age), discard that configuration message and recalculate using all rules

• Recalculate when:
  1. Receive better or newer configuration message on port \(p\): overwrite existing configuration message
  2. Timer ticks: increment message age in stored messages for each port
STP: Handling failures

90’s root ID: $r = 2$, $d = 1$
90’s message: $(2, 1, 90)$

<table>
<thead>
<tr>
<th>Switch 90</th>
<th>Root</th>
<th>Dist</th>
<th>Transmitter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port 1</td>
<td>2</td>
<td>1</td>
<td>11 66 90</td>
</tr>
<tr>
<td>Port 2</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Port 3</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Port 4</td>
<td>2</td>
<td>1</td>
<td>90</td>
</tr>
<tr>
<td>Port 5</td>
<td>2</td>
<td>1</td>
<td>90</td>
</tr>
</tbody>
</table>
STP: Handling new bridges

- Don’t want loops, even for short periods of time
- Switches 2, 5, 101 send messages immediately, with current age in table
- On power-up, switch puts its ports in new **pre-forwarding** state: sends configuration messages as if designated, but doesn’t forward data
The problem with extended LANs

- Switched LANs afford greater scalability, but extended LANs do not isolate traffic
  - Broadcast frames traverse the entire extended LAN

- Two consequences:
  1. Broadcast traffic does not scale, reducing overall performance
  2. Allows eavesdropping across LANs

- Therefore, extended LANs don’t scale
Virtual LANs (VLANs)

• Idea: The switch assigns each port a **color**, an identifier designating the VLAN that port belongs to

• Traffic isolation: colors = broadcast domains

• Easily reconfigurable port assignments

• Routing between VLANs: layer 3 routing functionality
VLAN example

- Configure ports on W, X, Y, and Z to be in appropriate VLANs
  - *Trunk ports* between B1 and B2 configured for both VLANs

- Bridge inserts VLAN header containing color between Ethernet header and payload

- If a packet contains a VLAN header, bridges only forward on matching-color or trunk ports
Why can’t Ethernet replace IP?

• L2 is point-to-point, L3 has source and destination

• Then Ethernet came, world got confused and thought Ethernet was a competitor to L3

• Perlman: “Should have been called ‘Etherlink.’”

• So, why can’t Ethernet replace IP?
  1. Flat addresses
  2. No hop count (loops would be a disaster)
  3. No fragmentation, reassembly, for heterogeneous networks
Multi-hop Networks and End-to-end Arguments
Pre-Reading: End-to-end Arguments in System Design

NEXT TIME