Lecture 6.1: Internetworking
Internetworking: Discussions

• Up to now, we have learned how to
  • Connect one node to another
  • Connect one node to an existing network, e.g. Ethernet
  • “Last-mile” link: Connect to a modest number of node together: Ethernet 1024 nodes, point-to-point: 2 nodes

• Next, we will learn how to build networks of global scale: How to *interconnect different types of links and networks*: **Internetworking**
For *Internetworking*, we shall look at few sub-problems:

- *Interconnect links of the same type*: **Switches**
- We consider an important of class switch: **Bridges** to *interconnect Ethernet segments*.
- We also look a way to *interconnect disparate networks and links*: **Gateways**, or now mostly known as **routers**. We shall focus on the **IP**
- Once we are able to interconnect a whole lot of links and networks with switches and routers, we will look at a way to *find a suitable path, or route* through a network:
  - Paths that are efficient, loop free, etc.: **Routing**
Internetworking: Our Goal

- Understanding the functions of **switches, bridges and routers**

- Discussing **Internet Protocol (IP)** for interconnecting networks

- Understanding **the concept of routing**
Switch

– A mechanism that allows us to interconnect links to form a large network

– A multi-input, multi-output device which transfers packets/frames \textit{from an input to one or more outputs}

– How about Hub/Repeater?
Switching Topology

- A switch implements a **star topology**
- Switches are MIMO devices

- Ports are numbered
Properties

– Even though a switch has a fixed number of inputs and outputs, which limits the number of hosts that can be connected to a single switch, *large networks can be built by interconnecting a number of switches*

– Adding a new host to the network by connecting it to a switch does not necessarily mean that the hosts already connected will get worse performance from the network
Properties

The last claim cannot be made for the shared media network (discussed in Chapter 2)

– It is impossible for two hosts on the same Ethernet to transmit continuously at 10Mbps because they share the same transmission medium

– Every host on a switched network has its own link to the switch

  • So it may be entirely possible for many hosts to transmit at the full link speed (bandwidth) provided that the switch is designed with enough aggregate capacity
Primary Job

• A switch’s primary job is to receive incoming packets on one of its links and to transmit them on some other link
  – This function is referred as switching and forwarding
Switches

- Link-layer device: smarter than hubs, take 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

- plug-and-play, self-learning
Properties

Allows 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' simultaneously, without collisions
  - not possible with dumb hub
Example: Institutional Network

to external network

router

mail server

web server

IP subnet
Switching and Forwarding

• How does a switch decide which output port to place each packet on?
  – It looks at the header of the packet for an identifier that it uses to make the decision
  – Two common approaches
    • Datagram or Connectionless approach
    • Virtual circuit or Connection-oriented approach
  – A third approach source routing is less common
Datagram/Connectionless

• Assumptions
  – Each host has a globally unique address, i.e. MAC
  – There is some way to identify the input and output ports of each switch
    • *We can use numbers*
    • We can use names
Datagram/Connectionless

• Datagrams
  – Key Idea
    • Every packet contains enough information to enable any switch to decide how to get it to destination
      – Every packet contains the complete destination address
Datagram/Connectionless

• **No dedicated** connection between communicating hosts: Packets may follow **independent paths** to the destination

• Packets are sent to the switch at any time (no contention)

• Source is **not aware** of the state of the destination

• Less prone to switch failures if alternative paths exist
An example network

- To decide how to forward a packet, a switch consults a *forwarding table* (sometimes called a *routing table*)
Forwarding Table

Entry: MAC address of host, port/interface to reach host: looks like a routing table!

Q: how are entries created, maintained in switch table?
Switch: Self-Learning

- 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>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
</tbody>
</table>

Switch table (initially empty)
Switch: Frame Filtering/Forwarding

When frame received:

1. record link associated with sending host
2. index switch table using MAC dest address
3. if entry found for destination
   then {
     if dest on segment from which frame arrived
     then drop the frame
     else forward the frame on interface indicated
   }
else flood → forward on all but the interface on which the frame arrived
Self-Learning/Forwarding Example

• Frame destination unknown: **flood**
  - Destination A location known: **selective send**

<table>
<thead>
<tr>
<th>MAC addr</th>
<th>Port</th>
<th>TTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>A'</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Forwarding table (initially empty)
Interconnecting Switches

- switches can be connected together

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

A: self learning! (works exactly the same as in single-switch case!)
Suppose C sends frame to I, I responds to C

Q: show switch tables and packet forwarding in $S_1$, $S_2$, $S_3$, $S_4$
Characteristics of Connectionless (Datagram) Network

– A host can send a packet anywhere at any time, since any packet that turns up at the switch can be immediately forwarded (assuming a correctly populated forwarding table)

– When a host sends a packet, it has no way of knowing if the network is capable of delivering it or if the destination host is even up and running

– Each packet is forwarded independently of previous packets that might have been sent to the same destination.
  
  • Thus two successive packets from host A to host B may follow completely different paths

– A switch or link failure might not have any serious effect on communication if it is possible to find an alternate route around the failure and update the forwarding table accordingly
Virtual Circuit Switching

– Uses the concept of *virtual circuit* (VC)
– Also called a connection-oriented model
– First set up a virtual connection from the source host to the destination host and then send the data
Virtual Circuit (VC) Switching

Two-stage process

– Connection setup

– Data Transfer

• Connection setup

– Establish “connection state” in each of the switches between the source and destination hosts

– The connection state for single connection consists of an entry in the “VC table” in each switch through which the connection passes
One entry in the VC table on a single switch contains

- **Incoming VC identifier (VCI)** (identifies the connection per link)
- **Outgoing VC identifier (VCI)** (possibly different than the incoming)
- **Incoming interface** (different than the VC Identifier, similar to a port)
- **Outgoing interface** (different than the VC identifier, similar to a port)

- The semantics for one such entry is
  - If a packet arrives on the designated incoming interface and that packet contains the designated VCI value in its header, then the packet should be sent out the specified outgoing interface with the specified outgoing VCI value first having been placed in its header

<table>
<thead>
<tr>
<th>Incoming Interface</th>
<th>Incoming VC</th>
<th>Outgoing Interface</th>
<th>Outgoing VC</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
<td>11</td>
</tr>
</tbody>
</table>
**Permanent VC (PVC):** Administrator sets up a permanent connection and configures VC table.

**Switched VC (SVC):** The host signals to the switches in order to establish a VC, *dynamically.*
Let’s assume that a network administrator wants to manually create a new virtual connection from host A to host B

– First the administrator identifies a path through the network from A to B
The administrator then picks a VCI value that is currently unused on each link for the connection

- For our example,
  - Suppose the VCI value 5 is chosen for the link from host A to switch 1
  - 11 is chosen for the link from switch 1 to switch 2
  - So the switch 1 will have an entry in the VC table

<table>
<thead>
<tr>
<th>Incoming Interface</th>
<th>Incoming VC</th>
<th>Outgoing Interface</th>
<th>Outgoing VC</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
<td>11</td>
</tr>
</tbody>
</table>
## PVC Example

<table>
<thead>
<tr>
<th>Switch</th>
<th>Incoming Interface</th>
<th>Incoming VCI</th>
<th>Outgoing Interface</th>
<th>Outgoing VCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>11</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>7</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

![Diagram of PVC example with switches and interfaces]
Data Transfer Stage
SVC Example

- A sends *setup msg* to switch 1 indicating the address of B

- Switch 1 setups incoming/outgoing interfaces and VCIs

- Connection setup msg is forwarded like a datagram to switch 2

- Switch 2 repeats the setup process

- Once data stage is over, connection is torn down
SVC: Observations

• Delay?

• Overhead (in terms of address)?

• A switch/link failure?

• How to get set up message to B?
• Bridges are **special switches** that *interconnect* *Ethernet networks*
• Act as **simple nodes** on each Ethernet
• What is Forwarding Table for this Bridge?
Problem with Flooding
Why Loop?

• How does an extended LAN come to have a loop in it?
  – Network is managed by more than one administrator
    • For example, it spans multiple departments in an organization
    • It is possible that no single person knows the entire configuration of the network
      – A bridge that closes a loop might be added without anyone knowing
  – Loops are built into the network to provide redundancy in case of failures

• Solution
  – Distributed Spanning Tree Algorithm
Preliminaries on Graph

- Graph $G(V, E)$: Collection of vertices and edges
  - $V$: Set of vertices
  - $E$: Set of edges, $e(x, y) = 1$, if $x, y$ connected, 0 otherwise
- **Path**: sequence of vertices with an edge between subsequent vertices (can be defined as sequence of edges as well)
A graph is said to be connected, if there is a path from any node to any other node.
(A) Cyclic Graph

- **Cycle**: a path with the same first and last vertex
- **Cyclic graph**: A graph that contains at least one cycle
- **Acyclic graph**: A graph with no cycles
- **Tree**: An acyclic connected graph (spanning tree, spans all vertices)
- **Forest**: A disconnected acyclic graph (a graph with many trees)
Mapping Extended LAN to Graph

- Think of the extended LAN as being represented by a graph that possibly has loops (cycles)

- A spanning tree is a sub-graph of this graph that covers all the vertices but contains no cycles
  - Spanning tree keeps all the vertices of the original graph but throws out some of the edges

Example of (a) a cyclic graph; (b) a corresponding spanning tree.
Mapping Extended LAN to Graph

- Bridges and LANs become vertices, ports become edges
- Goal: Make a tree that spans only LANs (red vertices): Spanning Tree Algorithm
Spanning Tree Algorithm

• Each bridge has a unique identifier
• Pick bridge with smallest ID, make it the root of the tree
Spanning Tree Algorithm

- Compute shortest path (in terms of hops) from each bridge to root
• Each LAN remains connected to the bridge with shortest path to the root. In case of a tie use smallest ID.
Distributed Spanning Tree: Message Exchange

Initially, all bridges consider themselves as roots
Broadcast on all ports ID, root bridge, and distance to root
If bridge hears a message from another bridge with smaller ID it adjusts root/distance, and forwards
Ex., Let \( (Y, d, X) \) denote (root, distance, bridge ID)
   
   \( B_3 \) receives \( (B_2, 0, B_2) \)
   \( B_3 \) accepts B2 as root
   \( B_3 \) adds one on its distance to B2, and forwards to B5 \( (B_2, 1, B_3) \)
   \( B_2 \) accepts B1 as root and updates B3 with \( (B_1, 1, B_2) \)
   \( B_5 \) accepts B1 as root and updates B3 with \( (B_1, 1, B_3) \)
   \( B_3 \) accepts B1 as root, and note that both B2, B5 are closer to root in both ports. Hence, B3 blocks all ports