Switching, Forwarding, and Queuing

Module Goals

At the conclusion of this module, students will be able to

- differentiate between routers, switches, and hubs
- explain destination-based forwarding
- explain various queuing disciplines
- show the operation of FIFO and fair queues
- compare and contrast the behavior or queuing disciplines

Definitions

- **ports**: a physical connection in or out of a device
- **switch**: a layer 2 (data link layer) device that builds a network by allowing devices using the same protocol to talk

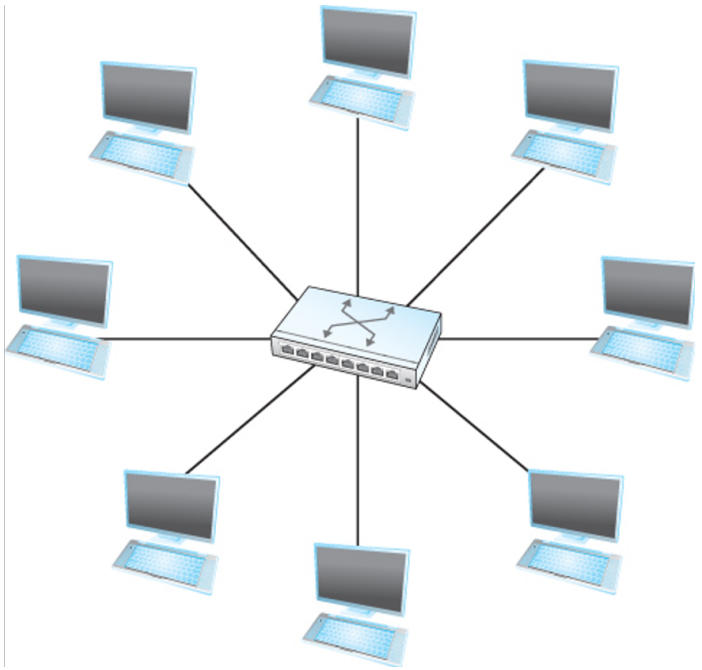
  (usually intelligent enough to route a packet out just one outgoing packet)
- **router**: a layer 3 (network layer) device that connects one or more networks together, regardless of the link layer protocol

  (almost unusual to see a router without some switch hardware or vice versa)
- **hub**: a layer 2 (maybe layer 1) device that connects devices together but is dumb

  (usually repeats whatever came in on all of the other output ports)

A Question

- the west host decides to talk to the southeast host
- data goes out from the west host to the switch—that's the easy part
- how does the switch know which output to use to reach the southeast host?



Switching and Forwarding

- look at the header of the packet for an identifier to use for the decision
- how header information is used caan vary, but there are two common approaches
  - **dataagram** or **connectionless** approach
  - **virtual circuit** or **connection-oriented** approach
- we'll only focus on the first approach

Switching Assumptions

- each host has a globally unique **address**
- there is some way to identify the input and output ports for each switch
  - we can use numbers (this is port 7)
  - we can use names (this is the port leading to host `DeepThought`)

What is a Datagram?

- as noted, at each level of communication, a “chunk” of data has different names
- at the transport layer, they are **segments**
- at the network layer, they are **datagrams** or **packets**
- at the link layer, they are **frames**

What is a Datagram?

- a datagram is defined as...

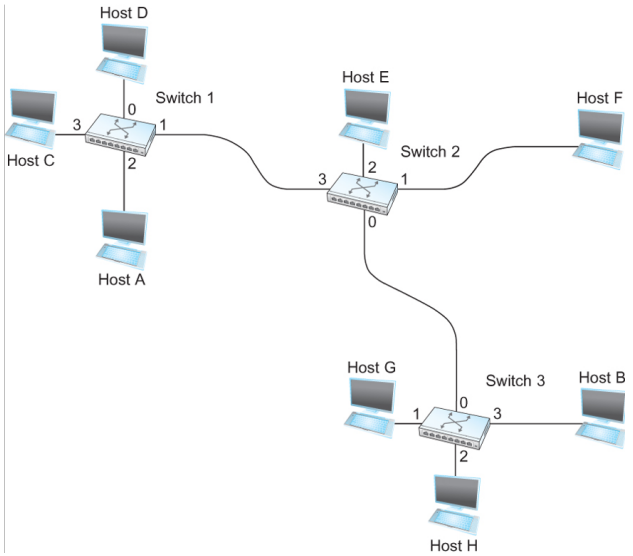
  a self-contained, independent entity of data carrying sufficient information to be routed from the source to the destination computer without reliance on earlier exchanges between this source and destination computer and the transporting network

- the key idea is that the datagram has a destination address that can be used by the switch to route it to the appropriate place

Switching and Forwarding

- to decide how to forward a packet, a switch consults a **forwarding table** (sometimes erroneously called a **routing table**—they are different!)
- for switch 2 the forwarding table might look like this:

Dest.	A	B	C	D	E	F	G	H
Port	3	0	3	3	2	1	0	0



- what happens when the topology starts changing? (stay tuned)

Connectionless Networks

- so you have a network that sends datagrams... what else can we say about this network?
- a host can sent a packet anywhere at any time!

  (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

Connectionless Networks

- each packet is forwarded independently of previous packets that might have been sent to the same destination

  (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

  (well, if we can find an alternate route around the failure and update the forwarding table)

Queuing

- it may be the case that lots of packets are arriving at the switch or router faster than they can be services
- in our earlier queuing theory discussions this was common but we didn't really define how things went into the queue
- this is actually a really important aspects, because how things are placed directly affects how soon they get to leave
- two main ideas: FIFO and Fair queuing

FIFO Queuing

- Also called **first-come, first-served** (FCFS) queuing
- idea: the first packet that arrives is the first packet to be transmitted
- if the queue is full, discard the packet
  - without regard to which flow the packet belongs or how important the packet is
  - called **tail drop**, since packets that arrive at the tail end of the queue are dropped
- tail drow and FIFO queuing are not inseparable
  - queuing just tells us how packets get sent
  - tail drop tells us how packets get dropped

Priority Queuing

- a simple variant on the FIFO queue is the priority queue
- each packet is marked with a priority (say, in the IP header) or is inferred from existing fields
- the routers then implement multiple FIFO queues with one for each priority class
- the routers always transmit packets out of the highest-priority queue with packets in it
- each "priority queue" is just a FIFO queue

When FIFO Isn't Sufficient

- philosophy: a packet is a packet is a packet (FIFO is stupid)
  - indiscriminate treatment doesn't allow packets to be sorted by flow
  - at the same time, FIFO with tail drop is the protocol followed by most Internet routers
- **fair queuing** (FQ) is an algorithm that has been proposed to address this problem
- idea: maintain a separate queue for each flow currently being handled and service these queues in a round-robin fashion

Is It That Simple?

- **no.**
- the main complication with FQ is that packets being processed at the router are not necessarily the same length
- to truly allocate the bandwidth of the outgoing link in a fair manner, it is necessary to take packet length into consideration
- for example, what happens if two flows are using a router, one with 500 byte packets and the other with 1000 byte packets?

Making Fair Queuing “Fair”

- what we really want is **bit-by-bit round-robin** where the router transmits a bit from flow 1, then a bit from flow 2, and so on
- its not feasible to interleave the bits from different packets

  The GQ mechanism therefore simulates this behavior by first determining when a given packet would finish being transmitted if it were being sent using bit-by-bit round-robin, and then using this finishing time to sequence the packets for transmission.

- wait, what?

Understanding Bit-by-Bit

- to understand the algorithm for approximating bit-by-bit round-robin, consider the behavior of a single flow
- for this flow, let
  - P_i denote the length of packet i
    (measured in ticks of the clock to transmit i)
  - S_i denote the time when the router starts to transmit packet i
  - F_i denote the time when the router finishes transmitting packet i
- clearly $F_i = S_i + P_i$

Understanding Bit-by-Bit

- when do we start transmitting packet i?
  (depends on whether packet i arrived before or after the router finishes transmitting packet $i-1$)
- if it was before, we just keep sending bits
  (remember, only one flow)
- if it was after, there was a time when nothing was transmitted
- Let A_i denote the time that packet i arrives at the router, then...

$$S_i = \max\left(F_{i-1}, A_i\right) \text{ and } F_i = \max(F_{i-1}, A_i) + P_i$$

Multiple Flows

- now for every flow, we calculate F_i for each packet that arrives
- all of the F_i values act as timestamps
- the next packet to be transmitted is always the packet that has the lowest timestamp

  (the packet that should finish transmission before all others)
- a freshly arrived packet, being shorter than something already in the transmit queue, may be promoted to early transmission

Summary

- switches build networks and routers connect networks
- forwarding tables tell a router what to do with a given packet
- when things back up, packets get queued using some queuing discipline

  (FIFO, priority, fair)