Link Layer and Wireless

CS144 Review Session 7

May 16, 2008

Ben Nham

Routers, Switches, and Hubs

- Routers are network layer devices
  - Modify IP datagram (decrement TTL)
  - Hosts and other routers must be aware of them
- Switches and hubs are link layer devices
  - Only care about frames, don’t modify IP datagram
  - Transparent to network

Hubs

- Operate as a repeater
  - Broadcast an incoming frame to all ports, except for the ingress port
  - Like having a longer Ethernet cable that all the hosts tap into
  - All ports are on single collision domain!
- Advantages: simple, restores signal, potentially fast since we don't have to buffer or examine frame
- Disadvantages: poor bandwidth due to collisions

Hub Question 1

- A 10-port hub is connected to 10 hosts using gigabit links. What is the maximum aggregate transfer rate of data flowing through this network?
  - All ports are part of the same collision domain--only one device can send at a time
  - Therefore, peak bandwidth is one gigabit

Hub Question 2

- Recall that 100Mbps Ethernet restricts cable lengths to 100m. Suppose we want to connect two hosts which are 1000m apart. Can we use 10 100m cables with 9 hubs in series to accomplish this?
  - No. Since all ports are on same collision domain, max network diameter (1km) is too large to meet the TRANSP > 2 * PROP constraint of CSMA/CD
  - In reality, the IEEE standard limits number of hubs in series and specifies maximum network diameter

Switches

- Must store and examine frame before forwarding
- Simple learning protocol—no configuration
  - Given incoming frame (MAC_{src}, MAC_{dst}) on port x:
  - Add (MAC_{src}, x) to switch table
  - Look up port for MAC_{dst} for in switch table
    - If entry is there, forward frame to that port
    - Else, broadcast frame to all ports (except ingress port)
  - Runs spanning tree protocol to prevent loops
- Collision domain is a single port—switch will make sure that the frame it sends out does not collide with another frame being sent on the same link

Memory Mapped Registers

- Can send a command to hardware through memory-mapped registers
- These addresses aren't mapped to physical memory, but rather to the HW device
  - Store to location writes to HW device over bus
  - Load from location reads from HW device over bus

Direct Memory Access

- With memory-mapped registers, CPU is constantly waiting on I/O operations to complete
  - I/O devices are usually slower than processors
  - Blocks processor
- Instead, better to have device write directly to physical memory
- This can be done through DMA
  - CPU pokes device, goes on to do something else
  - Device writes directly to memory
  - At some other time (like an interrupt), CPU goes back to fetch data that device wrote to memory

Polling vs. Interrupts

- Polling: CPU periodically checks for data from device
  - Inefficient if there's no data for CPU to process, and higher latency
  - Efficient if lots of data to process (batches context switches)
- Interrupts: Device interrupts CPU's current execution, making it jump to an interrupt handler
  - Efficient if it doesn't happen much, and lower latency since we don't wait for a polling timer to timeout
  - Can lead to livelock if too many interrupts (e.g. in network cards that process many packets)

Lab 5: Dynamic Routing

- Implement RIP (distance-vector) in Java
- You are given a router that already works, except it doesn't process RIP packets
  - Write RIP packet handling code that modifies routing table
  - Simulates the network topology locally (doesn't route real packets)
- References
  - Assignment webpage
  - Textbook: DV (4.5.2) and RIP (4.6.1)
  - RIP Version 2 RFC2453 (although you don't have to implement all the features)

Distance Vector Refresher

- Distributed, local routing algorithm
- Exchange routing tables with neighbors
- If we find a better cost path to any destination through our neighbors, update the cost

Pseudocode

- Define:
  - $D_x(y)$ = cost from x to y (x's routing table)
  - c(x,y) = link cost of edge (x,y)

```
while True:
    wait until we get a routing table from neighbor v

    for each new routing table entry received from v that goes to dest:
      entry = lookupRoutingTable(dest)
      if c(x,v) + D_v(dest) <= entry.curCost or entry.nextHop is v:
        entry.nextHop = v
        entry.cost = c(x,v) + D_v(dest)

    send routing table to all neighbors if D_x changed
```

- Also have to handle updates if a local link metric is updated or goes up/down

Basic Code Overview

- acceptPacket gets called with a new RIP update packet
  - RIPRoutingUpdate contains an ArrayList of RIPRoutingUpdate.Entry (get the ArrayList through getAllEntries() method)
  - Get cost to neighbor through getLocalLinkInfoForNextHop
- Send a packet using m_ports[PORT_UPDATE_OUT].pushOut(packet)
  - Packet is of type IPPacket
  - Build by creating a new RoutingTableUpdate, and calling addEntry to it
- notifyAlarm() gets called periodically, use it to do periodic tasks
  - Decrement TTL of routes learned from others
  - Send updates if necessary (if timeout interval has passed, or a route timed out)
- Get familiar with Java functions for manipulating IPs, applying netmasks
  - Netutils.applyNetMask
  - InetAddress class
- Use Clack JavaDoc from assignment webpage for reference

Additional Cases

- New network advertised
  - Add to routing table
- Ignore updates to one of your own prefixes if you know they're down already
  - Otherwise you could end up thinking you have a route to yourself that you know is already down!
- Local link metric has updated or local link has gone down
- Split horizon with poison reverse
  - If I learned a route from you as next hop, I send you infinite as the route cost in the update to you