CS4450

Computer Networks:
Architecture and Protocols

Lecture 9
Why Network Layer?
Fundamentals of Routing

Rachit Agarwal
Announcements

• I am back!
  • All emails answered, caught up on sleep, ready to roll!

• Live coding session on the 27th. Please bring your laptops.

• Prelim will be "in class" on 03/26. No make up.

• Problem Set 2 solutions are posted (Piazza)

• Project 1 posted (course webpage)

• Emails: Please please please
  • cc your TAs (Katie, Qizhe)
Goals for Today’s Lecture

• We are at a stage where you have a strong foundation in networks
  • Sharing networks, architectural principles, design goals
  • And, you may not realize, but you understand many new tradeoffs

• Why do we need network layer?
  • Why not just use switched Ethernet across the Internet?

• What do routing tables look like?
  • The right way to think about routing tables....

• Correctness definition for routing tables

• Our first network layer protocol: Link State
Recap of Link Layer so far
Recap: Link layer

• Traditional Link Layer: Broadcast Ethernet

• CSMA/CD
  • Random access on a broadcast channel
  • Exponential Backoff

• Why Frames?
  • To incorporate sentinel bits for identifying frame start/end
  • To incorporate link layer source and destination names
  • To incorporate CRC for checking correctness of received frames

• Modern Link Layer: Switched Ethernet
  • Why? Scalability limits of traditional Ethernet
    • Why? Detecting collisions on a broadcast channel
Recap: Spanning Tree definition

- Subgraph that includes all vertices but contains no cycles
  - Links not in the spanning tree are not used in forwarding frames
Recap: Spanning Tree Protocol

• Messages \((Y,d,X)\)
  • Proposing root \(Y\); from node \(X\); advertising a distance \(d\) to \(Y\)

• Initially each switch proposes itself as the root
  • that is, switch \(X\) announces \((X,0,X)\) to its neighbors

• At each switch \(Z\):
  WHENEVER a message \((Y,d,X)\) is received from \(X\):
    • IF \(Y\)'s id < current root
      • THEN set root = \(Y\); next-hop = \(X\)
    • IF Shortest distance to root > \(d + \) distance_from_X
      • THEN set shortest-distance-to-root = \(d + \) distance_from_X
    • IF root changed OR shortest distance to the root changed:
      • Send all neighbors message \((Y, \) shortest-distance-to-root, \(Z)\)
The end of Link Layer ....

And the beginning of network layer :-D
Questions?
Why do we need a network layer?

- Why not just use spanning trees across the entire network?
- Easy to design routing algorithms for (spanning) trees
  
  - **Step 1**: Source node “floods” its packet on its spanning tree links
  - **Step 2**: Whenever a node receives a packet:
    - Forwards incoming packet out to all links other than the one that sent the packet
Flooding on a Spanning Tree

- Sends packet to **every** node in the network

- **Step 1:** Ignore the links not belonging to the Spanning Tree

- **Step 2:** Source sends “flood” packet out every link (on spanning tree)

- **Step 3:** Send incoming packet out to all links **other than the one that sent the packet**
Flooding Example

Source

Destination

1

3

4

2

5

6

7
Flooding Example

Eventually all nodes are covered

Source

Destination

One copy of packet delivered to destination
Routing via Flooding on Spanning Tree ...

• Easy to design routing algorithms for (spanning) trees
  
  • **Step 1**: Source node “floods” its packet on its spanning tree links
  
  • **Step 2**: Whenever a node receives a packet:
    • Forwards incoming packet out to all links other than the one that sent the packet
  
• Amazing properties:
  
  • No routing tables needed!
  
  • No packets will ever loop.
  
  • At least (and exactly) one packet must reach the destination
    • Assuming no failures
Three fundamental issues!

Issue 1: Each host has to do unnecessary packet processing! (to decide whether the packet is destined to the host)
Three fundamental issues!

Issue 2: Higher latency!
(The packets unnecessarily traverse much longer paths)
Three fundamental issues!

Issue 3: Wasted bandwidth!
(2-6 and 3-1 packets unnecessarily have to share bandwidth)
Questions?
Why do we need a network layer?

• Network layer performs “routing” of packets to alleviate these issues

• Uses routing tables

• Lets understand routing tables first
Routing Packets via Routing Tables

- Routing tables allow finding path from source to destination

What path will a packet take from Cornell to MIT?
Routing Packets via Routing Tables

- Finding path for a packet from source to destination

How to specify whether the packet should take Path 1 or Path 2?
Each Switch stores a table indicating the next hop for corresponding destination of a packet (called a routing table).

- **Routing Table**

  - Suppose packet follows **Path 1: Cornell - S#1 - S#3 - MIT**

<table>
<thead>
<tr>
<th>DESTINATION</th>
<th>NEXT HOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORNELL</td>
<td>L1</td>
</tr>
<tr>
<td>MIT</td>
<td>L3</td>
</tr>
<tr>
<td>HARVARD</td>
<td>L4</td>
</tr>
</tbody>
</table>

**Switch #1**

- L1 to Switch #2
- L2 to Switch #3

<table>
<thead>
<tr>
<th>DESTINATION</th>
<th>NEXT HOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORNELL</td>
<td>L2</td>
</tr>
<tr>
<td>MIT</td>
<td>L5</td>
</tr>
<tr>
<td>HARVARD</td>
<td>L5</td>
</tr>
</tbody>
</table>

**Switch #2**

- L3 to Switch #3

<table>
<thead>
<tr>
<th>DESTINATION</th>
<th>NEXT HOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORNELL</td>
<td>L5</td>
</tr>
<tr>
<td>MIT</td>
<td>L6</td>
</tr>
<tr>
<td>HARVARD</td>
<td>L3</td>
</tr>
</tbody>
</table>

**Switch #3**

- L4 to Switch #1
- L5 to Switch #2
- L6 to MIT

**Harvard**

- L4 from Harvard
- L5 from Switch #3
- L6 from Switch #3

**MIT**

- L5 from Switch #2
Routing Table: The right way to think about them

Let's focus on one destination - MIT

<table>
<thead>
<tr>
<th>DESTINATION</th>
<th>NEXT HOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORNELL</td>
<td>L1</td>
</tr>
<tr>
<td>MIT</td>
<td>L3</td>
</tr>
<tr>
<td>HARVARD</td>
<td>L4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DESTINATION</th>
<th>NEXT HOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORNELL</td>
<td>L2</td>
</tr>
<tr>
<td>MIT</td>
<td>L5</td>
</tr>
<tr>
<td>HARVARD</td>
<td>L5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DESTINATION</th>
<th>NEXT HOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORNELL</td>
<td>L5</td>
</tr>
<tr>
<td>MIT</td>
<td>L6</td>
</tr>
<tr>
<td>HARVARD</td>
<td>L3</td>
</tr>
</tbody>
</table>

See something interesting?
Routing Table: The right way to think about them

- Lets focus on one destination - MIT

Routing table entries for a particular destination form a (directed) spanning tree with that destination as the root!!!!
Routing Table: The right way to think about them

• Routing tables are nothing but ....
  • A collection of (directed) spanning tree
  • One for each destination

• Routing Protocols
  • “n” spanning tree protocols running in parallel
“Valid Routing Tables” (routing state)

• Global routing state is valid if:
  • it always results in deliver packets to their destinations

• Goal of Routing Protocols
  • Compute a valid state
  • But how to tell if a routing state is valid?...
  • Think about it, what could make routing incorrect?
Validity of a Routing State

• Global routing state valid if and only if:
  • There are no dead ends (other than destination)
  • There are no loops

• A dead end is when there is no outgoing link
  • A packet arrives, but ..
    • the routing table does not have an outgoing link
    • And that node is not the destination

• A loop is when a packet cycles around the same set of nodes forever
Example: Routing with Dead Ends

• Suppose packet wants to go from Cornell to MIT using given state:

---

<table>
<thead>
<tr>
<th>DESTINATION</th>
<th>NEXT HOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORNELL</td>
<td>L1</td>
</tr>
<tr>
<td>MIT</td>
<td>L2</td>
</tr>
<tr>
<td>HARVARD</td>
<td>L4</td>
</tr>
</tbody>
</table>

---

Switch #1

Switch #2

Switch #3

Cornell

Harvard

MIT

<table>
<thead>
<tr>
<th>DESTINATION</th>
<th>NEXT HOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORNELL</td>
<td>L5</td>
</tr>
<tr>
<td>MIT</td>
<td>L6</td>
</tr>
<tr>
<td>HARVARD</td>
<td>L3</td>
</tr>
</tbody>
</table>

---

Dead End!
Packet never reaches MIT

No forwarding decision for MIT!
Example: Routing with Loops

• Suppose packet wants to go from Cornell to MIT using given state:

<table>
<thead>
<tr>
<th>DESTINATION</th>
<th>NEXT HOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORNELL</td>
<td>L1</td>
</tr>
<tr>
<td>MIT</td>
<td>L3</td>
</tr>
<tr>
<td>HARVARD</td>
<td>L4</td>
</tr>
</tbody>
</table>

Switch #1

Switch #2

Switch #3

- Loop!
- Packet never reaches MIT
Two Questions

• How can we verify given routing state is valid?

• How can we produce valid routing state?
Checking Validity of a Routing State

- Check validity of routing state for one destination at a time...

- For each node:
  - Mark the outgoing link with arrow for the required destination
  - There can only be one at each node

- Eliminate all links with no arrows

- Look what’s left. **State is valid if and only if**
  - Remaining graph is a spanning tree with destination as sink
  - Why is this true?
    - Tree -> No loops
    - Spanning (tree) -> No dead ends
Example 1
Example 1: Pick Destination
Example 1: Put Arrows on Outgoing Ports
Example 1: Remove unused Links

Leaves Spanning Tree: Valid
Example 2:
Example 2:

Is this valid?
Example 3:
Example 3:

Is this valid?
Checking Validity of a Routing State

• Simple to check validity of routing state for a particular destination

• Dead ends: nodes without arrows

• Loops: obvious, disconnected from destination and rest of the graph
Two Questions

• How can we **verify** given routing state is valid?

• How can we **produce** valid routing state?
Creating Valid Routing State

• Easy to avoid dead ends

• Avoiding loops is hard

• The key difference between routing protocols is how they avoid loops!
Four flavors of protocols

• Create Tree, route on tree
  • E.g., Spanning tree protocol (switched Ethernet)
  • **Good:** easy, no (persistent) loops, no dead ends
  • **Not-so-good:** unnecessary processing, high latency, low bandwidth

• Obtain a global view:
  • E.g., Link state

• Distributed route computation:
  • E.g., Distance vector
  • E.g., Border Gateway Protocol
Routing Metrics

• Routing goals: compute paths with minimum $X$
  • $X =$ number of “hops” (nodes in the middle)
  • $X =$ latency
  • $X =$ weight
  • $X =$ failure probability
  • …

• Generally assume every link has “cost” associated with it

• We want to minimize the cost of the entire path
  • We will focus on a subset of properties $X$, where:
  • Cost of a path = sum of costs of individual links/nodes on the path
  • E.g., number of hops and latency
#1: Create a Tree
#1: Create Tree Out of Topology

- Remove enough links to create a tree containing all nodes

- Sounds familiar? Spanning trees!

- If the topology has no loops, then just make sure not sending packets back from where they came
  - That causes an immediate loop

- Therefore, if no loops in topology and no formation of immediate loops ensures valid routing

- However... three challenges
  - Unnecessary host resources used to process packets
  - High latency
  - Low bandwidth (utilization)
#2: Global view
Two Aspects of Global View Method

• **Protocol**: What we focus on today
  - Where to create global view
  - How to create global view
  - Disseminating route computation (if necessary)
  - When to run route computation

• **Algorithm**: computing loop-free paths on graph
  - Straightforward to compute lowest cost paths
    - Using Dijkstra’s algorithm (please study; algorithms course)
  - We won’t spend time on this
**Where to create global view?**

- One option: Central server
  - Collects a global view
  - Computes the routing table for each node
  - “Installs” routing tables at each node
  - **Software-defined Networks: later in course**

- Second option: At each router
  - Each router collects a global view
  - Computes its own routing table using Link-state protocol

**Link-state routing protocol**
- OSPF is a specific implementation of link-state protocol
  - IETF RFC 2328 (IPv4) or 5340 (IPv6)
Overview of Link-State Routing

• Every router knows its local “link state”
  • Knows state of links to neighbors
  • Up/down, and associated cost

• A router floods its link state to all other routers
  • Uses a special packet — Link State Announcements (LSA)
  • Announcement is delivered to all nodes (next slide)
  • Hence, every router learns the entire network graph

• Runs route computation locally
  • Computing least cost paths from them to all other nodes
  • E.g., using Dijkstra’s algorithm
How does Flooding Work?

• “Link state announcement” (LSA) arrives on a link at a router

• That router:
  • Remembers the packet
  • Forwards the packet out all other links
  • Does not send it out the incoming link
  • Why?

• If a previously received announcement arrives again...
  • Router drops it (no need to forward again)
Link-State Routing
Each Node Then has a Global View
When to Initiate Flooding of announcements?

• Topology change
  • Link failures
  • Link recovery

• Configuration change
  • Link cost change (why would one change link cost?)

• Periodically
  • Refresh the link-state information
  • Typically (say) 30 minutes
  • Corrects for possible corruption of data
Making Floods Reliable

• Reliable Flooding
  • Ensure all nodes receive same link state announcements
    • No announcements dropped
  • Ensure all nodes use the latest version

• Suppose we can implement reliable flooding. How can it still fail?

• Can you ever have loops with link-state routing?

• Again: Can you ever have loops with link-state routing?
Fundamental Challenge with obtaining a Global View

• A global view of the network makes computing paths without loops easy
  • Many graph algorithms for computing loop-free paths
    • For e.g., Dijkstra’s Algorithm

• But, but, but ..... 

• Getting an accurate global view of network is challenging!
  • Especially in a “timely” manner
  • As the network is changing
Are Loops Still Possible?

A and D think this is the path to C
E-C link fails, but D doesn’t know yet

E thinks that this the path to C
E reaches C via D, D reaches C via E
Loop!
Inconsistent link-state views
- Some routers know about failure before others
- The shortest paths are no longer consistent
- Can cause transient forwarding loops
  - Transient loops are still a problem!
Convergence

- All routers have consistent routing information
  - E.g., all nodes having the same link-state database

- Forwarding is consistent after convergence
  - All nodes have the same link-state database
  - All nodes forward packets on same paths

- But while still converging, bad things can happen
Time to Reach Convergence

• Sources of convergence delay?
  • Time to detect failure
  • Time to flood link-state information (~longest RTT)
  • Time to recompute forwarding tables

• Performance problems during convergence period?
  • Dead ends
  • Looping packets
  • And some more we’ll see later ....
Link State is Conceptually Simple

- Everyone floods links information
- Everyone then knows graph of the network
- Everyone independently computes paths on the graph
- All the complexity is in the details
#3: Distributed Route Computation
#3: Distributed Route Computation

- Distributed algorithms to compute feasible route

- **Approach A**: Finding optimal route for maximizing/minimizing a metric

- **Approach B**: Finding feasible route via exchanging paths among switches
Welcome to the Network Layer!

• THE functionality: **delivering the data**

• **THE protocol: Internet Protocol (IP)**
  • To achieve its functionality (delivering the data), IP protocol has **three** responsibilities

• Addressing (next lecture)

• Encapsulating data into packets (next lecture)

• Routing (using a variety of protocols; several lectures)
Next lecture!