The Network Layer
Forwarding, Routing, and Addressing
(Part I)

Lecture 22
6.082 Spring 2007
April 30, 2007

Hari Balakrishnan

Layering in the Internet

End-to-End Layer
Everything else!
Reliability, integrity,
ordering, jitter ctrl,
congestion response, ...

Forwarding & routing
(and addressing)

HTTP, FTP, SMTP, ...
TCP, UDP
Ethernet, WiFi, ...

Today
IP

Application
Transport
Network
Data Link
Physical

Framing, coding, [limited]
rxmits, channel access
Modulation/demodulation
Network Layer Functions

- **Main goal**: ensure best-effort end-to-end connectivity
- **Addressing**: How to name nodes?
- **Forwarding**: What should each switch do to each packet?
- **Routing**: How to build routing tables to ensure that forwarding is correct?

Forwarding: It’s All in the Header

- Packet header is crucial: switches look at this info to decide what to do
  - E.g., to send a letter via USPS, the “To” (and “From”) fields are crucial
- Header depends on network layer protocol
  - Generally has **dst address**, **src address**, **demux info** (which higher layer should get packet)
  - Also usually packet length, hop limit (or “time to live”), **type-of-service** (e.g., priority level), **checksum**, ...
**IPv4 Header**

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version (ip_v)</td>
<td>0-2</td>
<td>Major version number</td>
</tr>
<tr>
<td>IP Header Length (IHL)</td>
<td>3-4</td>
<td>Length of IP header in 32-bit words</td>
</tr>
<tr>
<td>Type of Service (TOS)</td>
<td>5-7</td>
<td>How service is handled</td>
</tr>
<tr>
<td>Total Length (ip_len)</td>
<td>8-15</td>
<td>Total length of header and payload</td>
</tr>
<tr>
<td>Identification (ip_id)</td>
<td>16-23</td>
<td>16-bit value identifying fragment</td>
</tr>
<tr>
<td>Flags (ip(flags))</td>
<td>24-25</td>
<td>Flags to control fragments</td>
</tr>
<tr>
<td>Fragment Offset (ip_off)</td>
<td>26-29</td>
<td>Offset within the datagram for fragments</td>
</tr>
<tr>
<td>Time to Live (ip_ttl)</td>
<td>30-31</td>
<td>How long to live before discarding</td>
</tr>
<tr>
<td>Protocol (ip_proto)</td>
<td>32-34</td>
<td>Protocol number</td>
</tr>
<tr>
<td>Header Checksum (ip_sum)</td>
<td>35-39</td>
<td>Checksum for header</td>
</tr>
<tr>
<td>Source Address (ip_src)</td>
<td>40-43</td>
<td>Source node address</td>
</tr>
<tr>
<td>Destination Address (ip_dst)</td>
<td>44-47</td>
<td>Destination node address</td>
</tr>
<tr>
<td>Options</td>
<td>48-60</td>
<td>Options that can be carried in the header</td>
</tr>
<tr>
<td>Payload</td>
<td>61-64</td>
<td>Data payload</td>
</tr>
</tbody>
</table>

**Addressing**

- Addressing sounds trivial and unimportant, but it isn’t
- A good addressing scheme helps scalability
- Influences mobility (Can nodes move a little? A lot?)
- Affects, and is affected by, administration and network management
- For now, assume each node has a unique arbitrary address (“just a name”) with no structure
  - We’ll revisit when we discuss network scalability
Forwarding

- Core function is conceptually simple
  - `lookup(dst_addr)` in routing table returns next hop (i.e., outgoing link) for packet
  - `send(packet)` along outgoing link
- And do some book-keeping
  - When packet arrives, enqueue
    - May pick one of several queues if switch implements more than standard best-effort
  - Decrement hop limit field; if 0, discard packet
  - Recalculate checksum (in IP, header checksum)

(IP) Forwarding is Simple, but (IP) Routers are Extremely Complex

- Speed, speed, speed
- Feature creep
  - Security, multicast, quality of service, tunnels, ...  
- The “control path” (routing protocols, etc.) aren’t simple, as we’ll see...
  - Routing implemented in software even at the highest-speed IP routers
  - Complex and changing requirements
  - Need heavy configurability and control
- We’ll say more about the high points of IP router design next lecture
- Let’s now turn to `routing`
Routing Desiderata
(A minimal set: There are more...)

- Fault-tolerance: Links and nodes may fail
- Loop-freedom: Find paths that don’t cause loops when there are no failures
- Convergence: When network conditions change, protocol must eventually produce loop-free paths
- Correctness: In addition to loop-freedom, want:
  - Route validity: Any route produced must correspond to a valid path (e.g., no “blackholes”, loops, etc.)
  - Path visibility: If there’s some valid and usable path to a destination, protocol must find it
- Good paths: Pick paths with decent performance
- ... 

Why is Routing Hard?

- Inherently distributed problem
  - Information about links and neighbors is local to each node, but we want global reach
- Handling dynamic conditions difficult
  - Failures and recovery could be arbitrarily timed, messages could be lost, etc.
  - Mobility makes life even harder
- Scaling to large sizes difficult
- And on the Internet, routing policy makes it hard to even define what’s “correct routing”
  - A topic for later classes
Let’s Start Small

• Simple model: Consider a network of switches (nodes) connected by links
  • Assume network isn’t too big
  • Links and nodes may fail and recover arbitrarily, but failures are “fail stop”
    • I.e., if link/node fails, other directly connected nodes can discover that
• Model network as a graph, $G$
  • $G$ has a set of nodes (vertices), $V$
  • $G$ has a set of links (edges), $E$
  • Assume that each link $l$ has a cost, $C_l$

Shortest Path Routing on a Graph

• Each node wants to find the path with minimum aggregate cost to any other node in the graph
• Several possible approaches
  • Link-state protocols (today)
  • Vector protocols (next lecture)
Link-State Routing

- Conceptually, a two-step process
- Disseminate information about all the nodes and all the links to every other node
  - This process disseminates the graph
- Each node then runs the same local shortest path computation over the graph
  - If each node implements computation correctly and each node has the same graph, then resulting forwarding will work correctly
  - Each node makes local decision about next hop (rather than pick entire path, aka source routing)

Second Step First: Shortest Paths

- Many algorithms: We’ll study one due to Dijkstra

```
A --- 6 --- C ------ 2 --- F
|            |            |
6            4            8
B ------ 7 ------ D ------ 2
|                  |
5                  1
|                  |
B ------ 7 ------ D ------ 2
|                  |
```

- Key property of shortest paths:
  
  Suppose shortest path from X to Y goes through Z. Then, the path from X to Z must be a shortest path. [Why?]
Greedy Path Computation

- Find paths to all other nodes in order of non-decreasing path cost
  - Start with link with smallest cost (clearly no better way to get to that neighbor)
  - Maintain `nodes_done` list (nodes for which we’ve found shortest path) and `spcost(v)`, shortest path cost to node v
- Consider all nodes z not in `nodes_done` that are connected to some node y in `nodes_done`
  - Find $z^* = \text{MIN} (\text{over } z) \text{ of } spcost(y) + \text{cost}(y, z)$
  - Add $z^*$ to `nodes_done` and set `spcost(z*)`
- Continue until all nodes are in `nodes_done`

Dijkstra’s Algorithm Example

![Diagram of a graph with nodes A, B, C, D, E, F, G and edges with costs labeled: A to B (6), A to C (6), B to D (5), C to D (7), C to E (4), D to E (1), E to F (2), E to G (2), F to G (8). The path from A to G is highlighted with costs: A to C (6), C to E (4), E to G (2), total cost (12).]
First Step: Disseminating *Link-State*

- Each node sends periodic HELLO to all its neighbors
- Each node periodically broadcasts *Link-State Advertisements* (LSA)
  - If seq > last_heard, incorporate information and rebroadcast LSA to your neighbors
  - LSAs aren’t reliable messages, so periodic
- Periodic messages help handle dynamism: state in each node is “soft” and times out if not refreshed

![Graph Diagram]

Lab 10: Link-State Protocol Simulation

- Periodic HELLO and LSA messages
- Integrate LSA to form graph
  - Requires a “breadth first” traversal of graph
  - Go through all LSAs to count number of nodes
- Dijkstra’s shortest path computation
Summary

• The network layer implements the “glue” that achieves connectivity

• Three functions: addressing, forwarding, and routing

• Link-state protocols disseminate topology in distributed way
  • Each node runs identical (centralized) shortest path computation to produce a “shortest path tree” to all other destinations
  • Store results in routing table
  • When forwarding, consult routing table