Network Layer

Tyler Bletsch
Duke University

Slides are adapted from Brian Rogers (Duke)
TCP/IP Model

<table>
<thead>
<tr>
<th>OSI</th>
<th>TCP/IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Application</td>
</tr>
<tr>
<td>6</td>
<td>Presentation</td>
</tr>
<tr>
<td>5</td>
<td>Session</td>
</tr>
<tr>
<td>4</td>
<td>Transport</td>
</tr>
<tr>
<td>3</td>
<td>Network</td>
</tr>
<tr>
<td>2</td>
<td>Data link</td>
</tr>
<tr>
<td>1</td>
<td>Physical</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>TCP/IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td></td>
</tr>
<tr>
<td>Transport</td>
<td>Internet</td>
</tr>
<tr>
<td>Host-to-network</td>
<td></td>
</tr>
</tbody>
</table>

Not present in the model
Networking Layer

• Networking Layer’s job:
  – Get packets from source all the way to destination
  – In contrast to data link layer:
    • Move frames from one end of wire to the other

• Network Layer must:
  – Be aware of topology of communication subnet
    • i.e. the set of routers
  – Choose appropriate path through subnet
    • Find good path for communication
    • Avoid overloading some communication paths and routers
  – Deal with problems when src and dest are in different networks

• Example:
  – Internet and its IP network protocol layer
Host sends packet to nearest router
  - On its own LAN or over a point-to-point link to carrier
Packet stored at router until fully received
  - Checksum is verified
Packet forwarded onto next router along path
Network Layer Service

• Provides services to transport layer with these goals:
  – Services independent of router technology
  – Hides number, type, topology of routers
  – Network addresses provided to transport layer use uniform numbering scheme, even across LANs, WANs

• Historically, 2 ideas for network layer
  – Connectionless service
    • E.g. Internet
    • Makes sense with subnet is inherently unreliable
    • Higher layers should do error control and flow control
  – Connection-oriented service
    • E.g. ATM, derived from reliable telephone network system
    • Quality of service is a dominant factor
Connectionless Service

Initially

<table>
<thead>
<tr>
<th>A's Table</th>
<th>Later</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>B</td>
</tr>
<tr>
<td>E</td>
<td>C</td>
</tr>
<tr>
<td>F</td>
<td>C</td>
</tr>
</tbody>
</table>

C's Table

<table>
<thead>
<tr>
<th>A's Table</th>
<th>Later</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

E's Table

<table>
<thead>
<tr>
<th>A's Table</th>
<th>Later</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>E</td>
<td>-</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

- Packets injected & routed individually in subnet
- No advance setup
- Packets sometimes called datagrams here
  - Subnet = datagram subnet
- Routers have tables which tell them destination links
- Here we have 4 packets sent from H1 to H2
  - Packets 1-3 forwarded along A->C->E->F
  - Router A changed route table before packet 4 arrived
- Based on **routing algorithm**
Connection-Oriented Service

- Each connection between hosts (virtual circuit) stores an entry in router tables
  - Same route used for all traffic flowing along that connection
- Example, H1 and then H3 establish connection to H2
- Each packet carries an ID of the connection it belongs to
  - In example above, allows router C to distinguish packets in H3->H2 channel from H1->H2

<table>
<thead>
<tr>
<th>A’s Table</th>
<th>C’s Table</th>
<th>E’s Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1 1 C 1</td>
<td>A 1 E 1</td>
<td>C 1 F 1</td>
</tr>
<tr>
<td>H3 1 C 2</td>
<td>A 2 E 2</td>
<td>C 2 F 2</td>
</tr>
</tbody>
</table>
## Qualitative Comparison

<table>
<thead>
<tr>
<th>Issue</th>
<th>Datagram Subnet</th>
<th>Virtual-Circuit Subnet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circuit Setup</td>
<td>Not needed</td>
<td>Required</td>
</tr>
<tr>
<td>Addressing</td>
<td>Each packet contains the full src and dest address</td>
<td>Each packet contains a short VC number</td>
</tr>
<tr>
<td>State information</td>
<td>Routers do not need state info about connections</td>
<td>Each VC requires router table space per connection</td>
</tr>
<tr>
<td>Routing</td>
<td>Each packet is routed independently</td>
<td>Route chosen when VC is set up; all packets follow it</td>
</tr>
<tr>
<td>Effect of router failure</td>
<td>None, except for packets lost during the crash</td>
<td>All VCs that passed through the failed router are terminated</td>
</tr>
<tr>
<td>Quality of service</td>
<td>Difficult</td>
<td>Easy if enough resources can be allocated in advance for each VC</td>
</tr>
<tr>
<td>Congestion control</td>
<td>Difficult</td>
<td>Easy if enough resources can be allocated in advance for each VC</td>
</tr>
</tbody>
</table>
Routing

• Routing algorithm:
  – Determine which output line an incoming packet goes out on
  – May be a new decision for every incoming packet

• Routing vs. Forwarding
  – Forwarding: lookup outgoing line for a packet in routing table
  – Routing: Initialize and update routing tables

• Desirable routing algorithm properties:
  – Correctness, simplicity, robustness, stability, fairness, optimality
  – Some properties trade-off, e.g. fairness vs. optimality

• Adaptive vs. Non-adaptive routing:
  – Adaptive routing changes to reflect topology, current traffic, etc.
  – Also referred to as dynamic vs. static
Inter vs. Intra-domain routing

- Routing organized in two levels
- Intra-domain routing
  - Complete knowledge, strive for *optimal* paths
  - Scale to ~100 networks
- Inter-domain routing
  - Aggregated knowledge, scale to Internet
  - Dominated by *policy*
    - E.g., route through X, unless X is unavailable, then route through Y. Never route traffic from X to Y.
  - Policies reflect business agreements, can get complex
Optimality Principle

• General statement about optimal routes

• If router J is on the optimal path from router I to K, then:
  – Optimal path from J to K also falls along the same route

• Set of all optimal routes from all sources to a given destination forms a sink tree
  – With a root at the destination

• No loops, so each packet delivery is bounded in # hops

• Useful for comparison against other routing algorithms
Static Routing

• Shortest path routing
  – Build graph of subnet; routers are nodes, links are edges
  – Find shortest path between each pair of routers
  – What is “shortest”?
    • # hops vs. distance vs. queueing delay vs. transmission delay
    • Or some function that takes several factors into account
  – Use Dijkstra’s shortest path algorithm

• Flooding
  – Forward every incoming packet out on every other outgoing line
  – Some measures to reduce # of duplicate packets
    • Hop counter per packet; packet discarded on 0
    • Or selective flooding – only send out on lines going in right direction
  – Evaluates every path; and thus guaranteed to find shortest delay
Dynamic Routing

- Two classes of intra-domain routing algorithms
- Distance Vector (Bellman-Ford Shortest Path Algorithm)
  - Send best-known path info to neighbors; can figure out optimal routes over time
  - Requires only local state
  - Harder to debug
  - Can suffer from loops
  - Examples: RIP, BGP (Border Gateway Protocol: used between autonomous systems on the Internet)
- Link State (Dijkstra-Prim Shortest Path Algorithm)
  - Send adjacency info, each router builds map of network over time, then runs shortest-path algorithm locally
  - Each node has global view of the network
  - Simpler to debug
  - Requires global state, limits scalability
  - Example: OSPF (used within large networks, e.g. within companies)
Distance Vector

• Local routing algorithm
  – Also called **RIP** (Routing Information Protocol)

• Each node maintains a set of triples
  – \(<Destination, Cost, NextHop>\)

• Exchange updates *with neighbors*
  – Periodically (seconds to minutes)
  – Whenever table changes (*triggered* update)
Distance Vector Example

B only exchanges information with A and C
Distance Vector

- Local routing algorithm
- Each node maintains a set of triples
  - \(<\text{Destination}, \text{Cost}, \text{NextHop}>\)
- Exchange updates with neighbors
  - Periodically (seconds to minutes)
  - Whenever table changes (\textit{triggered} update)
- Each update is a list of pairs
  - \(<\text{Destination}, \text{Cost}>\)
- Update local table if receive a “better” route
  - Smaller cost
Nodes start with info of just direct neighbors; shares that info with those neighbors (e.g., B.)

Example: C talks to B

B’s routing table @ time = 0

<table>
<thead>
<tr>
<th>Destination</th>
<th>Cost</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>infinity</td>
<td>--</td>
</tr>
<tr>
<td>E</td>
<td>infinity</td>
<td>--</td>
</tr>
<tr>
<td>F</td>
<td>infinity</td>
<td>--</td>
</tr>
<tr>
<td>G</td>
<td>infinity</td>
<td>--</td>
</tr>
</tbody>
</table>
C sends info to B:
- B learns about D.
- B also learns another path to A, but it’s worse, so we ignore it.

B’s routing table @ time = 0

<table>
<thead>
<tr>
<th>Destination</th>
<th>Cost</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>E</td>
<td>infinity</td>
<td>--</td>
</tr>
<tr>
<td>F</td>
<td>infinity</td>
<td>--</td>
</tr>
<tr>
<td>G</td>
<td>infinity</td>
<td>--</td>
</tr>
</tbody>
</table>
Distance Vector

- Local routing algorithm
- Each node maintains a set of triples
  - \(<\text{Destination}, \text{Cost}, \text{NextHop}>\)
- Exchange updates with neighbors
  - Periodically (seconds to minutes)
  - Whenever table changes (\textit{triggered} update)
- Each update is a list of pairs
  - \(<\text{Destination}, \text{Cost}>\)
- Update local table if receive a “better” route
  - Smaller cost
- Refresh existing routes, delete if time out
Calculating Best Path

- Bellman-Ford equation:
  - $D_b(d)$ denote the current best distance from b to d
  - $C(b,c)$ denote the cost of a link from b to c
- Then $D_b(d) = \min(D_b(d), C(b,c) + D_c(d))$
- D is any additive metric
  - e.g., number of hops, queue length, delay

```
<table>
<thead>
<tr>
<th>Destination</th>
<th>Cost</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>infinity</td>
<td>--</td>
</tr>
<tr>
<td>E</td>
<td>infinity</td>
<td>--</td>
</tr>
<tr>
<td>F</td>
<td>infinity</td>
<td>--</td>
</tr>
<tr>
<td>G</td>
<td>infinity</td>
<td>--</td>
</tr>
</tbody>
</table>
```

$D_b(d) = \min(\text{infinity}, 1 + 1)$

$D_b(a) = \min(1, 1 + 1)$
Distance Vector Example

B’s routing table

<table>
<thead>
<tr>
<th>Destination</th>
<th>Cost</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td>C</td>
</tr>
</tbody>
</table>
Adapting to Failures

- F-G fails
• F-G fails
• F sets distance to G to infinity, propagates
Adapting to Failures

- F-G fails
- F sets distance to G to infinity, propagates
- A sets distance to G to infinity
Adapting to Failures

- F-G fails
- F sets distance to G to infinity, propagates
- A sets distance to G to infinity
- A receives periodic update from C with 2-hop path to G
Adapting to Failures

- F-G fails
- F sets distance to G to infinity, propagates
- A sets distance to G to infinity
- A receives periodic update from C with 2-hop path to G
- A sets distance to G to 3 and propagates
Adapting to Failures

- F-G fails
- F sets distance to G to infinity, propagates
- A sets distance to G to infinity
- A receives periodic update from C with 2-hop path to G
- A sets distance to G to 3 and propagates
- F sets distance to G to 4, through A
Count-to-Infinity Problem

- Link from A to E fails
- A advertises distance of infinity to E
- B and C advertise a distance of 2 to E
- B decides it can reach E in 3 hops through C
- A decides it can reach E in 4 hops through B
- C decides it can reach E in 5 hops through A, …
- When does this stop?
Good news travels fast
Assume A is down initially, comes up after some time
Good news travels at rate of 1 hop per exchange
Count-to-Infinity Problem

- Bad news travels slowly
- Assume A is up initially, goes down after some time
- No router ever has a value more than 1 higher than the minimum of all its neighbors
  - Gradually all work their way up to infinity; Need to bound infinity!
Avoiding Loops

- IP packet field prevents a packet from living forever
  - Does not *repair* a loop
- Simple approach: consider a small cost $n$ (e.g., 16) to be infinity
  - After $n$ rounds decide node is unavailable
  - But rounds can be long, this takes time
- Problem: distance vector based only on local information
Better Loop Avoidance

- **Split Horizon**
  - When sending updates to node A, don’t include routes you learned from A
  - Prevents B and C from sending cost 2 to A

- **Split Horizon with Poison Reverse**
  - Rather than not advertising routes learned from A, explicitly include cost of \( \infty \).
  - Faster to break out of loops, but increases advertisement sizes

- **But still…**
  - Split horizon/split horizon with poison reverse only help between two nodes
    - Can still get loop with three nodes involved
    - Might need to delay advertising routes after changes, but affects convergence time