Computer Networks

Lecture 2: Routing

Adrian Perrig
Network Security Group
ETH Zürich
Where we are in the Course

• More fun in the Network Layer!
  – We’ve covered packet forwarding
  – Now we’ll learn about routing

<table>
<thead>
<tr>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport</td>
</tr>
<tr>
<td>Network</td>
</tr>
<tr>
<td>Link</td>
</tr>
<tr>
<td>Physical</td>
</tr>
</tbody>
</table>
Routing versus Forwarding

- **Forwarding** is the process of sending a packet on its way
- **Routing** is the process of deciding in which direction to send traffic
Perspective on Bandwidth Allocation

- Routing allocates network bandwidth adapting to failures; other mechanisms used at other timescales

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Timescale / Adaptation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load-sensitive routing</td>
<td>Seconds / Traffic hotspots</td>
</tr>
<tr>
<td>Routing</td>
<td>Minutes / Equipment failures</td>
</tr>
<tr>
<td>Traffic Engineering</td>
<td>Hours / Network load</td>
</tr>
<tr>
<td>Network Provisioning</td>
<td>Months / Network customers</td>
</tr>
</tbody>
</table>
Delivery Models

• Different routing used for different delivery models

Unicast (§4.7)  Broadcast (§4.7.1)  Multicast (§4.7.2)  Anycast
Goals of Routing Algorithms

- We want several properties of any routing scheme:

<table>
<thead>
<tr>
<th>Property</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctness</td>
<td>Finds paths that work</td>
</tr>
<tr>
<td>Efficient paths</td>
<td>Uses network bandwidth well</td>
</tr>
<tr>
<td>Fair paths</td>
<td>Doesn’t starve any nodes</td>
</tr>
<tr>
<td>Fast convergence</td>
<td>Recovers quickly after changes</td>
</tr>
<tr>
<td>Scalability</td>
<td>Works well as network grows large</td>
</tr>
</tbody>
</table>
Rules of Routing Algorithms

- Decentralized, distributed setting
  - All nodes are alike; no controller
  - Nodes only know what they learn by exchanging messages with neighbors
  - Nodes operate concurrently
  - May be node/link/message failures
Topics

- IPv4, IPv6, NATs and all that
- Shortest path routing
- Distance Vector routing
- Flooding
- Link-state routing
- Equal-cost multi-path
- Inter-domain routing (BGP)
Shortest Path Routing (§5.2)

• Defining “best” paths with link costs
  – These are shortest path routes
What are “Best” paths anyhow?

• Many possibilities:
  – Latency, avoid circuitous paths
  – Bandwidth, avoid small pipes
  – Money, avoid expensive links
  – Hops, to reduce switching

• But only consider topology
  – Ignore workload, e.g., hotspots
Shortest Paths

We’ll approximate “best” by a cost function that captures the factors

– Often call lowest “shortest”

1. Assign each link a cost (distance)
2. Define best path between each pair of nodes as the path that has the lowest total cost (or is shortest)
3. Pick randomly to break any ties
Shortest Paths (2)

• Find the shortest path A ➔ E

• All links are bidirectional, with equal costs in each direction
  – Can extend model to unequal costs if needed
Shortest Paths (3)

• ABCE is a shortest path
• \( \text{dist(ABCE)} = 4 + 2 + 1 = 7 \)

• This is less than:
  – \( \text{dist(ABE)} = 8 \)
  – \( \text{dist(ABFE)} = 9 \)
  – \( \text{dist(AE)} = 10 \)
  – \( \text{dist(ABCDE)} = 10 \)
Shortest Paths (4)

- Optimality property:
  - Subpaths of shortest paths are also shortest paths
- ABCE is a shortest path
  So are ABC, AB, BCE, BC, CE
Sink Trees

• Sink tree for a destination is the union of all shortest paths towards the destination
  – Similarly source tree

• Find the sink tree for E
Sink Trees (2)

• Implications:
  – Only need to use destination to follow shortest paths
  – Each node only need to send to the next hop

• **Forwarding table** at a node
  – Lists next hop for each destination
  – Routing table may know more
Computing Shortest Paths with Dijkstra (§5.2.1)

- How to compute shortest path given the network topology
  - With Dijkstra’s algorithm

![Source tree for E](source_tree_image)
Edsger W. Dijkstra (1930-2002)

- Famous computer scientist
  - Programming languages
  - Distributed algorithms
  - Program verification

- Dijkstra’s algorithm, 1959
  - Single-source shortest paths, given network with non-negative link costs
Dijkstra’s Algorithm

Algorithm:

• Mark all nodes tentative, set distances from source to 0 (zero) for source, and $\infty$ (infinity) for all other nodes

• While tentative nodes remain:
  – Extract N, a node with lowest distance
  – Add link to N to the shortest path tree
  – Relax the distances of neighbors of N by lowering any better distance estimates
Dijkstra’s Algorithm (2)

• Initialization

We’ll compute shortest paths from A
Dijkstra’s Algorithm (3)

• Relax around A
Dijkstra’s Algorithm (4)

- Relax around B

Distance fell!
Dijkstra’s Algorithm (5)

- Relax around C

Distance fell again!
Dijkstra’s Algorithm (6)

- Relax around G (say)
Dijkstra’s Algorithm (7)

- Relax around F (say)

Relax has no effect
Dijkstra’s Algorithm (8)

- Relax around E
Dijkstra’s Algorithm (9)

• Relax around D
Dijkstra's Algorithm (10)

• Finally, H ... done
Dijkstra Comments

• Finds shortest paths in order of increasing distance from source
  – Leverages optimality property

• Runtime depends on efficiency of extracting min-cost node
  – Superlinear in network size (grows fast)

• Gives complete source/sink tree
  – More than needed for forwarding!
  – But requires complete topology
Distance Vector Routing (§5.2.2)

• How to compute shortest paths in a distributed network
  – The Distance Vector (DV) approach
Distance Vector Routing

• Simple, early routing approach
  – Used in ARPANET, and RIP (Routing Information Protocol)

• One of two main approaches to routing
  – Distributed version of Bellman-Ford
  – Works, but very slow convergence after some failures

• Link-state algorithms are now typically used in practice
  – More involved, better behavior
Distance Vector Setting

Each node computes its forwarding table in a distributed setting:

1. Nodes know only the cost to their neighbors; not the topology
2. Nodes communicate only with their neighbors using messages
3. All nodes run the same algorithm concurrently
4. Nodes and links may fail, messages may be lost
Distance Vector Algorithm

Each node maintains a vector of distances (and next hops) to all destinations

1. Initialize vector with 0 (zero) cost to self, \( \infty \) (infinity) to other destinations

2. Periodically send vector to neighbors

3. Update vector for each destination by selecting the shortest distance heard, after adding cost of neighbor link
   - Use the best neighbor for forwarding
Distance Vector Example

- Consider a simple network. Each node runs on its own
  - E.g., node A can only talk to nodes B and D
DV Example (2)

• First exchange, A hears from B, D and finds 1-hop routes
  – A always learns min(B+3, D+7)

<table>
<thead>
<tr>
<th>To</th>
<th>B says</th>
<th>D says</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>∞</td>
</tr>
<tr>
<td>C</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>D</td>
<td>∞</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B says</th>
<th>D says</th>
</tr>
</thead>
<tbody>
<tr>
<td>+3</td>
<td>+7</td>
</tr>
</tbody>
</table>

A learns
Cost Next

<table>
<thead>
<tr>
<th>Cost</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>∞</td>
<td>--</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
</tr>
<tr>
<td>∞</td>
<td>--</td>
</tr>
<tr>
<td>7</td>
<td>D</td>
</tr>
</tbody>
</table>

= learned better route
DV Example (3)

- First exchange for all nodes to find best 1-hop routes
  - E.g., B learns min(A+3, C+6, D+3)

<table>
<thead>
<tr>
<th></th>
<th>A says</th>
<th>B says</th>
<th>C says</th>
<th>D says</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>B</td>
<td>∞</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>C</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
<td>∞</td>
</tr>
<tr>
<td>D</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
</tr>
</tbody>
</table>

**A learns**

<table>
<thead>
<tr>
<th>To</th>
<th>Cost Next</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>--</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>∞</td>
<td>--</td>
</tr>
<tr>
<td>D</td>
<td>7</td>
<td>A</td>
</tr>
</tbody>
</table>

**B learns**

<table>
<thead>
<tr>
<th>To</th>
<th>Cost Next</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>--</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>7</td>
<td>B</td>
</tr>
</tbody>
</table>

**C learns**

<table>
<thead>
<tr>
<th>To</th>
<th>Cost Next</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>--</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>--</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>C</td>
</tr>
</tbody>
</table>

**D learns**

<table>
<thead>
<tr>
<th>To</th>
<th>Cost Next</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>7</td>
<td>D</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>--</td>
</tr>
</tbody>
</table>

- = learned better route
DV Example (4)

- Second exchange for all nodes to find best 2-hop routes

<table>
<thead>
<tr>
<th>To</th>
<th>A says</th>
<th>B says</th>
<th>C says</th>
<th>D says</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>3</td>
<td>∞</td>
<td>7</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>∞</td>
<td>6</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A learns Cost Next</th>
<th>B learns Cost Next</th>
<th>C learns Cost Next</th>
<th>D learns Cost Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -- 3 A</td>
<td>9 B 6 B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 B 0 -- 5 D 3 B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 D 5 D 0 -- 2 C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 B 3 D 2 D 0 --</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

= learned better route
DV Example (5)

- Third exchange for all nodes to find best 3-hop routes

<table>
<thead>
<tr>
<th>To</th>
<th>A says</th>
<th>B says</th>
<th>C says</th>
<th>D says</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>3</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>9</td>
<td>5</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

- A learns Cost Next: 0 -- 3
- B learns Cost Next: 3 B 0 -- 5
- C learns Cost Next: 8 B 5 D 0 -- 2
- D learns Cost Next: 6 B 3 D 2 D 0 --

= learned better route
DV Example (5)

- Fourth and subsequent exchanges; converged

<table>
<thead>
<tr>
<th>To</th>
<th>A says</th>
<th>B says</th>
<th>C says</th>
<th>D says</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>3</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>8</td>
<td>5</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>To</th>
<th>A learns Cost Next</th>
<th>B learns Cost Next</th>
<th>C learns Cost Next</th>
<th>D learns Cost Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0 --</td>
<td>3 A</td>
<td>8 D</td>
<td>6 B</td>
</tr>
<tr>
<td>B</td>
<td>3 B 0 --</td>
<td>5 D</td>
<td>3 B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>8 B 5 D 0 --</td>
<td>2 C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>6 B 3 D 2 D 0 --</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

= learned better route
Distance Vector Dynamics

• Adding routes:
  – News travels one hop per exchange

• Removing routes
  – When a node fails, no more exchanges, other nodes forget

• But partitions (unreachable nodes in divided network) are a problem
  – “Count to infinity” scenario
DV Dynamics (2)

- Good news travels quickly, bad news slowly (inferred)

Desired convergence

“Count to infinity” scenario
DV Dynamics (3)

• Various heuristics to address
  – e.g., “Split horizon, poisoned reverse”
    • Split horizon: omit routes learned from neighbor in updates sent to neighbor
    • Poisoned reverse: set metric = ∞ for routes learned from neighbor sent back to that neighbor

• But none are very effective
  – Can you come up with topology where split horizon / poisoned reverse does not work well?
  – Link state now favored in practice in intra-domain (LAN) settings
  – Except when very resource-limited
RIP (Routing Information Protocol)

• DV protocol with hop count as metric
  – Infinity is 16 hops; limits network size
  – Includes split horizon, poison reverse

• Routers send vectors every 30 secs
  – Runs on top of UDP
  – Timeout in 180 secs to detect failures

• RIPv1 specified in RFC1058 (1988)
Flooding

- How to broadcast a message to all nodes in the network with **flooding**
  - Simple mechanism, but inefficient
Flooding

• Rule used at each node:
  – Sends an incoming message on to all other neighbors
  – Remember the message so that it is only sent once over each link (called duplicate suppression)

• Inefficient because one node may receive multiple copies of message
Flooding (2)

- Consider a flood from A; first reaches B via AB, E via AE
Flooding (3)

• Next B floods BC, BE, BF, BG, and E floods EB, EC, ED, EF

F gets 2 copies

E and B send to each other
Flooding (4)

- C floods CD, CH; D floods DC; F floods FG; G floods GF

F gets another copy
Flooding (5)

- H has no-one to flood ... and we’re done

Each link carries the message, and in at least one direction
Flooding Details

• Remember message (to stop flood) using source and sequence number
  – Used for duplicate suppression, so same message is only sent once to neighbors
  – So subsequent message (with higher sequence number) will again be flooded

• To make flooding reliable, use stop-and-wait
  – So receiver acknowledges, and sender resends if needed
Link State Routing \((\Sect{5.2.1}, \Sect{5.3})\)

- How to compute shortest paths in a distributed network
  - The Link-State (LS) approach

\[
\text{Flood!} \quad \text{... then compute}
\]
Link-State Routing

• One of two approaches to routing
  – Trades more computation than distance vector for better dynamics

• Widely used in practice
  – Used in Internet/ARPANET from 1979
  – Modern networks use OSPF and IS-IS for intra-domain routing
Link-State Setting

Each node computes their forwarding table in the same distributed setting as distance vector:

1. Node knows only the cost to its neighbors; not the topology
2. Node can talk only to its neighbors using messages
3. Nodes run the same algorithm concurrently
4. Nodes/links may fail, messages may be lost
Link-State Algorithm

Proceeds in two phases:

1. Nodes flood topology in the form of link state packets
   - Each node learns full topology

2. Each node computes its own forwarding table
   - By running Dijkstra (or equivalent)
Phase 1: Topology Dissemination

- Each node floods link state packet (LSP) that describes their portion of the topology

Node E’s LSP flooded to A, B, C, D, and F

<table>
<thead>
<tr>
<th>Seq. #</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
</tr>
</tbody>
</table>
Phase 2: Route Computation

- Each node has full topology
  - By combining all LSPs

- Each node simply runs Dijkstra
  - Some replicated computation, but finds required routes directly
  - Compile forwarding table from sink/source tree
  - That’s it folks!
Forwarding Table

Source Tree for E (from Dijkstra)

E’s Forwarding Table

<table>
<thead>
<tr>
<th>To</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>C</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>
<tr>
<td>G</td>
<td>F</td>
</tr>
<tr>
<td>H</td>
<td>C</td>
</tr>
</tbody>
</table>
Handling Changes

• On change, flood updated LSPs, and re-compute routes
  – E.g., nodes adjacent to failed link or node initiate
Handling Changes (2)

• **Link failure**
  - Both nodes notice, send updated LSPs
  - Link is removed from topology

• **Node failure**
  - All neighbors notice a link has failed
  - Failed node can’t update its own LSP
  - But it is OK: all links to node removed
Handling Changes (3)

• Addition of a link or node
  – Add LSP of new node to topology
  – Old LSPs are updated with new link

• Additions are the easy case ...
Link-State Complications

• Things that can go wrong:
  – Seq. number reaches max, or is corrupted
  – Node crashes and loses seq. number
  – Network partitions then heals

• Strategy:
  – Include age on LSPs and forget old information that is not refreshed

• Much of the complexity is due to handling corner cases (as usual!)
## DV/LS Comparison

<table>
<thead>
<tr>
<th>Goal</th>
<th>Distance Vector</th>
<th>Link-State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctness</td>
<td>Distributed Bellman-Ford</td>
<td>Replicated Dijkstra</td>
</tr>
<tr>
<td>Efficient paths</td>
<td>Approx. with shortest paths</td>
<td>Approx. with shortest paths</td>
</tr>
<tr>
<td>Fair paths</td>
<td>Approx. with shortest paths</td>
<td>Approx. with shortest paths</td>
</tr>
<tr>
<td>Fast convergence</td>
<td>Slow – many exchanges</td>
<td>Fast – flood and compute</td>
</tr>
<tr>
<td>Scalability</td>
<td>Excellent: storage/compute</td>
<td>Moderate: storage/compute</td>
</tr>
</tbody>
</table>
IS-IS and OSPF Protocols

• Widely used in large enterprise and ISP networks
  – IS-IS = Intermediate System to Intermediate System
  – OSPF = Open Shortest Path First

• Link-state protocol with many added features
  – E.g., “Areas” for scalability
Equal-Cost Multi-Path Routing

• More on shortest path routes
  – Allow multiple shortest paths

Use ABE as well as ABCE from A→E
Multipath Routing

• Allow multiple routing paths from node to destination be used at once
  – Topology has them for redundancy
  – Using them can improve performance and reliability

• Questions:
  – How do we find multiple paths?
  – How do we send traffic along them?
Equal-Cost Multipath Routes

• One form of multipath routing
  – Extends shortest path model by keeping set if there are ties

• Consider A→E
  – ABE = 4 + 4 = 8
  – ABCE = 4 + 2 + 2 = 8
  – ABCDE = 4 + 2 + 1 + 1 = 8
  – Use them all!
Source “Trees”

- With ECMP, source/sink “tree” is a directed acyclic graph (DAG)
  - Each node has set of next hops
  - Still a compact representation
Source “Trees” (2)

• Find the source “tree” for E
  – Procedure is Dijkstra, simply remember set of next hops
  – Compile forwarding table similarly, may have set of next hops

• Straightforward to extend DV too
  – Just remember set of neighbors
Source “Trees” (3)

Source Tree for E

E’s Forwarding Table

<table>
<thead>
<tr>
<th>Node</th>
<th>Next hops</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B, C, D</td>
</tr>
<tr>
<td>B</td>
<td>B, C, D</td>
</tr>
<tr>
<td>C</td>
<td>C, D</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>
<tr>
<td>G</td>
<td>F</td>
</tr>
<tr>
<td>H</td>
<td>C, D</td>
</tr>
</tbody>
</table>
Forwarding with ECMP

• Could randomly pick a next hop for each packet based on destination
  – Balances load, but adds jitter

• Instead, try to send packets from a given source/destination pair on the same path
  – Source/destination pair is called a flow
  – Map flow identifier to single next hop
  – No jitter within flow, but less balanced
Multipath routes from F/E to C/H

E’s Forwarding Choices

<table>
<thead>
<tr>
<th>Flow</th>
<th>Possible next hops</th>
<th>Example choice</th>
</tr>
</thead>
<tbody>
<tr>
<td>F → H</td>
<td>C, D</td>
<td>D</td>
</tr>
<tr>
<td>F → C</td>
<td>C, D</td>
<td>D</td>
</tr>
<tr>
<td>E → H</td>
<td>C, D</td>
<td>C</td>
</tr>
<tr>
<td>E → C</td>
<td>C, D</td>
<td>C</td>
</tr>
</tbody>
</table>

Use both paths to get to one destination
Combining Hosts and Routers

• How routing protocols work with IP
  – The Host/Router distinction

- I route
- I don’t!
Recap from Lecture 1

• In the Internet:
  – Hosts on same network have IP addresses in the same IP prefix
    • Host performs ARP on IP address of destination
  – Hosts just send off-network traffic to the nearest router to handle
    • Host performs ARP on IP address of default router / gateway
  – Routers discover the routes to use
  – Routers use longest prefix matching to forward packets to next hop
Host/Router Combination

- Hosts attach to routers as IP prefixes
  - Router needs table to reach all hosts

**Single network (One IP prefix “P”)** → **IP router “A”** → **Rest of network**

LAN switch
Network Topology for Routing

- Group hosts under IP prefix connected directly to router
  - One entry for all hosts
Routing now works as before!

- Routers advertise IP prefixes for hosts
- Router addresses are “/32” prefixes
- Lets all routers find a path to hosts
- Hosts find by sending to their router
Hierarchical Routing

- How to scale routing with hierarchy in the form of regions
  - Route to regions, not individual nodes
Internet Growth

- 1.1 billion Internet hosts ...
- Likely to continue growth with mobile and IoT devices

Source: Internet Systems Consortium (www.isc.org)
Internet Routing Growth

- Internet growth translates into routing table growth (even using prefixes) ...

Source: By Mro (Own work), CC-BY-SA-3.0, via Wikimedia Commons
Impact of Routing Growth

1. Forwarding tables grow
   – Larger router memories, may increase lookup time

2. Routing messages grow
   – Need to keep all nodes informed of larger topology

3. Routing computation grows
   – Shortest path calculations grow faster than the size of the network
Techniques to Scale Routing

1. IP prefixes
   - Route to blocks of hosts

2. Network hierarchy
   - Route to network regions

3. IP prefix aggregation
   - Combine, and split, prefixes
Hierarchical Routing

• Introduce a larger routing unit
  – IP prefix (hosts) $\equiv$ from one host
  – Region, e.g., ISP network

• Route first to the region, then to the IP prefix within the region
  – Hide details within a region from outside of the region
Hierarchical Routing (2)

Hierarchical table for 1A

<table>
<thead>
<tr>
<th>Dest.</th>
<th>Line</th>
<th>Hops</th>
</tr>
</thead>
<tbody>
<tr>
<td>1A</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1B</td>
<td>1B</td>
<td>1</td>
</tr>
<tr>
<td>1C</td>
<td>1C</td>
<td>1</td>
</tr>
<tr>
<td>2A</td>
<td>1B</td>
<td>2</td>
</tr>
<tr>
<td>2B</td>
<td>1B</td>
<td>3</td>
</tr>
<tr>
<td>2C</td>
<td>1B</td>
<td>3</td>
</tr>
<tr>
<td>2D</td>
<td>1B</td>
<td>4</td>
</tr>
<tr>
<td>3A</td>
<td>1C</td>
<td>3</td>
</tr>
<tr>
<td>3B</td>
<td>1C</td>
<td>2</td>
</tr>
<tr>
<td>4A</td>
<td>1C</td>
<td>3</td>
</tr>
<tr>
<td>4B</td>
<td>1C</td>
<td>4</td>
</tr>
<tr>
<td>4C</td>
<td>1C</td>
<td>4</td>
</tr>
<tr>
<td>5A</td>
<td>1C</td>
<td>4</td>
</tr>
<tr>
<td>5B</td>
<td>1C</td>
<td>5</td>
</tr>
<tr>
<td>5C</td>
<td>1B</td>
<td>5</td>
</tr>
<tr>
<td>5D</td>
<td>1C</td>
<td>6</td>
</tr>
<tr>
<td>5E</td>
<td>1C</td>
<td>5</td>
</tr>
</tbody>
</table>

Full table for 1A

<table>
<thead>
<tr>
<th>Dest.</th>
<th>Line</th>
<th>Hops</th>
</tr>
</thead>
<tbody>
<tr>
<td>1A</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1B</td>
<td>1B</td>
<td>1</td>
</tr>
<tr>
<td>1C</td>
<td>1C</td>
<td>1</td>
</tr>
<tr>
<td>2A</td>
<td>1B</td>
<td>2</td>
</tr>
<tr>
<td>2B</td>
<td>1B</td>
<td>3</td>
</tr>
<tr>
<td>2C</td>
<td>1B</td>
<td>3</td>
</tr>
<tr>
<td>2D</td>
<td>1B</td>
<td>4</td>
</tr>
<tr>
<td>3A</td>
<td>1C</td>
<td>3</td>
</tr>
<tr>
<td>3B</td>
<td>1C</td>
<td>2</td>
</tr>
<tr>
<td>4A</td>
<td>1C</td>
<td>3</td>
</tr>
<tr>
<td>4B</td>
<td>1C</td>
<td>4</td>
</tr>
<tr>
<td>4C</td>
<td>1C</td>
<td>4</td>
</tr>
<tr>
<td>5A</td>
<td>1C</td>
<td>4</td>
</tr>
<tr>
<td>5B</td>
<td>1C</td>
<td>5</td>
</tr>
<tr>
<td>5C</td>
<td>1B</td>
<td>5</td>
</tr>
<tr>
<td>5D</td>
<td>1C</td>
<td>6</td>
</tr>
<tr>
<td>5E</td>
<td>1C</td>
<td>5</td>
</tr>
</tbody>
</table>
Hierarchical Routing (3)
Hierarchical Routing (4)

- Penalty is longer paths

1C is best route to region 5, except for destination 5C.
Observations

• Outside a region, nodes have one route to all hosts within the region
  – This gives savings in table size, messages and computation

• However, each node may have a different route to an outside region
  – Routing decisions are still made by individual nodes; there is no single decision made by a region
IP Prefix Aggregation and Subnets (§4.2.1)

- How to help scale routing by adjusting the size of IP prefixes
  - Split (subnets) and join (aggregation)
Recall

• IP addresses are allocated in blocks called IP prefixes, e.g., 18.31.0.0/16
  – Hosts on one network in same prefix

• A “/N” prefix has the first N bits fixed and contains $2^{32-N}$
  – 2 host addresses
  – E.g., “/24”
  – E.g., “/16”
Key Flexibility

• Routers keep track of prefix lengths
  – Use it for longest prefix matching

  Routers can change prefix lengths without affecting hosts

• More specific IP prefix
  – Longer prefix, fewer IP addresses

• Less specific IP prefix
  – Shorter prefix, more IP addresses
Prefixes and Hierarchy

• IP prefixes already help to scale routing, but we can go further
  – Can use a less specific prefix to name a region made up of several prefixes
• More information at: www.route-aggregation.net
Subnets and Aggregation

Two use cases for adjusting the size of IP prefixes; both reduce routing table size

1. Subnets
   - Internally split one less specific prefix into multiple more specific prefixes

2. Aggregation
   - Externally join multiple more specific prefixes into one large prefix
Subnets

- Internally split up one IP prefix

![Diagram showing subnets]

16K 128.208.0.0/18

32K 128.208.128.0/17

8K 128.208.96.0/19

One prefix sent to rest of Internet

128.208.0.0/16 (to Internet)

64K addresses

Company

Rest of Internet
Visualizing the Prefixes

<table>
<thead>
<tr>
<th>Prefix</th>
<th>3rd byte in binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>128.208.128/17</td>
<td>1000 0000</td>
</tr>
<tr>
<td>128.208.96/19</td>
<td>0110 0000</td>
</tr>
<tr>
<td>128.208.0/18</td>
<td>0000 0000</td>
</tr>
</tbody>
</table>

Questions:
What prefix is covered in addition by 128.208/16?
What if that prefix belongs to someone outside these networks?
Aggregation

- Externally join multiple separate IP prefixes

One prefix sent to rest of Internet

New York

192.24.0.0/19
(1 aggregate prefix)

London

192.24.0.0/21
(3 prefixes)

Cambridge

192.24.16.0/20

Oxford

192.24.8.0/22

Edinburgh

Rest of Internet

ISP
Visualizing the Prefixes

Questions:
What prefix is covered in addition by 192.24.0/19?
What if that prefix belongs to someone outside these networks?
Supplementary Material

• Note: the following material is purely for your enjoyment to satisfy your curiosity. It will not be tested on the exam.
Constructing a Routing Table

• Recall: Number of entries in the routing tables are increasing as the Internet is growing.

• How to construct a routing table with the least possible entries that still maintains all the information required for forwarding?

• Approach: Use subnets and prefix aggregation!

• We use an algorithm called Optimal Routing Table Constructor (OPTC), presented in the following paper: “Constructing Optimal IP Routing Tables” by Richard P. Draves et al., INFOCOM 1999.
## Constructing a Routing Table

### Routing Table at Router R

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>00*</td>
<td>2</td>
</tr>
<tr>
<td>10*</td>
<td>2</td>
</tr>
<tr>
<td>11*</td>
<td>3</td>
</tr>
</tbody>
</table>

### Binary Tree Representation of Routing Table

![Binary Tree Representation of Routing Table](image-url)
ORTC: First Pass

• Propagate information from root to tree leafs
  1. Normalize tree
  2. Initialize new leafs with information from nearest ancestor
  3. Remove information in interior nodes
ORTC: Second Pass

• Calculate most prevalent hops at every level of the routing table
  1. Percolate sets of next hops from leafs to root
  2. Percolate:

\[ A \# B = \begin{cases} 
A \cap B & \text{if } A \cap B \neq \emptyset \\
A \cup B & \text{if } A \cap B = \emptyset 
\end{cases} \]
ORTC: Third Pass

- Select next hops for prefixes and eliminate redundant information

  1. Each node has a set of possible next hops
  2. Each node will inherit a next hop from the closest ancestor
  3. If this inherited next hop is a member of the node’s set of potential next hops, it does not need a next hop on its own
Constructing an Optimal Routing Table

Optimal Routing Table

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Dst.</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>2</td>
</tr>
<tr>
<td>01*</td>
<td>1</td>
</tr>
<tr>
<td>11*</td>
<td>3</td>
</tr>
</tbody>
</table>
How to perform efficient Longest-Prefix Matching?
Ternary Content-Addressable Memory (TCAM)

• A 100 Gbps link may carry approx 200 M packets per second (64-byte packets)
  – Only 5 ns time to perform lookup per packet! (without parallelization or pipelining)
  – From http://thyme.rand.apnic.net : 700’000 prefixes announced

• TCAM is a specialized type of high-speed memory
  • Searches entire content in single clock cycle
  • Ternary: memory has three different states (0, 1, X)
  • Last state “X” referred to as “don't-care” or wildcard
  • Allows searches based on pattern-matching
Longest-Prefix Matching using TCAM

- The wildcard state helps defining partial entries in the routing table
  - E.g., 10.11.12.* (resp. 10.11.12.0/24) where * stands for any value in the last octet
- Locate all possible matches in single cycle
- Routing prefixes must be ordered by their prefix length
- Priority encoder finds first match
LPM using TCAM - Example

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>00X</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
<td>0X</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
<td>1X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Input: 10.11.12.65

Input binary: 0100 0001

Note: Numbers in **blue and bold** are in binary

Match!

more specific

less specific

Note: Numbers in **blue and bold** are in binary
TCAM Power Consumption

• Downside: expensive to build, consumes a lot of power, and thus generates a lot of heat that must be dissipated

• TCAM consumes close to 30% of the overall power consumed by a high-end router
  • Prefix growth requires larger TCAM in routers!

• Recent research targets: power-efficient TCAM cell, leveraging SRAM to emulate TCAM functionality, etc.
Finding Duplicates: Bloom Filter

- Problem: identify if an element is a duplicate
- Challenge: cannot store all previous elements
- Solution: Bloom filter provides probabilistic data structure for set membership testing

  - Setup: bit vector V with m bits, V = 0
  - Insert element e
    - Compute k hash functions \( h_i = H_i(e) \), where \( 0 < h_i \leq m \)
    - Set all bits \( V[h_i] = 1 \)
  - Test if element e’ has already been seen
    - Recompute k hash functions, test if all \( V[h_i] = \) 1
    - If all bits are 1 ☐ “seen before”, otherwise “new”
Bloom Filter

- **Initialization:**

  - Insertion of $e$: $H_1(e) = 5$, $H_2(e) = 2$

  - Test of $e' = e$: $H_1(e') = 5$, $H_2(e') = 2 \Rightarrow$ seen before

  - Test of $e'' \neq e$: $H_1(e'') = 5$, $H_2(e'') = 1 \Rightarrow$ new

  - Test of $e''' \neq e$: $H_1(e''') = 2$, $H_2(e''') = 5 \Rightarrow$ seen before?

  **No FN, constant time insertion and test**
Bloom Filter Parameters

- **m**: # of bits in BF
- **k**: # of hash functions
- **n**: # of inserted elements
- **$P(\text{FP}) = \left(1 - \left[1 - \frac{1}{m}\right]^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k$**
- **k to minimize $P(\text{FP})$**: $k = \frac{m}{n} \ln 2 \approx 0.7 \frac{m}{n}$,
- For that choice of $k$, resulting $P(\text{FP}) = p = 2^{-k} \approx 0.6185^{m/n}$.
- Given optimal $k$, choice of optimal $m = -\frac{n \ln p}{(\ln 2)^2}$.

Source: http://en.wikipedia.org/wiki/Bloom_filter
Bloom Filter Size Examples

- $n = 10^6$
- $p = 1\%$
- $m = 9.6 \times 10^6$ bits $\sim 1.2$ Mbytes
- $k \sim 6.6 \times 7$
- With $k = 7$, $p \sim 1\%$

- $p = 0.1\%$
- $m = 14.4 \times 10^6$ bits $\sim 1.8$ Mbytes
- $k = 10$
How to Find Duplicates in Practice?

- Problem: Bloom filter “fills up”
- Simple approach: Reset Bloom filter periodically
- Problems with simple approach?
  - Some false detections: fundamental problem with Bloom filters
  - Fail to provide “no FN” guarantee: some duplicates are not detected!
    - If duplicate arrives after reset, then it is not detected as duplicate
Duplicate Detection in Practice

- Define maximum packet propagation time $\delta$
- Require time synchronization, max sync error $\sigma$
- Split time up into time periods of duration $\Delta > \delta + 2\sigma$
- Keep two Bloom filters $B_0$ and $B_1$
- $B_0$ is reset at the beginning of time period $i$ and filled during period $i$, where $i \mod 2 = 0$
  - Analogous for $B_1$
- Each packet includes timestamp $t_s$ it is sent, in time period $T_s$
- On packet arrival at router at $t_c$, in time period $T_c$
  - Check if $t_s - \sigma \leq t_c \leq t_s + \delta + \sigma$; otherwise drop packet
  - Router checks both $B_0$ and $B_1$ to detect duplicate
  - If packet is not a duplicate, insert it into $B_i$ where $i = T_c \mod 2$
Details of Setting Time Duration

- $t^i_s$: timestamp of packet $i$
- $\delta^i$: propagation delay of packet $i$
- $\sigma$: max sync error

$\Rightarrow |(t^i_s + \delta^i) - t^c| \leq \sigma$

$\Rightarrow t^i_s + \delta^i - \sigma \leq t^c \leq t^i_s + \delta^i + \sigma$

$\Rightarrow t^i_s - \sigma \leq t^c \leq t^i_s + \delta + \sigma$ (because $0 \leq \delta^i \leq \delta$)

$\Rightarrow$ For any claimed $t^i_s$, packet is valid at router for at most $\delta + 2\sigma$ length of time

$\Rightarrow$ If duration of time period $\Delta > \delta + 2\sigma$, packet is valid for at most two intervals (current and the next one)
Quick Summary of Bloom Filters

- Basic operations
  - Insert an element
  - Membership test

- Properties
  - No FN, low FP
  - Time efficient: $O(1)$ membership checking
  - Space efficient: constant number of bits per element for a given FP rate
    - But still $O(N)$ memory overhead