Chapters 4,5: Network Layer

- Introduction (forwarding and routing)
- Review of queueing theory
- Router design and operation
- IP: Internet Protocol
  - IPv4 (datagram format, addressing, NAT, ICMP)
  - IPv6
- Routing algorithms
  - Link state, Distance Vector
- Routing in the Internet
  - Autonomous Systems
  - Routing protocols (OSPF, BGP)
- Generalized Forwarding & SDN

IPv6

- Initial motivation: 32-bit address space soon to be completely allocated.
- Additional motivation:
  - header format helps speed processing/forwarding
  - header changes to facilitate QoS

IPv6 datagram format:
  - fixed-length 40 byte header
  - no fragmentation allowed
**IPv6 Header**

*Priority:* identify priority among datagrams in flow; help distinguish packets that can be flow controlled and those that cannot

*Flow Label:* experimental - identify datagrams in same “flow” and treat them specially. Potential use?

*Next header:* identify upper layer protocol for data or optional IP header

---

**IPv6 Addressing**

- *BTW, how big is $2^{128}$???*
  - There are $1,000,000,000,000,000,000,000,000,000,000$ stars in the observable Universe.
  - $2^{128}$ provides more than 300 billion addresses for every such star!
Other Changes from IPv4

- **Checksum**: removed entirely to reduce processing time at each hop
- **Options**: allowed, but outside of header, indicated by “Next Header” field
- **ICMPv6**: new version of ICMP
  - additional message types, e.g. “Packet Too Big”
  - multicast group management functions

Header Extensions
Example optional headers

- Hop-by-Hop Option - Special options that require hop-by-hop processing
- Routing - Extended routing, like IPv4 loose source route
- Fragmentation - Fragmentation and reassembly (processed at destination)
- Authentication - Integrity and authentication, security
- Encapsulation - Confidentiality
- Destination Options - Optional information to be examined by the destination node

Transition From IPv4 To IPv6

- Not all routers are upgraded simultaneously
  - How will the network operate with mixed IPv4 and IPv6 routers?
- Tunneling: IPv6 carried as payload in IPv4 datagram among IPv4 routers. How?
Tunneling

Logical view:

Physical view:

Flow: A to B: IPv6
Flow: B to C: IPv6 inside IPv4
Flow: B to C: IPv6 inside IPv4
Flow: E to F: IPv6
IPv6 Status

- CIDR introduced in 1993 to stave off exhaustion of IPv4 addresses
- CIDR (plus NAT) has been very effective, but final phase of allocation began in 2011.
- Still, in 2013 only 16% of networks supported IPv6
- In 2014, IPv4 still carried more than 99% of all Internet traffic
- Google reports that (only) 8% of users reached Google services via IPv6...
- IPv6: 20 years in deployment - hard to change network infrastructure!
- Yet, look at app protocols developed in past 20 years social media, messaging, streaming, gaming...

Chapters 4,5: Network Layer

- Introduction (forwarding and routing)
- Review of queueing theory
- Router design and operation
- IP: Internet Protocol
  - IPv4 (datagram format, addressing, NAT, ICMP)
  - IPv6
- Routing algorithms
  - Link state, Distance Vector
- Routing in the Internet
  - Autonomous Systems
  - Routing protocols (OSPF, BGP)
- Generalized Forwarding & SDN
Interplay between routing, forwarding

Routing Algorithm classification

Static or dynamic?
Static:
- routes change slowly over time
- manual configuration
Dynamic:
- routes change more quickly
- periodic updates in response to link cost changes

Global or decentralized information?
Global:
- all routers have complete topology, link cost info
- “link state” algorithms
Decentralized:
- router knows physically-connected neighbors, link costs to neighbors
- iterative process of computation, exchange of info with neighbors
- “distance vector” algorithms
Graph abstraction

Graph: G = (N,E)

N = set of routers = \{ u, v, w, x, y, z \}

E = set of links = \{ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) \}

Remark: Real network routing algorithms typically use DIRECTED graphs.

Graph abstraction: costs

- \( c(x,x') = \) cost of link \((x,x')\)
  - e.g., \( c(w,z) = 5 \)

- cost could be
  - 1 (hop count)
  - inversely related to bandwidth,
  - inversely related to congestion,
  - count of packets in queue,
  - some combination of above

Cost of path \((x_1, x_2, x_3, ..., x_p)\) = \( c(x_1,x_2) + c(x_2,x_3) + ... + c(x_{p-1},x_p) \)

Question: What’s the least-cost path between u and z?

Routing algorithm: algorithm that finds least-cost path from source to destination.
**Principle of Optimality**

- If node B lies on an optimal path from node A to node C, then an optimal path from node B to node C also lies along the same path. Why does this property hold?

**Principal of Optimality**

- The result is that the set of optimal routes from all sources to a given destination form *sink tree* rooted at the destination.

- In general, is the sink tree unique?
A Link-State Routing Algorithm

Dijkstra’s algorithm
- In-memory graph of network
- Network topology, link costs known to all nodes
  - Accomplished by flooding “link state advertisements”
  - All nodes have same info
- Computes least cost paths from one node (source) to all other nodes
  - Gives forwarding table for that node
- Iterative: after k iterations, know least cost path to k destinations

Notation:
- \( c(x,y) \): link cost from node x to y; \( = \infty \) if not direct neighbors
- \( D(v) \): current value of cost of path from source to dest. v
- \( p(v) \): predecessor node along path from source to v
- \( M \): set of nodes whose least cost path definitively known

Basic Idea
- Find the shortest paths from a given source node to all other nodes
- Proceeds in stages - build the sink tree one branch at a time.
- By the kth stage, the shortest paths to the k nodes closest to the source have been determined (and added to set M)
- At (k + 1)st stage, that node not already in M that has the shortest path from the source is added to M
- As nodes are added to M, their path from the source is defined.
Dijsktra’s Algorithm

1. **Initialization:**
   2. \( M = \{ u \} \)
   3. for all nodes \( v \)
   4. if \( v \) adjacent to \( u \)
   5. then \( D(v) = c(u,v) \)
   6. else \( D(v) = \infty \)

7. **Loop**
   8. find \( w \) not in \( M \) such that \( D(w) \) is a minimum
   9. add \( w \) to \( M \)
   10. update \( D(v) \) for all \( v \) adjacent to \( w \) and not in \( M \):
       \[
       D(v) = \min(D(v), D(w) + c(w,v))
       \]
   11. /* new cost to \( v \) is either old cost to \( v \) or known shortest path cost to \( w \) plus cost from \( w \) to \( v \) */
   12. until all nodes in \( M \)

Dijsktra’s algorithm: example

<table>
<thead>
<tr>
<th>( M )</th>
<th>( D(v)_{p(v)} )</th>
<th>( D(w)_{p(w)} )</th>
<th>( D(x)_{p(x)} )</th>
<th>( D(y)_{p(y)} )</th>
<th>( D(z)_{p(z)} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dijkstra’s algorithm: result

Resulting shortest-path tree from u:

Resulting forwarding table in u:

<table>
<thead>
<tr>
<th>destination</th>
<th>link</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>(u,v)</td>
</tr>
<tr>
<td>x</td>
<td>(u,x)</td>
</tr>
<tr>
<td>y</td>
<td>(u,x)</td>
</tr>
<tr>
<td>w</td>
<td>(u,x)</td>
</tr>
<tr>
<td>z</td>
<td>(u,x)</td>
</tr>
</tbody>
</table>

Bellman-Ford Algorithm

- Proceeds in stages.
- Find the shortest paths from a given source subject to the constraint that the paths contain at most one link
- Next, find all that contain two links.
- ...and so on.
**Bellman-Ford**

- **Variables:**
  - $c(x,y) = \text{link cost}$
  - $h = \text{max number of links in path at current stage}$
  - $D^h(y) = \text{cost of least-cost path from source to node } y \text{ under the constraint of no more than } h \text{ links}$

**Algorithm**

- Let $s$ be the source node
- Initialize $D^0(n) = \infty$ for all $n \neq s$
- For each successive $h > 0$,
  - $D^{h+1}(n) = \min_j [D^h(j) + c(j,n)]$

- When does the algorithm halt?
Bellman-Ford Example

Source = u

<table>
<thead>
<tr>
<th>h</th>
<th>D_h(v)</th>
<th>D_h(w)</th>
<th>D_h(x)</th>
<th>D_h(y)</th>
<th>D_h(z)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Comparison

- Both Dijkstra’s algorithm and the Bellman-Ford algorithm converge to shortest path solutions under static conditions.
- Complexity?

- Suitability for distributed implementation?
Distance Vector Protocol

- $D_x(y)$ = estimate of least cost from $x$ to $y$
- Node $x$ knows cost to each neighbor $v$: $c(x,v)$
- Node $x$ maintains distance vector $D_x = [D_x(y): y \in N]$
- Node $x$ also maintains its neighbors’ distance vectors
  - For each neighbor $v$, $x$ maintains $D_v = [D_v(y): y \in N]$

Bellman-Ford Equation

Define

$$D_x(y) := \text{cost of least-cost path from source } x \text{ to } y$$

Then

$$D_x(y) = \min_v \{c(x,v) + D_v(y)\}$$

where min is taken over all neighbors $v$ of $x$
Distance vector protocol

**Basic idea:**
- From time to time, each node sends its own distance vector estimate to neighbors
- Asynchronous
- When a node \( x \) receives new DV estimate from neighbor, it updates its own DV using B-F equation:
  \[
  D_x(y) \leftarrow \min_v\{c(x,v) + D_v(y)\} \quad \text{for each node } y \in N
  \]
- Under normal conditions, the estimates \( D_x(y) \) converge to the actual least cost \( d_x(y) \)

**Distance Vector Algorithm**

Iterative, asynchronous:
- each local iteration caused by:
  - local link cost change
  - DV update message from neighbor

Distributed:
- each node notifies neighbors *only* when its DV changes
  - neighbors then notify their neighbors if necessary

Each node:
- *wait* for (change in local link cost or msg from neighbor)
- *recompute* estimates
- if DV to any dest has changed, *notify* neighbors
Distance Vector Example

![Distance Vector Diagram](distance_vector_diagram.png)

(a) Distance Vector Example

(b) Routing Table

Comparison of LS and DV algorithms

**Message complexity**
- **LS**: with $n$ nodes, $E$ links, $O(nE)$ msgs sent
- **DV**: exchange between neighbors only
  - convergence time varies

**Speed of Convergence**
- **LS**: $O(n^2)$ algorithm requires $O(nE)$ msgs
  - may have oscillations
- **DV**: convergence time varies
  - routing loops possible
  - “count-to-infinity” problem

**Robustness**: what happens if router malfunctions?

**LS**:
- node can advertise incorrect link cost
- each node computes only its own table

**DV**:
- DV node can advertise incorrect path cost
- each node’s table used by others
  - error propagates through network