Module 9
Network Layer
A note on the use of these ppt slides:
We’re making these slides freely available to all (faculty, students, readers). They’re in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a lot of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we’d like people to use our book!)
- If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2010
J.F Kurose and K.W. Ross, All Rights Reserved

Computer Networking: A Top Down Approach
5th edition.
Jim Kurose, Keith Ross
Addison-Wesley, April 2009.
Network layer

- transport segment from sending to receiving host
- on sending side encapsulates segments into datagrams
- on receiving side, delivers segments to transport layer
- network layer protocols in every host, router
- router examines header fields in all IP datagrams passing through it
Network Layer is Host-to-Host
# Internet Protocols

## Protocol Layers

<table>
<thead>
<tr>
<th>Application</th>
<th>FTP</th>
<th>Telnet</th>
<th>NFS</th>
<th>SMTP</th>
<th>HTTP ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport</td>
<td>TCP</td>
<td></td>
<td></td>
<td>UDP</td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td>IP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Link</td>
<td>X.25</td>
<td>Ethernet</td>
<td>Packet Radio</td>
<td>ATM</td>
<td>FDDI</td>
</tr>
<tr>
<td>Physical</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Segment</th>
<th>Datagram</th>
<th>Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Two Key Network-Layer Functions

- **Forwarding**: move packets from router’s input to appropriate router output

- **Routing**: determine route taken by packets from source to destination.
  - routing algorithms

- **Connection service**: before datagrams flow, two end hosts and intervening routers establish virtual connection (VC)
  - Needed in some network architectures: ATM, frame relay, X.25
  - Network vs transport layer connection service:
    - **network**: between two hosts (may also involve intervening routers in case of VCs)
    - **transport**: between two processes
Interplay Between Routing and Forwarding

routing algorithm

<table>
<thead>
<tr>
<th>header value</th>
<th>output link</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td>3</td>
</tr>
<tr>
<td>0101</td>
<td>2</td>
</tr>
<tr>
<td>0111</td>
<td>2</td>
</tr>
<tr>
<td>1001</td>
<td>1</td>
</tr>
</tbody>
</table>

value in arriving packet's header
Network layer connection and connection-less service

- Datagram network provides network-layer connectionless service
- Virtual Circuit (VC) network provides network-layer connection service
- Analogous to the transport-layer services, but:
  - service: host-to-host
  - no choice: network provides one or the other
  - implementation: in network core
Virtual Circuits

“source-to-destination path behaves much like telephone circuit”

➤ performance-wise

➤ network actions along source-to-destination path

• call setup, teardown for each call *before* data can flow
• each packet carries VC identifier (not destination host address)
• *every* router on source-destination path maintains “state” for each passing connection
• link, router resources (bandwidth, buffers) may be *allocated* to VC (dedicated resources = predictable service)
Datagram networks

- no call setup at network layer
- routers: no state about end-to-end connections
  - no network-level concept of “connection”
- packets forwarded using destination host address
  - packets between same source-destination pair may take different paths

Application
Transport
Network
Data Link
Physical

1. Send data
2. Receive data
Datagram Forwarding table

Routing algorithm

Local forwarding table

<table>
<thead>
<tr>
<th>dest address</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>address-range 1</td>
<td>3</td>
</tr>
<tr>
<td>address-range 2</td>
<td>2</td>
</tr>
<tr>
<td>address-range 3</td>
<td>2</td>
</tr>
<tr>
<td>address-range 4</td>
<td>1</td>
</tr>
</tbody>
</table>

4 billion IP addresses, so rather than list individual destination address list range of addresses (aggregate table entries)

IP destination address in arriving packet’s header
## Datagram Forwarding table

<table>
<thead>
<tr>
<th>Destination Address Range</th>
<th>Link Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111</td>
<td>0</td>
</tr>
<tr>
<td>11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111</td>
<td>1</td>
</tr>
<tr>
<td>11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111</td>
<td>2</td>
</tr>
<tr>
<td>otherwise</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Destination Address Range</th>
<th>Link Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>11001000 00010111 00010*** *********</td>
<td>0</td>
</tr>
<tr>
<td>11001000 00010111 00011000 *********</td>
<td>1</td>
</tr>
<tr>
<td>11001000 00010111 00011111 *********</td>
<td>2</td>
</tr>
<tr>
<td>otherwise</td>
<td>3</td>
</tr>
</tbody>
</table>
Datagram or VC network: why?

Internet (datagram)
- data exchange among computers
  - “elastic” service, no strict timing req.
- “smart” end systems (computers)
  - can adapt, perform control, error recovery
  - simple inside network, complexity at “edge”
- many link types
  - different characteristics
  - uniform service difficult

ATM (VC)
- evolved from telephony
- human conversation:
  - strict timing, reliability requirements
  - need for guaranteed service
- “dumb” end systems
  - telephones
  - complexity inside network
Router Architecture Overview

two key router functions:

• run routing algorithms/protocol (RIP, OSPF, BGP)
• *forwarding* datagrams from incoming to outgoing link
Input Port Functions

- **Physical layer:** bit-level reception
- **Data link layer:** e.g., Ethernet

### Decentralized switching:
- Given datagram destination, lookup output port using forwarding table in input port memory
- Goal: complete input port processing at ‘line speed’
- Queuing: if datagrams arrive faster than forwarding rate into switch fabric
The Internet Network layer

Host, router network layer functions:

- **Transport layer:** TCP, UDP
  - **Routing protocols**
    - path selection
    - RIP, OSPF, BGP
  - **IP protocol**
    - addressing conventions
    - datagram format
    - packet handling conventions
  - **ICMP protocol**
    - error reporting
    - router “signaling”

- **Network layer**
- **Physical layer**
IP datagram format

- IP protocol version number
- Header length (bytes)
- "Type" of data
- 16-bit identifier
- Time to live
- Upper layer
- Header checksum
- Total datagram length (bytes)
- Options (if any)
  - Data (variable length, typically a TCP or UDP segment)

32 bit source IP address
32 bit destination IP address
E.g. timestamp, record route taken, specify list of routers to visit.

Upper layer protocol to deliver payload to
Max number remaining hops (decremented at each router)
Fragmentation for fragmentation/reassembly

IP Fragmentation & Reassembly

- network links have MTU (maximum transmission unit): largest possible link-level frame.
  - different link types, different MTUs
- large IP datagram divided ("fragmented") within net
  - one datagram becomes several datagrams
  - "reassembled" only at final destination
  - IP header bits used to identify, order related fragments
IP Addressing: introduction

- **IP address**: 32-bit identifier for host, router interface

- **interface**: connection between host/router and physical link
  - Router’s typically have multiple interfaces
  - Host typically has one interface
  - IP addresses associated with each interface

223.1.1.1 = 11011111 00000001 00000001 00000001

223 1 1 1 1
Subnets

- **IP address:**
  - subnet part (high order bits)
  - host part (low order bits)

- **What’s a subnet?**
  - device interfaces with same subnet part of IP address
  - can physically reach each other without intervening router
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 (Cont)**

**Priority:** identify priority among datagrams in flow

**Flow Label:** identify datagrams in same “flow.”
(concept of “flow” not well defined).

**Next header:** identify upper layer protocol for data

<table>
<thead>
<tr>
<th>ver</th>
<th>pri</th>
<th>flow label</th>
</tr>
</thead>
<tbody>
<tr>
<td>payload len</td>
<td>next hdr</td>
<td>hop limit</td>
</tr>
<tr>
<td>source address</td>
<td>(128 bits)</td>
<td></td>
</tr>
<tr>
<td>destination address</td>
<td>(128 bits)</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

32 bits
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
Routing Algorithms – Interplay between routing, forwarding

Routing algorithm

<table>
<thead>
<tr>
<th>header value</th>
<th>output link</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td>3</td>
</tr>
<tr>
<td>0101</td>
<td>2</td>
</tr>
<tr>
<td>0111</td>
<td>2</td>
</tr>
<tr>
<td>1001</td>
<td>1</td>
</tr>
</tbody>
</table>

value in arriving packet’s header
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)\}
- $c(x,y) =$ cost of link $(x,y)$
  - e.g., $c(w,z) = 5$
  - cost could always be 1, or inversely related to bandwidth, or inversely related to congestion
- Cost of path $(x_1, x_2, x_3, \ldots, x_p) = c(x_1,x_2) + c(x_2,x_3) + \ldots + c(x_{p-1},x_p)$

Routing algorithm: algorithm that finds least-cost path
Routing Algorithm classification

Global or decentralized information?

Global:
- all routers have complete topology, link cost info
- “link state” (LS) algorithms

Decentralized:
- router knows physically-connected neighbors, link costs to neighbors
- iterative process of computation, exchange of info with neighbors
- “distance vector” (DV) algorithms

Static or dynamic?

Static:
- routes change slowly over time

Dynamic:
- routes change more quickly
  - periodic update
  - in response to link cost changes
A Link-State Routing Algorithm

Dijkstra’s algorithm
• net topology, link costs known to all nodes
  ➡ accomplished via “link state broadcast”
  ➡ all nodes have same info
• Dijkstra’s algorithm: 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
**Dijkstra’s Algorithm**

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

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

<table>
<thead>
<tr>
<th>Step</th>
<th>N'</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>0</td>
<td>u</td>
<td>2,u</td>
<td>5,u</td>
<td>1,u</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td>ux</td>
<td>2,u</td>
<td>4,x</td>
<td>2,x</td>
<td>∞</td>
<td>4,y</td>
</tr>
<tr>
<td>2</td>
<td>uxy</td>
<td>2,u</td>
<td>3,y</td>
<td>4,y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>uxyv</td>
<td></td>
<td>3,y</td>
<td></td>
<td>4,y</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>uxyvw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4,y</td>
</tr>
<tr>
<td>5</td>
<td>uxyv wz</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

![Graph Diagram]
Dijkstra’s algorithm: example (2)

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>
Distance Vector Algorithm

Bellman-Ford Equation (dynamic programming)

Define

\[ d_x(y) := \text{cost of least-cost path from } x \text{ to } y \]

Then

\[ d_x(y) = \min_v \{c(x,v) + d_v(y)\} \]

where min is taken over all neighbors \( v \) of \( x \)
Bellman-Ford example

Clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

$$d_u(z) = \min \{ c(u,v) + d_v(z), \ c(u,x) + d_x(z), \ c(u,w) + d_w(z) \}$$

$$= \min \{2 + 5, \ 1 + 3, \ 5 + 3\} = 4$$

Node that achieves minimum is next hop in shortest path $\rightarrow$ forwarding table
Distance Vector Algorithm

• Each node $x$ maintains the following
  ➔ Its own distance vector $D_x = [D_x(y) : y \in N]$ (N is the set of nodes)
    ✦ $D_x(y)$ = estimate of least cost from $x$ to $y$
  ➔ cost to each neighbor $v$: $c(x,v)$
  ➔ its neighbors’ distance vectors. For each neighbor $v$, $x$ maintains
    $D_v = [D_v(y) : y \in N]$  
• from time-to-time, each node sends its own distance vector estimate to neighbors
• when $x$ receives new DV estimate from neighbor, it updates its own DV using B-F equation:

  \[ D_x(y) \leftarrow \min_v \{c(x,v) + D_v(y)\} \]  for each node $y \in N$

• Under minor, natural conditions, the estimate $D_x(y)$ 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:

1. **wait** for (change in local link cost or msg from neighbor)
2. **recompute** estimates
3. if DV to any dest has changed, **notify** neighbors
node x table

<table>
<thead>
<tr>
<th></th>
<th>cost to</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>x</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>y</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>z</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>

node y table

<table>
<thead>
<tr>
<th></th>
<th>cost to</th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>∞</td>
</tr>
<tr>
<td>y</td>
<td>2</td>
</tr>
<tr>
<td>z</td>
<td>∞</td>
</tr>
</tbody>
</table>

node z table

<table>
<thead>
<tr>
<th></th>
<th>cost to</th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>∞</td>
</tr>
<tr>
<td>y</td>
<td>∞</td>
</tr>
<tr>
<td>z</td>
<td>7</td>
</tr>
</tbody>
</table>

\[ D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\} \]
\[ = \min\{2+0, 7+1\} = 2 \]

\[ D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\} \]
\[ = \min\{2+1, 7+0\} = 3 \]
\[ D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\} = \min\{2+0, 7+1\} = 2 \]

\[ D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\} = \min\{2+1, 7+0\} = 3 \]
Comparison of LS and DV algorithms

Message complexity

• **LS:** with \( n \) nodes, \( E \) links, \( O(nE) \) messages sent

• **DV:** exchange between neighbors only
  ➔ convergence time varies

Speed of Convergence

• **LS:** \( O(n^2) \) algorithm requires \( O(nE) \) messages
  ➔ may have oscillations

• **DV:** convergence time varies
  ➔ may be routing loops
  ➔ 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 propagate thru network
Hierarchical Routing

So far we assumed

• All routers are identical
• Network is “flat”
• These are not true in practice

scale: with 200 million destinations:
• can’t store all destinations in routing tables!
• routing table exchange would swamp links!

administrative autonomy
• internet = network of networks
• each network admin may want to control routing in its own network
Hierarchical Routing

• aggregate routers into regions, autonomous systems (AS)
• routers in same AS run same routing protocol
  ➡ intra-AS routing protocol
  ➡ routers in different AS can run different intra-AS routing protocol

**gateway router**
  ➡ at “edge” of its own AS
  ➡ has link to router in another AS
RIP (Routing Information Protocol)

- included in BSD-UNIX distribution in 1982
- distance vector algorithm
  - distance metric: # hops (max = 15 hops), each link has cost 1
  - DVs exchanged with neighbors every 30 sec in response message (aka advertisement)
  - each advertisement: list of up to 25 destination subnets (in IP addressing sense)

from router A to destination subnets:

<table>
<thead>
<tr>
<th>subnet</th>
<th>hops</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>1</td>
</tr>
<tr>
<td>v</td>
<td>2</td>
</tr>
<tr>
<td>w</td>
<td>2</td>
</tr>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>3</td>
</tr>
<tr>
<td>z</td>
<td>2</td>
</tr>
</tbody>
</table>
RIP: Example

routing table in router D

<table>
<thead>
<tr>
<th>destination subnet</th>
<th>next router</th>
<th># hops to dest</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>y</td>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>z</td>
<td>B</td>
<td>7</td>
</tr>
<tr>
<td>x</td>
<td>--</td>
<td>1</td>
</tr>
<tr>
<td>....</td>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>
RIP: Example

A-to-D advertisement

<table>
<thead>
<tr>
<th>dest</th>
<th>next</th>
<th>hops</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>x</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>z</td>
<td>C</td>
<td>4</td>
</tr>
</tbody>
</table>

routing table in router D

<table>
<thead>
<tr>
<th>destination subnet</th>
<th>next router</th>
<th># hops to dest</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>y</td>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>z</td>
<td>B</td>
<td>7</td>
</tr>
<tr>
<td>x</td>
<td>--</td>
<td>1</td>
</tr>
<tr>
<td>....</td>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>
OSPF (Open Shortest Path First)

• “open”: publicly available
• uses Link State algorithm
  ➪ LS packet dissemination
  ➪ topology map at each node
  ➪ route computation using Dijkstra’s algorithm

• OSPF advertisement carries one entry per neighbor router
• advertisements disseminated to entire AS (via flooding)
  ➪ carried in OSPF messages directly over IP (rather than TCP or UDP)
OSPF “advanced” features (not in RIP)

- **security**: all OSPF messages authenticated (to prevent malicious intrusion)
- **multiple same-cost paths** allowed (only one path in RIP)
- for each link, multiple cost metrics for different TOS (e.g., satellite link cost set “low” for best effort ToS; high for real time ToS)
- **integrated uni- and multicast support**:
  - Multicast OSPF (MOSPF) uses same topology data base as OSPF
- **hierarchical** OSPF in large domains.
Hierarchical OSPF

- **two-level hierarchy**: local area, backbone.
  - link-state advertisements only in area
  - each node has detailed area topology; only know direction (shortest path) to nets in other areas.

- **area border routers**: “summarize” distances to nets in own area, advertise to other Area Border routers.

- **backbone routers**: run OSPF routing limited to backbone.

- **boundary routers**: connect to other AS’s.