Routing Primer
Routing Outline

- Overview of Point-to-Point Routing (WAN)
- Routing Algorithm Classification
- Distance Vector Routing
- Link State Routing
- RIP
- OSPF
- BGP
Wide Area Network (WAN)

Interdomain level

Border routers

Autonomous system or domain

LAN level

Intradomain level

Border routers

Internet service provider

Leon-Garcia & Widjaja: Communication Networks
National Internet Service Providers

National service provider A

National service provider B

National service provider C
Network Layer

- transport segment from sending to receiving host.
- on sending side, encapsulates segments into datagram packets.
- 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.
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.

  **analogy:**
  - **routing**: process of planning trip from source to destination
  - **forwarding**: process of getting through single interchange
Interplay between Routing and Forwarding

Routing creates the tables.
Forwarding uses the tables.

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

K & R
Host, router network layer functions:

Transport Layer: TCP, UDP

Routing protocols
  • path selection
  • RIP, OSPF, BGP

Routing table

IP protocol
  • addressing conventions
  • datagram format
  • packet handling conventions

ICMP protocol
  • error reporting
  • router “signaling”

Data Link Layer

Physical Layer

Network Layer
Routing Algorithm Classification
Routing algorithm: that part of the Network Layer responsible for deciding on which output line to transmit an incoming packet.

- Remember: For virtual circuit subnets the routing decision is made ONLY at set up.

Algorithm properties: correctness, simplicity, robustness, stability, fairness, optimality, and scalability.
Routing is Graph Theory Problem

Figure 3.28 Network represented as a graph.

edges have costs
Routing Classification

Adaptive Routing

- based on current measurements of traffic and/or topology.
- centralized
- isolated
- distributed

Non-Adaptive Routing

- routing computed in advance and off-line
- flooding
- static routing using shortest path algorithms
Flooding

- Pure flooding :: every incoming packet to a node is sent out on every outgoing line.
  - Obvious adjustment - do not send out on arriving link (assuming full-duplex links).
  - The routing algorithm can use a hop counter (e.g., TTL) to dampen the flooding.
- Selective flooding :: only send on those lines going "approximately" in the right direction.
Centralized Routing

RCC

A
B
W
Z
Internetwork Routing [Halsall]

Adaptive Routing

Centralized

[IGP] Intradomain routing

Distance Vector routing [RIP]

Interdomain routing [BGP, IDRP]

[OSPF, IS-IS, PNNI]

Distributed

Exterior Gateway Protocols

Isolated

Exterior Gateway Protocols

Centralized [RCC]
Adaptive Routing Design

Design Issues:

1. How much **overhead** is incurred due to gathering the routing information and sending *routing packets*?

2. What is the time frame (i.e., the frequency) for sending *routing packets* in support of adaptive routing?

3. What is the **complexity** of the routing strategy?
Adaptive Routing

Basic functions:

1. Measurement of pertinent network data \{e.g. the cost metric\}.
2. Forwarding of information to where the routing computation will be done.
3. Compute the routing tables.
4. Convert the routing table information into a routing decision and then dispatch the data packet.
Shortest Path Routing

1. Bellman-Ford Algorithm [Distance Vector]
2. Dijkstra’s Algorithm [Link State]

What does it mean to be the shortest (or optimal) route?

We need a cost metric (edges in graph):

a. Minimize the number of hops along the path.
b. Minimize the mean packet delay.
c. Maximize the network throughput.
Distance Vector Routing
{Tanenbaum & Perlman version}
Distance Vector Routing

Historically known as the old ARPANET routing algorithm {or known as Bellman-Ford (BF) algorithm}.

BF Basic idea: each router maintains a Distance Vector table containing the distance between itself and ALL possible destination nodes.

Distances, based on a chosen metric, are computed using information from the neighbors’ distance vectors.

Distance Metric: usually hops or delay
Distance Vector Routing

Information kept by DV router

1. each router has an ID
2. associated with each link connected to a router, there is a link cost (static or dynamic).

Distance Vector Table Initialization

Distance to itself = 0
Distance to ALL other routers = infinity number
1. A router transmits its **distance vector** to each of its neighbors in a routing packet.

2. Each router receives and saves the most recently received **distance vector** from each of its neighbors.

3. A router recalculates its **distance vector** when:
   a. It receives a **distance vector** from a neighbor containing different information than before.
   b. It discovers that a link to a neighbor has gone down (i.e., a **topology change**).

   The DV calculation is based on minimizing the cost to each destination.
Distance Vector Example

**Figure 5-9.** (a) A subnet. (b) Input from A, I, H, K, and the new routing table for J.

![Diagram of a subnet with nodes A, B, C, D, E, F, G, H, I, J, K, and L connected by lines indicating routing paths.]

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>I</th>
<th>H</th>
<th>K</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>24</td>
<td>20</td>
<td>21</td>
<td>8 A</td>
</tr>
<tr>
<td>B</td>
<td>12</td>
<td>36</td>
<td>31</td>
<td>28</td>
<td>20 A</td>
</tr>
<tr>
<td>C</td>
<td>25</td>
<td>18</td>
<td>19</td>
<td>36</td>
<td>28 B</td>
</tr>
<tr>
<td>D</td>
<td>40</td>
<td>27</td>
<td>8</td>
<td>24</td>
<td>28 I</td>
</tr>
<tr>
<td>E</td>
<td>14</td>
<td>7</td>
<td>30</td>
<td>22</td>
<td>20 H</td>
</tr>
<tr>
<td>F</td>
<td>23</td>
<td>20</td>
<td>19</td>
<td>40</td>
<td>17 I</td>
</tr>
<tr>
<td>G</td>
<td>18</td>
<td>31</td>
<td>6</td>
<td>31</td>
<td>30 I</td>
</tr>
<tr>
<td>H</td>
<td>17</td>
<td>20</td>
<td>0</td>
<td>19</td>
<td>18 H</td>
</tr>
<tr>
<td>I</td>
<td>21</td>
<td>0</td>
<td>14</td>
<td>22</td>
<td>12 H</td>
</tr>
<tr>
<td>J</td>
<td>9</td>
<td>11</td>
<td>7</td>
<td>10</td>
<td>10 I</td>
</tr>
<tr>
<td>K</td>
<td>24</td>
<td>22</td>
<td>22</td>
<td>0</td>
<td>0 -</td>
</tr>
<tr>
<td>L</td>
<td>29</td>
<td>33</td>
<td>9</td>
<td>9</td>
<td>6 K</td>
</tr>
</tbody>
</table>

Vectors received from J's four neighbors:
- JA delay is 8
- JI delay is 10
- JH delay is 12
- JK delay is 6

New estimated delay from J:
- 8
- 20
- 28
- 20
- 17
- 30
- 18
- 12
- 10
- 0

**Tanenbaum**
Distance Vector Routing

{Kurose & Ross version}
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 \( x \).
Clearly, \( d_v(z) = 5 \), \( d_x(z) = 3 \), \( d_w(z) = 3 \)

B-F equation says:

\[
d_u(z) = \min \{ c(u,v) + d_v(z), \ c(u,x) + d_x(z), \ c(u,w) + d_w(z) \} = \min \{2 + 5, 1 + 3, 5 + 3\} = 4
\]

The node that achieves minimum is next hop in shortest path ➔ forwarding table. Namely, packets from \( u \) destined for \( z \) are forwarded out link between \( u \) and \( x \).
- $D_x(y)$ = 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 $D_v = [D_v(y): y \in N]$
Distance Vector Algorithm

DV Basic idea:

- From time-to-time, each node sends its own distance vector estimate to neighbors.
- Asynchronous
- When a node $x$ receives a new DV estimate from any neighbor $v$, it saves $v$'s distance vector and 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 minor, natural conditions, the estimate $D_x(y)$ converges to the actual least cost $d_x(y)$. 
Distance Vector Algorithm

Iterative,
a synchronous: 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 destination has
changed, notify neighbors
\[ D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\} \]
\[ = \min\{2+0, 7+1\} = 2 \]

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

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>y</td>
<td>\infty</td>
<td>\infty</td>
<td>\infty</td>
</tr>
<tr>
<td>z</td>
<td>\infty</td>
<td>\infty</td>
<td>\infty</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>\infty</td>
<td>\infty</td>
<td>\infty</td>
</tr>
<tr>
<td>y</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>z</td>
<td>7</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>\infty</td>
<td>\infty</td>
<td>\infty</td>
</tr>
<tr>
<td>y</td>
<td>\infty</td>
<td>\infty</td>
<td>\infty</td>
</tr>
<tr>
<td>z</td>
<td>7</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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

\[ = \min\{2+0, 7+1\} = 2 \]

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

\[ = \min\{2+1, 7+0\} = 3 \]

**node x table**

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>from x</td>
<td>0</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>from y</td>
<td>∞</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>from z</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>

**node y table**

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>from x</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>from y</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>from z</td>
<td>7</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**node z table**

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>from x</td>
<td>0</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>from y</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>from z</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Link cost changes:

- node detects local link cost change.
- updates routing info, recalculates distance vector.
- if DV changes, it notifies neighbors.

At time $t_0$, $y$ detects the link-cost change, updates its DV, and informs its neighbors.

"good news travels fast"

At time $t_1$, $z$ receives the update from $y$ and updates its table. It computes a new least cost to $x$ and sends its neighbors its DV.

At time $t_2$, $y$ receives $z$'s update and updates its distance table. $y$'s least costs do not change and hence $y$ does not send any message to $z$. 
Link cost changes:
- good news travels fast
- bad news travels slow - “count to infinity” problem!
- 44 iterations before algorithm stabilizes: see P&D page 248!

Possible solutions:
1. Keep 'infinity' small {depends on graph diameter}.
2. Split Horizon: node does not send those routes learned from a neighbor back to that neighbor.
3. Split Horizon with Poison Reverse:
   - If z routes through y to get to x, z tells y its (z's) distance to x is infinite (so y won’t route to x via z).

- Does this solve count to infinity problem?
Link State Algorithm

1. Each router is responsible for meeting its neighbors and learning their names.

2. Each router constructs a link state packet (LSP) which consists of a list of names and cost to reach each of its neighbors.

3. The LSP is transmitted to ALL other routers. Each router stores the most recently generated LSP from each other router.

4. Each router uses complete information on the network topology to compute the shortest path route to each destination node.
Reliable Flooding

Figure 4.18 Reliable LSP Flooding

(a) X A
   C B D
(b) X A
   C B D
(c) X A
   C B D
(d) X A
   C B D
The process of making sure all the nodes participating in the routing protocol get a copy of the link-state information from all the other nodes.

**LSP** contains:
- Sending router’s node ID
- List of connected neighbors with the associated link cost to each neighbor
- Sequence number
- Time-to-live (TTL) \{an aging mechanism\}
Reliable Flooding

• First two items enable route calculation.

• Last two items make process reliable
  - ACKs and checking for duplicates is needed.

• Periodic **Hello** packets used to determine the demise of a neighbor.

• The sequence numbers are not expected to wrap around.
  - ➔ this field needs to be large (64 bits) !!
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.
- 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 destination \(v\)
- \(p(v)\): predecessor node along path from source to \(v\)
- \(N'\): set of nodes whose least cost path is definitively known.
Dijsktra's Shortest Path Algorithm

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

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 \( N' \) :
12 \[ D(v) = \min (D(v), D(w) + c(w,v)) \]
13 /* new cost to \( v \) is either old cost to \( v \) or known
14 shortest path cost to \( w \) plus cost from \( w \) to \( v \) */
15 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>∞</td>
</tr>
<tr>
<td>2</td>
<td>uxy</td>
<td>2,u</td>
<td>3,y</td>
<td>4,y</td>
<td>4,y</td>
<td>4,y</td>
</tr>
<tr>
<td>3</td>
<td>uxyv</td>
<td>3,y</td>
<td>3,y</td>
<td>4,y</td>
<td>4,y</td>
<td>4,y</td>
</tr>
<tr>
<td>4</td>
<td>uxyvw</td>
<td>3,y</td>
<td>3,y</td>
<td>4,y</td>
<td>4,y</td>
<td>4,y</td>
</tr>
<tr>
<td>5</td>
<td>uxyvwz</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram showing the network with nodes u, v, w, x, y, and z, and edges connecting them with distances labeled on the edges.
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>
Dijkstra's Algorithm, Discussion

**Algorithm complexity:** n nodes
- each iteration: need to check all nodes, w, not in N
- \( n(n+1)/2 \) comparisons: \( O(n^2) \)
- more efficient implementations possible: \( O(n \log n) \)

**Oscillations possible:**
- e.g., link cost = amount of carried traffic

![Dijkstra's Algorithm Diagram](image-url)
Intra-AS Routing

- also known as **Interior Gateway Protocols (IGP)**
- most common Intra-AS routing protocols:
  - **RIP**: Routing Information Protocol
  - **OSPF**: Open Shortest Path First
  - **IGRP**: Interior Gateway Routing Protocol (Cisco proprietary)
Routing Information Protocol (RIP)

- RIP had widespread use because it was distributed with BSD Unix in "routed", a router management daemon in 1982.
- **RIP** - most used Distance Vector protocol.
- RFC1058 in June 1988
- Runs over UDP.
- Metric = hop count
- **BIG problem is max. hop count = 16**
  - RIP limited to running on small networks (or AS's that have a small diameter)!!
Routing Information Protocol (RIP)

- Sends DV packets every 30 seconds (or faster) as Response Messages (also called advertisements).
- Each advertisement: list of up to 25 destination subnets within AS.
- Upgraded to RIPv2

From router A to subnets:

<table>
<thead>
<tr>
<th>destination</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>
**Figure 4.17 RIP Packet Format**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>Must be zero</td>
</tr>
<tr>
<td>Version</td>
<td></td>
</tr>
<tr>
<td>Family of net 1</td>
<td>Address of net 1</td>
</tr>
<tr>
<td>Address of net 1</td>
<td></td>
</tr>
<tr>
<td>Distance to net 1</td>
<td></td>
</tr>
<tr>
<td>Family of net 2</td>
<td>Address of net 2</td>
</tr>
<tr>
<td>Address of net 2</td>
<td></td>
</tr>
<tr>
<td>Distance to net 2</td>
<td></td>
</tr>
</tbody>
</table>

(network_address, distance) pairs
RIPv2

- Allows routing on a subnet (subnet masks)
- Has an authentication mechanism
- Tries to deal with multicast
- Uses route tags
- Has the ability for router to announce routes on behalf of another router.
### RIPv2 Packets

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>Must be zero</td>
</tr>
<tr>
<td>Version</td>
<td></td>
</tr>
<tr>
<td>Family of net 1</td>
<td>Route Tags</td>
</tr>
<tr>
<td>Address prefix of net 1</td>
<td></td>
</tr>
<tr>
<td>Mask of net 1</td>
<td></td>
</tr>
<tr>
<td>Distance to net 1</td>
<td></td>
</tr>
<tr>
<td>Family of net 2</td>
<td>Route Tags</td>
</tr>
<tr>
<td>Address prefix of net 2</td>
<td></td>
</tr>
<tr>
<td>Mask of net 2</td>
<td></td>
</tr>
<tr>
<td>Distance to net 2</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.31 RIPv2 Packet Format
OSPF (Open Shortest Path First)

- “open” :: publicly available (due to IETF)
- uses Link State algorithm
  - LS packet dissemination
  - topology map at each node
  - route computation uses 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).

* However hierarchy (partitioning domains into areas) reduces flooding impact.
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; high for real time).
- **integrated uni-** and **multicast** support:
  - Multicast OSPF (MOSPF) uses same topology data base as OSPF.
- **hierarchical** OSPF used in large domains.
Partitioning Domains

Figure 4.2 A domain divided into areas
Hierarchical OSPF

The image illustrates a hierarchical OSPF network. The network is divided into areas, with the backbone router connecting these areas. The diagram shows:

- **Backbone**: The central network that connects different areas.
- **Area 1, Area 2, Area 3**: Different areas in the network.
- **Border Routers**: Connectors between areas.
- **Internal Routers**: Routers within each area.

The network structure helps in managing and optimizing routing traffic more efficiently.
Hierarchical OSPF

- Two-Level Hierarchy: local area, backbone.
  - Link-State Advertisements (LSAs) only in area
  - each node has detailed area topology; only knows 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.
## OSPF LSA Types

1. Router link advertisement **[Hello message]**
2. Network link advertisement
3. Network summary link advertisement
4. AS border router’s summary link advertisement
5. AS external link advertisement
Figure 5-65. The relation between AS’s, backbones, and areas in OSPF
Internet Inter-AS routing: BGP

- BGP (Border Gateway Protocol): the de facto standard

- BGP provides each AS a means to:
  1. Obtain subnet reachability information from neighboring ASs.
  2. Propagate reachability information to all AS-internal routers.
  3. Determine “good” routes to subnets based on reachability information and policy.

- allows subnet to advertise its existence to rest of Internet: “I am here!”
Routing Primer Summary

- Routers forward and route over WANs
  - Produce look up tables in routers

- Routing Classification:
  - Adaptive or non-adaptive
  - Interdomain and Intradomain

- Distance Vector Routing (DV)
  - Perlman version
  - Tanenbaum example
  - K&R version
Routing Primer Summary

- **Link State Routing (LS)**
  - Uses reliable flooding; Dijkstra’s SP algorithm

- **RIP**
  - Old ARPA routing; unicast DV routing

- **OSPF**
  - Two-Level Hierarchical LS routing
  - Five LSA types for router communication

- **BGP**
  - Interdomain routing using reachability