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 (service classes)
  - Reduction of routing table size
  - Multicast support
  - Support for mobile hosts
  - Support coexistence with other protos, e.g., IPv4

IPv6 datagram format:
- Fixed-length 40 byte 7 element header
- No fragmentation allowed

IPv6 Header

*Version:* IPv4 or IPv6

*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
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
- **Extension header examples**:
  - Routing, fragmentation, authentication, encrypted security payload, destination options

Transition From IPv4 To IPv6

- Not all routers can be upgraded simultaneous
  - No “flag days”
  - How will the network operate with mixed IPv4 and IPv6 routers?
- **Tunneling**: IPv6 carried as payload in IPv4 datagram among IPv4 routers
Network layer: Status

- Network layer functions
- IP
- Routing and forwarding
- NAT
- ARP
- IPv6
- Routing

Internet routing

Our routing study thus far – idealization
- All routers identical
- Network “flat”
... not true in practice

Administrative autonomy
- Internet = network of networks
- Each network admin may want to control routing in its own network
- Aggregate routers into regions, “autonomous systems” (AS)
- Routers in same AS run same routing protocol
  - “Inter-AS” routing protocol
  - Routers in different AS can run different inter-AS routing protocol

Scale: With 200 million destinations:
- Can’t store all dest’s in routing tables!
- Routing table exchange would swamp links!
Interconnected ASes

- Forwarding table is configured by both intra- and inter-AS routing algorithm
  - Intra-AS sets entries for internal dests
  - Inter-AS & Intra-As sets entries for external dests

Inter-AS Tasks

- Suppose router in AS1 receives datagram for dest outside of AS1
  - Router should forward packet towards an AS-border router, but which one?

**AS1 needs ...**

1. ... to learn which dests are reachable through AS2 and which through AS3
2. ... to propagate this reachability info to all routers in AS1

Job of inter-AS routing!
Intra-AS routing

- Also known as Interior Gateway Protocols (IGP)
- Most common Intra-AS routing protocols:

  - **RIP:** Routing Information Protocol
    - Distance vector protocol (based on Bellman-Ford)
    - Routers periodically exchange reachability info with their neighbors
    - Distance metric: hop count
    - Advantage: simple, minimal communication overhead
    - Disadvantage: long convergence times, loop detection

Intra-AS routing protocols

- **OSPF:** Open Shortest Path First
  - Link state protocol (based on Dijkstra)
  - Routers periodically flood immediate reachability information to all other routers
  - Distance metric: administrative weight
  - Advantage: fast convergence
  - Disadvantage: complexity and communication overhead

- **ISIS:** Intermediate-System-to-Intermediate-System (ISO 10589) (link state)

- **IGRP:** Interior Gateway Routing Protocol (Cisco proprietary) (distance vector)

- **EIGRP:** Enhanced Interior Gateway Routing Protocol (Cisco proprietary) (enhanced distance vector)
Interplay between routing and forwarding

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) \}

Path: Sequence of edges (routers)

Remark: Graph abstr. is useful in other network contexts
Example: P2P, where \( N \) is set of peers
and \( E \) is set of TCP connections
Graph abstraction: Costs

- $c(x,x') =$ cost of link $(x,x')$
  - e.g., $c(w,z) = 5$
- Cost can be always 1, or inversely related to bandwidth, or inversely related to congestion

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: alg. that finds “good” path (typically: least cost path)

Routing algorithm classification

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

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
- Computes least cost paths from one node (‘source”) to all other nodes
  - Gives routing table for that node
- Iterative: after \( k \) iterations, know least cost path to \( k \) dest.’s

Notation:

- \( c(i,j) \): Link cost from node \( i \) to \( j \). Cost infinite 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 \( v \)
- \( N' \): Set of nodes whose least cost path definitively known

Dijsktra’s algorithm

1. Initialization for \( A \):
2. \( N' = \{A\} \)
3. for all nodes \( v \)
4. if \( v \) adjacent to \( A \)
5. then \( D(v) = c(A,v) \)
6. else \( D(v) = 1 \)
7. 
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 shortest path cost to \( w \) plus cost from \( w \) to \( v \) */
14. 
15. until all nodes in \( N' \)
Dijkstra’s algorithm: Example

<table>
<thead>
<tr>
<th>Step</th>
<th>start N’</th>
<th>D(B),p(B)</th>
<th>D(C),p(C)</th>
<th>D(D),p(D)</th>
<th>D(E),p(E)</th>
<th>D(F),p(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>2,A</td>
<td>5,A</td>
<td>1,A</td>
<td>infinity</td>
<td>infinity</td>
</tr>
<tr>
<td>1</td>
<td>AD</td>
<td>2,A</td>
<td>4,D</td>
<td>2,D</td>
<td>infinity</td>
<td>infinity</td>
</tr>
<tr>
<td>2</td>
<td>ADE</td>
<td>2,A</td>
<td>3,E</td>
<td>4,E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ADEB</td>
<td>2,A</td>
<td>3,E</td>
<td>4,E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ADEBC</td>
<td>2,A</td>
<td>3,E</td>
<td>4,E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ADEBCF</td>
<td>2,A</td>
<td>3,E</td>
<td>4,E</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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>B</td>
<td>(A,B)</td>
</tr>
<tr>
<td>D</td>
<td>(A,D)</td>
</tr>
<tr>
<td>E</td>
<td>(A,D)</td>
</tr>
<tr>
<td>C</td>
<td>(A,D)</td>
</tr>
<tr>
<td>F</td>
<td>(A,D)</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

Distance vector algorithm

Bellman-Ford Equation (dynamic programming)

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

Then
\[
d_x(y) = \min \{ c_v(x,v) + d_v(y) \}
\]

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

Clearly, \( d_u(z) = 5 \), \( d_x(z) = 3 \), \( d_w(z) = 3 \)

Bellman-Ford 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 yields minimum is next hop in shortest path \( \rightarrow \) forwarding table

---

**Distance vector algorithm (2)**

- \( D_x(y) \) = estimate of least cost from \( x \) to \( y \)
- Distance vector: \( D_x = [D_x(y): y \in N] \)
- Node \( x \) knows cost to each neighbor \( y \): \( c(x,y) \)
- Node \( x \) maintains \( D_x = [D_x(y): y \in N] \)
- Node \( x \) also maintains its neighbors’ distance vectors
  - For each neighbor \( y \), \( x \) maintains \( D_y = [D_y(y): y \in N] \)
Distance vector algorithm (3)

Basic idea:
- Each node periodically sends its own distance vector estimate to neighbors
- 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 “natural” conditions the estimates of $D_x(y)$ converge to the actual least cost $d_x(y)$

Distance vector algorithm (4)

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

Distributed:
- Each node notifies neighbors only when its Distance Vector changes
  - Neighbors then notify their neighbors if necessary

Each node:
- \textbf{wait} for (change in local link cost of msg from neighbor)
- \textbf{recompute} estimates
- if Distance Vector to any dest has changed, \textbf{notify} neighbors
### Distance vector routing: Overview

**Iterative, asynchronous:**
- each local iteration caused by:
  - Local link cost change
  - Message from neighbor: its least cost path change from neighbor

**Distributed:**
- Each node notifies neighbors
  - only when its least cost path to any destination changes
  - Neighbors then notify their neighbors if necessary

**Each node:**
- **wait** for (change in local link cost of msg from neighbor)
- **recompute** distance table
- if least cost path to any dest has changed, **notify** neighbors

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

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

At each node, $x$:

1. Initialization:
2. for all destinations $y$ in $N$:
   3. $D_x(y) = 1$ if $y$ is not a neighbor
   4. $D_x(y) = c(x,y)$ if $y$ is a neighbor
5. for each neighbor $w$
6. $D_w(y) = 1$ for all destinations $z$ in $N$
7. for each neighbor $w$
8. send distance vector $D_x = [D_x(y) \mid y \in N]$ to $w$

Distance vector algorithm (2.):

9. loop
10. wait (until I see a link cost change to neighbor $w$
11. or until I receive update from neighbor $w$)
12. for each $y$ in $N$:
13. $D_x(y) = \min_v \{c(x,v) + D_v(y)\}$
14. if $D_x(y)$ changed for any destination $y$
15. send DV $D_x = [D_x(y) \mid y \in N]$ to all neighbors
16. forever
Distance vector (DV): Link cost changes

Link cost changes:
- Node detects local link cost change
- Updates routing info, recalculates distance vector
- If DV changes, notify neighbors

“Good news travels fast”

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

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 $z$. 

<table>
<thead>
<tr>
<th>Node Y Table</th>
<th>Cost To</th>
<th>Node Z Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>From</td>
<td>X Y Z</td>
<td>From</td>
</tr>
<tr>
<td>Y</td>
<td>0 1</td>
<td>1 0 1</td>
</tr>
<tr>
<td>Z</td>
<td>5 1 0</td>
<td>5 1 0</td>
</tr>
<tr>
<td>Node X Table</td>
<td>Cost To</td>
<td>Node Y Table</td>
</tr>
<tr>
<td>--------------</td>
<td>---------</td>
<td>--------------</td>
</tr>
<tr>
<td>From</td>
<td>X Y Z</td>
<td>From</td>
</tr>
<tr>
<td>Y</td>
<td>4 0 1</td>
<td>1 0 1</td>
</tr>
<tr>
<td>Z</td>
<td>5 1 0</td>
<td>2 1 0</td>
</tr>
</tbody>
</table>
Distance vector: Link cost changes (2.)

Link cost changes:
- Good news travels fast
- Bad news travels slow

\[ Dy(x) = \min\{c(y,x) + Dx(x), c(y,z) + Dz(x)\} \]
\[ = \min\{60 + 0, 1 + 5\} = 6 \]

\[ Dy(x) = \min\{c(y,x) + Dx(x), c(y,z) + Dz(x)\} \]
\[ = \min\{60 + 0, 1 + 7\} = 8 \]
**Distance vector: Link cost changes (3.)**

- Good news travels fast
- Bad news travels slow - “count to infinity” problem!
  - 44 iterations before algorithm stabilizes: see text
- What happens here?

**Poissoned 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)
- Will this completely solve count to infinity problem?

**Comparison of LS and DV algorithms**

**Message complexity**
- **LS:** with $n$ nodes, $E$ links, $O(nE)$ msgs sent each
- **DV:** $O(d)$ messages, many times
  - $d$ is node degree

**Speed of Convergence**
- **LS:** $O(n \log n)$ algorithm requires $O(nE)$ msgs
  - 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:**
- Node can advertise incorrect path cost
- Each node’s table used by others; error propagate through network
Internet Inter-AS routing: BGP

- The de facto standard: Border Gateway Protocol (BGP)
- BGP provides each AS a means to:
  1. Obtain subnet reachability information from neighboring ASs
  2. Propagate reachability information to all routers in the AS
  3. Determine “good” routes to subnets based on reachability information and routing policy.
- Allows a subnet to advertise its existence to rest of the Internet: "I am here"
- Issues:
  - Which routing algorithm?
  - How are routes advertised?
  - How to implement routing policies?

BGP Basics

- Pairs of routers (BGP peers) exchange routing info over semi-permanent TCP connections: BGP sessions
- Note that BGP sessions do not correspond to physical links.
- When AS2 advertises a prefix to AS1, AS2 is promising it will forward any datagrams destined to that prefix towards the prefix.
  - AS2 can aggregate prefixes in its advertisement
BGP is a path vector protocol

- Distance vector algorithm with extra information
  - Two important attributes:
    - **AS-PATH**: contains all ASs along the way: AS 67 AS 17
    - **NEXT-HOP**: Indicates the specific internal-AS router to next-hop AS.
  - Path can be used to make routing decisions, e.g., to avoid loops
  - Pure distance vector does not enable policies
  - Link state does not scale and exposes policies
- When advertising a prefix, advert includes BGP attributes
  - Prefix + other attributes = “route”
- When gateway router receives route advertisement, uses ingress filters to accept/decline
  - Can make decision based on ASes on path, e.g., to avoid loops

BGP messages

- Peers exchange BGP messages using TCP
  - **OPEN**:
    - Opens TCP conn. to peer
    - Authenticates sender
  - **UPDATE**:
    - Advertises new routes (or withdraws old)
  - **KEEPALIVE**:
    - Keeps conn alive in absence of UPDATES, ACKs OPEN request
  - **NOTIFICATION**:
    - Reports errors in previous msg; closes a connection

Process:
- **Initialization**: Open ⇒ Updates for all routes
- **Ongoing**: Updates for changed routes
**BGP route processing**

- Receive BGP Updates
- Apply Policy = filter routes & tweak attributes
- Based on Attribute Values
- Best and Alternate Routes
- Apply policies to Best Routes!
- Transmit BGP Updates

**Routing policy**

- Reflects goals of network provider
  - Which routes to accept from other ASes
  - How to manipulate the accepted routes
  - How to propagate routes through network
  - How to manipulate routes before they leave the AS
  - Which routes to send to another AS
Routing policy: Examples

- Honor business relationships
  (E.g., customers get full-table; peers only customer prefixes)
  (E.g., prefer customer routes over peer routes over upstream routes)

- Allow customers a choice of route
  (E.g., on customer request do not export prefix to AS x, etc.)

- Enable customer traffic engineering
  (E.g., prepend x times to all peers or to specified AS)

- Enable DDoS defense for customers
  (E.g., blackholing by rewriting the next hop)

- ...

BGP routing policy

- A, B, C are provider networks
- X, W, Y are customer (of provider networks)
- X is dual-homed: attached to two networks
  - X does not want to route from B via X to C
  - .. so X will not advertise to B a route to C
BGP routing policy (2.)

- A advertises to B the path AW
- B advertises to X the path BAW
- Should B advertise to C the path BAW?
  - No way! B gets no "revenue" for routing CBAW since neither W nor C are B's customers
  - B wants to force C to route to W via A
  - B wants to route only to/from its customers!

Local preference attribute

- Path with highest local preference wins
- Allows providers to prefer routes
BGP route selection

- Router learn > 1 route to some prefix
- Router must select best route.
- **Elimination rules:**
  1. Local preference value attribute: policy decision
  2. Shortest AS-PATH
  3. Best MED (multi-exit-discriminator)
  4. Closest NEXT-HOP router: hot potato routing
  5. Additional criteria
  6. IP address of peer

Different types of ASes

- **Providers:** Offer connectivity to direct customer
  offer transit to other ISPs
- **Customers:** Buy connectivity from providers
- **Peers:** Exchange customers traffic at no cost
- **Siblings:** others

<table>
<thead>
<tr>
<th></th>
<th>Own Routes</th>
<th>Customer’s Routes</th>
<th>Sibling’s Route</th>
<th>Provider’s Route</th>
<th>Peer’s Route</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exporting to a Provider</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Exporting to a Customer</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Exporting to a Peer</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>
**OSPF (Open Shortest Path First)**

- “Open”: Specification publicly available
- Uses the Link State algorithm
  - State: per router info about itself and attached networks
  - OSPF advertisements: propagates state
  - Link state database: state of all routers
  - Topology map: derived from link state database

**OSPFv2: Components**

- Who is my neighbor?
  - Hello Protocol
- With whom I want to talk? (LAN!!)
  - Designated router/Backup designated router concept
- What info am I missing?
  - Database synchronization
- How do I distribute info?
  - Advertisements disseminated to entire Autonomous System (via reliable flooding)
  - OSPF messages directly over IP (rather than TCP or UDP)
- Route computation
  - From link state database with Dijkstra’s algorithm
  - Supports equal-cost path routing
**OSPF “advanced” features**

- **Security**: All OSPF messages are authenticated (to prevent malicious intrusion)
- **Multiple same-cost paths** allowed
- For each link, multiple cost metrics for different TOS (eg, 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** in large domains

---

**Hierarchical OSPF**

[Diagram showing hierarchical OSPF with areas and routers]
Hierarchical OSPF (2.)

- Two-level hierarchy: Local area and backbone.
  - Link-state advertisements only in respective areas.
  - Nodes in each area have detailed area topology; only know direction (shortest path) to networks in other areas.
- Area Border routers “summarize” distances to networks in the area and advertise them to other Area Border routers.
- Backbone routers: Run an OSPF routing algorithm limited to the backbone.
- Boundary routers: Connect to other ASs.

Why different Intra- and Inter-AS routing?

Policy:
- Inter-AS: Admin wants control over how its traffic routed, who routes through its net.
- Intra-AS: Single admin, so no policy decisions needed

Scale:
- Hierarchical routing saves table size, reduced update traffic

Performance:
- Intra-AS: Can focus on performance
- Inter-AS: Policy may dominate over performance

We need BOTH!