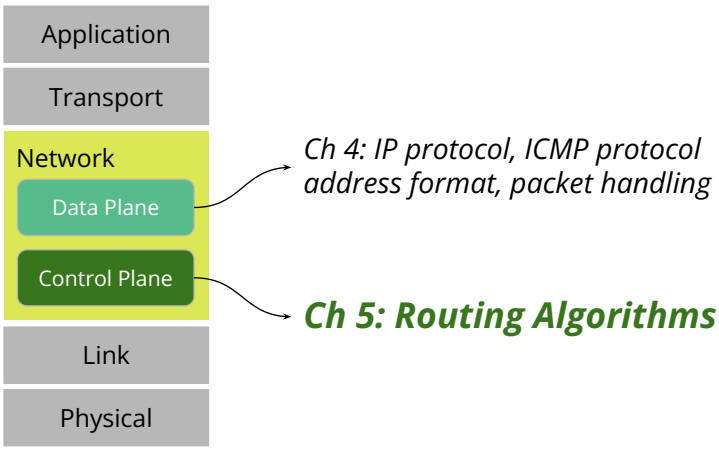
Chapter 05
Network Layer

Control Plane

Network Layer: Data and Control Planes



Network Layers

Two important planes & functions

- **Data Plane**: actions inside each *individual routers* (forwarding)
- **Control Plane**: “coordinated” actions among all the routers (routing)
  - Traditional routing algorithms
    - Centralized: Dijkstra like algorithm (link state)
    - Distributed: Bellman-Ford algorithm (distance vector)
  - Network management
  - Network configuration

Routing Algorithms:
How To Find Best Paths
From One Source to Many Destinations

Road Navigation

These apps have a ***global map*** of the entire world!
Current traffic conditions

Routing Algorithms (Executed by Routers)

- Goal: compute “good” paths/routes from senders to receivers
  - *Not just a single path!*
  - Good: “cheapest”, “fastest”, “shortest”, “least congested”, ...
- Two well-known algorithms
  - Dijkstra like based on **link state**
  - Bellman-Ford based on **distance vector**
- Modeled using a weighted graph
  - Nodes/Vertices are routers
  - Edges are network links
  - Weights are the cost of using (direct) links. Interpretation of cost: time, level of congestion, number of hops, etc.

Dijkstra vs. Bellman Ford

	Dijkstra	**Bellman-Ford**
Execution Mode	Centralized	Decentralized
Information Needed	Each router requires the complete graph	Each router needs to know only its immediate neighbors
Key Step in each iteration	Improve the cost only to neighbors of the best vertex	Improve the cost over all edges in the graph
Advantage	Globally Faster	Globally Slower
Local advantage	*None: every node runs the same amount of work*	*Faster: workload of each node depends on number of neighbors*
Limitation	Can't handle negative edges	Can handle negative edges

Link State

(Current) State of a link:

- Is it up or down?
- Its IP address and network mask
- What type of network it is connected to

Dijkstra Link-State Algorithm

- Centralized, requires knowledge of the entire network topology
- Iterative, computes the least cost from a SINGLE source node (u) to ALL other destinations nodes
  - The output can be used as the forwarding table at u
  - Each router runs the Dijkstra algorithm using its own node as the source to compute its forwarding table
- After k iterations, the algorithm knows the least cost path to k destinations
- Route Oscillation is possible when cost is computed based on dynamic properties (such as the current congestion level, the current amount of traffic)

Shortest Path Algorithm (1958)

```
// Shortest paths from source S to all other nodes
function Dijkstra(S) {
    for each vertex v {
        dist[v] = INFINITY
        pred[v] = UNDEFINED
        add v to Unvisited
    }
    dist[S] = 0
    while Unvisited is not empty {
        u = vertex in Unvisited with minimum dist[u]
        remove u from Unvisited
        for each neighbor v of u in Q {
            alt = dist[u] + cost(u,v)
            if alt < dist[v] {
                dist[v] = alt
                pred[v] = u
            }
        }
    }
}
```

Dijkstra Visualization

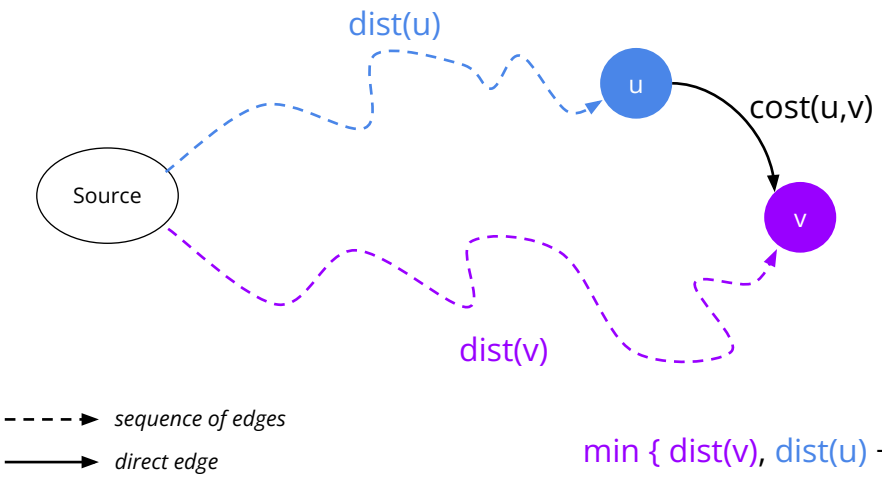
```
// Compute shortest paths from source S to all other nodes
function BellmanFord(S) {
    for each vertex v {
        dist[v] = INFINITY
        pred[v] = UNDEFINED
    }

    dist[S] = 0
    repeat M-1 times { // M is the number of vertices


        for each edge (u,v) {
            alt = dist[u] + cost(u,v)
            if alt < dist[v] {
                dist[v] = alt
                pred[v] = u
            }
        }
    }
}
```

only this portion runs in each router

Shortcuts in Dijskstra/Bellman-Ford



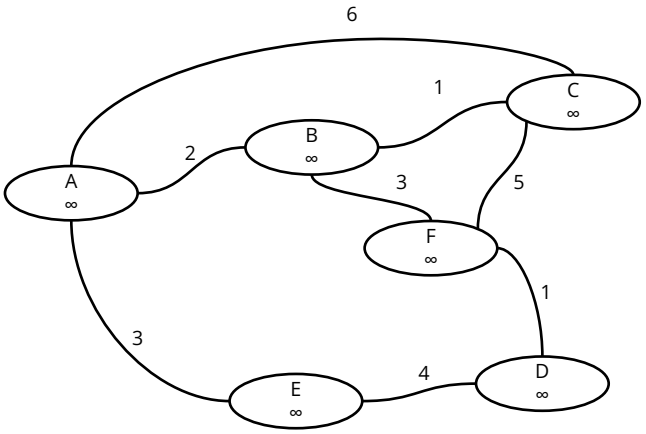
which route is better to reach v?

min { dist(v), dist(u) + cost(u,v) }

Bellman-Ford Algorithm

- Decentralized / distributed algorithm
  - Each node/router computes its best path to the destination (“distance vector”) and broadcast this information to all its neighbors
  - Upon receiving a set of distance vectors (from its neighbors), a node updates its best path calculation (and send the updated cost to its neighbors)
- Based on dynamic programming approach
- Iterative
  - At time 0 every node has the distance vector available on at itself
  - At (the end of) time 1 the distance vector of a node has propagated to nodes 1 hop away from that node
  - At (the end of) time k the distance vector of a node has propagated to nodes k hops away from that node

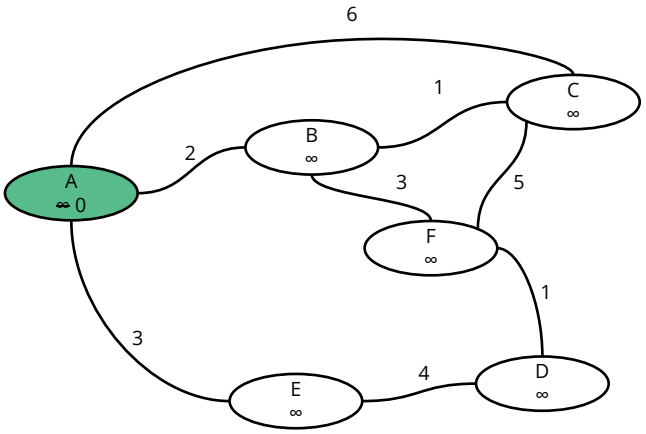
Dijkstra Shortest Path Example



Node	dist	pred
A	∞	?
B	∞	?
C	∞	?
D	∞	?
E	∞	?
F	∞	?

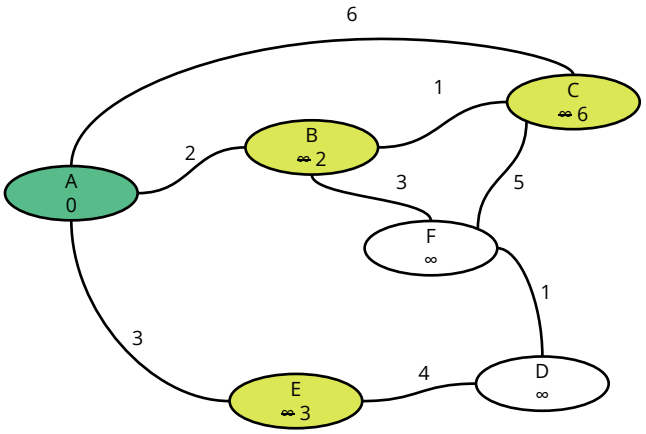
Dijkstra Shortest Path Example

Source A



Node	dist	pred
A	0	None
B	∞	?
C	∞	?
D	∞	?
E	∞	?
F	∞	?

Dijkstra Shortest Path Example

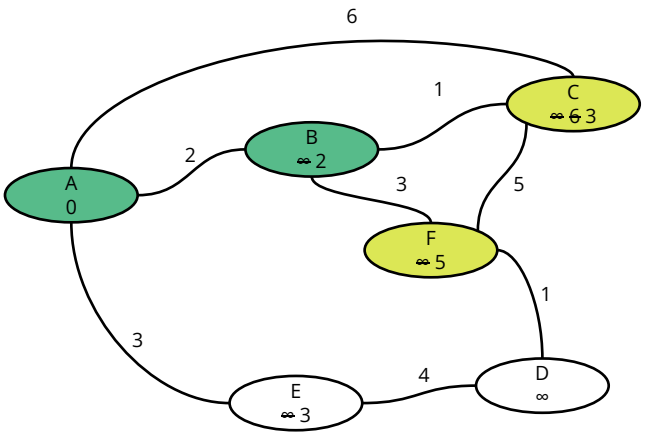


A *unvisited* neighbors: B, C, E

Node	dist	pred
A	0	None
B	~~∞~~ 2	A
C	~~∞~~ 6	A
D	∞	?
E	~~∞~~ 3	A
F	∞	?

Best *unvisited* neighbor: B

Dijkstra Shortest Path Example

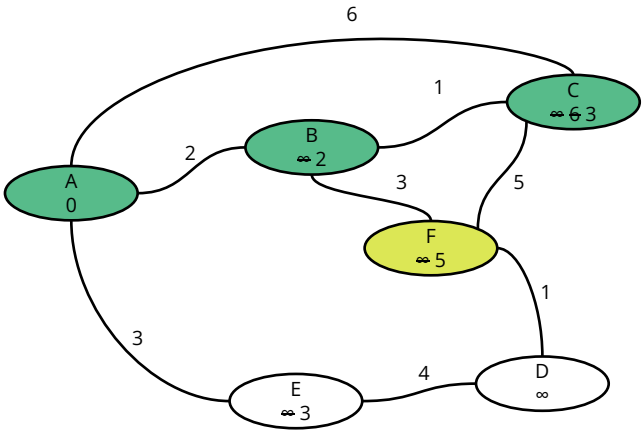


B *unvisited* neighbor(s): C, F

Node	dist	pred
A	0	None
B	2	A
C	6 ⇒ 3	~~A~~ B
D	∞	?
E	3	A
F	∞ ⇒ 5	B

Best *unvisited* node: C (or E)

Dijkstra Shortest Path Example

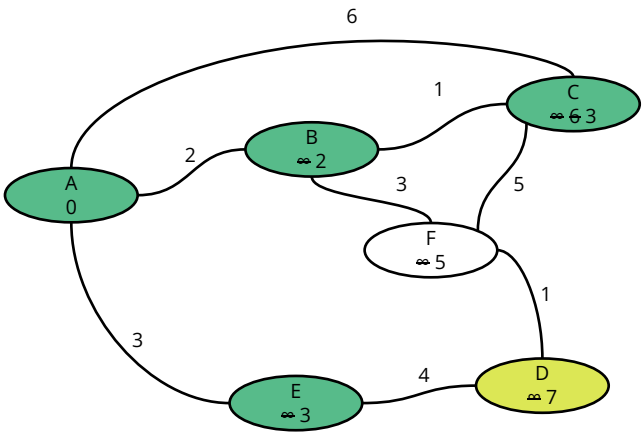


C *unvisited* neighbor(s): F

Node	dist	pred
A	0	None
B	2	A
C	3	B
D	∞	?
E	3	A
F	5	B

F unchanged
Best *unvisited* node: E

Dijkstra Shortest Path Example

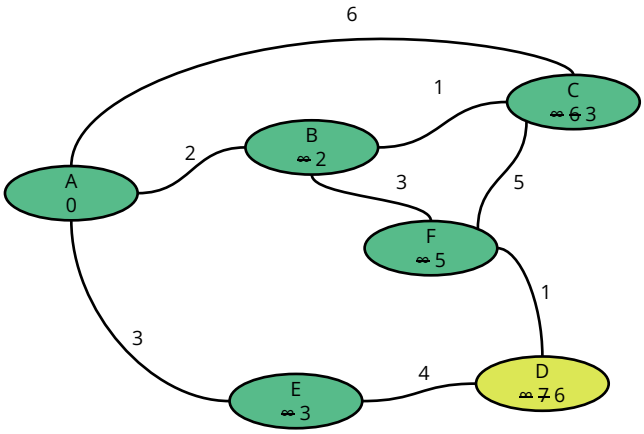


E *unvisited* neighbor(s): D

Node	dist	pred
A	0	None
B	2	A
C	3	~~A~~ B
D	∞ ⇒ 7	E
E	3	A
F	5	B

Best *unvisited* node: F

Dijkstra Shortest Path Example

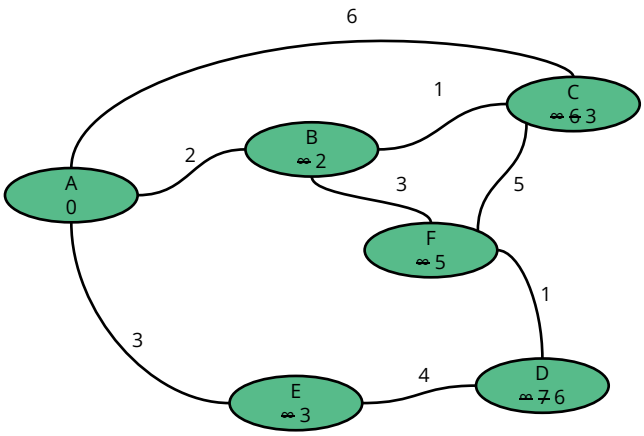


F *unvisited* neighbor(s): D

Node	dist	pred
A	0	None
B	2	A
C	3	~~A~~ B
D	~~7~~ 6	~~E~~ F
E	3	A
F	5	B

Best *unvisited* node: D

Dijkstra Shortest Path Example

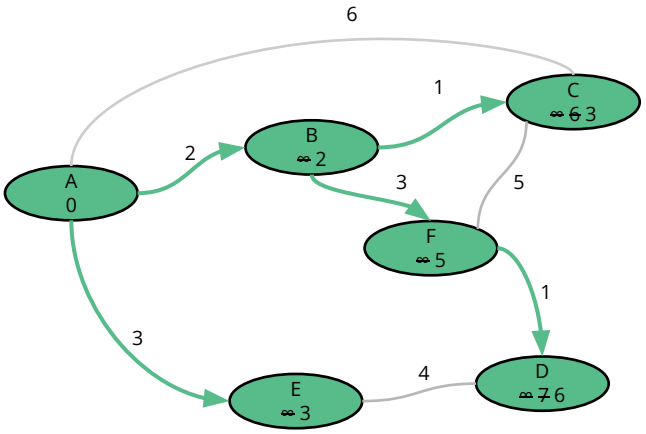


D *unvisited* neighbor(s): None

Node	dist	pred
A	0	None
B	2	A
C	3	~~A~~ B
D	~~7~~ 6	~~E~~ F
E	3	A
F	5	B

Best *unvisited* node: D

Dijkstra Shortest Path Example

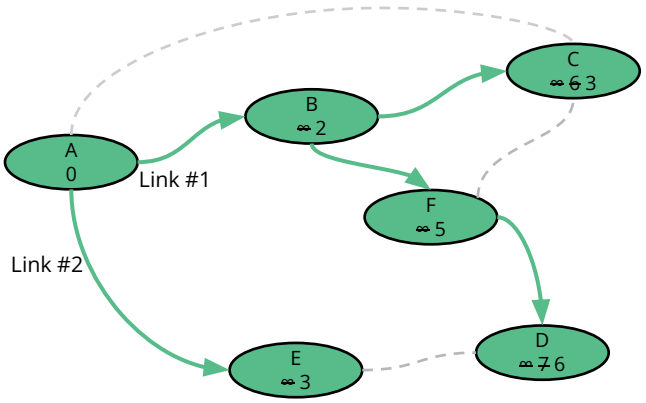


D *unvisited* neighbor(s): None

Node	dist	pred
A	0	None
B	2	A
C	3	~~A~~ B
D	~~7~~ 6	~~E~~ F
E	3	A
F	5	B

Best *unvisited* node: None

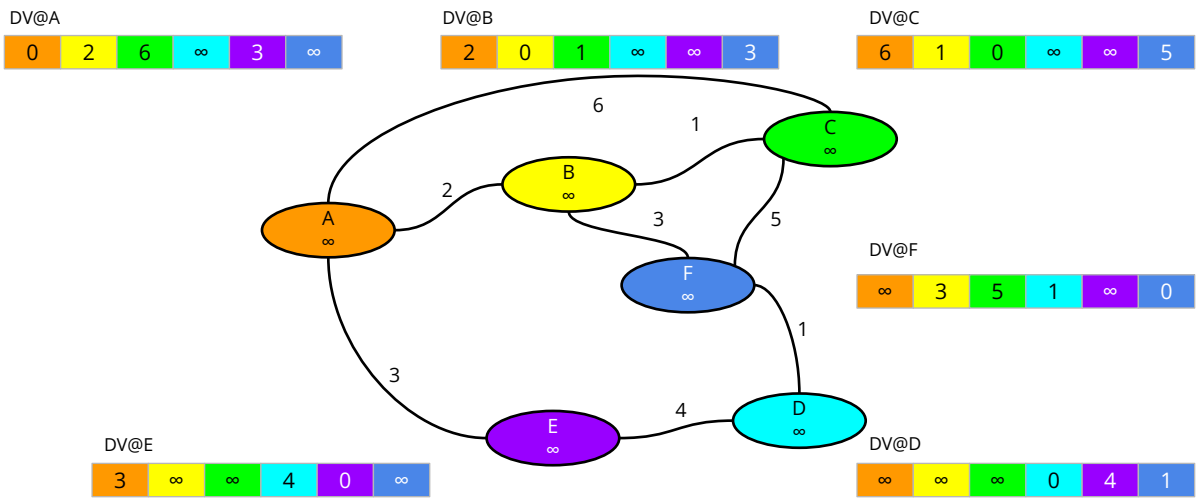
Forwarding Table for Node A



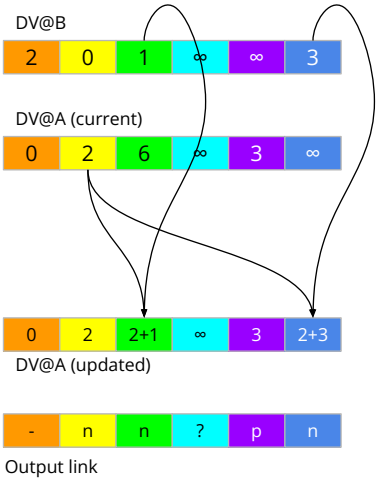
Dest	Path	Output Link
B	A⇒B	#1
C	A⇒B⇒C	#1
D	A⇒B⇒F⇒D	#1
E	A⇒E	#2
F	A⇒B⇒F	#1

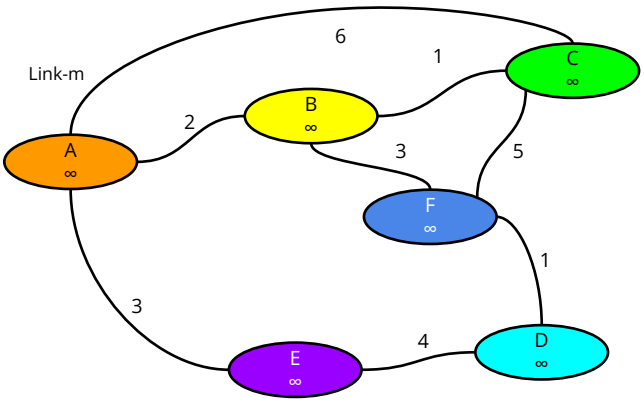
Bellman-Ford Algorithm

Bellman-Ford: Initial Distance Vectors

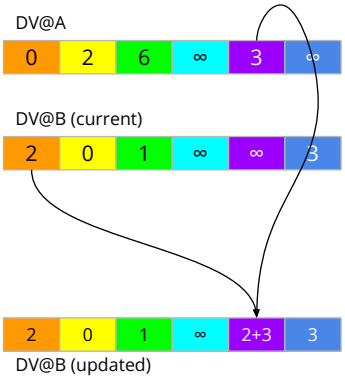


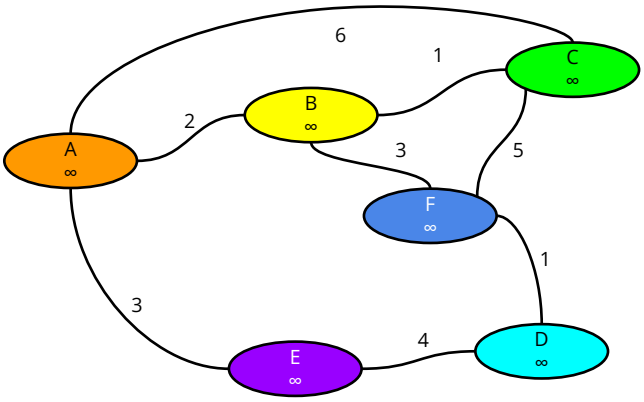
Bellman-Ford: A receives update from B (thru Link-n)



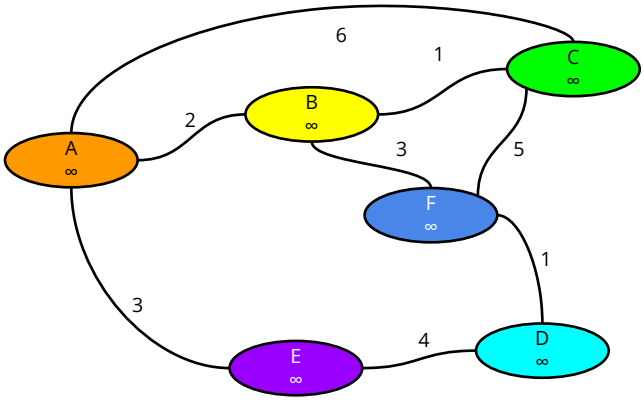


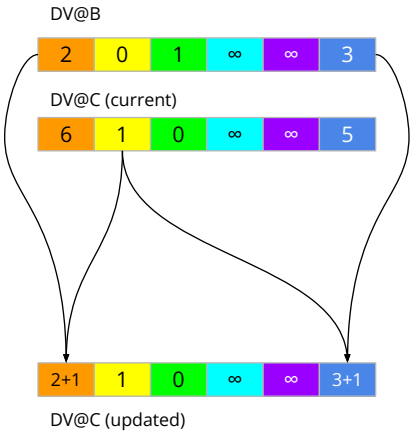
Bellman-Ford: B receives update from A



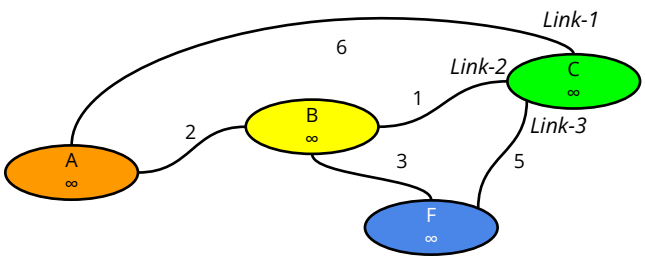


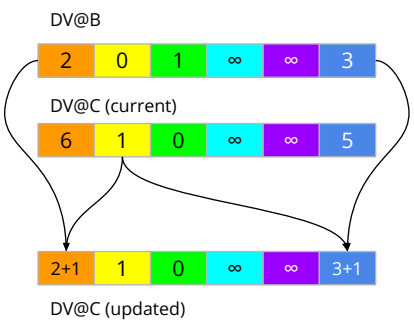
Bellman-Ford: C receives update from B





Bellman-Ford: Building Forwarding Table





Dest	Output Link (current)	Output Link (updated)
A	Link-1	Link-2
F	Link-3	Link-2

update from B arrives at C from Link-2

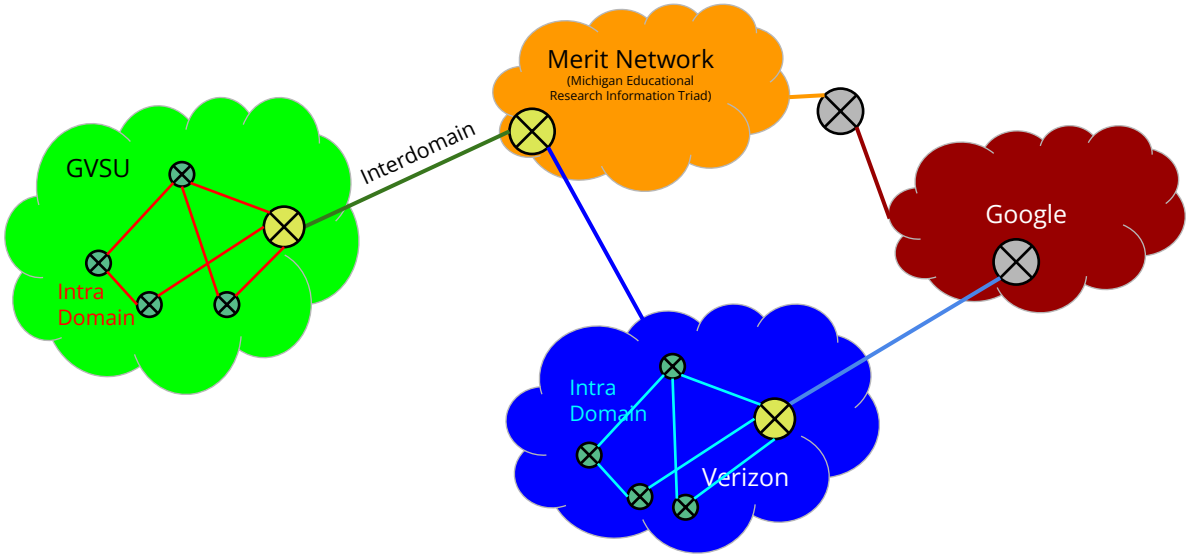
Bellman-Ford: Actions per Node

```
repeat {
    wait for update from neighbors
    recalculate my distanceVector
    readjust my routing table
    if my distanceVector has changed {
        send my distanceVector to my neighbors
    }
}
```

Implementation Issues

- Both Dijkstra & Bellman-Ford algorithms require routers to exchange information to neighbors
- **Issue**: Running either algorithm on a huge network is NOT scalable
- **Solution**
  - split the network into groups/domain/regions/autonomous systems
    - *typically* one AS per organization
      - GVSU Allendale, GVSU Pew Campus, AT&T, Comcast, Verizon, etc.
  - distinguish between intra-domain routing and inter-domain routing
    - Intra-domain: GVSU Wifi (in MAK) and GVSU Wifi (in Fieldhouse)
    - Inter domain: Verizon and GVSU network

Intra-Domain vs Inter-Domain



Intra Domain Routing

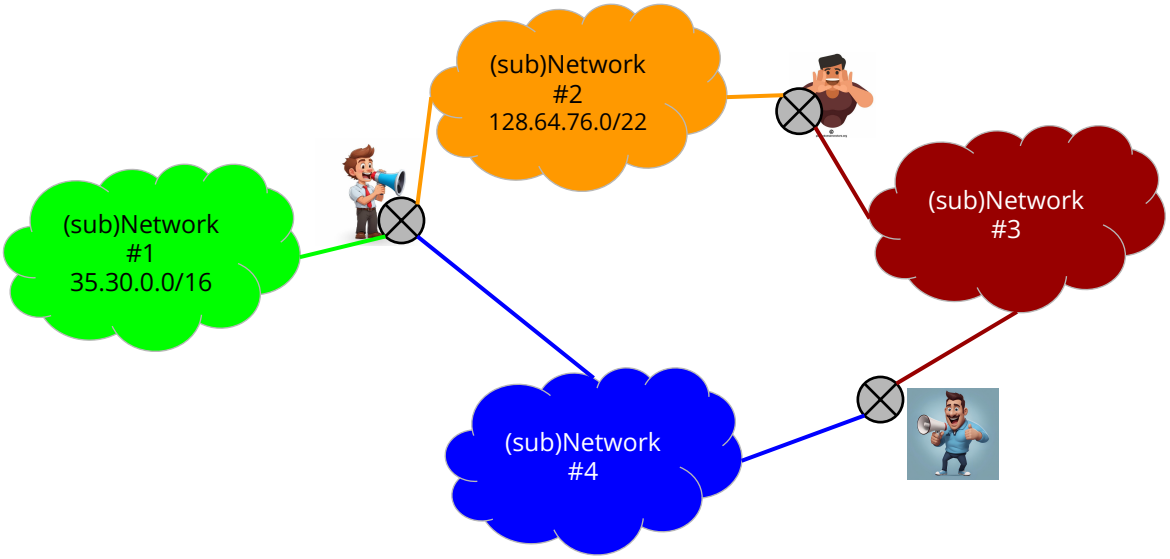
- RIP (RFC1723 Nov 1994)
  - Routers exchange distance vectors every 30 seconds
  - Limited to 50 hops
  - Link cost is solely based on hop counts, not actual propagation time
  - No longer widely used
- Enhanced Interior Gateway Routing Protocol (RFC7868)
  - Based on distance vector (Bellman-Ford algorithm)
- OSPF: Link-State Dijkstra Shortest Path Algorithm
  - Version 2: RFC2328 (1998) IPv4 only
  - Version 3: RCC5340 (2008) IPv6

OSPF

OSPF General Idea

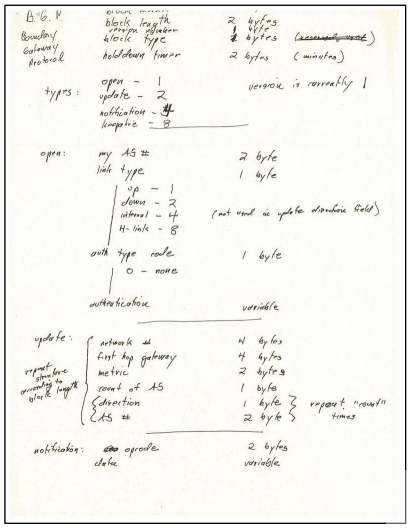
- Initialization/Flood Stage
  - Announce your presence (Link State Advertisement) to your neighbors (HELLO)
  - Receives announcement from your neighbors
- Build The Map (of the “area”) ⇒ Link State Database
  - Re-announce when self changes (LS Update)
  - Request specific details from neighbors (LS Request & LS Ack)
- Routing Stage
  - Execute the Dijkstra Shortest Path Algorithm

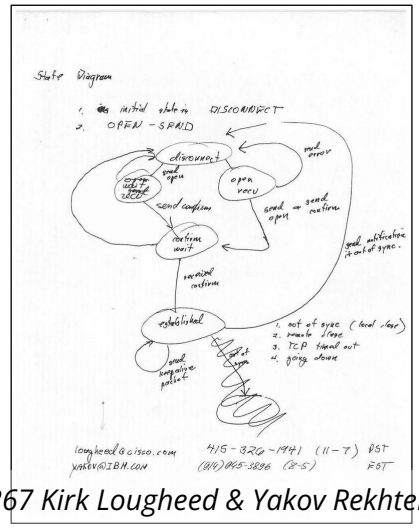
Link State **Announcements**



Inter-Domain Routing
Border Gateway Protocol (BGP)

BGP on two napkins over conference lunch break



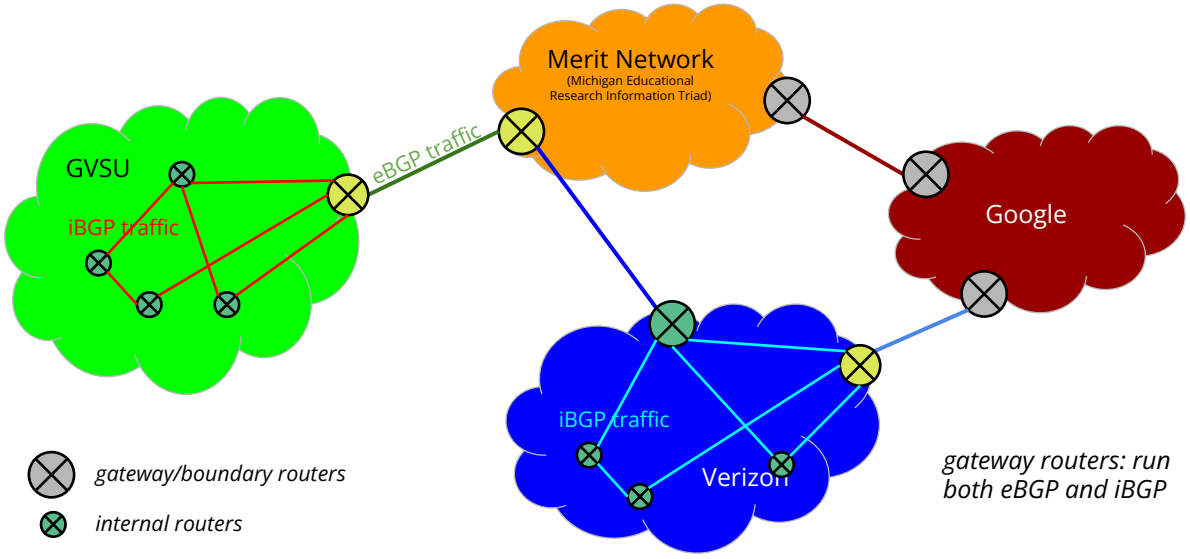


RFC1267 Kirk Lougheed & Yakov Rekhter

BGP

- BGP-3 RFC1267 (Oct 1991) “BGP on Napkins”
- BGP-4 RFC 4271 (Jan 2006)
- Built on-top of TCP (Port 179)
- Inter Autonomous System Routing Protocol
  - Autonomous System (AS) = “Domains”
- Like a coin, BGP has two sides
  - External (eBGP) : object reachability info from neighboring autonomous systems
  - Internal (iBGP): propage reachability info to all the routers within the autonomous system

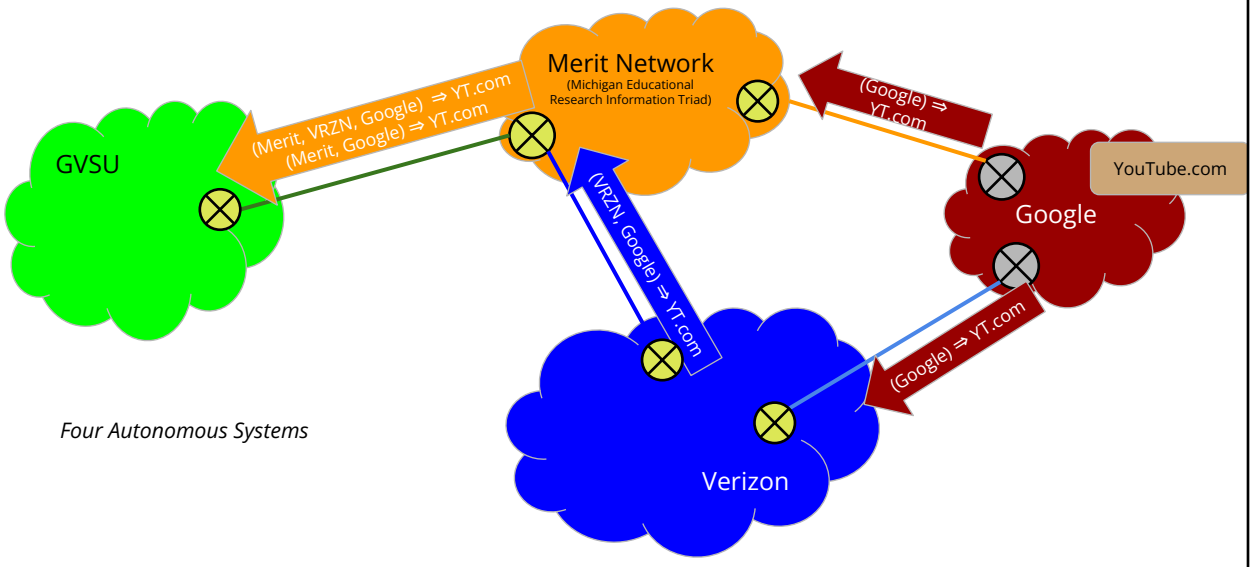
BGP: External BGP & Internal BGP



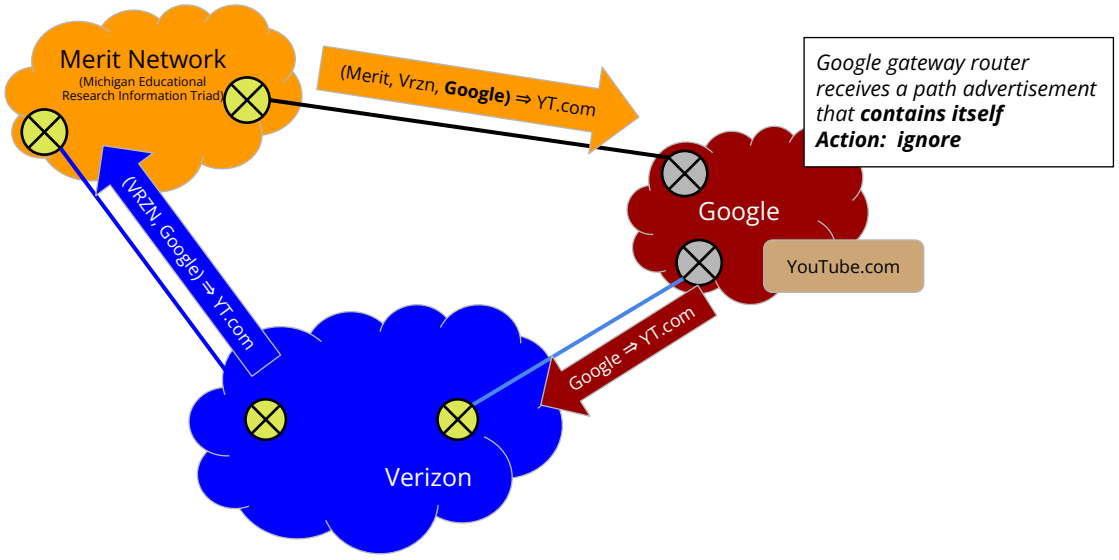
BGP: AS Path Vector & Routing Policy

- Distance vectors in Bellman-Ford algorithm contain **numbers** representing cost to reach particular nodes
- Vectors advertised in BGP contain AS *path to reach particular nodes*
  - Hence BGP is also called “Path Vector” protocol
- A router can decide whether path details in incoming advertisement will be *re-advertised* to its neighbors or *filtered out* altogether
  - Real world examples
    - Verizon network may **not** want to carry transit traffic from AT&T
    - Verizon customers may have to pay ROAMING charges when their traffic are routed via other provider

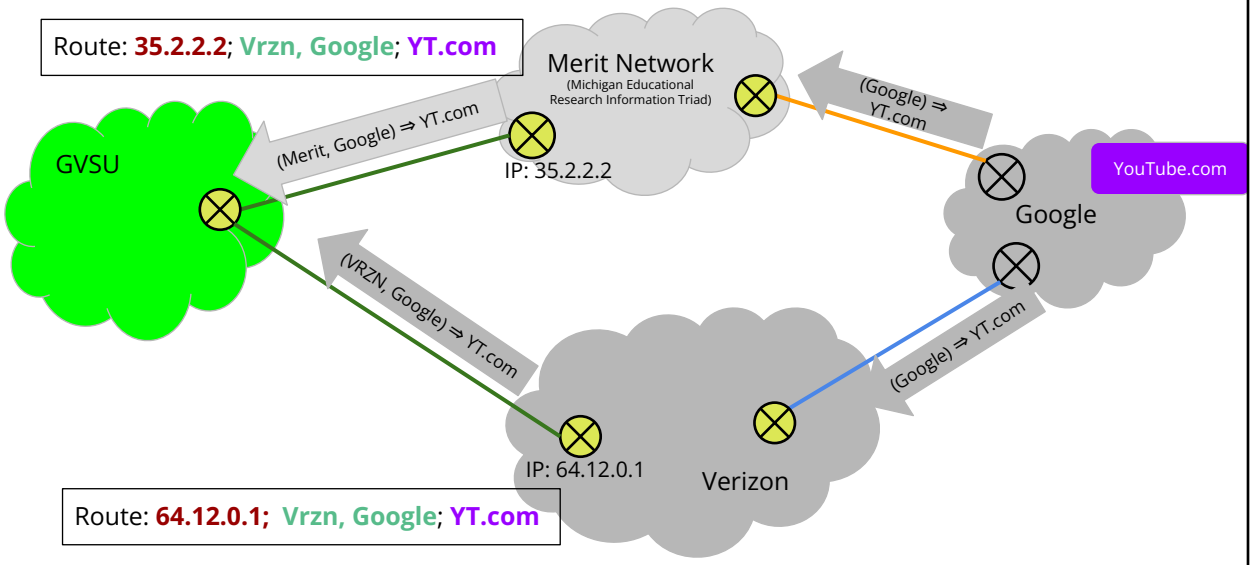
eBGP: **AS Path** Advertisement from Google to GVSU



eBGP: **Loop Detection** in Path Advertisement



BGP Routes: Next-Hop + AS-Path + Destination



BGP: Route Selection Criteria

1. Prefer shorter AS path
2. Among the paths with the same AS Path length, prefer closer Next Hop

Route: **35.2.2.2**; **Vrzn, Merit, Google**; **YT.com** ❌: Longer AS Path

Route: **35.2.2.2**; **Vrzn, Google**; **YT.com** ✔: Shorter AS Path

Route: **35.2.2.2**; **Vrzn, Google**; **YT.com**

Route: **64.12.0.1**; **Vrzn, Google**; **YT.com**

Choose route with shorter RTT

RTT(35.2.2.2) vs. RTT(64.12.0.1)

ICMP
Internet Control Message Protocol

ICMP: Internet Control Message Protocol

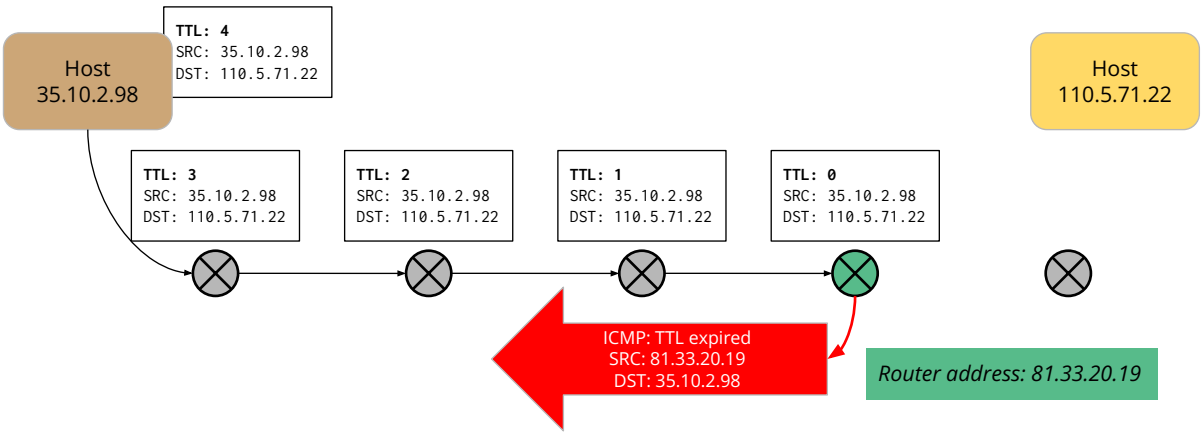
- ICMP
- Used by host / routers for Network Layer Information
  - Errors
  - Echo request/reply (PING)
- ICMP messages are carried as payload in IP datagrams

ICMP Message Type/Code

Type	Description
3	Destination Unreachable
4	No space in buffer
5	Datagram redirected
8	Echo
0	Echo reply
11	Time expired
12	Parameter error
13	Timestamp
14	Tinestano reply

Type	Code	Description
4	0	No space in buffer
5	0	Redirect datagrams for the network
	1	Protocol unreachable
	2	Port unreachable
	3	Fragmentation needed
	5	Source field failed
11	0	TTL expired
	1	Fragment reassembly time exceeded

IP Time-To-Live Expiration



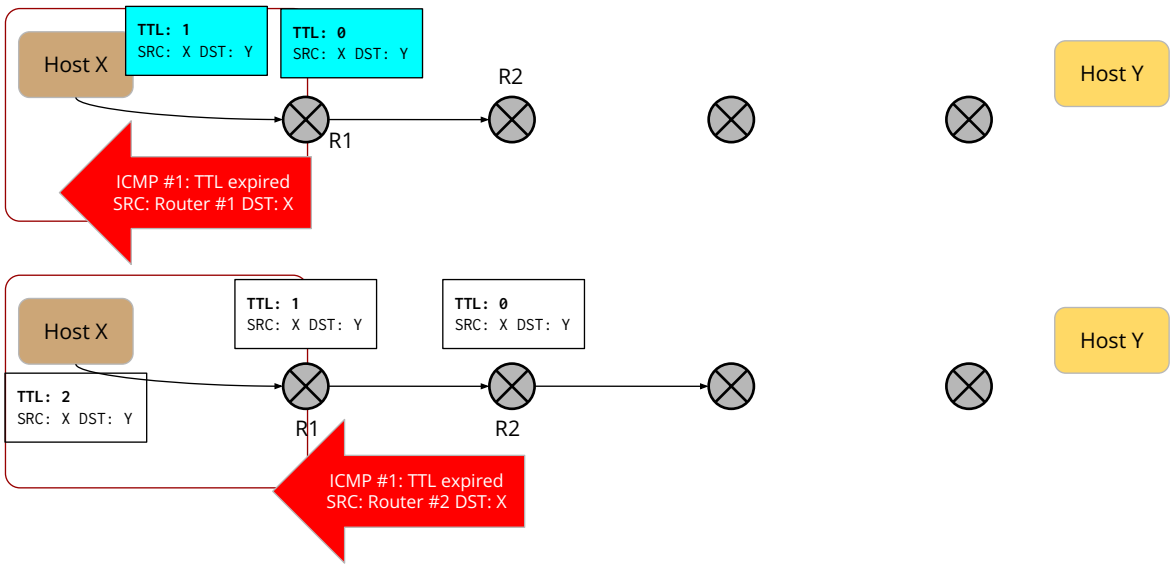
traceroute

```
traceroute to computing.gvsu.edu (104.17.88.18), 30 hops max, 60 byte packets

1  * * *
2  * * *
3  * * *
4  * * *
5  148.61.0.126 (148.61.0.126)  2.874 ms  2.921 ms  3.661 ms
6  irbx643.gdrp-eberhard-c1.mich.net (192.122.182.137)  1.078 ms  1.078 ms  1.112 ms
7  et-5-3-0x3.dtrt-wsudc-c1.mich.net (207.72.230.47)  4.705 ms  4.739 ms  4.716 ms
8  et-4-1-5x3.sfld-cor-123net.mich.net (207.72.230.128)  4.631 ms  4.680 ms  4.612 ms
9  static.det-ix.net (209.124.52.26)  5.144 ms  5.514 ms  5.298 ms
10  104.17.88.18 (104.17.88.18)  4.581 ms  4.618 ms  4.628 ms
```

no response within 5 seconds

TraceRoute Implementation: UDP + IP + Increasing TTLs



TraceRoute Implementation: UDP + IP + Increasing TTLs