U3 NETWORK LAYER

What is Network Layer?

→ The network layer is responsible for end-to-end packet (→ datagrams) delivery across multiple networks

→ It has 2 main operations :

i) Forwarding → Data Plane Function

→ Moving a packet within a router from input port to correct output port

ii) Routing → Control Plane

→ Deciding which path packets should take b/w source & destination networks

Data Plane is like the worker and Control Plane is like the planner

Router doesn't run application & transport layer
It runs only network, link & physical layer
→ If router receives TCP segment, it won't process it cuz it only sees IP header

Forwarding (Data Plane)

→ When a packet arrives at a router

i) It checks the destination IP Address in the header

ii) Looks up this address in Forwarding table

iii) Sends the packet to correct output port

{Also note, forwarding & switching are terms used interchangeably!}

{All this happens in single router}

ex -

Destination IP Prefix	Output Port
192.168.1.0/24	Port 1
192.168.2.0/24	Port 2
Default	Port 3

If destination is 192.168.2.45 ⇒ Router forwards to port 2

→ Forwarding is performed by line-cards in hardware for speed (i/p ports, o/p ports, switching fabric are in hardware)

→ It must happen in nanoseconds because links are in 100 Gbps and decisions must be made quick

Routing (Control Plane)

→ Determines path taken by packets as they flow from sender to receiver

→ Uses routing algorithms & protocols to exchange information

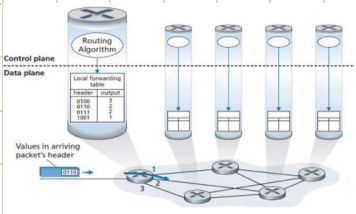
Component	Examples
Routing Algorithm	Dijkstra (Link-State), Bellman-Ford (Distance-Vector)
Routing Protocol	OSPF, RIP, BGP

Forwarding → per-router, per-packet immediate action
Routing → Global, Network-wide path computation

→ Each router builds a routing table & from that, forwarding table is derived

→ Control Plane operations are done using software & usually much longer (usually seconds or more)

Traditional Approach (Routing + Forwarding)



→ Each router is self contained (performs both data-plane & control-plane functions)

→ Router updates the forwarding table independantly using routing algorithms

Inputs of routing algorithm are obtained using routing protocols

Routing protocols allow routers to exchange their forwarding tables with one another

If forwarding table is manually configured, it is error prone (Static Routing → Control Plane inactive)

ex - If a packet with header field value of 0110 arrives to a router, the router indexes into its forwarding table & determines that output link interface for this packet is interface & router internally forwards there.

SDN Approach (Software Defined Networking)

→ The issue with traditional approach is that it is distributed, so if there are many routers, each must run its own OSPF / BGP instance. If one link fails, all routers must recompute & exchange updates like delay, high overhead etc.,

→ SDN solves the issue by centralizing the control plane

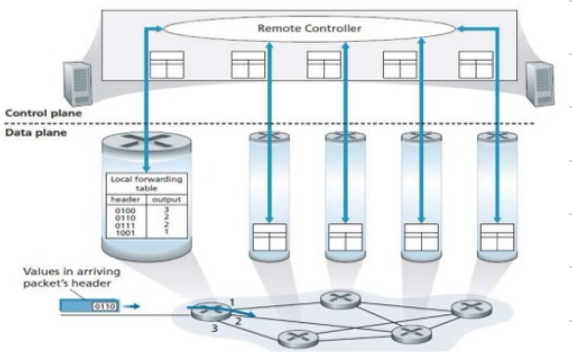
→ Routers & switches become simple forwarding devices

→ A central controller (software) decides how packets should flow & controller installs forwarding tables into routers using a standard protocol (OpenFlow)

→ The goal of SDN is to provide open, user-controlled management of the forwarding hardware in a network

→ Control plane can have one or more remote controllers depending on scale of network
If multiple controllers are used, can form peer-to-peer high-speed, reliable distributed n/w control

→ All switches in data plane must obtain consistent view of data delivery.



Network Service Models

→ They define what kind of service the network layer provides to transport layer

Types of Service	Description
i) Guaranteed Delivery	Every packet is delivered (Reliable)
ii) Guarenteed Delivery with Bounded Delivery	Delivery within fixed max delay
iii) In-Order Delivery	Packets arrive in same order as sent
iv) Guaranteed Bandwidth	Minimum Throughput guaranteed
v) Security	Encryption / Decryption at Source & Destination
vi) Best Effort	No Guarantee (Maybe lost / delayed / out-of-order)

→ Internet uses Best-Effort Service meaning IP tries its best but doesn't ensure delivery, order or timing

Internal Functions of a Router

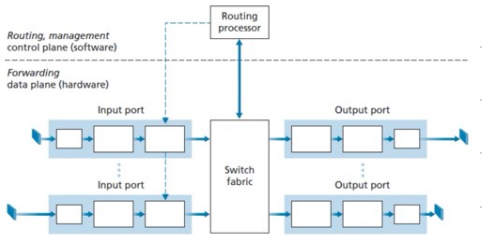
→ Router Architecture

→ Router is like a small computer built to move packets as fast as possible

→ It has 4 main functional blocks :

i) Routing Processor ↳ Control Plane

ii) Switching Fabric
iii) Input Port
iv) Output Port
} → Data Plane



→ Input Port Functions

When a packet arrives :

i) PHY operations (Link termination)

→ Converts the signal on the medium (optical/electrical) to bitstream (symbols/frames)

ii) Link-layer Operations

→ Error checking, removes headers (Frame to datagram)

iii) Queue Management

→ If packets arrive faster than they can be switched, they wait in an input queue

iv) Header Inspection + Lookup

→ Reads IP header

→ Checks version, header checksum

→ Finds destination IP

→ Looks it up in the Forwarding Table & decides output port. Filters suspicious packets

→ Forwarding Mechanism

→ Forwarding table is created by the routing processor & copied to line cards (input ports)

→ Each entry : Destination Prefix → Output Port

→ Destination - based Forwarding

→ Default traditional approach : Choose port based on destination IP

→ But not sufficient for 4 billion (2^{32}) addresses

→ Longest Prefix Match

→ If multiple prefixes match the IP, router picks the one with longest matching prefix

ex - 128.96.39.0/24 → Port 0

128.96.39.128/25 → Port 1

So for destination 128.96.39.130 both match but /25 is longer, so it uses port 1

→ Hardware Support

→ To speed the lookups, routers use CAM, TCAM

ex - Cisco 8500 uses 64K CAM entries per input port

CAM - Content Addressable Memory
TCAM - Ternary CAM
- Supports "don't cares" bits for prefix matching

→ **Switching Fabric** → Router's internal N/w

→ After output port is known, the packet moves through the switching fabric

→ 3 Hardware designs are common :

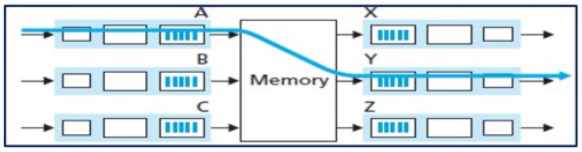
i) **Switching via Memory**

→ Store packets destined for an output port in memory

→ Very slow ⇒ One packet a time

→ Only supports half of memory bandwidth rate

ex - Cisco Catalyst 8500 router (old & slow)



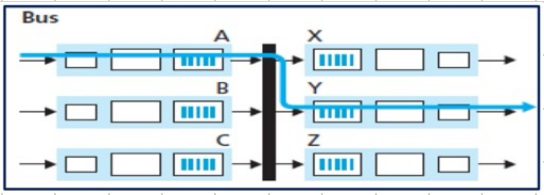
ii) **Switching via Bus**

→ Labels of o/p port is pre-pended to the packet & passed on to the bus

→ All o/p ports receive the packet but only one of them forwards the packet

→ Before forwarding, label is removed

ex - Cisco 6500 switches packets over 32 Gbps bus

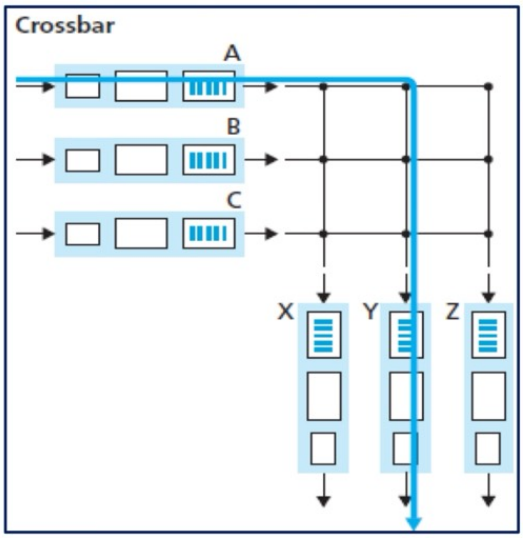


iii) **Switching via Crossbar (Interconnection N/w)**

→ Capable of switching multiple packets destined for different output ports (non-overlapping interconnection)

→ Switch controller activates the interconnection according to port selection

ex - Cisco 12000 & Cisco 7000 Series



→ **Output Port Processing**

→ Steps before packet leaves

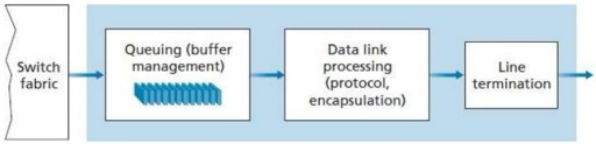
i) Queuing (if needed) (May also lead to packet loss)

ii) Checksum Recalculation - Since TTL decreases by 1

iii) Encapsulation - Add link-layer frame header/trailers

iv) Transmission - Converts bits to signals via Physical interface

→ Queue Management



→ Queue appears at input & output ports

→ Queues form because :

i) Mismatch b/w input arrival rate & switch fabric speed

ii) Mismatch b/w output rate & link speed

→ This causes buffer to fill up & packets drop (Loss)

→ Active Queue Management

→ These algorithms are deployed to take packet dropping decisions

→ Random Early Detection

→ Monitors buffer occupancy

→ Randomly drops/marks packets before full & signals TCP to slow down

→ Scheduling Policies

i) FCFS - First Come, First - Served (No priorities)

ii) WFQ - Weighted Fair Queuing (Fair bandwidth share among flows)

→ Buffer Size Formula, $B = \frac{RTT \times C}{\sqrt{N}}$

RTT : Round - Trip Time

C : Link Capacity

N : No. of Flows

→ Head - of - Line (HOL) Blocking

→ Occurs in input queued routers when :

i) Switching Fabric is slower than Line rate

ii) The packet at front of queue (For busy output port) blocks packets behind it

→ This results to throughput ≤ 58% of link capacity

→ Solutions to this is :

i) Parallel input queues

ii) Scheduling Algorithms

iii) Faster Fabric

→ Routing Processor (Control Plane Tasks)

→ Runs routing protocols like OSPF, BGP, RIP

→ It updates forwarding table

→ In SDN Networks, it communicates with remote controller

Path of Packet : I/P Port → Link Layer Decapsulation → Forwarding Table Lookup → Switching Fabric → O/P Port → Link Layer Encapsulation → Physical Transmission to next hop

Routing Overview & Types

→ Routing is the process of selecting a path for packets to travel across a network (from source to destination) possibly across many routers

→ A router must determine :

i) Which next hop to forward each packet to

ii) What route gives the lowest cost (delay, hop count etc.,)

→ Routing decisions come from routing algorithms & routing protocols

- Calculates best paths — ex - Dijkstra, Bellman-Ford
- Lets routers exchange info about n/ws — ex - OSPF, BGP, RIP

→ **Routing Algorithms**

→ Algorithms which compute the end-to-end paths based on the information obtained using routing protocols.

→ These algorithms are based on Graph Theory

→ Types include Centralized & Distributed, Static & Dynamic, Load-sensitive & Load-insensitive

→ **Types of Routing**

i) Unicast - Packet is sent to single interface

ii) Anycast - Packet is sent to the nearest group interfaces (in terms of routing distance) → streaming services

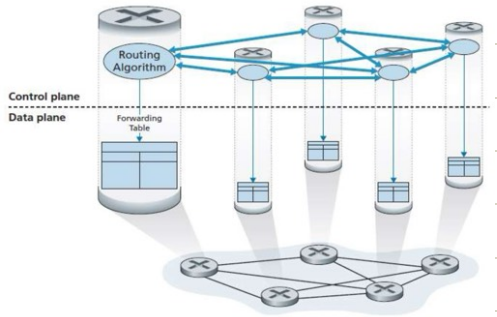
iii) Multicast - Packet is sent to multiple interfaces

→ **Routing Control Types**

i) Per-Router Control

→ Each router runs its own control logic

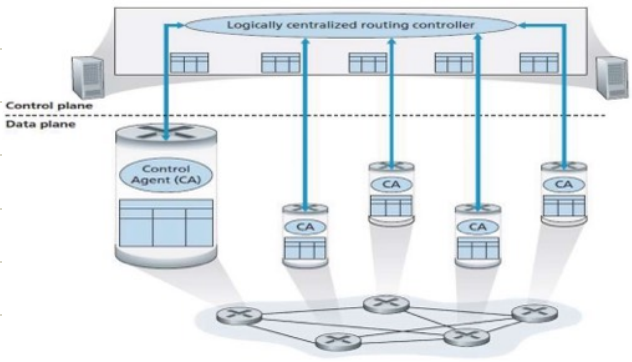
ex - Traditional OSPF / BGP



ii) Centralized Control

→ Single controller decides routers for all routers

ex - SDN Controller



→ **Routing Process**

i) Neighbor Discovery - detect directly connected routers

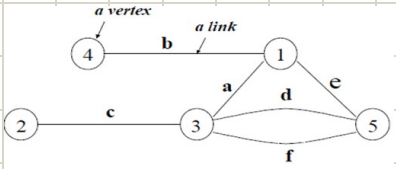
ii) Exchange Information - Routing protocols share link costs or full topology

iii) Compute Paths - Each router runs algorithm to find shortes paths

iv) Build Forwarding Table - Destination Prefix → Next hop / Output port

v) Forward Packets - Data plane uses the table for actual transmission

Graph Theory basics



→ Assume a topology of a network which is represented as a graph $G = \{E, N\}$

→ E : set of edges (link costs)

N : set of vertices / nodes (routers)

→ The ultimate goal is to find a least-cost path from one node to another without looping

→ Spanning Tree is a sub-graph which consists of all nodes & no loops

→ Criteria for evaluating MST Algorithm

i) Convergence time to steady state solution

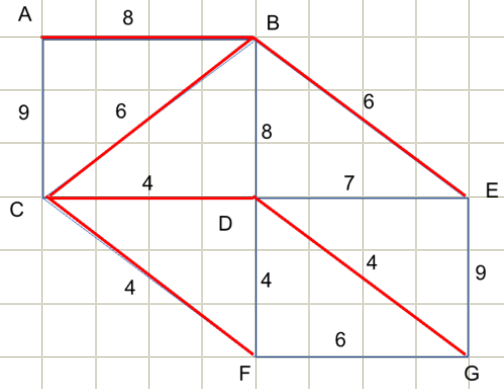
ii) Robustness to topology changes

iii) Computation complexity

iv) Amount of control messages exchanged

v) Size of buffer to store the information

ex- The minimum spanning tree for the network is in red



Link State Algorithm *

→ The main concept is that each router

i) knows entire n/w topology (all routers + link costs)

ii) Broadcasts its link-state to all other routers

iii) Then independantly computes shortest paths to all routers using Dijkstra's Algorithm

→ Now assume node u is the source

```
1  Initialization:
2    N' = {u}
3    for all nodes v
4      if v is a neighbor of u
5        then D(v) = c(u,v)
6      else D(v) = ∞
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 each neighbor v of 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    least path cost to w plus cost from w to v */
15 until N'= N
```

D(v): Current cost estimate from u to v

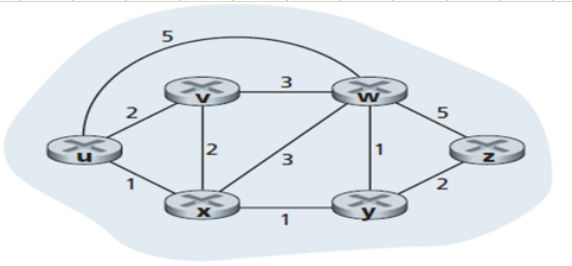
p(v): Predecessor of node v

N': Set of nodes for case of Known shortest path

$$D(v) = \min_{w \in N'} \left[D(v), D(w) + c(w,v) \right]$$

→ Order of Complexity is $O(N^2)$

ex -



Step 1 : Initialize a source, in this case pick u & find the link cost to directly attached neighbours

step	N' (path)	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)	D(z), p(z)
0	u	2, u	5, u	1, u	∞	∞

Step 2 : Now pick the node with least cost, in this case x & add to N'

Update the table again to find the least cost to that node

step	N'	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)	D(z), p(z)
0	u	2, u	5, u	1, u	∞	∞
1	ux	2, u	(1+3), w		(1+1), y	∞

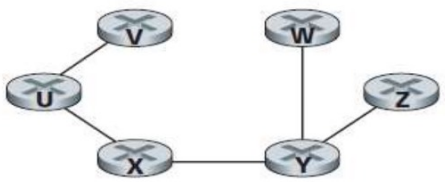
Step 3 : Now update the table again based on least cost. Here we have a tie b/w nodes v & y.

Dijkstra's algorithm doesn't require a unique choice - both are valid. Finish rest of the table.

step	N'	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)	D(z), p(z)
0	u	2, u	5, u	1, u	∞	∞
1	ux	2, u	(1+3), x		(1+1), x	∞
2	uxy	2, u	3, y			4, y
3	uxyv		3, y			4, y
4	uxyvw					4, y
5	uxyvwz					

Step 4 : When LS algorithm terminates, we have for each node, its predecessor & least-cost path

So we can construct entire path from source to all destinations. Hence, we construct forwarding path



Destination	Link
v	(u, v)
w	(u, x)
x	(u, x)
y	(u, x)
z	(u, x)

Distance Vector Algorithm

→ The main concept is that each router

i) knows only its direct neighbours & cost to reach them

ii) Periodically exchanges information (distance vectors) with its neighbours

iii) Uses Bellman-Ford Equation to update routes iteratively

→

At each node, x:

```
1  Initialization:
2    for all destinations y in N:
3       Dx(y) = c(x,y) /* if y is not a neighbor then c(x,y) = ∞ */
4    for each neighbor w
5       Dw(y) = ? for all destinations y in N
6    for each neighbor w
7       send distance vector  Dx = [Dx(y): y in N] to w
8
9  loop
10    wait  (until I see a link cost change to some neighbor w or
11             until I receive a distance vector from some neighbor w)
12
13    for each y in N:
14       Dx(y) = min_v{c(x,v) + Dv(y)}
15
16 if Dx(y) changed for any destination y
17       send distance vector Dx = [Dx(y): y in N] to all neighbors
18
19 forever
```

$D_v(y)$: v's current cost to reach y

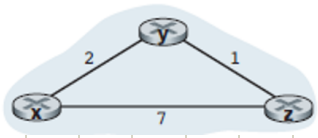
$C(x,v)$: Cost of link from x to neighbour y

→ Here since it keeps updating forever, any small change can lead to large no. of iterations (causing ∞ loop) which is reffered to as count-to-infinity problem

$$D_x(y) = \min_v [c(x,v) + D_v(y)]$$

Can fix it using Route Poisoning by conveying neighbour that route cost back through it is ∞

ex -



Step 1 : Assume each router knows only the cost to direct neighbour

For node x :

$$D_x(y) = \min [c(x,y) + D_y(y) \ , \ c(x,z) + D_z(y)] = \min [2+0 \ , \ 7+\infty] = 2$$ (if not known yet, use ∞)

$$D_x(z) = \min [c(x,z) + D_z(z) \ , \ c(x,y) + D_y(z)] = \min [7+0 \ , \ 2+\infty] = 7$$

For node y :

$$D_y(x) = \min [c(y,x) + D_x(x) \ , \ c(y,z) + D_z(x)] = \min [2+0 \ , \ 1+\infty] = 2$$

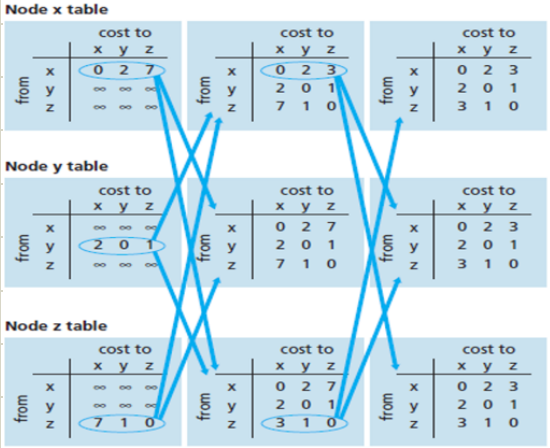
$$D_y(z) = \min [c(y,z) + D_z(z) \ , \ c(y,x) + D_z(z)] = \min [1+0 \ , \ 2+\infty] = 1$$

For node z :

$$D_z(y) = \min [c(z,x) + D_x(x) \ , \ c(z,y) + D_y(x)] = \min [7+0 \ , \ 1+\infty] = 7$$

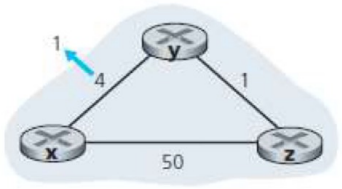
$$D_z(x) = \min [c(z,y) + D_y(y) \ , \ c(z,x) + D_x(y)] = \min [1+0 \ , \ 7+\infty] = 1$$

Step 2 : Iterate this till all tables are exactly same by exchanging DVs after all 3 nodes



→ Effect of changes in link cost

ex - Assume initial DV will be same for x, y and z



	x	y	z
$D_x(\cdot)$	0	4	5
$D_y(\cdot)$	4	0	1
$D_z(\cdot)$	5	1	0

$c(x,y) = c(y,x) = 1$

$c(y,z) = c(z,y) = 1$

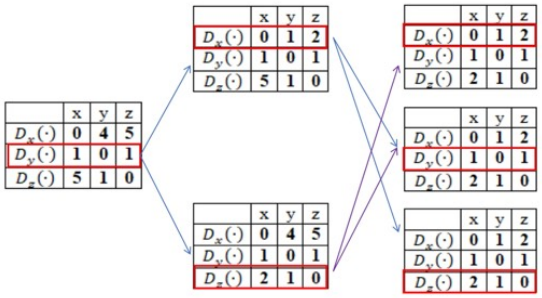
$c(x,z) = c(z,x) = 50$

Suppose link cost $c(x,y)$ changes to 1 & it is detected by y

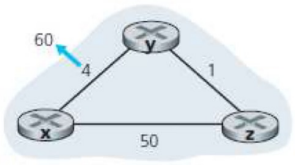
Then new table is

	x	y	z
$D_x(\cdot)$	0	4	5
$D_y(\cdot)$	1	0	1
$D_z(\cdot)$	5	1	0

And it will be broadcasted as follows



ex - Now assume link cost $c(x,y)$ changes to 60 instead of 1



	x	y	z
$D_x(\cdot)$	0	4	5
$D_y(\cdot)$	4	0	1
$D_z(\cdot)$	5	1	0

$c(x,y) = c(y,x) = 60$

$c(y,z) = c(z,y) = 1$

$c(z,x) = c(x,z) = 50$

After y detects the change,

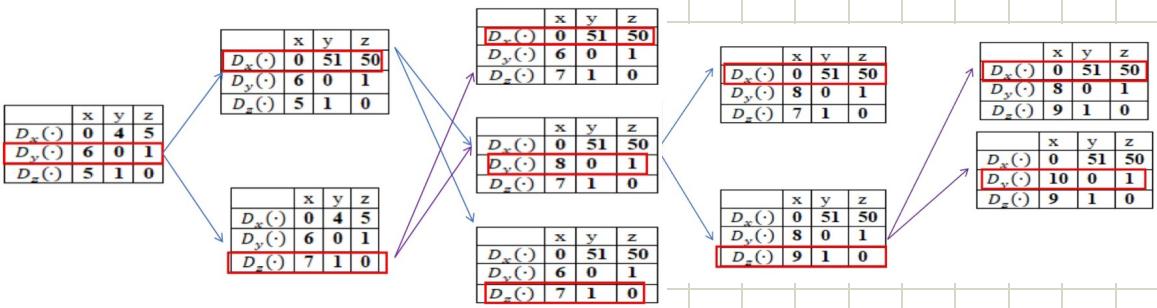
	x	y	z
$D_x(\cdot)$	0	4	5
$D_y(\cdot)$	6	0	1
$D_z(\cdot)$	5	1	0

Takes this even though this path doesn't exist

$D_y(x) = \min[c(y,x) + D_x(x), c(y,z) + D_z(x)] = \min[60+0, 1+5] = 6$

$D_y(z) = \min[c(y,x) + D_x(z), c(y,z) + D_z(z)] = \min[60+5, 1+0] = 1$

Then by broadcasting



→ In a distributed n/w, as old values of $D_y(x)$ & $D_y(z)$ are used, it takes a really long time to realize the true state information. Router y didn't know old values were being used & it believed it was the shortest path, triggering a large no. of iterations which is reffered as count-to-infinity problem

OSPF (Open Shortest Path First)

→ Routing Protocols

- → Routing Protocols allow routers to exchange their forwarding table entries, helping to learn paths to various networks automatically
- → Each ISP manages its own routers independantly & the internet is a collection of ISP networks, therefore, there isn't a single routing protocol that is used globally.
- → Each ISP's network is viewed as an Autonomous System (AS) - a group of routers & links managed by one organization
- → Sometimes large ISP have multiple AS's internally to reduce complexity

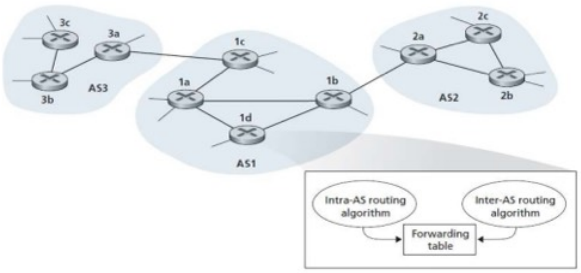
→ Autonomous Systems

- → An AS is a set of routers & links under a single administrative authority
- → There are 2 types of routers :
  - i) Gateway Router - Connects the AS to other AS's
  - ii) Non-Gateway Router - Operates only within the AS
- → Stub AS is an AS that generates / receives packets but doesn't forward traffic for others
- → AS Number (ASN) is a unique 16-bit ID assigned to every AS which is assigned by Regional Internet Registries (RIRs) under ICANN (ex- APNIC, ARIN, RIPE NCC)

→ Routing Updates and Policy

- → Routers may update locally / globally based on who manages them
  - → Local updates ⇒ intra AS (ex- RIP, OSPF, IS-IS, EIGRP) → Routing info b/w routers within AS
  - → Global updates ⇒ inter AS (ex- BGP) → Routing info b/w routers across ASes
- → The n/w admin can use either Link-state (LS) or Distance-Vector (DV) algorithms inside the AS
- → The routing configuration depends on cost of forwarding own & other networks' traffic & business & economic policies (ISPs decide which traffic to carry)

→ Intra-AS Routing - OSPF



- → Described in RFC 2328
- → Uses Dijkstra's Shortest Path First algorithm
- → Link-State Routing protocol
- → Scope only inside one organization (Intra AS)

Parameter	Description
Administrative Distance	110
IP header Protocol Number	89
Security	Supports digital authentication
Equal-Cost Multipath (ECMP)	Allows multiple same-cost routes
Hop Count Limit	None

→ RIP has 15
OSPF has 0

Message Type	Description
Hello	Used to discover who the neighbors are
Link State Update	Provide the sender's costs to its neighbors
Link state ACK	Acknowledges link state update
Database description	Announces which updates the sender has
Link state request	Request information from partner

→ OSPF operation Summary

1) Neighbor Information
   → Forms neighbor relationships (adjacencies) with routers in same area
   → Uses Hello packets to multicast address (224.0.0.5)

2) Link-State Advertisements (LSAs)
   → OSPF advertises the status of its directly connected links, instead of distance
   → LSAs are sent only when link change occurs & refreshed every 30 minutes

3) Cost Metric
   → Calculated based on link bandwidth
   → Higher bandwidth = Lower Cost
   → $\text{Cost} = \frac{10^8}{\text{Bandwidth (bps)}}$

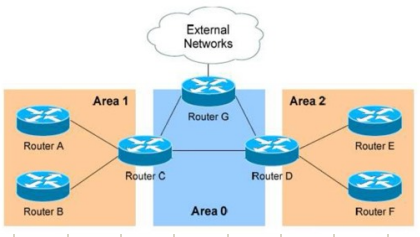
4) Hello & Dead intervals
   → Hello interval : 10s
   → Dead interval : 40s (If no Hello, neighbor considered down)

→ OSPF Tables
   → Neighbor Table - List of all neighboring routers (adjacencies)
   → Topology Table (Link-State Database) - All possible routes from LSAs
   → Routing Table - Best route for each known network (Output of Dijkstra's Algorithm)

→ Hello Packets (Neighbor Discovery)
   → Exchanged to build & maintain adjacencies
   → Sent to multicast 224.0.0.5
   → Contains Area ID, Area Type (↳ Stub, NSSA etc.), Prefix & Subnet Mask, Hello Interval, Dead interval, Network Type (Broadcast, Point-to-point etc.,) & authentication information

→ Purpose of Areas
   → Areas minimize LSA flooding traffic & computation
   → Each area floods LSAs only within the AS itself
   → Hierarchy + Multicasting → Reduced Complexity & Overhead

→ Area Zero (Backbone Area)
   → It is the backbone or transit area required for OSPF to function
   → Rule is that all other areas must connect (directly or indirectly) to area 0 & Traffic b/w non-backbone areas must pass through Area 0.



→ Designated Router (DR) & Backup DR (BDR)

→ They are used to reduce LSA traffic on broadcast networks

→ Working :

i) All routers send LSAs to the DR

ii) DR distributes LSAs to all other routers

iii) BDR takes over if DR fails

→ Election Criteria :

→ Router with highest OSPF priority (more links, higher bandwidth) is DR

→ Second highest is BDR

→ Types of Routers in OSPF

→ Internal Router

→ All interfaces in same area

→ Area Border Router (ABR)

→ Connects one or more areas to backbone (Area 0)

→ Backbone Router

→ Atleast one interface in area 0

→ AS Boundary Router (ASBR)

→ Connects OSPF domain to another AS (ex- BGP)

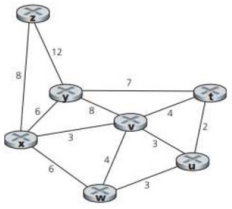
→ Routing Control & Policy

→ Each AS has control over what routing information it advertises or accepts

ex - AS1 may choose not to advertise networks announced by AS3

→ The purpose of this is to maintain control of internal traffic & implement routing policies & economic constraints.

Q. Consider the following network. With the indicated link costs, use Dijkstra's shortest-path algorithm to compute the shortest path from x to all network nodes. Show how the algorithm works by computing a table similar to Table shown below.

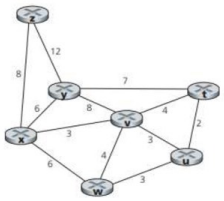


Iteration	N'	D(u),p(u)	D(t), p(t)	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)
0	z						

A.

Iteration	N'	D(u), p(u)	D(t), p(t)	D(v), p(v)	D(w), p(w)	D(y), p(y)	D(z), p(z)
0	x	∞	∞	3, x	6, x	6, x	8, x
1	xv	6, v	7, v		6, x	6, x	8, x
2	xvu		7, v		6, x	6, x	8, x
3	xvuw		7, v			6, x	8, x
4	xvuwy		7, v				8, x
5	xvuwyt						8, x
6	xvuwytz						

Q. Consider the network shown below. Using Dijkstra's algorithm, and showing your work using a table shown below, do the following: 1) Compute the shortest path from t to all network nodes. 2) Compute the shortest path from u to all network nodes. 3) Compute the shortest path from v to all network nodes. 4) Compute the shortest path from w to all network nodes. 5) Compute the shortest path from y to all network nodes. 6) Compute the shortest path from z to all network nodes.



Iteration	N'	D(u),p(u)	D(t), p(t)	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)
0	z						

A. 1)

Iteration	N'	D(x), p(x)	D(u), p(u)	D(v), p(v)	D(w), p(w)	D(y), p(y)	D(z), p(z)
0	t	∞	2, t	4, t	∞	7, t	∞
1	tu	∞		4, t	5, u	7, t	∞
2	tuv	7, v			5, u	7, t	∞
3	tuvw	7, v				7, t	∞
4	tuvwx					7, t	15, x
5	tuvwxy						15, x
6	tuvwxyz						

2)

Iteration	N'	D(x), p(x)	D(z), p(z)	D(v), p(v)	D(w), p(w)	D(y), p(y)	D(z), p(z)
0	u	∞	2, u	3, u	3, u	∞	∞
1	ut	∞		3, u	3, u	9, t	∞
2	utv	6, v			3, u	9, t	∞
3	utvw	6, v				9, t	∞
4	utvwx					9, t	14, x
5	utvwxy						14, x
6	utvwxyz						

3)

Iteration	N'	D(x), p(x)	D(v), p(v)	D(t), p(t)	D(w), p(w)	D(y), p(y)	D(z), p(z)
0	v	3,v	3,v	4,v	4,v	8, v	∞
1	vx		3,v	4,v	4,v	8,v	11,v
2	vxu			4,v	4,v	8,v	11, v
3	vxut				4,v	8,v	11,v
4	vxutw					8,v	11, v
5	vxutwy						11,v
6	vxutwyz						

4)

Iteration	N'	D(x), p(x)	D(v), p(v)	D(v), p(v)	D(t), p(t)	D(y), p(y)	D(z), p(z)
0	w	6, w	3, w	4, w	∞	∞	∞
1	wu	6, w		4, w	5, u	∞	∞
2	wuv	6, w			5, u	12, v	∞
3	wuvt	6, w				12, v	∞
4	wuvtx					12, v	14, x
5	wuvtxy						14, x
6	wutvxyz						

5)

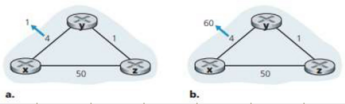
Iteration	N'	D(x), p(x)	D(v), p(v)	D(v), p(v)	D(w), p(w)	D(t), p(t)	D(z), p(z)
0	y	6, y	∞	8, y	∞	7, y	12, y
1	yx		∞	8, y	12, x	7, y	12, y
2	yxt		9, t	8, y	12, x		12, y
3	yxtv		9, t		12, x		12, y
4	yxtvu				12, x		12, y
5	yxtvuw						12, y
6	yxtvuwz						

6)

Iteration	N'	D(x), p(x)	D(v), p(v)	D(v), p(v)	D(w), p(w)	D(y), p(y)	D(t), p(t)
0	z	8, z	∞	∞	∞	12, z	∞
1	zx		∞	11, x	14, x	12, z	∞
2	zxv		14, v		14, x	12, z	15, v
3	zxvy		14, v		14, x		15, v
4	zxvyu				14, x		15, v
5	zxvyuw						15, v
6	zxvyuwt						

Q.

- Consider Figure shown below. Suppose there is another router w, connected to router y and z. The costs of all links are given as follows: c(x,y) = 4 ; c(x,z) = 50 ; c(y,w) = 1 ; c(z,w) = 1 ; c(y,z) = 3. Suppose that poisoned reverse is used in the distance-vector routing algorithm.
- A) When the distance vector routing is stabilized, router w, y, and z inform their distances to x to each other. What distance values do they tell each other?
- B) Now suppose that the link cost between x and y increases to 60. Will there be a count-to-infinity problem even if poisoned reverse is used? Why or why not? If there is a count-to-infinity problem, then how many iterations are needed for the distance-vector routing to reach a stable state again? Justify your answer.
- c. How do you modify c(y,z) such that there is no count-to-infinity problem at all if c(y,x) changes from 4 to 60?

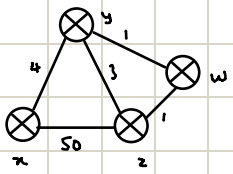


a. b.

127

A. A)

Router z	Informs w, $D_z(x) = \infty$ Informs y, $D_z(x) = 6$
Router w	Informs y, $D_w(x) = \infty$ Informs z, $D_w(x) = 5$
Router y	Informs w, $D_y(x) = 4$ Informs z, $D_y(x) = 4$



B) Yes, there will be count-to-infinity problem. The following table shows converging process.

	t_0	t_1	t_2	t_3	t_4
Router z	Informs w, $D_z(x) = \infty$ Informs y, $D_z(x) = 6$		No Change	Informs w, $D_z(x) = \infty$ Informs y, $D_z(x) = 11$	
Router w	Informs y, $D_w(x) = \infty$ Informs z, $D_w(x) = 5$		Informs y, $D_w(x) = \infty$ Informs z, $D_w(x) = 10$		No Change
Router y	Informs w, $D_y(x) = 4$ Informs z, $D_y(x) = 4$	Informs w, $D_y(x) = 9$ Informs z, $D_y(x) = \infty$		No Change	Informs w, $D_y(x) = 14$ Informs z, $D_y(x) = \infty$

w, y, z form a loop. Following these iterations, at t_{27}, z detects that least cost to x is 50 directly.
at t_{29}, w learns least cost to x is 51 via z
at t_{30}, y updates least cost to x is 52 via w
at t_{31}, no updating, routing stabilized

	t_0	t_1	t_2	t_3	t_4
Router z	Informs w, $D_z(x) = 50$ Informs y, $D_z(x) = 50$				via w, ∞ via y, 55 via z, 50
Router w		Informs y, $D_w(x) = \infty$ Informs z, $D_w(x) = 50$	Informs y, $D_w(x) = 51$ Informs z, $D_w(x) = \infty$		via w, ∞ via y, ∞ via z, 51
Router y		Informs w, $D_y(x) = 53$ Informs z, $D_y(x) = \infty$		Informs w, $D_y(x) = \infty$ Informs z, $D_y(x) = 52$	via w, 52 via y, 60 via z, 53

c) Cut the link between y and z

BGP (Border Gateway Protocol)

→ When a packet travels across multiple AS's and must use Inter-AS routing protocol. Intra-AS only works within one AS

→ Inter-AS protocols must co-ordinate b/w ASes that are administratively independant, So, all ASes on the internet use the same Inter-AS protocol, called Border Gateway Protocol

→ BGP is defined in RFC 4271

→ BGP is like glue which connects 1000s of ISPs & organizations worldwide

→ **Nature of BGP**

→ It is decentralized & asynchronous protocol (Routers operate independantly & exchange info gradually)

→ Packets aren't routed to single destination IPs, but to CIDR-ized prefixes, which represent blocks of IP Addresses

ex - 138.16.68.0 ⇒ represents 1024 IPs

and each router's forwarding table entry has (prefix, interface) like (138.16.68.0/22, I1)

So, BGP works at prefix level & not per IP

→ **Functions of BGP**

→ Obtain Prefix Reachability Information

→ Routers learn what prefixes exist & where they can be reached

→ Each subnet "announces" itself via BGP advertisements: "I exist, and I can be reached here"

→ Without BGP, subnets would be isolated (unknown to rest of the world)

→ Determine the Best Route

→ A router may receive multiple routes to same prefix

→ It uses BGP's route-selection rules to select the best route of them all

→ It isn't always the shortest path → policy & cost influence the choice

→ **Types of routers in an AS**

→ There can be 2 types

i) Gateway Router

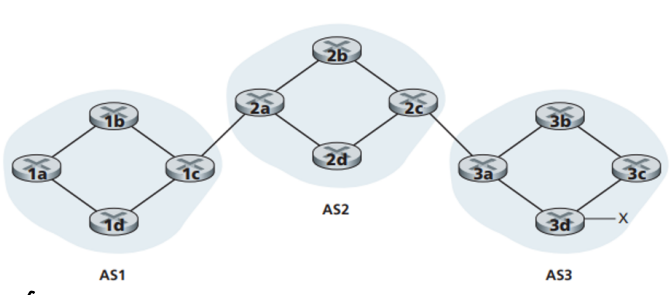
→ At the edge of an AS

→ Connects to routers in other ASes

ii) Internal Router

→ Connects only within its own AS

→ To hosts or internal routers



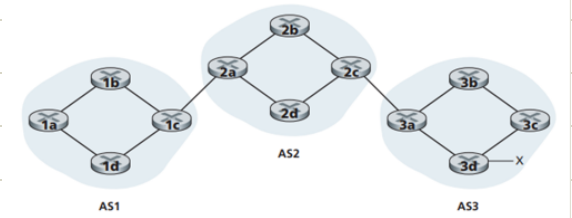
↳ In AS1,

1c is gateway router

1a, 1b, 1d are internal routers

→ Propagating Reachability Information

→ Lets assume the previous example,



Here, consider a subnet x inside AS3

1) AS3 → AS2

AS3 advertises "AS3 x" to AS2 (prefix x exists in AS3)

2) AS2 → AS1

AS2 advertises "AS2 AS3 x" to AS1 (x reachable via AS2 then AS3)

→ Each AS not only learns that x exists but also the path leading to x

→ BGP Connections

→ Routers, not ASes, actually exchange BGP information

→ Routers establish TCP connections on port 179 called BGP sessions

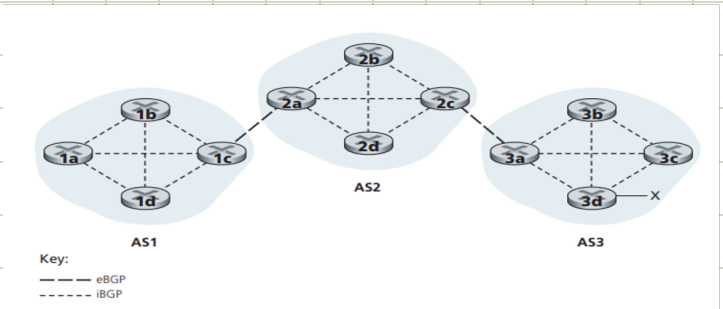
→ 2 Types of sessions exist :

i) External BGP (eBGP) - Between routers in different ASes

ii) Internal BGP (iBGP) - Between routers in same AS

→ Each TCP connection + BGP message = BGP connection

ex -



If AS3 had to advertise subnet x :

i) Gateway 3a → 2c : Sends eBGP message "AS3 x"

ii) Router 2c → all routers in AS2 : Sends iBGP message "AS3 x"

iii) Gateway 2a → 1c : Sends eBGP message "AS2 AS3 x"

iv) Router 1c → all routers in AS1 : Sends iBGP message "AS2 AS3 x"

Now every router in AS1 & AS2 knows prefix x exists & AS path that leads to x

→ Route Attributes

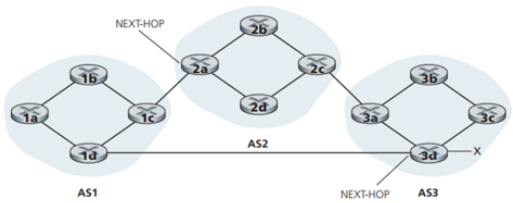
→ Each BGP advertisement carries attributes, forming BGP route (Prefix + Attributes)

→ AS-PATH - List of ASes the route has passed through. Prevents loops

NEXT-HOP - IP Address of the router that begins the AS-PATH (the next router to send packet to)

→ Multi-EXIT-DISCRIMINATOR, LOCAL-PREF, ATOMIC-AGGREGRATE, AGGREGATOR

ex -



There are 2 routes from AS1 to subnet x :

Route 1 → via AS2 AS3

Route 2 → via AS3 directly

Then, AS-PATH : "AS2 AS3" and "AS3"

NEXT-HOP : For "AS2 AS3 x" → IP of router 2a (Left interface)

For "AS3 x" → IP of router 3d (Left-most interface)

→ **Route Representation**

→ Each BGP can be represented as

(NEXT-HOP, AS-PATH, Destination Prefix)

ex - (2a-IP, AS2 AS3, x)

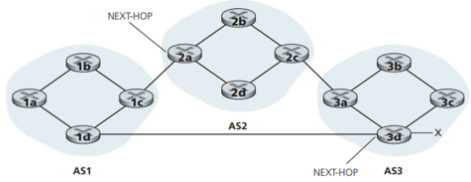
(3d-IP, AS3, x)

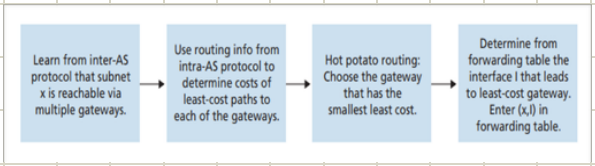
→ **Hot Potato Routing**

→ When a router learns multiple routes to a prefix, it chooses a route that has least intra-AS cost to reach NEXT-HOP

→ In reference, it tries to throw the packet (potato) out of its AS (hand) as quickly as possible

ex -





Router 1b :

- Cost to reach router 2a = 2 hops
- Cost to reach router 3d = 3 hops

So it chooses route via 2a (less cost) & adds (x, I) to forwarding table

→ Hot Potato routing is a selfish algorithm - it tries to reduce the cost in its own AS while ignoring the other components of the end-to-end costs outside its AS

→ In this, 2 routers in same AS may choose 2 different AS paths to same prefix

→ **Forwarding Table**

→ To forward packets for external prefixes, Router uses both Inter (→ BGP) & Intra (→ OSPF) AS protocols

→ Final entry format : (prefix, outgoing interface)

IP Anycast

→ Anycast is a routing method, not an address type like unicast / multicast

→ A single IP Address is assigned to multiple servers located in different geographical locations

→ When a device sends data to that IP, BGP routes it to nearest (best) server (lowest AS hop)

→ The purpose of IP Anycast is to replicate identical content across multiple servers (ex- CDN, DNS) and allow each user to access the nearest server, improving latency & reliability

→ **Working of IP Anycast**

i) Each server is assigned the same IP address

ii) Each server advertises that IP address via BGP

iii) Routers receive multiple advertisements for same IP - from different ASes or regions

iv) Each router independantly chooses best route (usually lowest cost route)

v) Traffic to that IP is then forwarded to the closest server according to BGP's route selection

ex- Assume a CDN assigns same IP address to servers in India, Germany & USA each location advertises that IP using BGP

Now a router in India will see 3 routes:

Route 1 : via AS hop to India Server (1 hop)

Route 2 : via AS hop to Germany Server (3 hops)

Route 3 : via AS hop to USA Server (4 hops)

Router picks 1 hop route & connects to India Server

→ **Advantages**

i) It has low latency because user is connected to nearest location

ii) The load is distributed & traffic is balanced naturally across servers

iii) If one server fails, routers automatically select another route

iv) It works across IPv4 & IPv6 with standard BGP

→ **Disadvantages**

i) It isn't ideal for TCP based services like CDN video streaming because if routing changes mid-connection, packets of same TCP flow could reach different servers

→ Hence, CDNs rarely use IP anycast for content delivery, However, DNS Systems use it extensively because DNS uses stateless UDP (No session problem)

→ **DNS Example**

→ There are 13 root DNS IP addresses (A-M)

→ Each IP corresponds to 100s of physical DNS servers deployed & each of those servers advertises the same IP using BGP

→ When a user queries one of those root IPs, BGP routes the query to closest root server geographically (or) topologically, ensuring fast DNS resolution & high reliability

Routing Policy

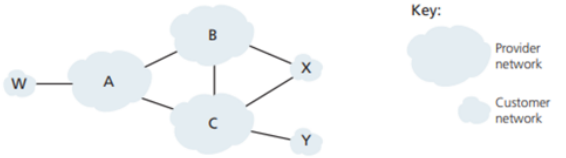
→ BGP isn't only about shortest paths, it's policy driven

→ As routing policies can override shortest-path or hot-potato choices

→ Route preference is influenced by LOCAL-PREFERENCE attribute which is set by each AS's admin

→ Local policies define who carries whose traffic & how routes are advertised

ex -



Assume, W, X, Y → Access ISPs

A, B, C → Backbone provider N/ws

X is multihomed (connected to B & C)

Key Assumptions

→ Access ISPs (W, X, Y) only send/receive their own traffic

→ Backbone providers (A, B, C) carry transit traffic for their customers

→ Each AS advertises routes according to business relationships (Customer ↔ Provider, Peer ↔ Peer)

Stub Network (Access ISP) behavior

→ X is connected to 2 providers

→ X mustn't carry transit traffic b/w them (no B → X → C forwarding)

→ So to enforce this, X advertises B & C that it has no routes beyond itself & even if X knows a route X → C → Y, it doesn't advertise to B.

→ As a result, B doesn't forward traffic to C via X

→ This selective advertisement implements the "stub" (access-only) behavior

Provider Network Behavior

→ Now B learns from A that A can reach W

So, B can reach W via B → A → W

→ B advertises B → A → W to it's customer X, so X can reach W through B

→ However B doesn't advertise B → A → W to C as A & C are peers and they can connect directly instead of adding additional costs.

→ Rule of Thumb - Backbone ISP should carry traffic only if either source/destination belong to customer

Policy Enforcement via BGP

→ BGP allows ISP to :

- Accept, filter or modify advertisements based on policy rules
- Control traffic flow according to economic relationships
- Maintain scalable, independant & profitable network management

Generalized Forwarding & SDN Control Plane

→ **Traditional (Destination-Based) Forwarding**

- → Router checks only destination IP in the header
- → 2 Step operation:
  1) Match - Look up destination IP in forwarding table
  2) Action - Send to corresponding output port
- → Forwarding is a fixed logic implemented in hardware

→ **SDN Approach**

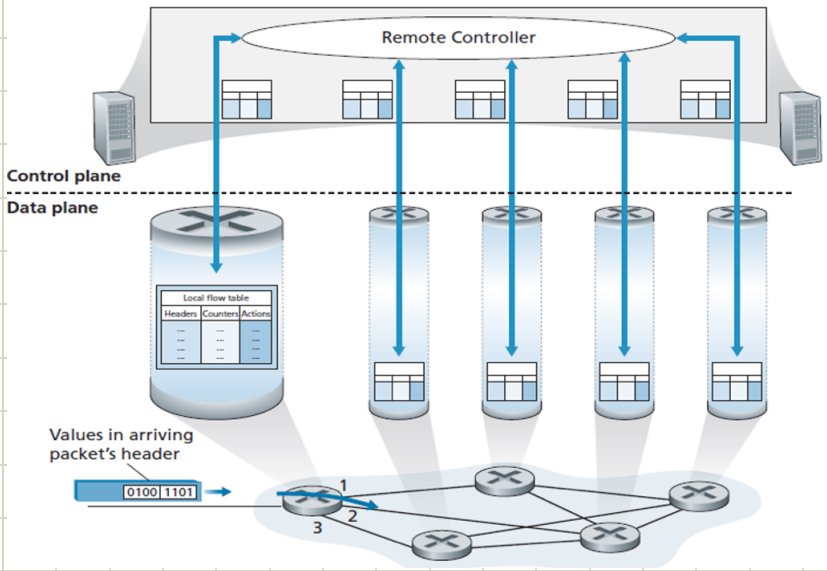
- → Switching & Output port processing remain the same
- → But forwarding becomes generalized – no longer restricted to destination IP
- → Since forwarding is implemented in software, it can be programmed & updated dynamically

→ **Generalized Forwarding (Match + Action model)**

- → The core idea is SDN switches uses a Match + Action model
- → Each packet arriving at SDN undergoes:
  i) Match
    - → Compare multiple header fields (from link, network & transport layers) to the flow table
    - → Compares destination & source IP, MAC, port numbers, VLAN ID etc.,
  ii) Action
    - → Perform actions like forward to output port(s), Drop/Block, Modify header (ex- NAT), Send a copy to controller (for inspection or new rule installation), Load balance traffic
- → Traditional forwarding becomes a special case of generalized forwarding

→ **Packet Switch Terminology**

- → Because forwarding can depend on link-layer, network layer, or transport layer headers, SDN devices are called packet switches, not strictly routers or L2 switches



Match + Action table in each packet switch with table being computed, installed and updated by remote controller

→ Flow Table (Match + Action Table)

→ Control through Remote Controller

→ Each switch has Match + Action Table managed by remote controller

→ The controller computes, installs & updates the entries in flow table

→ This setup defines SDN Control Plane

→ Open Flow Standard

→ OpenFlow 1.0 defines how match + action tables are configured & protocol for controller - switch interaction

→ P4 (Programming Protocol-independant Packet Processors) is a high level language for programming & computing flow tables. It includes constructs like variables, conditions, functions etc., for packet level processing

→ Structure of Flow Table Entry

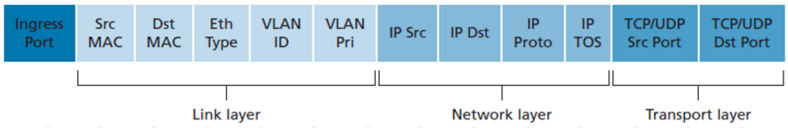
→ Each entry includes :

1) Header Field Values (Match Fields)
   → Packet header values match (ex- IP, MAC, TCP Port, VLAN etc.,)
2) Counters
   → Track no. of packets matched, bytes processed or time since last match
3) Actions
   → Specifies what to do when match occurs

→ Matching is done in TCAM (Ternary Content - Addressable Memory) for high speed lookups

If no match, packet is dropped or sent to controller

→ Matching Fields in OpenFlow 1.0



Common Header Fields Used :

i) Link Layer (Ethernet) :
   → Source & Destination MAC Addresses, VLAN ID, VLAN priority, Ethernet type (indicates high-layer protocol)

ii) Network Layer (IP) :
   → Source & Destination IP Addresses, IP protocol field, Type of Service (ToS)

iii) Transport Layer :
   → Source & Destination port numbers

→ Matching all fields is impractical as it leads to trade-off between space & flexibility

Priority values are assigned to flow entries to resolve conflicts

→ Actions in OpenFlow 1.0

i) Forwarding - Sends to one or more o/p ports ; broadcast or multicast ; or sends to controller
ii) Drop - Flow entry with no action field ⇒ drop the packet
iii) Modify field - Rewrite upto 10 header fields (Layer 2-4, except IP protocol). Enables NAT, VLAN, tagging etc.,

→ ICMP (Supporting Protocol in Network Layer)

→ ICMP or Internet Control Message Protocol

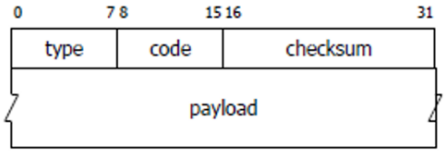
→ Defined in RFC 792

→ Used by routers / hosts to exchange control messages, it isn't user data

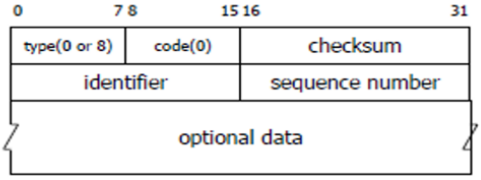
ex - Network / Host / Port unreachable, TTL expired, Echo / Echo reply

ICMP Type	Code	Description
0	0	echo reply (to ping)
3	0	destination network unreachable
3	1	destination host unreachable
3	2	destination protocol unreachable
3	3	destination port unreachable
3	6	destination network unknown
3	7	destination host unknown
4	0	source quench (congestion control)
8	0	echo request
9	0	router advertisement
10	0	router discovery
11	0	TTL expired
12	0	IP header bad

→ ICMP Message Format

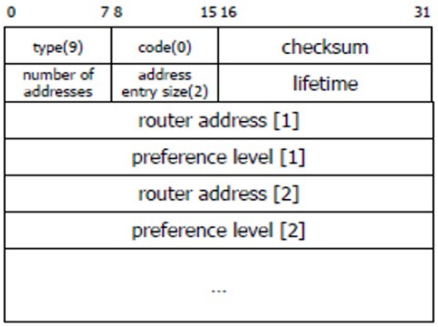


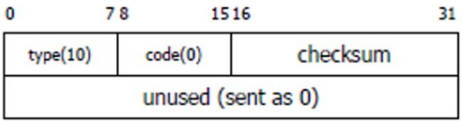
→ ICMP Echo and Traceroute



→ ICMP Solicitation & Discovery messages

- Discovered in RFC 1256
- Discovery messages are sent out every 7 - 10 minutes

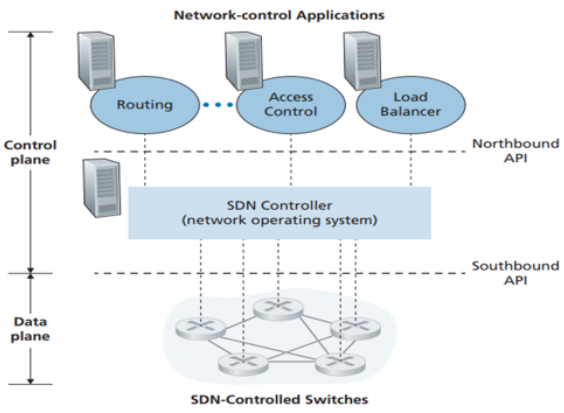




→ SDN Architecture Overview

→ SDN is characterized by :

1) Flow based forwarding
2) Seperation of data & control planes
3) Externalized network control functions
4) Programmable network



(1) Flow-Based Forwarding

→ Forwarding can depend on any header field (L2, L3, L4)

→ OpenFlow 1.0 allows forwarding based on 11 header fields

→ Control plane (via controller) computes & installs flow rules into switches

(2) Seperation of data & control planes

→ Data plane comprises the switches executing "match + action" rules in flow tables (→ switches = simple + fast)

→ Control plane comprises servers & software managing these rules via controller (→ Controller = smart + centralized logic)

(3) Externalized network control functions

→ Control software runs on external servers, not on routers/switches

→ Components :

1) SDN Controller (Network Operating System)

→ Maintains global network state (links, hosts, switches)

→ Provides APIs to applications

2) Network-Control Applications

→ Use controller APIs to configure network behavior like routing (→ Dijkstra for path setup), access control (→ Firewall), load balancing

→ Basically, controller is logically centralized but physically distributed for scalability & reliability

(4) Programmable network

→ Network logic implemented as software applications on top of controller APIs

→ Enables innovation & vendor independance

ex- Routing apps, Access control, Load balancing apps
↳ Path selection ↳ Firewalling

→ This unbundles network components

i) Data Plane (Switch)

ii) Control Plane (Controller)

iii) Network Apps

→ SDN Control Plane - Role

→ Computes flow tables & installs them into switches

→ Updates rules dynamically when network events occur (ex- link failure)

→ Coordinates flow entries across multiple switches for :

i) End-to-end routing paths

ii) Distributed firewalling

iii) QoS enforcement

SDN Controller

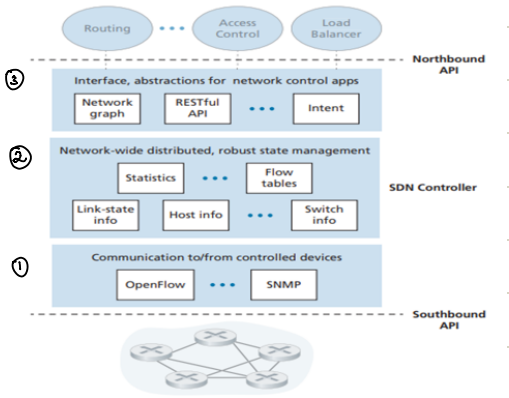
→ The SDN Control Plane consists of 2 major components

i) SDN Controller

ii) SDN Network-Control Applications

→ The controller is the central software system that connects data plane & applications defining the network behavior

→ Generic SDN has 3 layers



1) Lowest Layer - Communication Layer (Southbound interface)

→ It enables communication b/w SDN controller & SDN controlled devices (switches, routers, hosts)

→ The main functions are:

i) Exchange control messages with switches

ii) Receive event notifications from devices like:

Link up/down, New device joins, Heartbeat or Status messages

→ This layer keeps the controller's network state up-to-date

→ Communication occurs through the Southbound interface

ex - OpenFlow is the most widely used southbound protocol

2) Middle Layer - Network wide state Management Layer

→ Maintains a complete & real time view of the entire network's state

→ Contains info about switches, hosts, links, Flow tables, Counters and statistics from switches

→ It provides network data to upper-layer applications for decision making

→ Ensures that flow tables are correctly computed and synchronized

ex of state info - Link Utilization, Topology maps, Packet counts, Flow aging timers etc.,

3) Top Layer - Interface to Network-Control Applications (Northbound Interface)

→ It provides an API for applications (routing, load balancing, firewall etc.,) to interact with controller

→ Main functions are:

i) Allows applications to read/write network state & flow tables

ii) Supports event notifications (ex- Topology change, Device status change)

iii) Enables application-driven control of network devices

→ Usually it is implemented using a RESTful API (HTTP-based request-response model)

→ Logical vs Physical Centralization

→ From the network's pov, controller behaves like a single central brain but in practise, it is implemented across multiple servers for Fault tolerance, High availability, Scalability

→ Challenges of this include :

i) Maintaing event order, consistency & consensus across servers

ii) Addressed by distributed systems principles like Lamport, Lampson Algorithms

ex- Open Daylight (ODL) - modular, Java based platform

Open Network Operating System (ONOS) - Distributed, high performance controller

→ OpenFlow Protocol (Southbound Interface Protocol)

→ This protocol enables communication b/w controller & OpenFlow-enabled switches

→ Operates over TCP (Port 6653 is default)

→ Messages from Controller to Switch

i) Configuration - Query or set a switch's configuration parameters

ii) Modify state - Add, delete or modify flow table entries & set port properties

iii) Read state - Retrieve statistics and counter values from the switch

iv) Send packet - Send a specific packet (in message payload) out of a given port

→ Messages from Switch to Controller

i) Flow Removed - Notifies controller when flow entry is removed (timeout or rule change)

ii) Port Status - Informs controller of port changes (ex- link failure or reactivation)

iii) Packet in - Sent when incoming packet has no match in flow table. Controller takes final call

→ Working

i) A packet arrives at a switch

ii) The switch checks the flow table

iii) If no match, sends "packet-in" message to controller

iv) Controller decides an action (install rule, drop, forward etc.,)

v) Sends "Modify - State" or "Send - Packet" message back to switch

Data Plane and Control Plane Interactions

→ In traditional n/w :

→ Each router runs its own control plane logic (ex- OSPF or Dijkstra)

→ Routers exchange link-state updates directly with each other

→ Routing decisions are distributed & vendor-specific

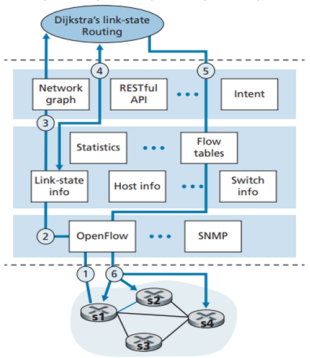
→ In SDN :

→ Control logic runs centrally in the controller

→ Switches act only as data plane devices & not as complete routes

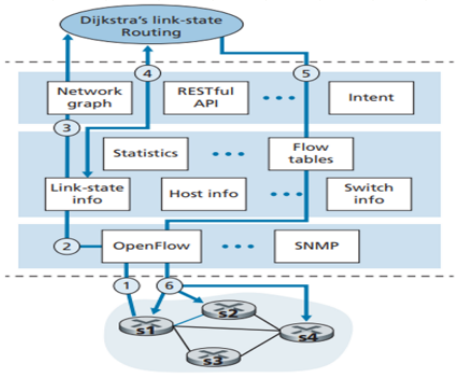
→ Switches send event updates to controller & not to switches

↳ like link failure



→ This shift fundamentally changes how network reacts & updates to events

→ Dijkstra's Algorithm in SDN Context



→ Consider the example in the diagram

→ Shortest path is implemented by Dijkstra's algorithm

→ Algorithm runs as SDN application, outside the switches

→ Communication b/w switches & controller occurs through OpenFlow

→ Now assume a scenario of link failure b/w S1 & S2 to trace how control plane & data plane interact

1) Switch s1 detects link failure
   → s1 sends an OpenFlow Port-Status message the controller, reporting link-down event
2) SDN Controller
   → Receives the event, and passes it to Link-State manger which updates link-state database in state-management layer
3) Link-State Routing Application
   → The routing app has registered to be notified upon chance, hence, it has been notified
   ↳ In control plane
4) Routing Application Processing
   → The app retrieves updated topology from state manager & recomputes new shortest path using Dijkstra's algorithm
5) Flow Table Manager
   → Determines which switches' flow table must be updated (Here s1, s3, s4)
6) Controller to Switches (OpenFlow)
   → Controller sends Modify-State messages via OpenFlow to update the flow of s1, s3, s4 to reflect new routes

→ By this, s1 now forwards packets destined for s2 via s4, and s4 forwards packets from s1 towards s2. s2 receives packets through the new path

→ From this we can observe:
   → SDN controller replaces the distributed per-router control plane
   → It enables:
      → Easy policy or algorithm changes (ex- switch from shortest path to custom load balancing)
      → Simplified management (all logic centralized)
      → Vendor independance (logic implemented in open software)

→ With just one software update, SDN enable ISP can completely redefine routing behavior w/o touching individual routers