CS348 Notes
Intro to Layer 3 Routing
Intra - Domain Routing
Video Numbers: 15, 16, 17

OjMaha

I have prepared these notes by watching the videos from Networks Playlist. The following notes may be asynchronous and irrelevant to what Prof. Vinay teaches in class (cuz I do not pay attention during lectures lol). Further, these notes might not cover *everything* as explained in the video lectures. Consider these to be a supplemental read :). If you find any errors, do notify me so they can be edited.

Layer-2 switching provided a way to connect two LANs. It prevented multiple collisions if the LAN was directly connected. Thanks to the switch's intelligence.

But, it is not possible to scale it to billions of devices.

Why? The spanning tree might not be optimal. Some ports are not being used so optimal paths may get discarded. Hence, poor resource utilization. Further, too many forwards. Forwarding Rate $\propto$ no. of hosts.

The forwarding table will be of size O(N). Lookup also tough.

↳ flat addressing. → map from MAC address to port number.

↳ 2^{48} diff MAC addresses.

Even a small LAN may have a wide range of MAC addresses.

↳ Layer 3 switching uses IP addresses : hierarchical addressing

thus removing flat addressing problem.

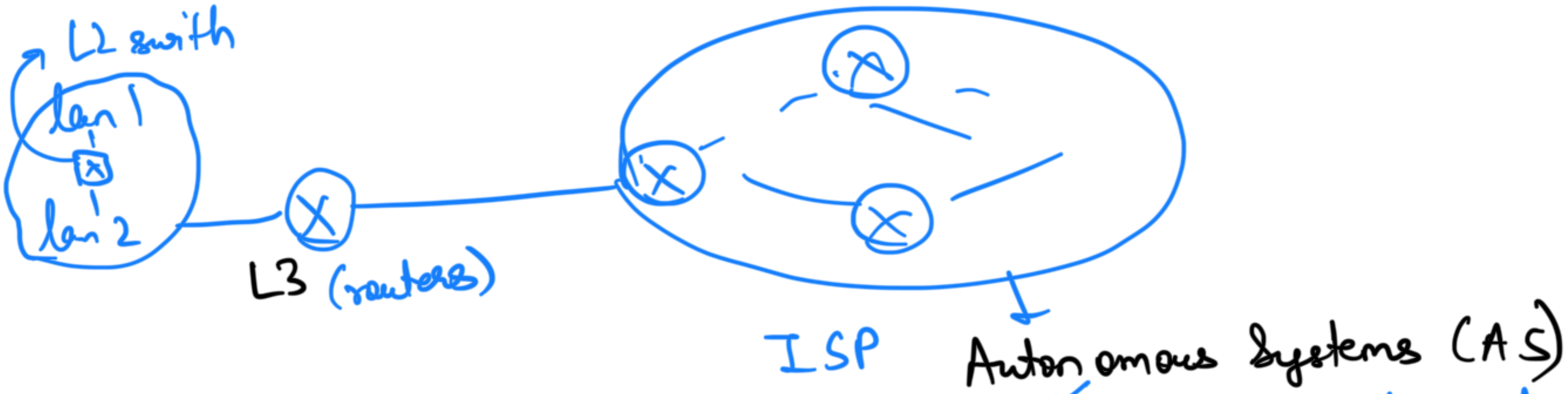
Another issue is that stability. If root fails then SPT has to be reconstructed. If "hello" messages from root don't come ⇒ reconstruct SPT.

Same problem if any switch fails.

For larger LANs → reconstruct SPT more often.

There is no common addressing scheme or communication protocol globally making scalability of Ethernet Switching a pain in the ass.

L2 switch is blind to IP addresses. L3 switch forwards based on IP address.



cuz they can choose any routing protocol inside.

Intra-domain routing: routing protocols within AS

Inter-domain routing: " " between AS.

↳ BGP (Border gateway protocol)

Intra-domain Routing

Distance Vector

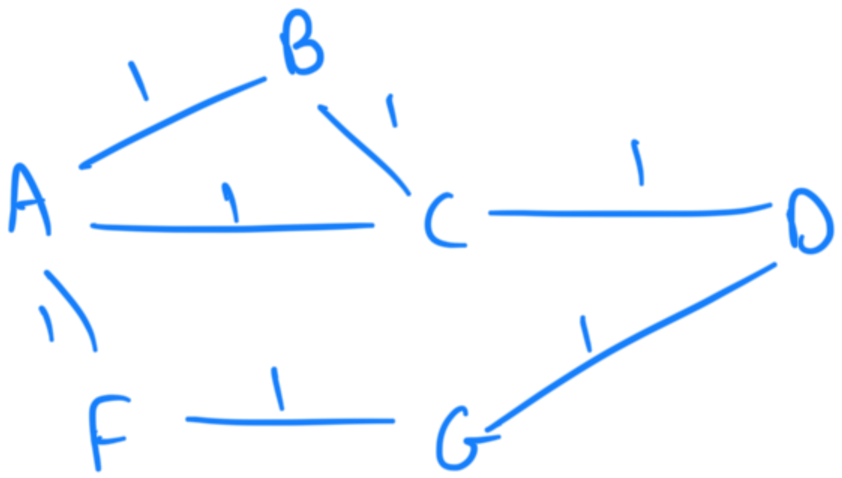
RIP: Routing info protocol

Link State Routing

OSPF: open shortest path first

IS-IS: intermediate system to IS.

Distance Vector: RIP



step1:

A sends out: (A,0) and hears (B,0); (C,0); (F,0).

Routing table @ A :

Dest''	next hop	Cost
A	—	0
B	B	1
C	C	1
F	F	1

Step2:

A sends out to own neighbors : (A,0) (B,1) (C,1) (F,1). → current shortest path is to F.

hears from B : (C,1) (A,1)

from C : (D,1) (B,1) (A,1)

from F : (A,1) (G,1)

Now; routing table becomes.

Dest.	Next hop	Cost
A	-	0
B	B	1
C	C	1
F	F	1
D	C	2
E	E	1

(event triggers routing update)

⇑

G F 2

triggered update

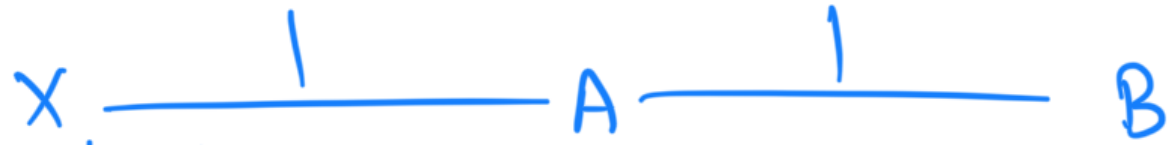
Now, suppose F-G link fails. Then F tells A : (G, ∞) & updates its table. The advertisements keep coming periodically & C tells A: (G,2)

Then entry becomes G C 3 .

periodic update

(tell neighbour (dest, dist))

Count to Infinity Problem :



Dest	next	cost
A	A	1
B	A	2

X	X	1
B	B	1

A	A	1
X	A	2

Now say X–A fails.

A sends trigger update to B : (X, ∞)

Say B sent its periodic update at almost the same time : $(X, 2)$; $(A, 1)$

Now A first updates X – ∞. Then from info he recd from B; it updates it to X B 3. (which is evidently false).

Note that B correctly updates it to X – ∞.

Then in next periodic update:

A tells B : $(X, 3)$; $(B, 1)$.

B receives & updates entry to X A 4.) increasing dist.

B receives & updates

B tells A : (X,4) ; (A,1)

Then A updates entry to X B 5 . ∞

B thinks A is next hop to X. A thinks B is the next hop to X.

RIP: allows a max distance of 16. Cost =16 ⇒ can't reach destination.

To solve count-to-∞ problem;

Split-Horizon :

Do not advertise info about a destination if the neighbor is the next hop to the destination.

Now, when X-A fails :

A tells B : (X,∞)

B tells A nothing abt. distance to X. (hehe RG kaat li)

Thus B updates its table correctly now as X - ∞ .

Moral: sometimes it is good to cut RG.

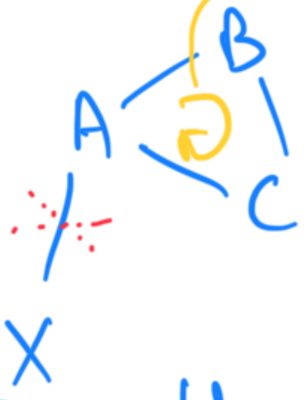
Split Horizon with Poison Reverse:

A node tells its next hop to a destⁿ that its distance to the destination is ∞. (misinfo)

irrespective of whether failure has occurred or not.

B sends advertisements to A that (X, ∞) since A is the next hop otw to X. Note that A doesn't and shouldn't care about this info since A is closer to X than B by one by virtue of the fact that A is the next hop toward X from B.

Example: routing loop issue.



Each one has their own routing table. Now suppose A-X fails. A will tell (X, ∞) to B & C. But now suppose message to C gets lost.

Say We use split horizon here.

B & C are silent to A abt. X. B tells C : (X, ∞). C tells B : (X, 2)

it didn't receive msg.

Now B thinks there is some path to X via C in 2 units.

Now B [illegible]

B's table: X C 3 } after it tells A.

A's table : X B 4

{ C still has entry of X A 2 }

Now A tells C : (X, 4). Loop Created !!

Now, the distance keep increasing in the loop. C updates X A 5. B updates X C 6 . . . .

C thinks A is the next hop. A thinks B is the next hop. B thinks C is the next hop.

RIP:

Use Distance Vector, Cost of all links are 1. Max cost = 16 ; (16 = ∞).

It can be used only in smaller networks coz wut if no. of hops are actually >16.

Pros: Distance Vector is easy to implement

Cons: Count to ∞, routing loops. It takes time for routing table to converge.

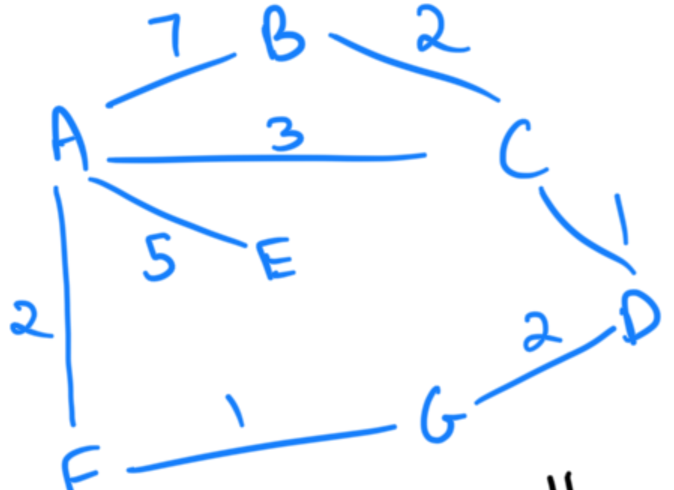
Link-State-Routing :

Each node sends to all others (broadcast) info about cost to immediate neighbors.

Say D has 2 neighbours $(A, \overset{cost}{1})$; $(C, \overset{cost}{5})$. It sends this info to all nodes (even non-neighbors)

Dijkstra Algo :

Each node finds shortest path tree to all other nodes in network.



T = {A}

T = {A, F}

T = {A, F, G}

T = {A, F, G, C}

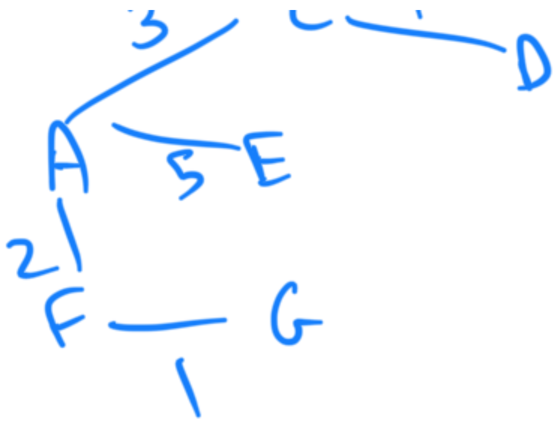
T = {A, F, G, C, D}

T = {A, F, G, C, D, E, B}

Keep adding shortest neighbors. Maintain current cost to reach leaf, add node with min^m distance greater than or equal to this cost.

⇓

Then A's routing table:



Thus; A's routing table:

Dest	Next	Cost
B	C	5
C	C	3
D	C	4
E	E	5
F	F	2
G	F	3

When a link fails wut to do?

Say A-F fails.

→ A and F broadcast to all that their link has failed.

→ All rerun Dijkstra algo.

Pros: No routing loops, count-to-∞. Convergence of routing table is faster.

Cons: Algo is more complicated (Now, is it really?) than distance vector.

Now, how to choose the link weights (costs)?

choose bandwidth, latency as parameters maybe.

Let's talk a bit abt <u>ARPANET</u>.

It had two types of speeds 56kbps and 9.6kbps. There were two links;
Satellite Links, Terrestrial Links (self explanatory)

Idea 1: Use latency.

time for packets ahead in q.

time for first bit to reach

bandwidth

latency of this pkt = queuing delay + speed of light delay + transmission delay

(P/C)

packet size.

Calculate avg latency of all packets in a time window.
Give wt. accordingly.

Issues:

1) A lot of oscillations. A particular path (A→B) is preferred in a time window. Then due to queuing delay then (A→C) is preferred in next then again A→B. The latencies flip baar baar. Under heavy load, routing path

oscillations occur. Queuing delays go up → increased link wt → new shortest path
(on paths used heavily)

2) End-to-End latency (eg: A to B) keeps varying which might affect APPL. layer performance.

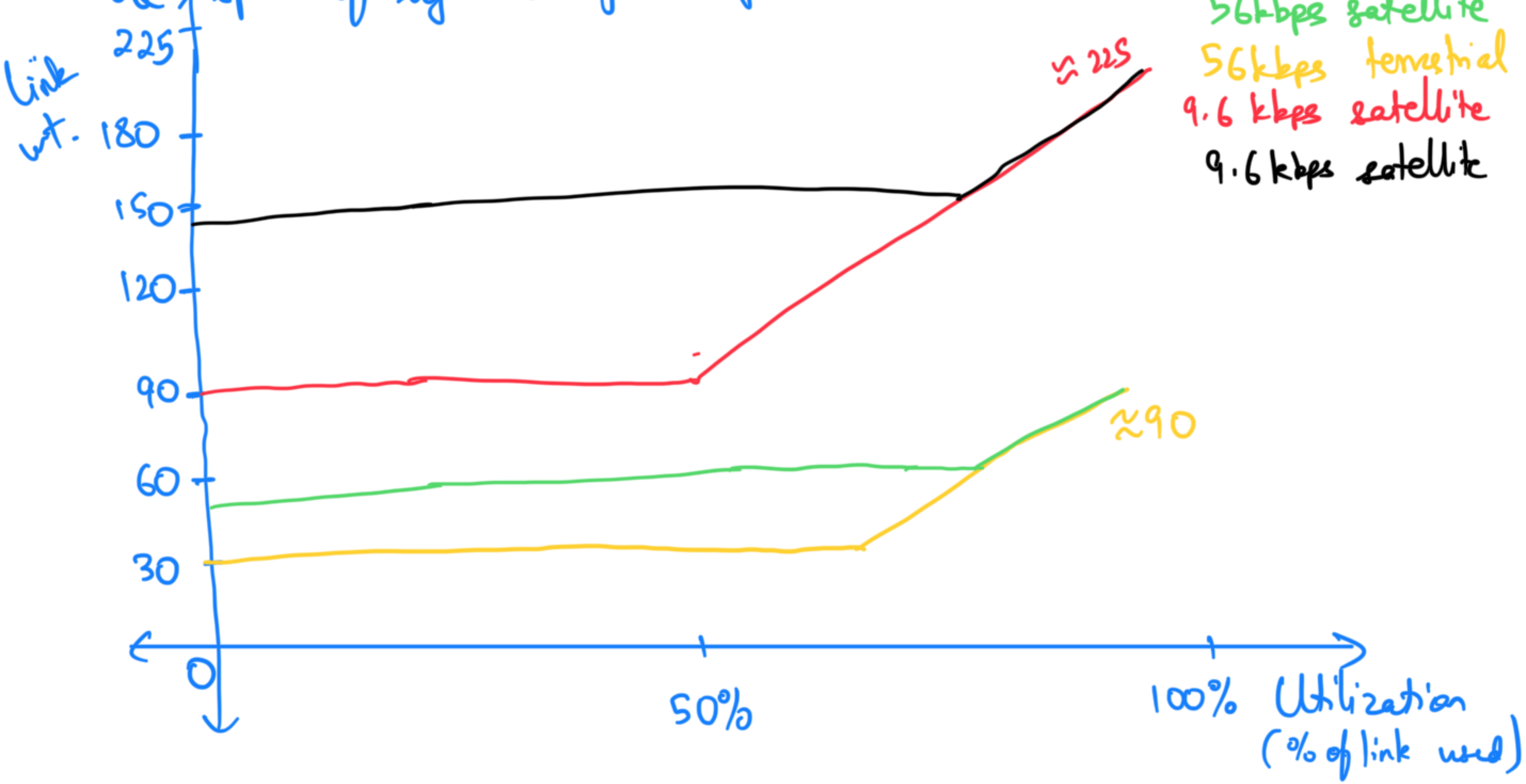
3) Packets may reach re-ordered.

4) Routing Loops are Possible. How?
If we have a large network, it takes time for weights to converge so for the meantime (when it is converging) if A receives packet to send to X; it might think B is next hop and B might also think that A is next hop. Note that this happens cuz info abt nodes haven't fully reached yet.

5) The range of link weights was very large resulting in over-penalization.
126 links of 56 kbps had same wt as single 9.6 kbps link.
↳ too high weight.

6) Satellite links penalized too much.

9 kbps terrestrial had lower wt than 56kbps satellite (under few conditions) as speed of light delay is high.



We avoid routing loops by measuring over a large time period. Weights are changed infrequently.

OSPF: weight of link = $\max\left(\frac{10^8}{\quad}, 1\right)$

(link speed (bps))