Table of contents

Introduction

In this lecture we are focusing on the network layer and on a specific function of the network layer, which is routing within a single administrative domain. Let's remember that our overarching theme of the course is to understand what it takes for two hosts to exchange data. In this lecture, we will zoom into the protocols that we need so that data can travel over a good path from the source to the destination within a single administrative domain. We'll learn about the two flavors of intradomain routing algorithms which are the link-state and distance-vector algorithms. We'll look at example protocols such as RIP and OSPF. We will also look at challenges that intradomain routing protocols face such as convergence delay. Finally, we will look at how routing protocols are used for purposes that go beyond determining a good path; how we can use routing for traffic engineering purposes so that we can steer traffic through the network avoiding congested links for example.

Readings and Additional Resources

Important readings:

Experience in Black-box OSPF Measurement

http://conferences.sigcomm.org/imc/2001/imw2001-papers/82.pdf (Links to an external site.)

Book references:

If you have access to the Kurose-Ross book and to the Peterson book, below you can find the list of the chapters we talked about in this lecture. As we have mentioned in the course schedule, purchasing the books is not required.

Kurose-Ross (Edition 6): 4.5.1, 4.5.2, 4.6.1, 4.6.2

Optional Readings:

Hot Potatoes HeatUp BGP Routing

https://www.cs.princeton.edu/~jrex/papers/hotpotato.pdf (Links to an external site.)

Traffic Engineering With Traditional IP Routing Protocols

https://www.cs.princeton.edu/~jrex/teaching/spring2005/reading/fortz02.pdf (Links to an external site.)

Dynamics of Hot-Potato Routing in IP Networks

https://www.cs.princeton.edu/~jrex/papers/sigmetrics04.pdf

OSPF Monitoring: Architecture, Design and Deployment Experience

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.86.8523&rep=rep1&type=pdf

Routing Algorithms

Let's assume that we have two hosts that have established a connection between them using TCP or UDP as we saw in the previous lecture. Each of the two hosts know the default router (or first-hop router as we say). When a host sends a packet, the packet is first transferred to that default router. But what happens after that? In this lecture we will see the algorithms that we need so that when a packet leaves the default router of the sending host, it will travel over a path towards the default router of the destination host.

A packet will be able to travel from the sending host to the destination host with the help of intermediate routers. When a packet arrives at a router, the router is responsible to consult a forwarding table and then to determine the outgoing link interface to forward the packet. So by **forwarding** we refer to transferring a packet from an incoming link to an outgoing link within a *single* router. We will talk about forwarding in a following lecture.

By **routing** we refer to how routers work together using routing protocols to determine the good paths (or good routes as we call them) over which the packets travel from the source to the destination node. When we have routers that belong to the same administrative domain we refer to the routing that takes place as intradomain routing. But when the routers belong to different administrative domains, we refer to interdomain routing.

In this lecture we focus on **intradomain routing algorithms** or Interior Gateway Protocols (IGPs). The two major classes of algorithms that we have are: A) link-state and B) distance-vector algorithms. For the following algorithms we represent each router as a node, and a link between two routers as an edge. Each edge is associated with a cost.

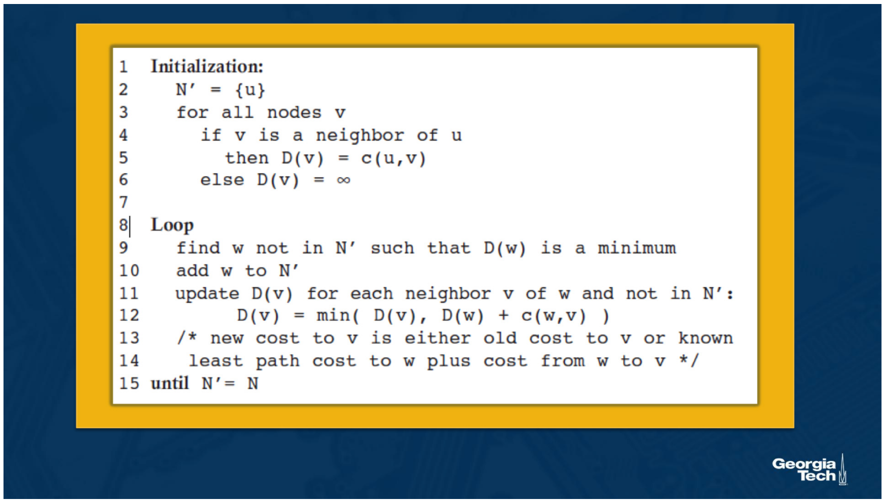
Linkstate Routing Algorithm

In this topic, we will talk about the link state routing protocols, and more specifically about Dijkstra's algorithm. In the linkstate routing protocol, the link costs and the network topology are known to all nodes (for example by broadcasting these values).

Let's introduce some basic terminology. Our source node is denoted u, and v denotes every other node in the network. The cost of the current least-cost path from u to v is $D(v)$. The previous node along the current least cost path from u to v is denoted $p(v)$. Finally, N' denotes the subset of nodes along the current least-cost path from u to v.

Initialization: The algorithm starts with an initialization step, where we initialize all the currently known least-cost paths from u to its directly attached neighbors. We know these costs because they are the costs of the immediate links. For nodes in the network that are not directly attached to u, we initialize the path cost to infinity. We also initialize the set N' to include only the source node u.

Iterations: After the initialization step, the algorithm follows with a loop that is executed for every destination node v in the network. At each iteration, we look at the set of nodes that are not in N', and we identify the node (say w) with the least-cost path from the previous iteration. We add that node w into N'. For



```
Initialization:
  N' = {u}
  for all nodes v
    if v is a neighbor of u
      then D(v) = c(u,v)
    else D(v) = ∞

Loop
  find w not in N' such that D(w) is a minimum
  add w to N'
  update D(v) for each neighbor v of w and not in N':
       D(v) = min( D(v), D(w) + c(w,v) )
  /* new cost to v is either old cost to v or known
   least path cost to w plus cost from w to v */
until N'= N
```

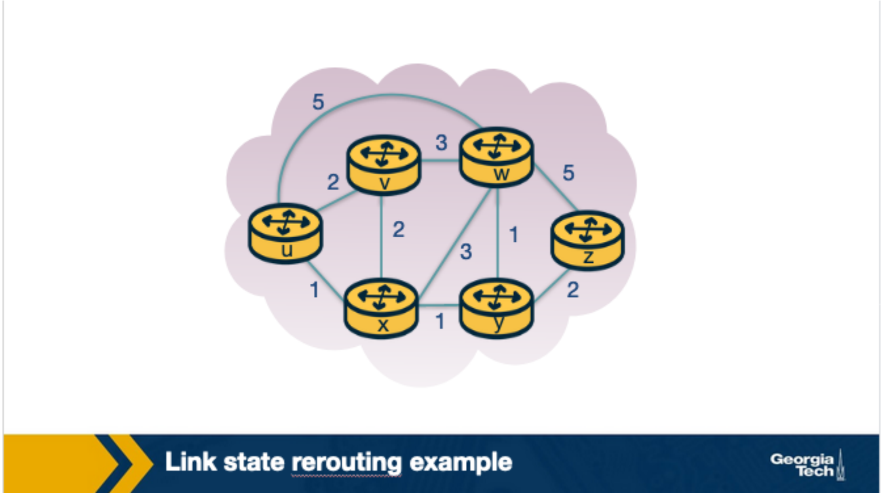
Figure 1: image

every neighbor v of w, we update $D(v)$ with the new cost, which is either the old cost from u to v (from the previous iteration) or the known least-cost path cost from source node u to w, plus the cost from w to v, whichever of the two quantities is the minimum.

The algorithm exits by returning the shortest paths, and their costs, from the source node u to every other node v in the network.

Linkstate Routing Algorithm - Example

Let's look at an example of the linkstate routing algorithm. We have the graph below and we consider our source node to be u. Our goal is to compute the least-cost paths from u to all nodes v in the network.



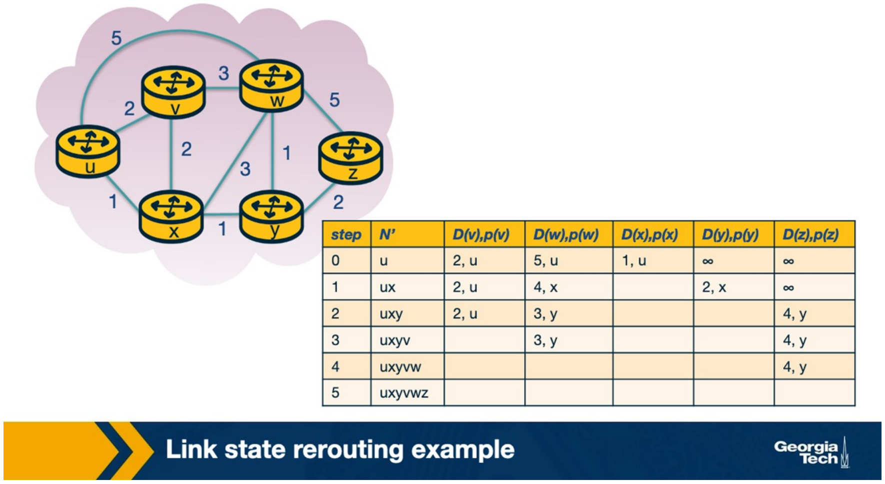
Link state rerouting example

Figure 2: image

We start with the initialization step, where we set all the currently known least-cost paths from u to its directly attached neighbors v, x and w. For the rest of the nodes in the network we set the cost to infinity, because they are not immediate neighbors to source node u. We also initialize the set N' to include only the source node u. The first row in our table represents the initialization step.

In the first iteration, we look among the nodes that are not yet in N', and we select the node with the least cost from the previous iteration. In this case, this is node x. Then, we update D for all the immediate neighbors of x, which in this case are nodes v and w. For example, we update $D(w)$ as the minimum between: the cost we had from the previous iteration which is 5, and the cost from u to x (1) plus cost from x to w (3). The minimum between the two is 4. We update the second row in our table.

We continue in a similar manner for the rest of the nodes in the table. The algorithm exits in the 5th iteration.



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	4, x		2, x	∞
2	uxy	2, u	3, y			4, y
3	uxyv		3, y			4, y
4	uxyvw					4, y
5	uxyvwz					

Link state rerouting example

Georgia Tech

Figure 3: image

Linkstate Routing Algorithm - Computational Complexity

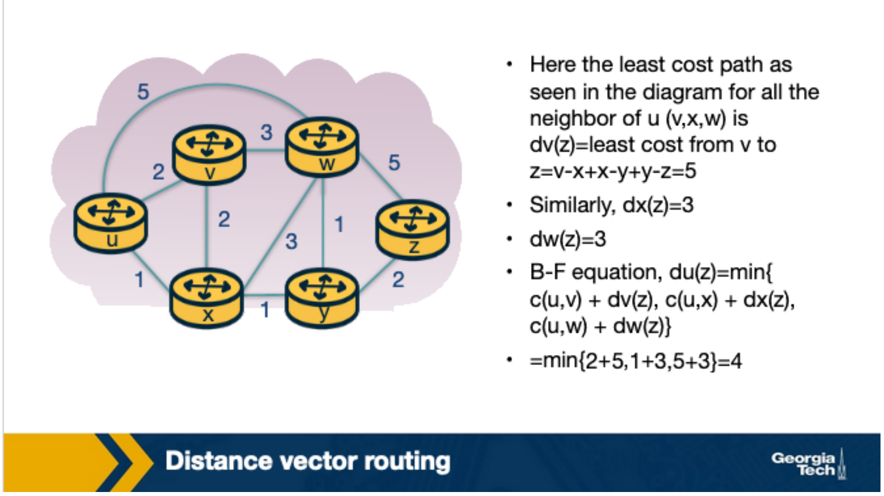
What is the computational complexity of the linkstate routing algorithm? In other words, in the worst case, how many computations are needed to find the least-cost paths from the source to all destinations in the network? In the first iteration, we need to search through all nodes to find the node with the minimum path cost. But as we proceed in the next iterations, this number decreases. So, in the second iteration we search through $(n-1)$ nodes. This decrease continues at every step. So, by the end of the algorithm, after we go through all the iterations, we will need to search through $n(n+1)/2$ nodes. Thus, the complexity of the algorithm is on the order of n squared $O(n^2)$.

Distance Vector Routing

In this section, we will talk about the distance vector (DV) routing algorithm. The DV routing algorithm is iterative (the algorithm iterates until the neighbors do not have new updates to send to each other), asynchronous (the algorithm does not require the nodes to be synchronized with each other), and distributed (direct nodes send information to one another, and then they resend their results back after performing their own calculations, so the calculations are not happening in a centralized manner).

The DV algorithm is based on the Bellman Ford algorithm. Each node maintains its own distance vector, with the costs to reach every other node in the network. Then, from time to time, each node sends its own distance vector to its neighbor nodes. The neighbor nodes in turn, receive that distance vector and they use it to update their own distance vectors. In other words, the neighboring nodes exchange their distance vectors to update their own view of the network.

How the vector update is happening? Each node x updates its own distance vector using the Bellman Ford equation: $D_x(y) = \min_v[c(x,v) + D_v(y)]$ for each destination node y in the network. A node x computes the least cost to reach destination node y by considering the options that it has to reach y through each of its neighbors v. So, node x considers the cost to reach neighbor v, and then it adds the least cost from that neighbor v to the final destination y. It calculates that quantity over all neighbors v and it takes the minimum.

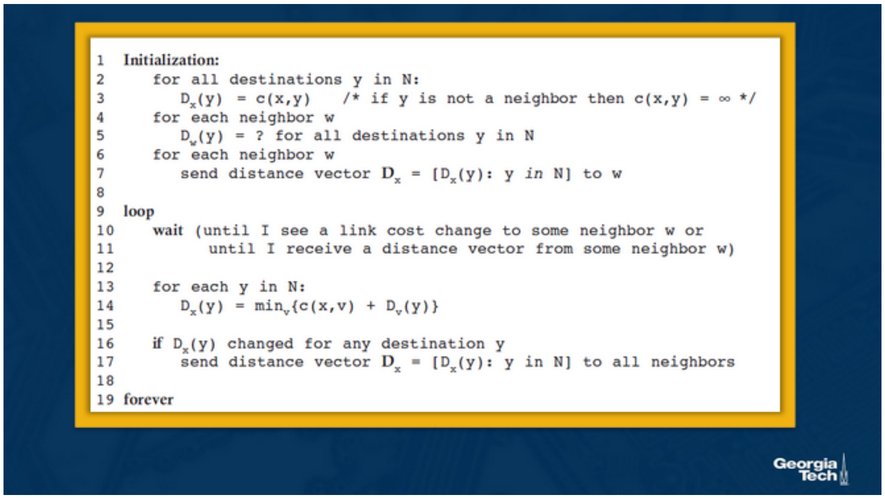


- Here the least cost path as seen in the diagram for all the neighbor of u (v,x,w) is dv(z)=least cost from v to z=v-x+x-y+y-z=5
- Similarly, dx(z)=3
- dw(z)=3
- B-F equation, du(z)=min{ c(u,v) + dv(z), c(u,x) + dx(z), c(u,w) + dw(z)}
- =min{2+5,1+3,5+3}=4

Distance vector routing

Figure 4: image

Formally, the DV algorithm is as follows:



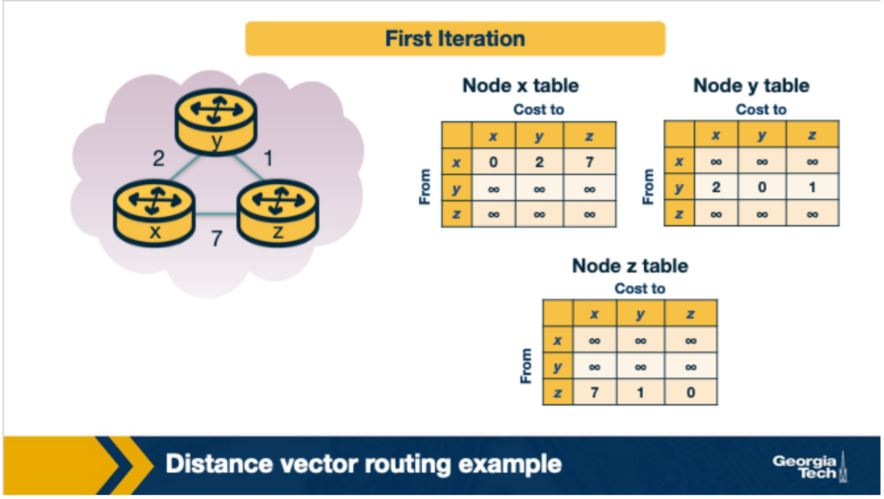
```
Initialization:
   for all destinations y in N:
      $D_x(y) = c(x,y)$   /* if y is not a neighbor then $c(x,y) = \infty$ */
   for each neighbor w
      $D_w(y) = ?$ for all destinations y in N
   for each neighbor w
      send distance vector $\mathbf{D}_x = [D_x(y)$: y in N] to w

loop
   wait (until I see a link cost change to some neighbor w or
         until I receive a distance vector from some neighbor w)

   for each y in N:
      $D_x(y) = \min_v\{c(x,v) + D_v(y)\}$

   if $D_x(y)$ changed for any destination y
      send distance vector $\mathbf{D}_x = [D_x(y)$: y in N] to all neighbors

forever
```

Figure 5: image

Distance Vector Routing Example

Now, let's see an example of the distance vector routing algorithm. Let's consider the three node network shown here:

In the first iteration, each node has its own view of the network, which is represented by an individual table. Every row in the table is the distance vector of each node. Node x has its own table, and the same is true for nodes y and z. We note that in the first iteration, node x does not have any information about y's and z's distance vectors. Thus, these values are set to infinity.



First Iteration

Node x table

Cost to

		x	y	z
From	x	0	2	7
	y	∞	∞	∞
	z	∞	∞	∞

Node y table

Cost to

		x	y	z
From	x	∞	∞	∞
	y	2	0	1
	z	∞	∞	∞

Node z table

Cost to

		x	y	z
From	x	∞	∞	∞
	y	∞	∞	∞
	z	7	1	0

Distance vector routing example

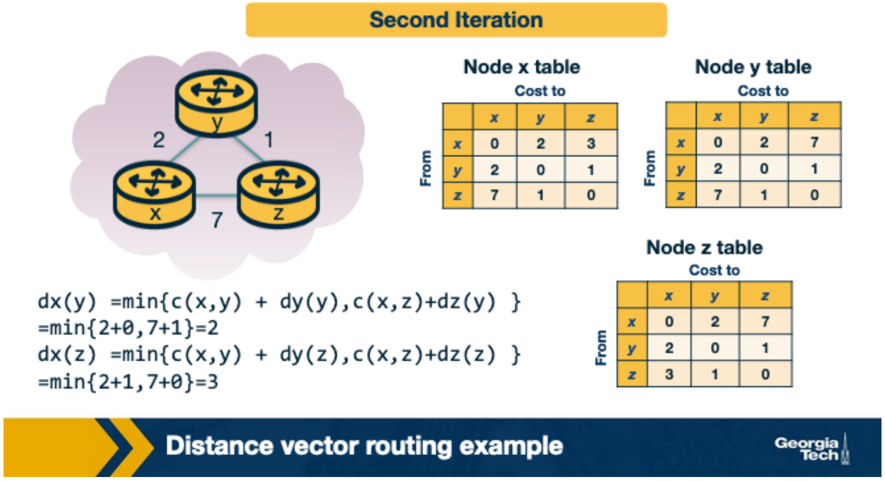
Figure 6: image

In the second iteration, the nodes exchange their distance vectors and they update their individual views of the network. Node x computes its new distance vector, using the Bellman Ford equation for every destination node y and z. For each destination, node x compares the cost to reach that destination through a neighbor node:

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

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

At the same time, node x receives the distance vectors from y and z from the first iteration. So, it updates its table to reflect its view of the network accordingly. Nodes y and z repeat the same steps to update their own tables.



Second Iteration

Node x table

Cost to

		x	y	z
From	x	0	2	3
	y	2	0	1
	z	7	1	0

Node y table

Cost to

		x	y	z
From	x	0	2	7
	y	2	0	1
	z	7	1	0

```
dx(y) =min{c(x,y) + dy(y),c(x,z)+dz(y) }
=min{2+0,7+1}=2
dx(z) =min{c(x,y) + dy(z),c(x,z)+dz(z) }
=min{2+1,7+0}=3
```

Node z table

Cost to

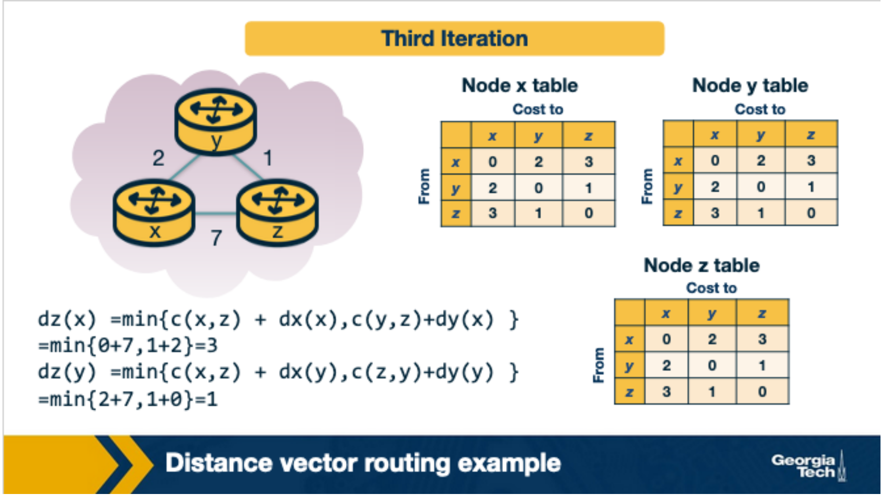
		x	y	z
From	x	0	2	7
	y	2	0	1
	z	3	1	0

Distance vector routing example

Figure 7: image

In the third iteration, the nodes get the distance vectors from the previous iteration (if they have changed), and they repeat the same calculations. Finally, each node has its own routing table.

Finally, at this point, there are no further updates sent from the nodes; thus, the nodes are not doing any further calculations on their distance vectors. The nodes enter a waiting mode, until there is a change in the link costs.



Third Iteration

Node x table

Cost to

From	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

Node y table

Cost to

From	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

Node z table

Cost to

From	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

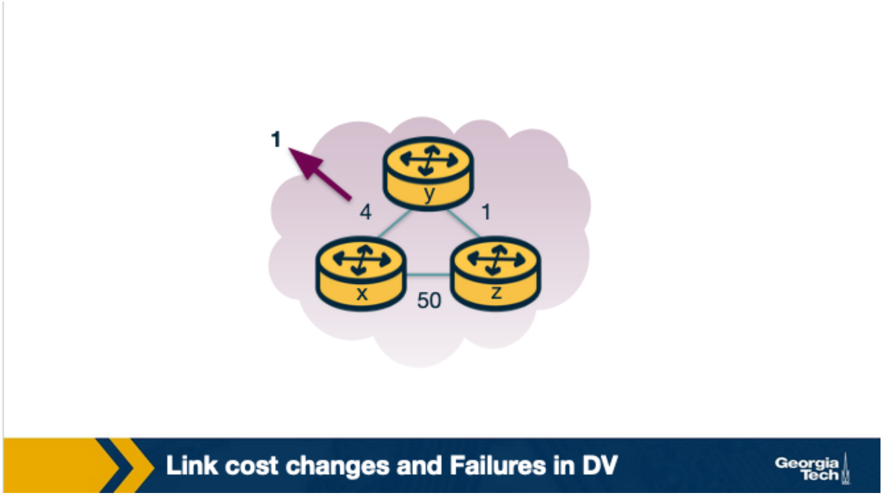
```
dz(x) =min{c(x,z) + dx(x),c(y,z)+dy(x) }
=min{0+7,1+2}=3
dz(y) =min{c(x,z) + dx(y),c(z,y)+dy(y) }
=min{2+7,1+0}=1
```

Distance vector routing example

Figure 8: image

Link Cost Changes and Failures in DV - Count to Infinity Problem

Now, we will see what happens when a node identifies that a link connecting it to one of its neighbors as changed. Let's consider the following example topology below:



Link cost changes and Failures in DV

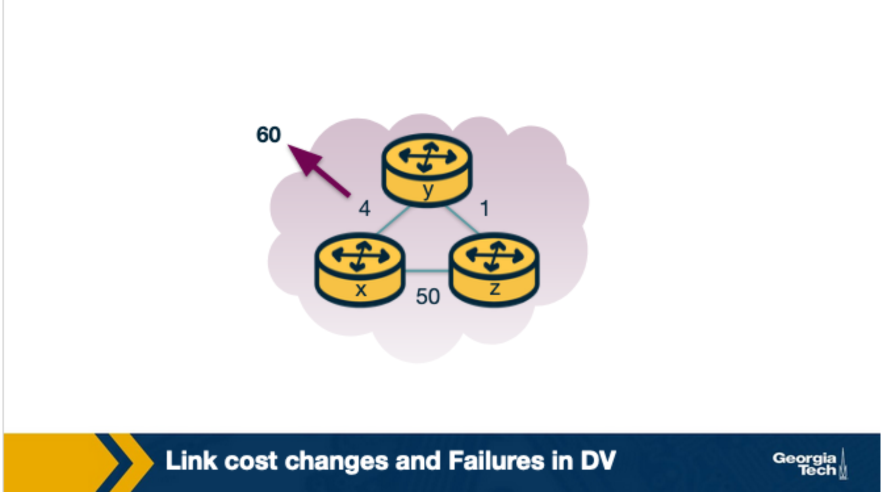
Figure 9: image

Let's assume that the link cost between $x - y$ changes from 4 to 1.

1. At time t_0, y detects that cost to x has changed from 4 to 1, so it updates its distance vector and sends it to its neighbors.
2. At time t_1, z receives the update from y. Now it thinks that is can reach x through y with a cost of 2. And it sends its new distance vector to its neighbors.
3. At time t_2, y receives update from z. The y distance vector does not change its distance vector and does not send updates.

In this scenario, we note that the fact that there was a decrease in the link cost, it propagated quickly among the nodes, as it only took a few iterations.

Unfortunately, this is not always the case. Let's consider the following scenario where a link cost increases by a large amount.



Link cost changes and Failures in DV

Figure 10: image

Let's assume that the link $y - x$ has a new cost of 60.

1. At t_0, y detects that cost has changed, now it will update its distance vector thinking that it can still reach x through z with a total cost of $5 + 1 = 6$.
2. At t_1, we have a routing loop, where z thinks it can reach x through y, and y thinks it can reach x through z. This will cause the packets to bounce back and forth between y and z until their tables change.
3. z and y keep updating each other about their new cost to reach x. For example, y computes its new cost to be 6, and it informs z. Then, z computes its new cost to be 7, and it informs y, and so on.

This back and forth continues for a total of 44 iterations, at which point z computes its cost to be larger than 50, and at that point it will prefer to reach x directly rather than through y. In contrast to the previous scenario, this link cost change took a long time to propagate among the nodes of the network. This is known as the count-to-infinity problem.

Poison Reverse

A solution to the previous problem is the following idea, called poison reverse: since z reaches x through y, z will advertise to y that the distance to x is infinity ($D_z(x) = \infty$). However, z knows that this is not true and $D_z(x) = 5$. z tells this lie to y, as long as it knows that it can reach x via y. Since y assumes that z has no path to x except via y, it will never send packets to x via z. So, z *poisons* the path from z to y.

Things change when the cost from x to y changes to 60. y will update its table and send packets to x directly with cost $D_y(x) = 60$. It will inform z about its new cost to x, after this update is received. Then, z will immediately shift its route to x to be via the direct (z, x) link at cost 50. Since there is a new path to x, z will inform y that $D_z(x) = 50$.

When y receives this update from z, y will update $D_y(x) = 51 = c(y, z) + D_z(x)$.

Since z is now on least cost path of y to reach x, y poisons the reverse path from z to x. y tells z that $D_y(x) = \infty$, even though y knows that $D_y(x) = 51$.

This technique will solve the problem with 2 nodes, however poisoned reverse will not solve a general count to infinity problem involving 3 or more nodes that are not directly connected.

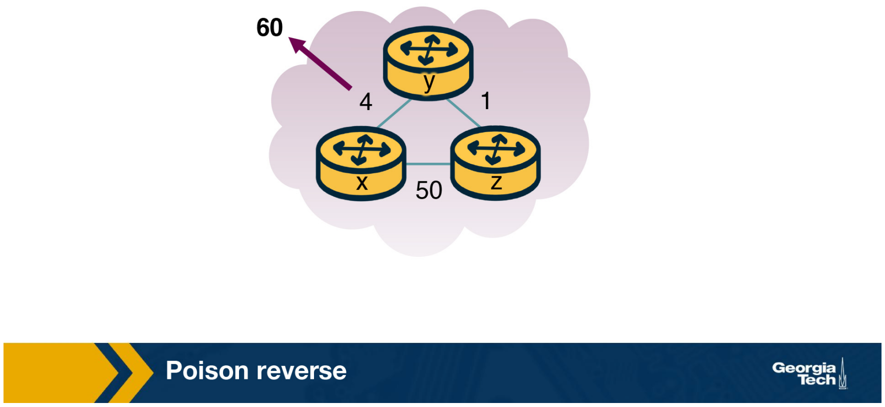


Figure 11: image

Distance Vector Routing Protocol Example: RIP

The **Routing Information Protocol (RIP)** is based on the Distance Vector protocol. The first version, released as a part of the BSD version of Unix, uses hop count as a metric (i.e. assumes link cost as 1). The metric for choosing a path could be shortest distance, lowest cost or a load-balanced path. In RIP, routing updates are exchanged between neighbors periodically, using a RIP response message, as opposed to distance vectors in the DV Protocols. These messages, called RIP advertisements, contain information about sender's distances to destination subnets.

Let's look at a simple RIP example to illustrate how it works. The figure below shows a portion of the network. Here, A, B, C and D denote the routers and w, x, y and z denote the subnet masks.

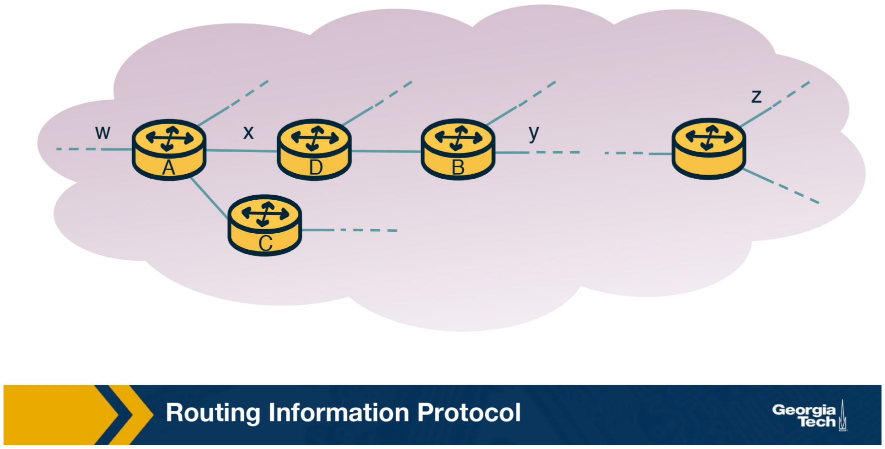
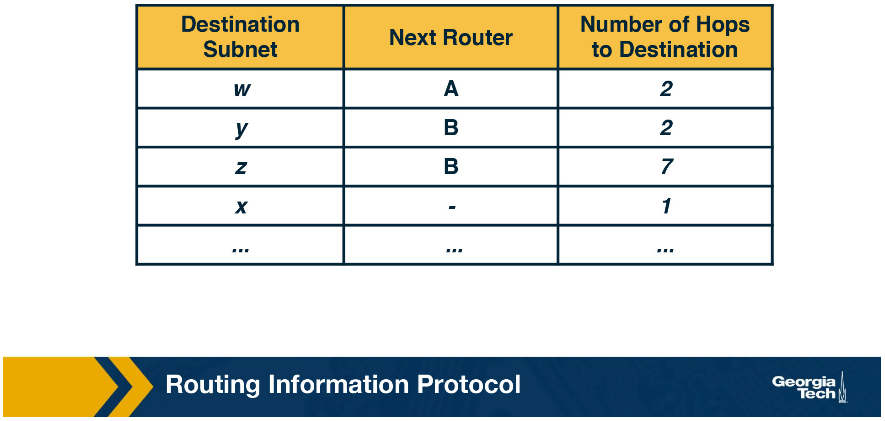


Figure 12: image

Each router maintains a **routing table**, which contains its own distance vector as well as the router's forwarding table. If we have a look at the routing table of Router D, we will see that it has three columns: destination subnet, identification of the next router along the shortest path to the destination, and the number of hops to get to the destination along the shortest path. A routing table will have one row for each subnet in the autonomous system (AS).



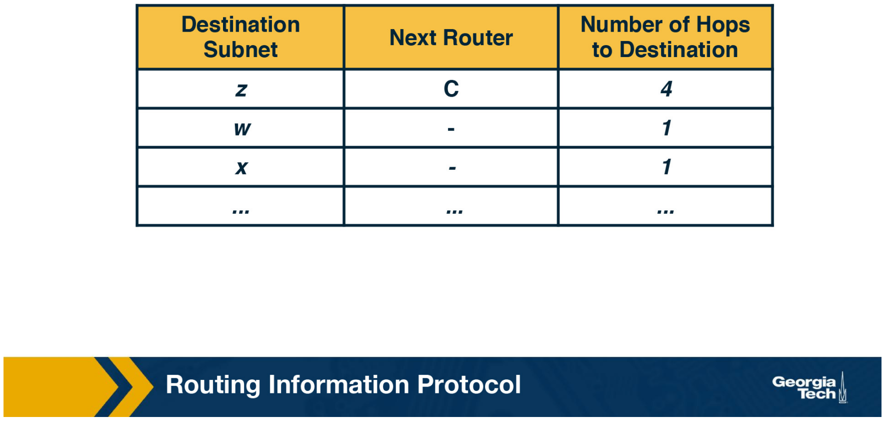
Destination Subnet	Next Router	Number of Hops to Destination
w	A	*2*
y	B	*2*
z	B	*7*
x	-	*1*
...	...	...

Routing Information Protocol

Georgia Tech

Figure 13: image

For this example, the table in the above figure indicates that to send a datagram from router D to destination subnet w, the datagram should first be forwarded to neighboring router A; the table also indicates that destination subnet w is two hops away along the shortest path. Now if router D receives from router A the advertisement (the routing table information of router A) shown in the figure below it merges the advertisement with the old routing table.



Destination Subnet	Next Router	Number of Hops to Destination
z	C	*4*
w	-	*1*
x	-	*1*
...	...	...

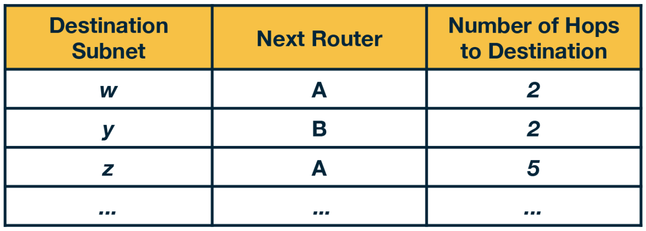
Routing Information Protocol

Georgia Tech

Figure 14: image

In particular, router D learns that there is now a path through router A to subnet z that is shorter than the path through router B. Therefore, router D updates its table to account for the new shortest path. The updated routing table is shown in the figure below. As the Distance Vector algorithm is in the process of converging or as new links or routers are getting added to the AS, the shortest path is changing.

Each node maintains a RIP Table (Routing Table), which will have one row for each subnet in the AS. RIP version 2 allows subnet entries to be aggregated using route aggregation techniques. If a router does not hear from its neighbor at least once every 180 seconds, that neighbor is considered to be no longer reachable (broken link). In this case, the local routing table is modified and changes are propagated. Routers send request and response messages over UDP, using port number 520, which is layered on top of network-layer IP protocol. RIP is actually implemented as an application-level process.



Destination Subnet	Next Router	Number of Hops to Destination
w	A	*2*
y	B	*2*
z	A	*5*
...	...	...

Routing Information Protocol

Figure 15: image

Some of the challenges with RIP include updating routes, reducing convergence time, and avoiding loops/count-to-infinity problems.

Linkstate Routing Protocol Example: OSPF

Open Shortest Path First (OSPF) is a routing protocol that uses a link state routing algorithm to find the best path between the source and the destination router. OSPF was introduced as an advancement of the RIP Protocol, operating in upper-tier ISPs. It is a link-state protocol that uses flooding of link-state information and a Dijkstra least-cost path algorithm. Advances include authentication of messages exchanged between routers, the option to use multiple same-cost paths, and support for hierarchy within a single routing domain.

As we have seen already, a link state routing algorithm is a dynamic routing algorithm in which each router shares knowledge of its neighbors with every other router in the network. The network topology that is built as a result can be viewed as a directed graph with preset weights for each edge assigned by the administrator.

Hierarchy. An OSPF autonomous system can be configured hierarchically into areas. Each area runs its own OSPF link-state routing algorithm, with each router in an area broadcasting its link state to all other routers in that area. Within each area, one or more area border routers are responsible for routing packets outside the area. Exactly one OSPF area in the AS is configured to be the backbone area. The primary role of the backbone area is to route traffic between the other areas in the AS. The backbone always contains all area border routers in the AS and may contain non-border routers as well. For packets routing between two different areas, it is required that the packet be sent through an area border router, through the backbone and then to the area border router within the destination area, before finally reaching the destination.

Operation. First, a graph (topological map) of the entire AS is constructed. Then, considering itself as the root node, each router computes the shortest-path tree to all subnets, by running Djikstra's algorithm locally. The link costs have been pre-configured by a network administrator. The administrator has a variety of choices while configuring the link costs. For instance, he may choose to set them to be inversely proportional to link capacity, or set them all to one. Given a set of link weights, OSFP provides the mechanisms for determining least-cost path routing. Whenever there is a change in a link's state, the router broadcasts routing information to all other routers in the AS, not just to its neighboring routers. It also broadcasts a link's state periodically even if its state hasn't changed.

Link State Advertisements. Every router within a domain that operates on OSPF uses Link State Advertisements (LSAs). LSA communicates the router's local routing topology to all other local routers in the same OSPF area. In practice, LSA is used for building a database (called the link state database) containing all the link states. LSAs are typically flooded to every router in the domain. This helps form a consistent network topology view. Any change in the topology requires corresponding changes in LSAs.

Refresh rate for LSAs. OSPF typically has a refresh rate for LSAs, which has a default period of 30 minutes. If a link comes alive before this refresh period is reached, then routers connected to that link ensure LSA flooding. Since the process of flooding can happen multiple times, every router receives multiple copies of refreshes or changes - and stores the first received LSA change as new, and the subsequent ones as duplicates.

Processing OSPF Messages in the Router

In the previous section, we looked at OSPF fundamentals and how it operates using Link State Advertisements (LSA). In this section we will look at how the OSPF messages are processed in the router in more detail.

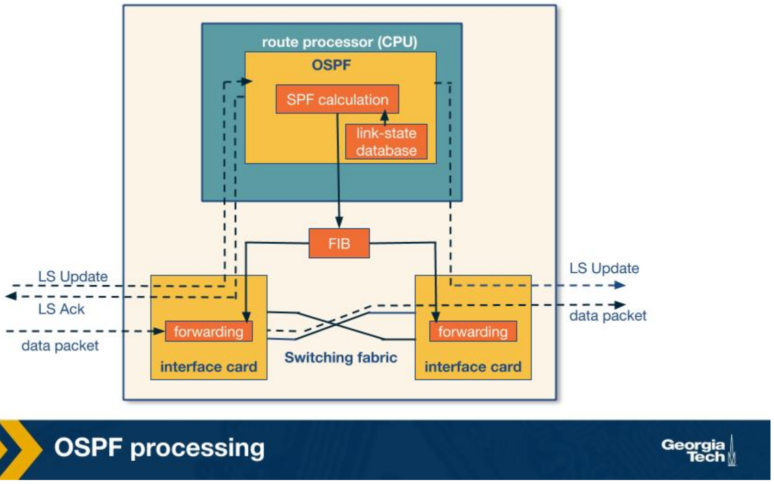


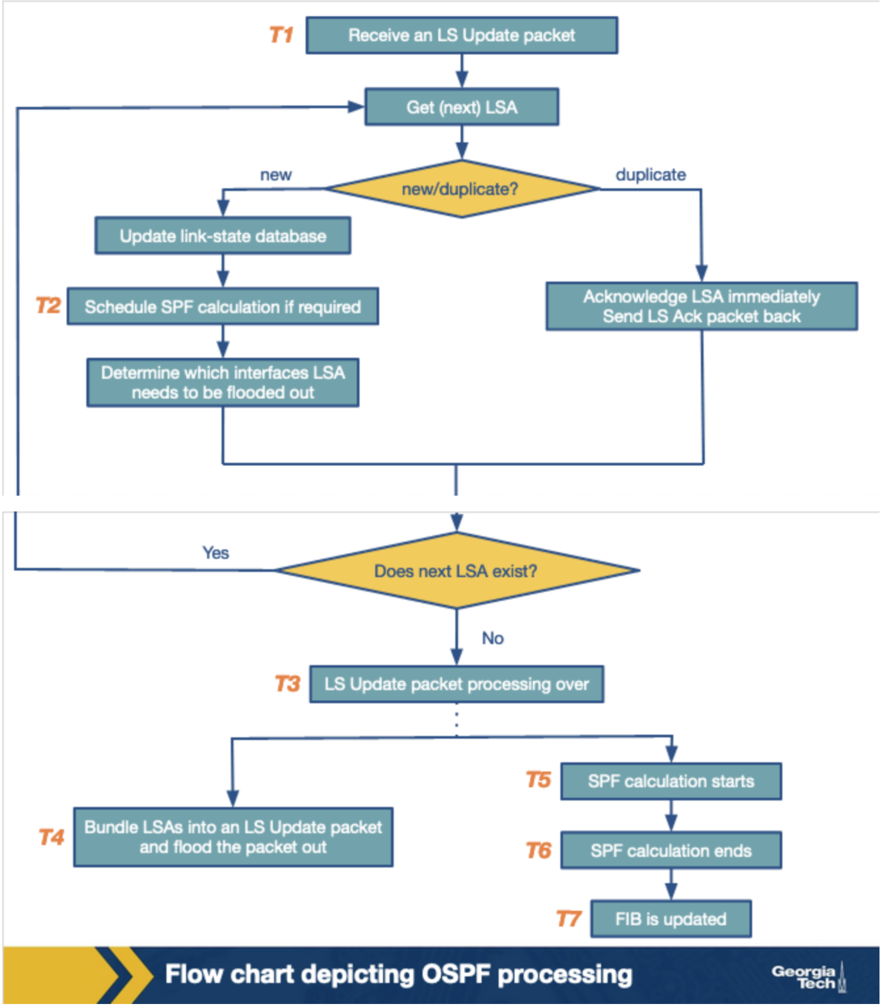
Figure 16: image

To do this, let's begin with a simple model of a router given in the figure above. The router consists of a route processor (which is the main processing unit) and interface cards that receive data packets which are forwarded via a switching fabric. Let us break down router processing in a few steps:

1. Initially, the LS update packets which contain LSAs from a neighboring router reaches the current router's OSPF (which is the route processor). This is the first trigger for the route processor. As the LS Updates reach the router, a consistent view of the topology is being formed and this information is stored in the link-state database. Entries of LSAs correspond to the topology which is actually visible from the current router.
2. Using this information from the link-state database, the current router calculates the shortest path using shortest path first (SPF) algorithm. The result of this step is fed to the Forwarding Information Base (FIB)
3. The information in the FIB is used when a data packet arrives at an interface card of the router, where the next hop for the packet is decided and its forwarded to the outgoing interface card.

To further understand OSPF processing, let's look at the following flow chart and view it in time slices (T1, T2, . . . , T7).

We've already noted that the processing tasks begin at the receipt of an LS update packet (**T1**). For every LSA unpacked from the update packet, the OSPF protocol checks whether it is a new or a duplicate LSA. This is done by referring to the link-state database, and checking for the sequence number of the LSA to a matching LSA instance in the database. For every new LSA, the database is updated, an SPF calculation is



T1 Receive an LS Update packet

Get (next) LSA

new/duplicate?

new

duplicate

Update link-state database

T2 Schedule SPF calculation if required

Determine which interfaces LSA needs to be flooded out

Acknowledge LSA immediately
Send LS Ack packet back

Does next LSA exist?

Yes

No

T3 LS Update packet processing over

T4 Bundle LSAs into an LS Update packet and flood the packet out

T5 SPF calculation starts

T6 SPF calculation ends

T7 FIB is updated

Flow chart depicting OSPF processing

Georgia Tech

Figure 17: image

scheduled (**T2**) and it's determined which interface the LSA needs to be flooded out of. In modern routers, the when of LSA flooding can be based on a timer.

When all the LSAs from an LS update packet have been processed (**T3**), the LSAs are prepared and flooded out as an LS Update packet to the next router (**T4**). After this, we move on to the actual execution of SPF calculation within the router (**T5** and **T6**). Since SPF calculation is a CPU-intensive task, SPF calculations are scheduled and carried out over a period of time (usually when LSA's are changed) so as to offset the CPU costs. After the SPF calculation is completed, the FIB is updated (**T7**).

Hot Potato Routing

In large networks, routers rely both on interdomain and intradomain routing protocols to route the traffic. The routers within the network use the intradomain routing protocols to find the best path to route the traffic within the network. In case when the final destination of the traffic is outside the network, then the traffic will travel towards the networks exit (egress points) before leaving the network. In some cases there are multiple egress points that the routers can choose from. These egress points (routers) can be equally good in the sense that they offer similarly good external paths to the final destination.

In this case, hot potato routing is a technique/practice of choosing a path within the network, by choosing the closest egress point based on intradomain path cost (Interior Gateway Protocol/IGP cost).

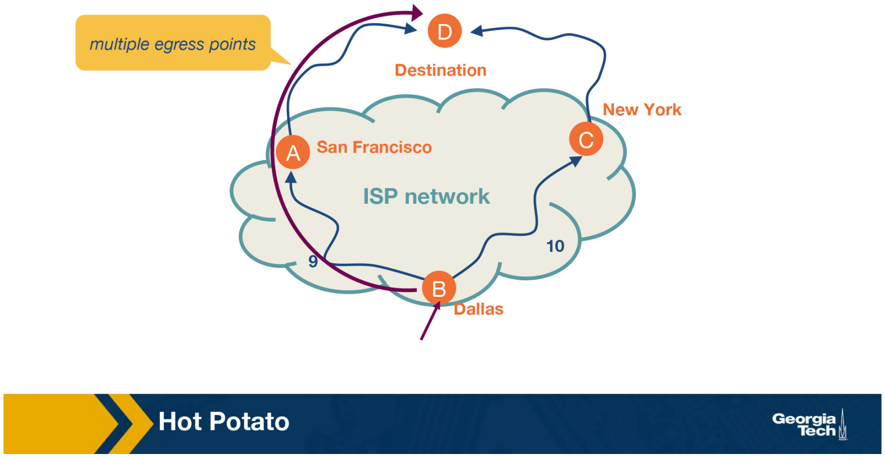


Figure 18: image

Let's look at an example. In the figure above, we have a network, and specifically we are looking at the router located in Dallas. Rhe router needs to forward traffic towards a destination. It could do so via New York or San Francisco. We assume that both egress points offer BGP (Border Gateway Protocol) path costs, so they are equally good egress points. In this case, the router has multiple egress points. We see that the IGP path cost for SF is 9 while the path cost for NY is 10. Thus, the router uses hot potato routing to choose to send the traffic to the destination via SF.

Hot potato routing simplifies computations for the routers as they are already aware of the IGP path costs. It makes sure that the path remains consistent, since the next router in the path will also choose to send the packet to the same egress point. Hot potato routing also effectively reduces the network's resource consumption by getting the traffic out as soon as possible.