Journal of Sustainable Development in Africa (Volume 9, No.3, 2007)
ISSN: 1520-5509
Fayetteville State University, Fayetteville, North Carolina

Path Routing Algorithm Optimization in Inter-Networks

C Gombiro, F.S Masaraneyi, G Chengetanai and T Chagonda

Abstract

This paper introduces algorithms for path routing in inter-networks. These are the algorithms that help us compute routes for stations sharing a broadcast channel.

We will consider each link connecting two nodes or stations in an inter-network to an optical fibre, and each optical fibre can support the same set of wavelengths. For each communication request of connecting station A with station B, the controller of the network has to designate a path between the two stations (path routing) and has to designate a path between the two stations (path routing) and has to allocate to this path one wavelength (wavelength allocation).

To transfer data from one station to another across an inter-network, we need to determine the route or path to be traversed. This should ideally be the optimal route between the two stations. The optimal path can be decided by routing algorithms, which can designate based on various factors, such as:

- *Capacity*: The throughput of the circuit in bits per second
- *Delay*: The mean transport delay associated with a link
- *Expense*: The actual cost associated with using a link
- *Error*: The mean residual error probability associated with the circuit.

Key words: Routing algorithm, Inter-Network.

Introduction

This paper introduces the underlying concepts widely in path routing in Inter-Networking. These concepts include path routing algorithms and their role in routing protocols.

The topic of routing in Inter-Networking has been covered in Computer Science literature for decades, and has achieved commercial popularity worldwide. Early networks were implemented in

homogeneous environments, which made them very simple to design and implement. Only relatively, recently have the large-scale Inter-Networks become popular.

Routing involves two basic activities, these are determining ***optimal routing path*** (the shortest and less cost path) and ***transporting information groups*** (typically called packets) through an intranet or extranet.

After a close consideration we thought of analyzing all possible algorithms that are and/ or currently in use, and try and try to find a relative way towards addressing the shortfalls of these algorithms. These shortfalls include among others, failure to determine the shortest possible path from source to destination, inefficient use of granted wavelength especially in optical network.

We intend to identify and study quite a number of algorithms and determine the least cost path/route in Inter-Networks. These include among others, static algorithms, hierarchical algorithms, inter-domain algorithms, intra-domain algorithms, host intelligent algorithms and distant vector algorithms.

Routing Principles

Routing is the act of moving information across an inter-network from a source to a destination along the way, at least one intermediate mode typically in encountered. A device called a router the job of selecting the path that a message should follow to its intended destination.

The topic of routing has been covered in Computer Science literature for more that two decades, but routing achieved commercial popularity as late as the mid- 1980. The primary reasons for this time lag is that network in the 1970s were simple, homogeneous environments.

Routing Components

Routing involves two basic activities:

- Optimal path determination and
- Transporting information groups (typically called packets) through an inter-network.

Path Determination

Routing protocols use metrics to evaluate what path will be best for a packet to travel. A metric is a standard measurement, such as path bandwidth, that is used by routing algorithms to determine the optimal path to a destination. To aid the process of path determination, routing algorithms initialize

and maintain routing tables, which contain route information. Route information varies depending on the routing algorithm used.

Routing algorithms fill routing tables with a variety of information.
Destination/next hop association tell a router that a particular destination can be reached optimally by sending the packet to a particular router representing the "*next hop*" on the way to the final destination. When a router receives an incoming packet, it checks the destination address and attempts to associate this address with a net hop. Fig 1.1 shows how the architecture of the next hop association.

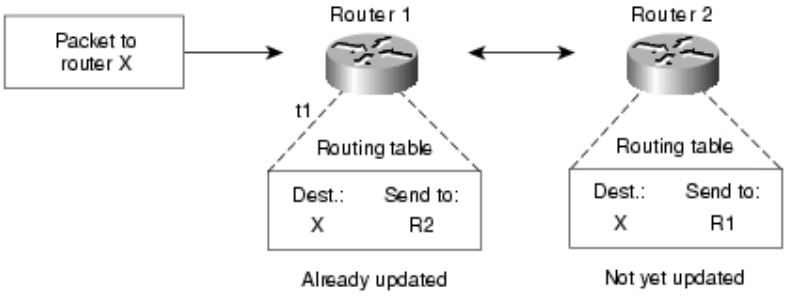

Fig 1.1: Destination/ Next Hop Association (for optimal path determination)

Routing tables also can contain other information, such as data about the desirability of a path. Routers compare metrics to determine optimal routes, and these metrics differ depending on the design of the routing algorithm used. A variety of common metrics will be introduced and described later in this chapter.

Routers communicate with each another and maintain their routing tables through the transmission of variety of messages. The routing update message is one such message that generally consists of all or portion of routing table. By analyzing routing updates from all other routers, router can build a detailed picture of network topology. A link-state advertisement, another example of a message sent between routers to determine optimal routes to network destinations.

Routing Algorithms

Routing algorithms can be differentiated based on several key characteristics. First, the particular goals of the algorithms designer affect the operation of the resulting routing protocol. Secondly, various types of routing algorithms exist, and each algorithms use a variety of metrics that affects calculation of optimal routes. This has resulted in us analyzing the attributes of these routing algorithms

The design and implementation of routing algorithms often have one or more of the following design goals.

- Optimality
- Simplicity and low overhead
- Robustness and stability
- Rapid convergence
- Flexibility

With *optimality* usually we will be referring to the capability of the routing algorithm to select the best route, which depends on the metric and metric weightings used to make the calculation. For example, one routing algorithm may use a number of hops and delays, but it may weigh delay more heavily in the calculation. However, routing protocols must define their metric calculation algorithms.

Classification of Routing Algorithms

There exits two broad ways of classifying routing algorithms.

a. Determine whether they are centralized or decentralized

A "*global routing algorithm*" computes the least cost path between source and destination using complete, global knowledge about the network. That is, the algorithm takes the connectivity between all nodes and all links costs as inputs. This then requires that the algorithm somehow obtain this information before actually performing the calculation. The calculation itself can be run at one site (centralized global routing algorithms) or replicated at multiple sites. The key distinguishing feature here, however, is that a global algorithm has complete information about connectivity and link costs. In practice, algorithms with global state information are often referred to as **link state algorithms**, since the algorithm must be aware of the state (cost) of each link in the network.

In a "*decentralized routing algorithm"*, calculation of the least cost path is carried out in an iterative, distributed manner. No node has complete information about the costs of all network links. Instead, each node begins with only knowledge of the costs of its own directly attached links and then through an iterative process of calculation and exchange of information with its neighbouring nodes (i.e nodes which are at the "other end "of links to which itself is attached) gradually calculates the least cost path to a destination, or set of destinations.

"*Distance vector algorithm*" is one example of decentralized routing algorithm. A node never actually knows a complete path from source to destination; instead, it only knows the least cost path, and the cost of the path from itself to the destination.

b. Classify routing algorithms according to whether they are non-adaptive and adaptive routing.

Non –Adaptive Routing

These algorithms do not base their routing decisions on measurements or estimates of current traffic or topology and usually traffic as well. Adaptive algorithms differ in where they get their information, when they change routes and the metrics that they use for optimization.

Adaptive Routing

These algorithms change their routing decisions to reflect changes in the topology and usually traffic as well. Adaptive algorithms differ in where they get their information when they change routes and the metrics that thy use for optimization.

Routing Tables

A metric is standard of measurement, such as path length or cost, which is used by routing algorithms to determine the optimal path to a destination. To aid the process of path determination, routing algorithms initialize and maintain routing tables. These contain route information, which varies depending on the routing algorithm used.

Routing Principles

In order to transfer packets from a sending host to the destination host, the network layer must determine the path or route that the packets are to follow. Whether the network layer provides a datagram service (in which case different packets between given host-destination pair may take different routes) or a virtual circuit service (in path), the network layer must nonetheless determine the path for a packet. This is the job of the network layer **routing protocol**.

At the heart of any routing protocol is the algorithm (the “routing algorithm”) that determines the path for a packet. The purpose of a routing algorithm is as follows: given a set of routers, a routing algorithm finds a “good” path from a source to destination. Typically, a “good” path is one, which has “least cost,” but will see that in practice, “real world” concerns such as policy issues (e.g., a rule such as “router X”, belonging to organization Y should not forward any packets originating from the network owned by organization Z”) also come into play.

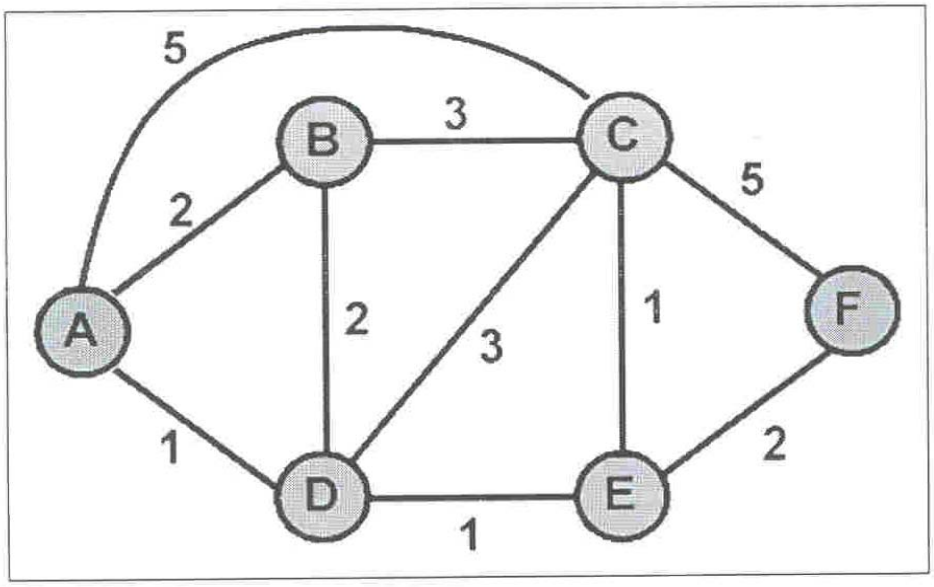

Fig 1.2: Abstract model of a network

In Fig 12 above, the nodes in the graph represent routers - the points at which packet routing decisions are made - and the lines ("edges" in graph theory terminology) connecting these nodes represent the physical links between these routers. A link also has a value representing the "cost" *of* sending a packet across the link. The cost may reflect the level *of* congestion on that link (e.g., the current average delay *for* a packet across that link) or the physical distance traversed by that link (e.g., a transoceanic link might have a higher cost than a terrestrial link). *For* our current purposes, we will take the link costs as given and won't worry about *how* they are determined.

Given the graph abstraction, the problem *of* finding the least cost path from a source *to* a destination requires identifying a series *of* links such that:

- The first link in the path is connected *to* the source
- The last link in the path is connected *to* the destination
- *For* all i, the i and i-1st link in the path are connected *to* the same node
- *For* the **least cost path**, the sum *of* the cost *of* the links on the path is the minimum over all possible paths between the source and destination. Note that if all link costs are the same, the least cost path is also the shortest path (ie. "The path crossing the smallest number of links between the source and the destination).

In Fig 1.2 above, for example, the least cost path between nodes A (source) and C (destination) is along the path ADEC (We will find it notationally easier to refer to the path in terms of the nodes on the path, rather than the links on the path).

The Internet basically uses only two types of algorithms. These are: The Link State Routing Algorithm and the Distance Vector Algorithm.

Analysis of Routing Algorithms

Static Routing Algorithms

Shortest Path Routing: Several algorithms can be used for computing path between two nodes. Let us consider **Dijkstra's** algorithm to compute the shortest path. The idea is to build a graph of the subnet, with which each node of the graph representing a router and each arc representing a communication line called a link. Here each node labelled with its distance from the source node along the best-known path. Initially, no path are known, so all nodes are labelled infinity. As the algorithm proceeds and paths are found, the labels may change, reflecting better paths. A label may be tentative or permanent. Initially, all labels re tentative. When it is discovered that label represents the shortest path from the source to that node, it is made permanent and changed thereafter.

Flooding: This requires no network information. A source node sends a packet to every one of its neighbours. At each node, an incoming packet is retransmitted on all outgoing links except for the link on which it arrived. In flooding we need to control the incessant retransmission of packets as the number of packets in circulation just from a single source packet grows without bound. To control this we include a hop-count field with each packet. The count is initially set to some maximum value. When the count reaches zero, the packet is discarded.

Dynamic Routing Algorithms

Distance Vector Routing: These algorithms operate by having each router maintain a table that gives the best-known distance to each destination and the line to use to get there. These tables are updated by exchanging information with neighbours. This algorithm is also called the distributed Bellman-Ford or the Food-Fulkerson routing algorithm. In this algorithm each router maintains a routing table indexed by and containing one entry for each router in the subnet. This entry contains two parts:

- The preferred outgoing line to use for that destination
- An estimate of the time or distance to that destination

The router is assumed to know the distance to each of its neighbours. The metric used could be the number of hops, time delay or total number of packets queued along the path.

Link State Routing: The idea behind link State Routing can be summarized as follows:

- Find the neighbouring routers and their network address
- Measure the delay of cost or the cost to reach each neighbour
- Construct a packet containing what it knows about its neighbours
- Send this packet to all other routers
- Compute the shortest path to every other router

In effect, the complete topology and all delays are experimentally measured and made known to all other routers. Then we can use Djikstra's algorithm to find the shortest path to every other router.

The Link State Routing Algorithm

In the link state algorithm, the network topology and all link costs are known before hand (i.e. they are available as input to the link state algorithm). In practice this is accomplished by having each node **broadcast** the identities and costs of its attached links to *all* other routers in the network.

This link state broadcast can be accomplished without the nodes having to initially know the identities of all other nodes in the network. A node needs only to know the identities and costs to its directly attached neighbours and it will then learn about the topology of the rest of the network by receiving link state broadcast from other nodes. The result of the nodes' link state broadcast is that all nodes have an identical and complete view of the network. Each node can then run the link state algorithm and compute the same set of least cost paths as every other node.

An example of the link state algorithm we are going to present is the Dijkstra's algorithm. It computes the least cost path *from* one node (the source, which we will refer to as A) to all other nodes in the network. Dijkstra's algorithm is iterative and has the property that after the k^{th} iteration of the algorithm, the least cost paths are known to k destination nodes, and among the least cost paths *to* all destination nodes, this k path will have the k smallest costs. Let us define the following notation:

- c(i,j): link cost from node i *to* node j. If nodes i and j are not directly connected, then c(i,j) = infty. We will assume for simplicity that c(i,j) equals c(j,i).
- D(v): the cost of path from the source node to destination v that has currently (as of this iteration of the algorithm) the least cost.
- p(v): previous node (neighbour of v) along current least cost path from source to v.
- N: set of nodes whose shortest path from the source is definitively known.

The link state algorithm consists of an initialization step followed by a loop. The number of times the loop is executed is equal to the number of nodes in the network. Upon termination, the algorithm will have calculated the shortest paths from the source node to every other node in the network.

Link State (LS) Algorithm:

1. *Initialization:*
2. N={A}
3. For all nodes v
4. if v adjacent to a
5. then D(v)=c(A,v)
6. else D(v)=infty
7. loop
8. find w not in N such that D(w) is a minimum
9. add w to N
10. update D(v) for all v adjacent to w and not in N:
11. D(v)=min(D(v),D(w) + c(w,v))
12. /*new cost to v is either old cost to v or known shortest path cost to w plus cost from w to v */
13. until all nodes in N

As an example, let us consider the network in Fig 1.2 and compute and compute the shortest path from A to all possible destinations. A tabular summary of the algorithm's computation is shown in Table 1.1, where each line in the table gives the values of the algorithms variables at the end of the iteration. Let us consider the few first steps in detail:

Table 1.1: Steps in running the link state algorithm on network in Fig 1.2

STEP	N	D(B),p(B)	D(C),P(C)	D(D),P(D)	D(E),P(E)	D(F),p(F)
0	A	2,A	S,A	1,A	lofty	infty
1	AD	2,A	4,D		2,D	infty
2	ADE	2,A	3,E			4,E
3	ADEB		3E			4E
4	ADEBC					4E
5	ADEBCF					

- **In the initialization step**, the currently known least path costs from A to its directly attached neighbours; B, C and D are initialized to 2, 5 and 1 respectively. Note in particular that the cost to C is set to 5 (even though we will soon see that a lesser cost path does indeed exists) since this is cost of the direct (one hop) link from A to C. The costs to E and F are set to infinity since they are not directly connected to A.

- **In the first iteration**, we look among those nodes not yet added to the set N and find that

node with the least cost as of the end of the previous iteration. That node is D, with a cost of 1, and thus D is added to the set N. Line 12 of the LS algorithm is then performed to update D(v) for all nodes v, yielding the results shown in the second line (step 1) in Table 1.1. The cost of the path to B is unchanged. The cost of the path to C (which was 5 at the end of the initialization) through node D is found to have a cost of 4. Hence this lower cost path is selected and C's predecessor along the shortest path from A is set to D. Similarly, the cost to E (through D) is computed to be 2, and the table is updated accordingly.

- **In the second iteration**, nodes Band E are found to have the shortest path costs (2), and we break the tie arbitrarily and add E to the set N so that N now contains A, 0, and E. The cost to the remaining nodes not yet in N, i.e, nodes B, C and F are updated via line 12 of the LS algorithm, yielding the results shown in the third row in table 1.1.

When the LS algorithm terminates, we have for each node, its predecessor along the least cost path from the source node. For each predecessor, we also have *its* predecessor and so in this manner we can construct the entire path from the source to all destinations.

What is the computation complexity of this algorithm? That is, given *n* nodes (not counting the source), how much computation must be done in the worst case to find the least cost paths from the source to all destinations? In the first iteration, we need to search through all *n* nodes to determine the node, *w,* not in N that has the minimum cost. In the second iteration, we need to check *n-l* nodes to determine the minimum c t; in the third iteration *n-2* nodes and so on. Overall, the total number of nodes we need to search through over all the iterations is n*(n+l)/2, and thus we say that the above implementation of the link state algorithm has worst case complexity of order *n* squared: $O(n^2)$. (A more sophisticated implementation of this algorithm, using a data structure known as a heap, can find the minimum in line 9 in logarithmic rather than linear time, thus reducing the complexity).

Finally, let us consider a pathology that can arise with the use of link state routing. Fig 1.3 shows a simple network topology where link costs are equal to the load carried on the link, e.g., reflecting the delay that would be experienced. In this example, link costs are not symmetric, i.e., c(A,B) equals c(B,A) only if the load carried on both directions on the AB link is the same. In this example, node D originates a unit of traffic destined for A, node B also originates a unit of traffic destined for A, and node C injects an amount of traffic equal to e, also destined for A. The initial routing is shown in Fig 1.3, with the link costs corresponding to the amount of traffic carried.

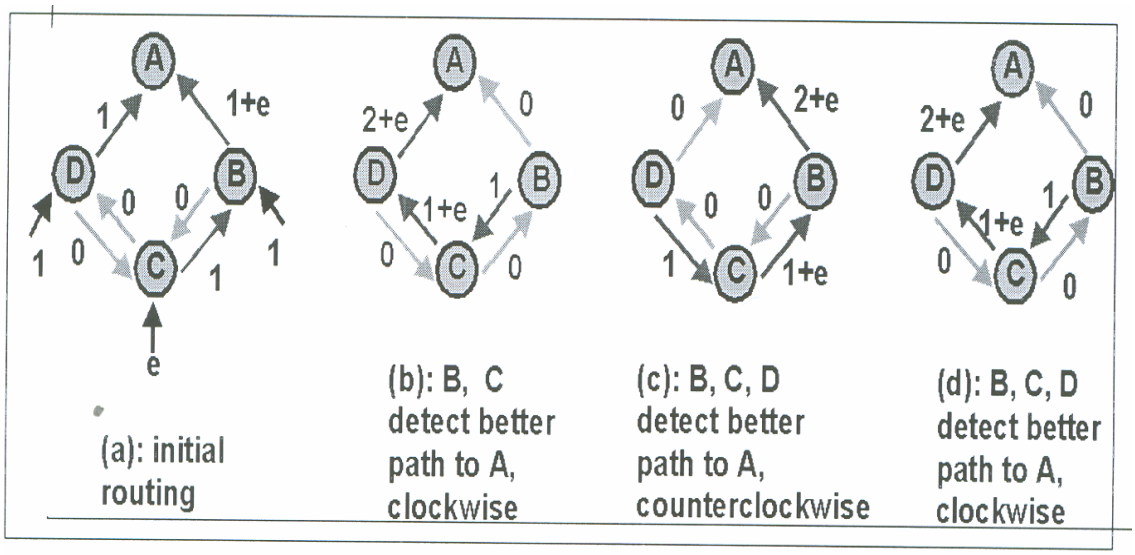

Fig 1.3: Oscillations with Link State routing

When the lS algorithm is next run, node C determines (based on the link costs shown in Fig 1.3) that the clockwise path to A has a cost of 1, while the counterclockwise path to A (which it had been using) has a cost of 1 +e. Hence C's least cost path to A is now clockwise. Similarly, B determines that its new least cost path to A is also clockwise, resulting in the routing and resulting path costs shown in Fig 1.3. When the lS algorithm is run next, nodes B, C and D all detect that a zero cost path to A in the counterclockwise direction and all route their traffic to the counterclockwise routes. The next time the lS algorithm is run, B, C, and D all then route their traffic to the clockwise

The Distance Vector Routing Algorithm

While the LS algorithm is a centralized algorithm, the other principal routing algorithm used in the Internet is an *asynchronous, iterative, distributed* distance vector (DV) algorithm. It is distributed in that each node receives some information from one or more of its *directly* attached neighbours, performs a calculation, and may then distribute the results of its calculation back to its neighbours. It is iterative in that this process continues on until no more information is exchanged between neighbours. (Interestingly, we will see that the algorithm is self-terminating -- there is no "signal" that the computation should stop; it just stops). The algorithm is asynchronous in that it does not require all of the nodes to operate in lock step with each other. An asynchronous, iterative, self-terminating, distributed algorithm is much more "interesting" and "fun" than a centralized algorithm (that is, such algorithms have great appeal to computer scientists and engineers).

The principal data structure in the DV algorithm is the **distance table** maintained at each node. Each node's distance table has a row for each destination in the network and a column for each of its directly attached neighbours. Consider a node X that is interested in routing to destination Y via its directly attached neighbour Z. Node XiS distance table entry Dx(y,Z) is the sum of the cost of the

direct one hop link between X and Z, c(X,Z), plus neighbour Z's currently known minimum cost path from itself (Z) to Y. That is:

$$\mathbf{D^x(y,Z) = c(X,Z) + min_w\{D^z(y,w)\}} \qquad \mathbf{(3\text{-}1)}$$

The min_w term in equation 3-1 is taken over all of Z's directly attached neighbors (including X, as we shall soon see).

Equation **3-1** suggests the form of the neighbour-to-neighbour communication that will take place in the DV algorithm -- each node must know the cost of each of its neighbours minimum cost path to each destination Thus, whenever a node computes a new minimum cost to some destination, it must inform its neighbours of this new minimum cost.

Before presenting the DV algorithm, let us consider the network topology and the distance table shown for node E in Fig 1.4. We first look at the row for destination A.

- Clearly the cost to get to A from E via the direct connection to A has a cost of 1. Hence $D^E(A,A) = 1$.

- Let us now consider the value of $D^E(A,D)$ - the cost to get from E to A, given that the first step along the path is D. In this case, the distance table entry is the cost to get from E to 0 (a cost of 2) plus whatever the minimum cost it is to get from 0 to A. Note that the minimum cost from 0 to A is 3. A path that passes right back through E! Nonetheless, we record the fact that the minimum cost from E to A given that the first step is via D has a cost of 5.

- Similarly, we find that the distance table entry via neighbour B is $D^E(A,B) = 14$.

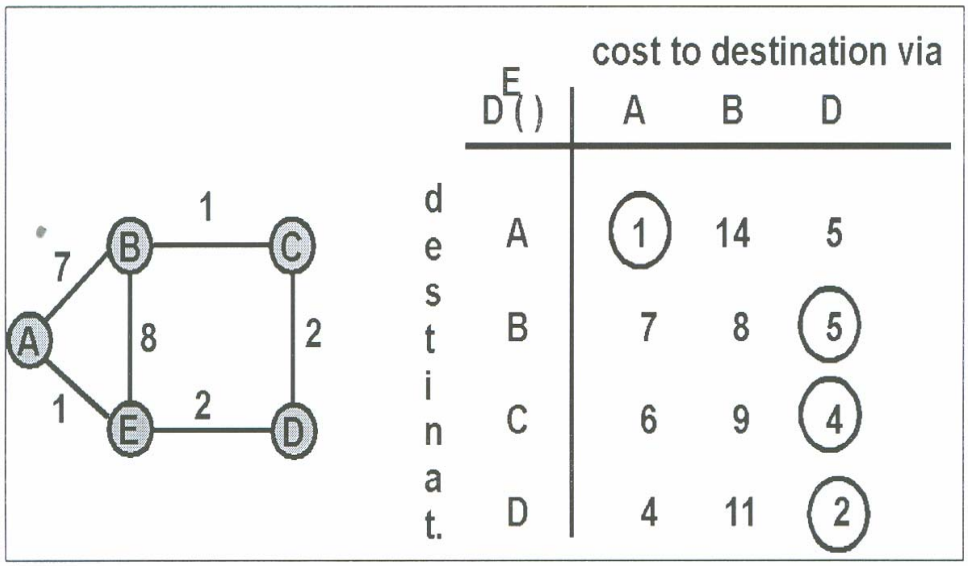

Figure1.4: A Distance Table Example

A circled entry in the distance table gives the cost of the least cost path to the corresponding destination (row). The column with the circled entry identifies the next node along the least cost path to the destination. Thus, a node's routing table (which indicates which outgoing link should be used to forward packets to a given destination) is easily constructed from the node's distance table.

In discussing the distance table entries for node E above, we informally took a global view, knowing the costs of all links in the network. The distance vector algorithm we will now present is *decentralized* and does not use such global information. Indeed, the only information a node will have is the costs of the links to its directly attached neighbours, and information it receives from these directly attached neighbours. The distance vector algorithm we will analyze is also known as the Bellman-Ford algorithm. It is used in many routing algorithms in practice, including: Internet BGP, ISO IDRP, Novell IPX, and many others.

Distance Vector (DV) Algorithm.

At each node, X:

1 *Initialization:*
2 for all adjacent nodes v:
3 DX(*,v) = infty
4 DX(v,v) = c(X,v)
5 for all destinations, y.
6 send minwD(y,w) to each neighbor j* w over all X's neighbors * j
7 *loop* **wait** (until I see a link cost change to neighbor V

8 or until I receive update from neighbor V)
9 /*space*/
10/*space*/
12 if (c(X,V) changes by d)
13 /* *change cost to all dest* r s *via neighbor v by d* */
14 /* *note: d could be positive or negative* */
15 for all destinations y: DX(y,V) = DX(y,V) + d
16/*space*/
17 **else if** (update received from V wrt destination Y)
18/* *shortest path from* V *to some* Y *has changed* *j
19 /* V *has sent a new mlue for its minI<' DV(Y,w)* */
20 /* *call this received new mlue* is *"newml"*
21 for the single destination y: DX(Y,V) = c(X,V) + newval
22/*space*/
23 if we have a new $\min_w$ DX(Y,w)for any destination Y
24 send new value of $\min_w$ DX(Y,w) to all neighbours
25/*space*/
26 ***forever***

The key steps are lines 15 and 21, where a node updates its distance table entries in response *to* either a change *of cost of* an attached link or the receipt *of* an update message from a neighbour. The other key step is line 24, where a node sends an update *to* its neighbours if its minimum *cost* path *to* a destination has changed.

Fig 1.5 illustrates the operation *of* the DV algorithm for the three-node network shown at the top *of* the figure. The operation *of* the algorithm is illustrated in a synchronous manner, where all nodes simultaneously receive messages from their neighbours, compute new distance table entries, and inform their neighbours *of* any changes in their new least path costs.

The circled distance table entries in Fig 1.5 show the current least path *cost to* a destination. An entry in red indicates that a new minimum *cost* has been computed (in either line 4 *of* the DV algorithm (initialization) or line 21). In such cases an update message will be sent (line 24 *of* the DV algorithm) *to* the node's neighbors as represented by the red arrows between columns in Fig 1.5

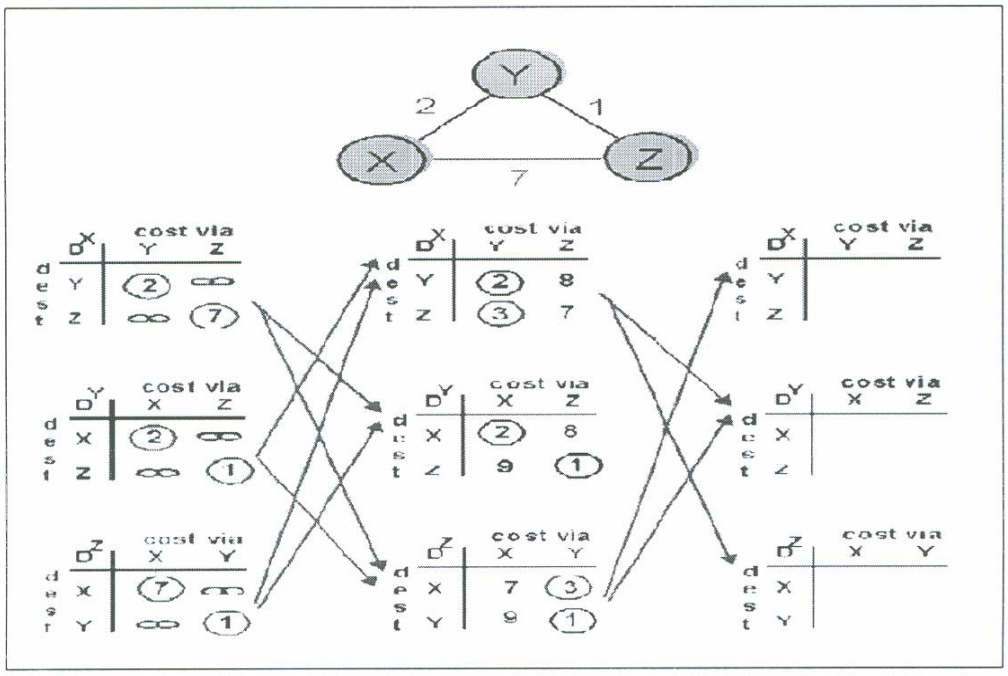

Fig 1.5: Distance Vector Algorithm: example

The leftmost column in Fig 1.5 shows the distance table entries for nodes X, Y, and Z after the initialization step. Let us now consider how node X computes the distance table shown in the middle column of Fig 1.5 after receiving updates from nodes Y and Z. As a result of receiving the updates from Y and Z, X computes in line 21 of the DV algorithm:

$$D^X(Y,Z) = c(X,Z) + \min_w D^Z(Y,w)$$
$$= 7 + 1$$
$$= 8$$

$$D^X(Z,Y) = c(X,Y) + \min_w D^Y(Z,w)$$
$$= 2 + 1$$
$$= 3$$

It is important to note that the only reason that X knows about the terms $\min_w D^Z(Y,w)$ and $\min_w D^Y(Z,w)$ is because nodes Z and Y have sent those values to X (and are received by X in line 10 of the DV algorithm). As an exercise, verify the distance tables computed by Y and Z in the middle column of Fig 1.5

The value $D^X(Z,Y) = 3$ means that X's minimum cost to Z has changed from 7 to 3. Hence, X sends updates to Y and Z informing them of this new least cost to Z. Note that X need not update Y and Z about its cost to Y since this has not changed. Note also that Y's recomputation of its distance table

in the middle column of Fig 1.5 *does* result in new distance entries, but *does not* result in a change of Y's least cost path to nodes X and Z. Hence Y does *not* send updates to X and Z.

The process of receiving updated costs from neighbours, recomputation of distance table entries, and updating neighbours of changed costs of the least cost path to a destination continues until no update messages are sent. At this point, since no update messages are sent, no further distance table calculations will occur and the algorithm enters a quiescent state, i.e., all nodes are performing the wait in line 9 of the DV algorithm. The algorithm would remain in the quiescent state until a link cost changes, as discussed below.

The Distance Vector Algorithm: link Cost Changes and link Failure

When a node running the DV algorithm detects a change in the link cost from itself to a neighbour (line 12) it updates its distance table (line 15) and, if there is a change in the cost of the least cost path, updates its neighbours (lines 23 and 24). Fig 1.6 illustrates this behaviour for a scenario where the link cost from Y to X changes from 4 to 1. We focus here only on Y and Z's distance table entries to destination (row) X.

- At time t_0, Y detects the link cost change (the cost has changed from 4 to 1) and informs its neighbours of this change since the cost of a minimum cost path has changed.
- At time t_1, Z receives the update from Y and then updates its table. Since it computes a new least cost to X (it has decreased from a cost of 5 to a cost of 2), it informs its neighbours.

- At time t_2, Y has receives Z's update and has updates its distance table. Y's least costs have not changed (although its cost to X via Z has changed) and hence Y does *not* send any message to Z. The algorithm comes to a quiescent state.

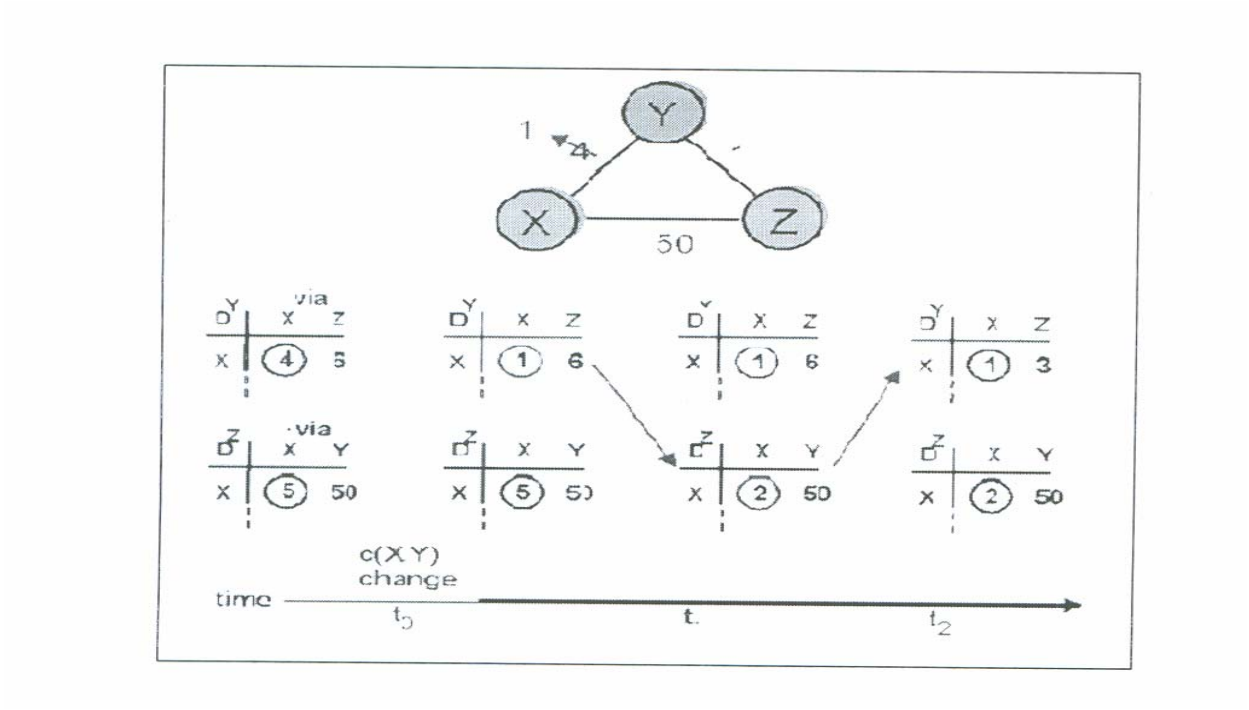

Fig 1.6: Link Cost change: good news travels fast

In Fig 1.6, only two iterations are required for the DV algorithm to reach a quiescent state. The "good news" about the decreased cost between X and Y has propagated fast through the network.

Let us now consider what can happen when a link cost *increases.* Suppose that the link cost between X and Y increases from 4 to 60.

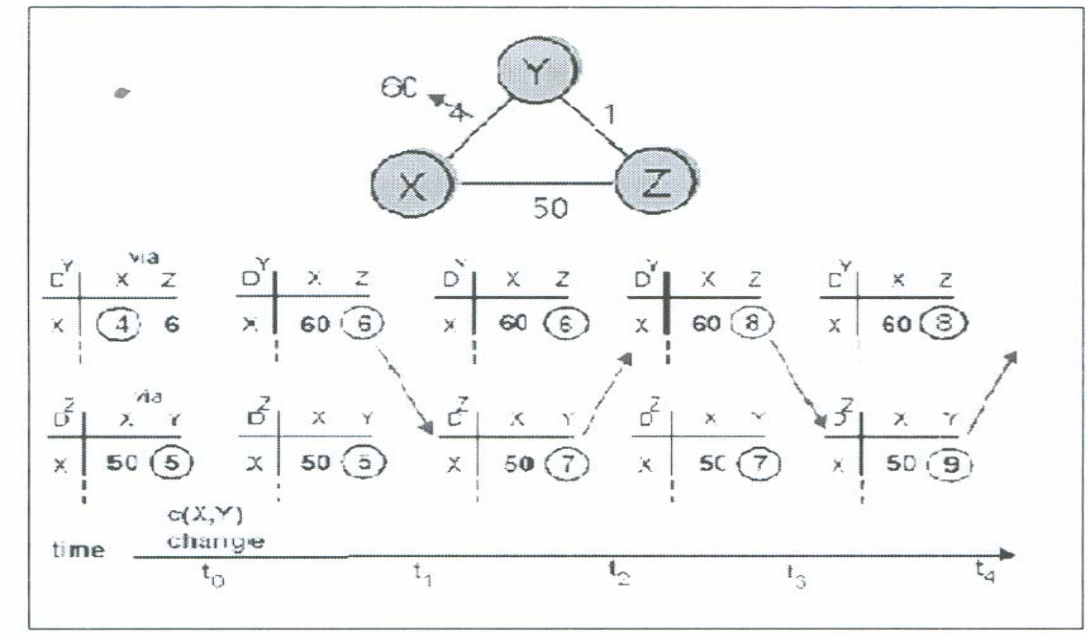

Fig 1.7: Link cost changes: bad news travels slow and causes loops

- At time to Y detects the link cost change (the cost has changed from 4 to 60). Y computes its new minimum cost path to X to have a cost of 6 via node Z. Of course, with our global view of the network, we can see that this new cost via Z is *wrong.* But the only information node Y has is that its direct cost to X is 60 and that Z has last told Y that Z could get to X with a cost of 5. So in order to get to X, Y would now route through Z, fully expecting that Z will be able to get to X with a cost of 5. As of t_1 we have a **routing loop** -- in order to get to X, Y routes through Z, and Z routes through Y. A routing loop is like a black hole -- a packet arriving at Y or Z as of t_1 will bounce back and forth between these two nodes forever, or until the routing tables are changed.

- Since node Y has computed a new minimum cost to X, it informs Z of this new cost at time t_1.

- Sometime after t_1, Z receives the new least cost to X via Y (Y has told Z that Y's new minimum cost is 6). Z knows it can get to Y with a cost of 1 and hence computes a new least cost to X (still via Y) of 7. Since Y's least cost to X has increased, it then informs Y of its new cost at t_2.

- In a similar manner, Y then updates its table and informs Z of a new cost of 9. Z then updates its table and informs Y of a new cost of 10, and so on.

How long will the process continue? You should convince yourself that the loop will persist for 44 iterations (message exchanges between Y and Z) -- until Z eventually computes its path via Y to be larger than 50. At this point, Z will (finally) determine that its least cost path to X is via its direct connection to X. Y will then route to X via Z.

The result of the "bad news" about the increase in link cost has indeed travelled slowly. What would have happened if the link cost change of c(Y,X) had been from 4 to 10,000 and the cost c(Z,X) had been 9,999? Because of such scenarios, the problem we have seen is sometimes referred to as the "*count-to-infinity*" problem.

A Comparison of link state and distance vector routing

The link state and distance vector algorithms we make a comparison of their attributes. Their attributes are of much importance to our study, as we will regard them as our foundation.

- **Message Complexity**. We have seen that LS requires each node to know the cost of each link in the network. This requires O(nE) messages to be sent, where n is the number of

nodes in the network and E is the number of links. Also, whenever a link cost changes, the new link cost must be sent to *all* nodes. The DV algorithm requires message exchanges between directly connected neighbours at each iteration. We have seen that the time needed for the algorithm to converge can depend on many factors. When link costs change, the DV algorithm will propagate the results of the changed link cost *only* if the new link cost results in a changed least cost path for one of the nodes attached to that link.

- **Speed of Convergence**. We have seen that 15 is an $O(n^2)$ algorithm requiring O(nE) messages. It can potentially suffer form oscillations. The DV algorithm can converge slowly (depending on the relative path costs, as we saw in Figure 4.2-7) and can have routing loops while the algorithm is converging. DV also suffers from the count to infinity problem.

- **Robustness**. What can happen is a router fails, misbehaves, or is sabotaged? Under 15, a router could broadcast an incorrect cost for one of its attached links (but no others). A node could also corrupt or drop any 15 broadcast packets it receives as part of link state broadcast. But an LS node is only computing its own routing tables; other nodes are performing the similar calculations for themselves. This means route calculations are somewhat separated under 15, providing a degree of robustness. Under DV, a node can advertise incorrect least path costs to any/all destinations. More generally, we note that at each iteration, a node's calculation in DV is passed on to its neighbour and then indirectly to its neighbour's neighbour on the next iteration. In this sense, an incorrect node calculation can be diffused through the entire network under DV.

Conclusion

We have managed to define routing algorithms, their classification and identify some insight as to how they work. Each algorithm however has its own strengths and weaknesses. Non- adaptive methods are simpler to use but may not provide optimal performance, especially in situations where the network load changes dynamically. In such cases, algorithms such as Link State Routing are preferable as the improvement in performance makes the added complexity worthwhile. We have as well, briefly covered some of the routing methods in use and basic relevant terminology.

References

1. A Zanasi ,N.FF Ebecken & C.A Brebbia(2004), Data Mining V-Data Mining text mining and their business application
2. Hagit Attiya, Jennifer Welch (2004), Distributed Computer fundamentals second editions.
3. E.W Dijkstra (1965), Solutions of a problem in current programming control.
4. Alber to Leon-Garcia (2004), Comunication Network Fundamentals concepts and key

architecture,pp.516-518.

5. Karose, James F (2005), Computer Networking: a top down approach featuring internet .

6. Cisco ,www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/routing.htm

7. S. Bansal, R. Shorey and A. Misra, Comparing the Routing Energy Overheads of Ad-Hoc Routing Protocols, to appear in IEEE Wireless Communications and Networking Conference (WCNC) 2003, March 2003, New Orleans, USA.

8. John M. McQuillan, Isaac Richer and Eric C. Rosen, ARPANet Routing Algorithm Improvements, BBN Report No. 3803, Cambridge, April 1978

9. John M. McQuillan, Isaac Richer and Eric C. Rosen, The New Routing Algorithm for the ARPANE*t*, IEEE Trans. on Comm., 28(5), pp. 711-719, 1980

10. Josh Seeger and Atul Khanna, Reducing Routing Overhead in a Growing *DDN*, MILCOMM '86, IEEE, 1986 .

11. Kurose, James E. and Ross, Keith W. (2004). Computer Networking. Benjamin/Cummings. ISBN 0321227352.

12. Doyle, Jeff (2005). Routing TCP/IP, Volume I, Second Ed.. Cisco Press. ISBN 1-58705-202-4. Ciscopress ISBN 1587052024

13. E. W. Dijkstra: A note on two problems in connexion with graphs. In: Numerische Mathematik. 1 (1959), S. 269–271

14. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 24.3: Dijkstra's algorithm, pp.595–601.