Journal of the Maharaja Sayajirao University of Baroda
ISSN :0025-0422

STATIC AND DYNAMIC ROUTING PROTOCOLS FOR WIRELESS NETWORKS: AN EFFECTIVE APPROACH

Dr. R. Jayaprakash, Assistant Professor, Department of Computer Science,
Nallamuthu Gounder Mahalingam College, Pollachi, Coimbatore:: jpinfosoft@gmail.com

ABSTRACT
The routing protocol area allows you to check the optimum path for digital communication between network nodes. Routers use them to share routing information with alternative routers to with dynamism build global routing tables. Link-state protocols don't "route by rumor." Instead, routers send updates advertising the state of their links (a link may be a directly-connected network). All routers apprehend the state of all existing links among their locus, and store this info in an exceedingly topology table. A distance-vector routing protocol begins by advertising directly-connected networks to its neighbors. These updates area unit sent often (RIP – each thirty seconds; IGRP – each ninety seconds). Neighbors can add the routes from these updates to their own routing tables. Every neighbor trusts this information fully, and can forward their full routing table (connected and learned routes) to each alternative neighbor. Thus, routers absolutely and blindly think about neighbors for route info, a plan referred to as routing by rumor.
Keywords: RIP, IGRP, Router, IGP, EGP, and Links.

I. INTRODUCTION

The zone unit of routing protocols used to check the most advantageous channel for electronic communication between network nodes. Routers use them to contribute to routing data with alternative routers to dynamically build international routing tables. The routing protocols area unit engaged once your organization's network grows to the purpose wherever static routes area unit unmanageable. Fashionable enterprise networks would like dynamic routing tables that mechanically change if there are a unit any traffic or topology changes.

II. DIFFERENT TYPES OF ROUTING PROTOCOLS

There square measure 2 major categories of routing protocols: Exterior entryway Protocol (EGP) and Interior entryway Protocol (IGP). EGP is employed to exchange routing info between autonomous systems. as an example, EGP is employed in knowledge transfers between ISPs (Internet Service Providers) to ISPs or between autonomous systems to ISPs. Whereas, IGP (Interior entryway Protocol) is employed for exchanging routing info between routers at intervals associate degree autonomous system, like knowledge transfers at intervals your organization's native space network (LAN). IGP are often additional classified into 2 categories: Distance-Vector and Link-State Routing Protocols.

Distance-Vector Routing Protocols, routers communicate with neighboring routers, sporadically informing them regarding topology changes.

Whereas in link-state routing protocol, routers produce a roadmap of however they're connected within the network. By calculative the simplest path from that router to each potential destination within the network, link state routing protocols type the routing table. RIP (Routing info Protocol), RIPv2, IGRP (Interior Gateway Routing Protocol), and EIGRP (Enhanced IGRP) square measure a part of Distance-Vector Routing Protocols. However, OSPF (Open Shortest Path First) and IS-IS (Intermediate System to Intermediate System) square measure a part of Link-State Routing Protocols.

Table 1: Nature of Routing Protocols

<table>
<tr><td colspan="6">Nature of Routing Protocols</td></tr>
<tr><td>Exterior Gateway Protocols (EGP)</td><td colspan="5">Interior Gateway Protocols (IGP)</td></tr>
<tr><td rowspan="2">Border Gateway Protocol (BGP)</td><td colspan="3">Distance – Vector</td><td colspan="2">Link – State</td></tr>
<tr><td>RIP</td><td>IGRP</td><td>EIGRP</td><td>OSPF</td><td>IS – IS</td></tr>
</table>

III. STATIC VS. DYNAMIC ROUTING

There are two basic methods of building a routing table:

- Static Routing - Stable
- Dynamic Routing – Occurrence of change at run time

A static (stable) routing table is formed, maintained, and updated by a network administrator, manually. A static route each to each network should be organized on every router for full property. This provides a granular level of management over routing, however quickly becomes impractical on massive networks. Routers won't share static routes with one another, so reducing CPU/RAM overhead and saving information measure. However, static routing isn't fault-tolerant, as any modification to the routing infrastructure (such as a link happening, or a replacement network added) needs manual intervention. Routers operational during a strictly static setting cannot seamlessly select a much better route if a link becomes unavailable. Static routes have Associate Degree body Distance (AD) of one, and so square measure continuously most well-liked over dynamic routes, unless the default AD is modified. A static route with associate degree adjusted AD is termed a floating static route, and is roofed in bigger detail in another guide.

A dynamic (change) routing table is formed, maintained, and updated by a routing protocol running on the router. Samples of routing protocols embody RIP (Routing information Protocol), EIGRP (Enhanced Interior Gate-way Routing Protocol), and OSPF (Open Shortest Path First). Specific dynamic routing protocols square measure lined in nice detail in different guides. Routers do share dynamic routing info with one another that will increase processor, RAM, and information measure usage. However, routing protocols square measure capable of dynamically selecting a special (or better) path once there's a modification to the routing transportation. Don't confuse routing protocols with routed protocols:

1. A routed protocol may be a Layer three protocol that applies logical addresses to devices and routes information between one or a lot of networks (such as net Protocol)
2. A routing protocol dynamically builds the network, topology, and next hop info in routing tables (such as RIP, EIGRP, etc.)

The following briefly outlines the pros and cons of *static* routing:

Static Routing- Advantages

- Minimal CPU/Memory overhead
- No bandwidth overhead (updates are not shared among routers)
- Granular control on how traffic is routed

Static Routing- Disadvantages

- Infrastructure changes must be manually adjusted
- No "dynamic" fault tolerance if a link goes down
- Impractical on large network

The following briefly outlines the advantages and disadvantages of dynamic routing:

Advantages of Dynamic Routing

- Simpler to configure on larger networks
- Will dynamically choose a different (or better) route if a link goes down
- Ability to load balance between multiple links

Disadvantages of Dynamic Routing

- Updates are shared between routers, thus consuming bandwidth
- Routing protocols put additional load on router CPU/RAM
- The choice of the "best route" is in the hands of the routing protocol, and not the network administrator

DYNAMIC ROUTING CATEGORIES

There are two distinct types of dynamic routing protocols:

- Distance-vector protocols
- Link-state protocols

Examples of distance-vector protocols include RIP and IGRP. Examples of link-state protocols include OSPF and IS-IS. EIGRP exhibits both distance-vector and link-state characteristics, and is considered an *amalgam* protocol.

IV. LINK-STATE ROUTING PROTOCOLS

Link-state routing protocols were developed to alleviate the convergence and loop issues of distance-vector protocols. Link-state protocols maintain three separate tables:

- Neighbor table – contains a list of all neighbors, and the interface each neighbor is connected off of. Neighbors are formed by sending packets.
- Topology table – otherwise known as the "link-state" table, contains a map of all links within an area, including each link's status.
- Shortest-Path table – contains the *finest* routes to each particular destination (otherwise known as the "routing" table")

Link-state protocols do *not* "route by rumor." Instead, routers send updates advertising the *state* of their *links* (a link is a directly-connected network). All routers know the state of all existing links within their region, and store this information in a topology table. All routers within an area have *indistinguishable* topology tables. Table 2 displays the characteristics of link-state protocols.

Table 2: Link-State Protocol Summary

Individuality	Explanation
Periodic updates	Only when changes occur. OSPF, for example, also sends all summary information every 30 minutes by default.
Broadcast updates	Only devices running routing algorithms listen to these updates. Updates are sent to a multicast address.
Database	A database contains all topological information from which an IP routing table is assembled.
Algorithm	Dijkstra Algorithm for OSPF.
Convergence	Updates are faster and convergence times are reduced.
CPU/memory	Higher CPU and memory requirements to maintain link-state databases.
Examples	OSPF and IS-IS.

Link-state protocols don't "route by rumor." Instead, routers send updates advertising the state of their links (a link may be a directly-connected network). All routers grasp the state of all existing links at intervals their region, and store this data in an exceedingly topology table. All routers at intervals a neighborhood have indistinguishable topology tables. Table 2 displays the characteristics of link-state protocols.

The best route to every link (network) is keep within the routing (or shortest path) table. If the state of a link changes, like a router interface failing, an advert containing solely this link-state modification are going to be sent to any or all routers at intervals that space. Every router can change its topology table consequently, and can calculate a replacement best route if needed. By maintaining uniform topologies table among all routers at intervals a neighborhood. Link-state protocols will converge terribly quickly and square measure un-attackable to routing loops. Additionally, as a result of updates square measure sent solely throughout a link-state modification, and contain solely the modification (and not the total table), link-state protocols square measure less information measure intensive than distance-vector protocols. However, the 3 link-state tables utilize a lot of RAM and electronic equipment on the router itself. Link-state protocols utilize some variety of price, sometimes supported information measure, to calculate a route's metric. The Dijkstra formula is employed to see the shortest path.

DIJKSTRA'S ALGORITHM

The reason why BFS doesn't work for weighted graphs is extremely easy we will not guarantee that the vertex at the front of the queue is that the vertex nearest to s. it's definitely the nearest in terms of the amount of edges wont to reach it, however not in terms of edge weights. However we will fix this simply. Rather than employing a plain queue, able to use a priority queue during which vertices are sorted by their increasing distt[ ] price. Then at every iteration, we are going to decide the vertex, u, with smallest distt[u] price and decision relax(u, v) on all of its neighbors, v. the sole distinction is that currently we have a tendency to add the burden of the sting (u, v) to our distance rather than simply adding one.

```
bool relax( int u, int v )
{
int newDistt = distt[u] + weight[u][v];
if( distt[v] <= newDistt ) return false;
distt[v] = newDistt;
return true;
}
```

The proof of correctness is strictly constant as for BFS ¬ constant loop invariant holds. However, the rule solely works as long as we have a tendency to don't have edges with negative weights. Otherwise, there's no guarantee that after we decide u because the nearest vertex, distt[v] for a few alternative vertex v won't become smaller than distt[u] at it slow within the future.

```
Algorithm1:
O(n^2+(m+n)log(n)) Dijkstra's
int graph[128][128]; // -1 means "no edge"
int n;  // number of vertices (at most 128)
 int distt[128];
//Compares 2 vertices first by distance and then by vertex number
struct ltDistt   {
Bool operator() (int u,int v) const {
return make_pair(distt[u],u)<make_pair(distt[v],v);
}
}
void dijkstra(int s)
{
for(int i=0;i<n;i++)distt[i]=INT_MAX;
distt[s]=0;
set<int,ltDistt>q;
q.insert(s);
while(!q.empty())   {
int u= *q.begin();// like u=q.front()
q.erase(q.begin());// like q.pop()
for(int v=0;v<n;v++)
if(graph[u][v]! = -1) {
Int newDistt=distt[u]+graph[u][v];
If(newDistt < distt[v]) //relaxation
{
If(q.count(v))q.erase(v);
Distt[v] = newDistt;
q.insert(v));
}
}
```

```
}
}
```

There square measure multiple ways that to implement Dijkstra's rule. the most challenge is maintaining a priority queue of vertices that has three operations –inserting new vertices to the queue, removing the vertex with smallest distt[], and decreasing the distt[] worth of some vertex throughout relaxation. We are able to use a group to represent the queue. Within the following example, assume that graph[i][j] contains the burden of the sting (i, j).
The period is n*log(n) for removing n vertices from the queue, and m*log(n) for inserting into and change the queue for every edge, and n*n for running the 'for(v)'loop for every vertex u. we are able to avoid the quadratic value by mistreatment Associate in Nursing contiguity list, for a total of O((m+n)log(n)). In our own way to implement the priority queue is to scan the distt[] array when to seek out the closer vertex, u.

Algorithm 2: O(n^2) Dijkstra's

```
int graph[128][128],n;
int distt[128];
bool done[128];
void dijkstra(int s)
{
for(int i=0;i<n;i++)
{
distt[i] = INT_MAX;
done[i] = false;
}
Distt[s] = 0;
while(true)
{
//find the vertex with the smallest distt[]value
int u = -1,bestDist = INT_MAX;
for(int i=0;i<n;i++)if(!done[i]&&distt[i]<bestDist)
{
u = i;
bestDist = distt[i];
}
if(bestDist == INT_MAX)
break;
//relax neighbouring edges
for(int v=0;v<n;v++)
if(!done [v]&& graph[u][v]! = -1)
{
If(distt[v]>distt[u]+graph[u][v])
distt[v]=distt[u]+graph[u][v];
}
Done[u] = true;
}
}
```

We have to introduce a brand new array, done (). We have a tendency to may additionally decision it "black[]" as a result of its true for those vertices that have left the queue. First, we have a tendency to initialize done to false and dist () to eternity. Within the most loop, we have a tendency to scan the distt () array to search out the vertex, u, with negligible distt() worth that's not black however. If we

won't realize one, we have a tendency to break from the loop. Otherwise, we have a tendency to relax all of U's adjacent edges. This apparently low-tech methodology is admittedly pretty clever in terms of period. the most while() loop executes at the most n times as a result of at the tip we have a tendency to perpetually set done[u] to true for a few u, and that we will solely do this n times before they're all true. Within the loop, we have a tendency to do O(n) add 2 straightforward loops. The entire is O(n2) , that is quicker than the primary deed as long because the graph is fairly dense (m>n a pair of /log(n)). this can be if we have a tendency to do use associate nearness list within the initial implementation; otherwise, the second can nearly always be faster). Dijkstra's formula is incredibly quick, however it suffers from its inability to manage negative edge weights. Having negative edges in a {very} graph may additionally introduce negative weight cycles that create a re-think the very definition of "shortest path". as luck would have it, there's associate formula that's additional tolerant to having negative edges –the attendant Ford formula.

V. DISTANCE-VECTOR ROUTING PROTOCOLS

All distance-vector routing protocols share several key characteristics:

- Periodic updates of the full routing table are sent to routing neighbors.
- Distance-vector protocols suffer from slow convergence, and are highly susceptible to loops.
- Some form of *distance* is used to analyze a route's metric.
- The Bellman-Ford algorithm is used to establish the shortest path.

A distance-vector routing protocol begins by advertising directly-connected networks to its neighbors. These updates area unit sent frequently (RIP – each thirty seconds; IGRP – each ninety seconds). Neighbors can add the routes from these updates to their own routing tables. Every neighbor trusts this so as fully, and can forward their full routing table (connected and learned routes) to each alternative neighbor. Thus, routers absolutely (and blindly) suppose neighbors for route data, an idea referred to as routing by rumor. There is a unit many disadvantages to the current behavior. as a result of routing data is propagated from neighbor to neighbor via periodic updates, distance-vector protocols suffer from slow convergence. This, additionally to blind religion of neighbor updates, ends up in distance-vector protocols being extremely vulnerable to routing loops. Table one describes the individuality of distance vector protocols. Distance-vector protocols utilize some sort of distance to calculate a route's metric. RIP uses hop count as its distance metric, and IGRP uses a composite of information measure and delay.

Table 3: Distance Vector Protocol Summary

Individuality	Description
Periodic updates	Periodic updates are sent at a set interval. For IP RIP, this interval is 30 seconds.
Broadcast updates	Updates are sent to the broadcast address 255.255.255.255. Only devices running routing algorithms listen to these updates.
Full table updates	When an update is sent, the entire routing table is sent.
Triggered updates	Also known as Flash updates, these are sent when a change occurs outside the update interval.
Split horizon	You use this method to stop routing loops. Updates are not sent out an outgoing interface from which the source network was received. This saves on bandwidth as well.
Count to infinity	This is the maximum hop count. For RIP, it is 15 and for IGRP, it is 255.
Algorithm	One algorithm example is Bellman-Ford for RIP.
Examples	RIP and IGRP are examples of distance vector protocols.

DISTRIBUTED BELLMAN FORD ALGORITHM

Distributed Ford additionally referred to as Distance Vector Routing rule a accepted shortest path routing rule with time quality of O(|V||E|) wherever, V is vertices and E is edges. This rule takes care of negative weight cycles.

The Ford rule may be a Dynamic Programming rule that solves the shortest path downside. it's at the structure of the graph, and iteratively generates a stronger resolution from a previous one, till it reaches the most effective resolution. Bellman-Ford will handle negative weights without delay, as a result of it uses the complete graph to enhance an answer. the thought is to begin with a base case resolution S0, a collection containing the shortest distances from s to any or all vertices, mistreatment no edge in the slightest degree. within the base case, d[s] = 0, and d[v] = ∞ for all alternative vertices v. we have a tendency to then proceed to relax each edge once, building the set S1. This new set is associate degree improvement over S0, as a result of it contains all the shortest distances mistreatment one edge of d[v] is lowest in S1 if the shortest path from s to v uses one edge. Now, we have a tendency to repeat this method iteratively, building S2 from S1, then S3 from S2, and so on... every set S_k contains all the shortest distances from s mistreatment k edges. Distance d[v] is lowest in S_k if the shortest path from s to v uses at the most k edges.

```
Algorithm: Bellmen Ford Algorithm
Vector< pair<int, int> > Edge List;          // A list of direct edges (u,v)
int graph[128][128];                         // Gives the weight
int n,distt[128];

void bellman-ford(int s)
{
//Initialize our solution to the BASE CASE S0
for( int i=0; i<n; i++ )
distt[i] = INT_MAX;
distt[s] = 0;
for( int k=0; k<n-1; k++ )
{     //n-1 iterations
// Builds a better solution Sk+1 from Sk
for( int j=0; j< EdgeList.size(); j++ )
{  // Try for every edge
int u= EdgeList[j].first, v=EdgeList[j].second;
if( distt[u] < INT_MAX  && distt[v] > dist[u] + graph[u][v] ) //relax
distt[v] = distt[u] + graph[u][v];
}
}
// ...Now we have the best solution after n-1 iterations
}
```

We tend to begin with a base case S0, and repeatedly relax each edge to come up with Sk+1 from Sk. Note that within the relaxation step, we tend to don'trelax a foothold if distt[u] is time, or otherwise we tend to could get overflow within the addition (conceptually we tend to ne'er wish to relax such a foothold anyway). additionally note that the order of mistreatment the perimeters will have an effect on the intermediate sets S_k, as a result of we tend to could 1st relax a foothold (u,v), then relax another edge (v,w) within the same step, whereas selecting the reverse order of those 2 edges might not relax them each. However, we tend to currently show that Sn-1 is exclusive, and contains the shortest distance doable from s to any vertex v.

Proposition: (Accuracy of Bellman-Ford) Let S_k denotes the set of distances from s specified d[v] is borderline in S_k if the shortest path from s to v uses at the most k edges. Then the Bellman-Ford rule builds S0, S1, ..., Sn-1 iteratively. Also, Sn-1 is that the best answer and its distinctive proof. We've got antecedently establish that the Bellman-Ford rule generates S0, S1, ..., Sn-1 iteratively within the

higher than paragraphs. Now, assumptive that negative weight cycles accessible from the supply don't exist within the graph, Sn-1 can contain the shortest doable distances from s to the other vertices. this can be as a result of any come in the graph can go in a cycle if we tend to use over n-1 edges, and since negative cycles don't exist, we tend to ne'er wish to use these positive weight cycles as a part of a shortest path. And, as a result of Sn-1 contains the most effective distances, it's exclusive. QED. So, the Bellman-Ford rule is correct, however will it continually terminate? It will, as we tend to solely have 2 loops, one in succession n-1 iterations, and also the different browsing all edges. Hence, the rule continually terminates, and contains a run time of O (n*m). Whereas the Bellman-Ford rule will handle negative weight edges pronto, the correctness of the rule breaks down once negative weight cycles exist that's accessible from s. However, the character of the rule permits United States to discover these negative weight cycles. the thought is that, if a negative weight cycle exist, then Sn-1 are constant as Sn, Sn+1, Sn+2, ... If we tend to run the iteration step over n-1 times, we are going to not be ever-changing the solution. On the opposite hand, if a negative weight cycle exist, then one amongst its edges should have negative weight, and any such edge are often relaxed more even once n-1 iterations, decreasing a number of the distances. Hence, to discover negative weight cycles, we tend to simply got to run the Bellman-Ford rule, and once it terminates, check whether or not we are able to relax any edges. If we can, then that edge is accessible from a negative weight cycle, and also the cycle is additionally accessible from the supply.

```
Detecting negative weight cycles in a graph
vector< pair<int,int> > EdgeList; // A list of directed edges (u,v)
int graph[128][128]; // Gives the weight
int n, distt[128];
int main()
{
// ...Set up the graph
bellman-ford(0); // Run bellman-ford on s=0
// Check for negative weight cycles reachable from s
for( int j=0; j< EdgeList.size(); j++ )
{   // Try for every edge
int u= EdgeList[j].first, v= EdgeList[j].second;
if( distt[u] < INT_MAX && distt[v] > distt[u] + graph[u][v] ) // can relax
cout << "Negative cycle reachable from s exists." << end1;
return 1;
}
cout << "No negative cycle detected, shortest distance found." << end1;
return 0;
}
```

Bellman-Ford is slower than Dijkstra's, but with this added functionality of handling negative weights and detecting negative cycles easily, it can be more useful in some cases. In particular, in a directed acyclic graph (one with no cycles), we can use Bellman-Ford to find the longest path from s to any vertices v, by simply changing all the positive weights to negative, and vice versa. Note that finding the longest path in a general graph is N.

VI. CONCLUSION

Distance-vector protocols suffer from slow convergence, and area unit extremely vulnerable to loops. The Bellman-Ford algorithmic rule is employed to see the shortest path. If the state of a link changes, like a router interface failing, a poster containing solely this link-state modification are sent to all or any routers among that space. Every router can regulate its topology table consequently, and can calculate a brand new best route if needed. Maintain a uniform topology routing table among all routers in a location. Link-state protocols will converge terribly quickly and area unit proof against routing loops.

REFERENCES

[1] Larry L. Peterson, Bruce S. Davie, "Computer Networks: A Systems Approach," 3rd Edition, Morgan Kaufmann, MIT, 2003.
[2] Andrew S. Tanenbaum, "Computer Networks, "Fourth Edition, New jersey, Prentice Hall, Inc., 2003. [3] James F. Kurose & K. W. Ross, "Computer Networking: A Top-down Approach Featuring the internet," 2nd Ed, Peaison Education Asia, 2003.
[4] V. Gambiroza, P. Yuan, L. Balzano, Y. Liu, S. Sheafor, and E. Knightly, "Design, Analysis, and Implementation of DVSR: A Fair HighPerformance Protocol for packet Rings," In IEEE/ACM Transaction on Network, Vol. 12, No. 1, 2004.
[5] William Stalling, "Data and Computer Communications, pearson Education, Inc., Publishing as Prentice Hall, New Jersey, 2011.
[6] Z.Xu and et.al, "A more Efficient Distance Vector Routing Algorithm," In proc. IEEE MILCOM '97 Proceeding, Vol.2, 1997.
[7] Roch Guerin and Ariel Orda, " Computing Shortest Paths for any Number of Hops," In IEEE Transaction on Network, Vol. 10, No. 5, 2002.
[8] T.H Comrmen, C.E. Leiserson, and R.L. Rivest, "An Introduction to Algorithms," Second Edition, MIT Press, Boston, 2002.
[9] Y. Mourtada,"Routing Basics & Protocol," Internet draft, 2000.
[10] Jayaprakash R., Balasubramanian R. (2020) DLBPS: Dynamic Load Balancing Privacy Path Selection Routing in Wireless Networks. In: Sengodan T., Murugappan M., Misra S. (eds) Advances in Electrical and Computer Technologies. Lecture Notes in Electrical Engineering, vol 672. Springer, Singapore. https://doi.org/10.1007/978-981-15-5558-9_70
[11] Jayaprakash, R., & Radha, B. (2020). An Implementation of Trusted Key Management Protocol (TKMP) in Wireless Network. Journal of Computational and Theoretical Nanoscience, 17(12), 5243-5249.