# Exploring Routing Algorithms in Networks

Rohan Rajesh Talesara
*Computer Science Dept.*
*PES University*
Bangalore, India
rtalesara77@gmail.com

Shalini Sai Prasad
*Computer Science Dept.*
*PES University*
Bangalore, India
shalinisaiprasad@gmail.com

Saahitya E
*Computer Science Dept.*
*PES University*
Bangalore, India
saahitya.e@gmail.com

***Abstract*—The main aim of this project is to compare different routing algorithms - link state algorithms and distance vector algorithms, with and without threading to observe the efficiency of these algorithms and understand the algorithmic principles behind them.**

***Index Terms*—Routing algorithms, link-state, distance-vector, Bellman-Ford, Floyd-Warshall, Dijkstra**

## I. Introduction

15% of the total internet bandwidth is consumed by people watching Netflix, according to a report by Sandvine [1]. This makes Netflix the top downstream application worldwide. Routing is important because it is the process of selecting the best path for data traffic to travel in packets across a network.

Routing is at worst glorified a path-finding algorithm in a graph, and at best it is a complex process that takes into consideration factors like dynamic removal of links or traffic fluctuations in a network to ensure that packets travel to the destination in an optimized way. In 2018, there was 102,960 Petabytes of fixed internet traffic and 16,646 Petabytes of mobile traffic. This statistic is expected to triple for mobile traffic, increase by 80% for fixed internet traffic and, on the whole, increase by 22% every year [2].

With new applications on the internet like 3D videos, Virtual Reality(VR) and Augmented Reality(AR), there is a new class of applications that consumes huge amounts of bandwidth(1 gigabit per second) and requires very low latency (refresh rate of 90 times per second). This has led to a strain and bottleneck on the finite bandwidth installed around the world that must accommodate this massive surge in internet traffic. This is where routing plays a crucial part in ensuring that packets travel through a network efficiently without leading to congestion.

## II. Global Routing Algorithms

This class of Routing involves having global and complete information of the graph. The global routing table is built by computing the least cost path between each pair of nodes in the network. Here, all the routing information is calculated at one site at regular intervals and the respective routing tables are sent to each and every node in the network and every router learns the network topology. A distinguishing feature about global routing algorithms is that any of these algorithms that is running has all the information about connectivity and every link's cost. These algorithms are also called link state(LS) algorithms since the algorithm keeps track of all the link costs in the network.

We have implemented three of these algorithms :

- Floyd-Warshall's Algorithm
- Bellman-Ford's Algorithm
- Dijkstra's Algorithm

### *A. Floyd-Warshall's Algorithm Pseudo-Code*

```
FLOYD-WARSHALL(W)

//W is the adjaceny matrix
N <- rows(W)
D^0 <- W
for k <- 1 to N
  for i <- 1 to N
```

```
    for j <- 1 to N
      D^k[i,j]<-min(D^(k-1)[i,j],\
      D^(k-1)[i,k]+ D^(k-1)[k,j])
return D^n
```

### B. Bellman-Ford's Algorithm Pseudo-Code

```
BELLMAN-FORD(W(V, E), s)

//W is the adjacency matrix with
//V vertices and E edges
//s is source node
for v  belongs to V
  d[v] <- infinity

d[s] <- 0

for i <- 1 to n-1
  for(u, v) belongs to E do
    d[v] <- min(d[v], d[u]+w(u,v))

return d[]

`
```

### C. Dijkstra's Algorithm Pseudo-Code

```
DIJKSTRA(W(V, E), s)

//W is the adjacency matrix with
//V vertices and E edges
//s is source
create vertex set Q
for v belongs to V;
  dist[v] <- infinity
  add v to Q

dist[s] <- 0

while Q is not empty
  u <- min(d[])
  remove u from Q;
  for each neighbor v of u
    alt <- dist[u]+w(u, v)
    if alt < dist[v]
      dist[v] <- alt

return dist[]
```

## III. Implementation of Link State Routing Algorithms

### A. Floyd-Warshall's Algorithm

- Time complexity $O(n^3)$
- All pairs shortest path finding algorithm
- Uses dynamic programming

To create the routing table we use the transpose of a parent matrix. The parent matrix is filled when we update the edge of a node j from a source i when there is a smaller path from i to j through k. The parent of node [i,j] is the parent of node [k,j]. This parent matrix represents which neighbor the packet comes through to reach the destination. It is also the transpose of the routing table. We separately parse through parent matrix to obtain the final routing table.

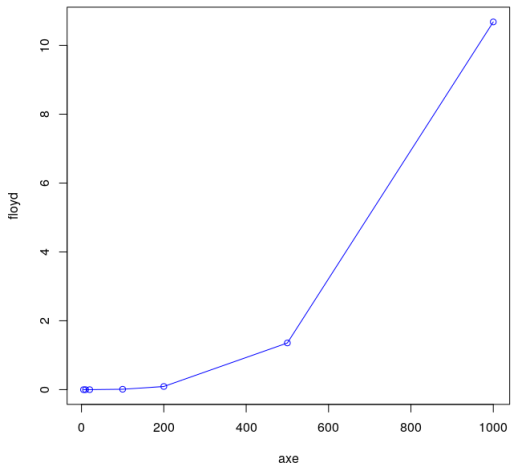


Fig. 1. Time for Floyd-Warshall for n nodes

### B. Bellman-Ford's Algorithm

- Time complexity $O(n^2)$
- Single source shortest path finding algorithm
- Uses dynamic programming

Since Bellman-Ford's algorithm works on a singe source, we create a wrapper function for it. The wrapper calls the algorithm with each node in the graph as a source and constructs the routing table with the shortest paths from each

node to all the other nodes of the graph. The algorithm is slightly optimised by not letting it run n-1 times for each node and breaking out of the loop when no nodes in the graph get updated.

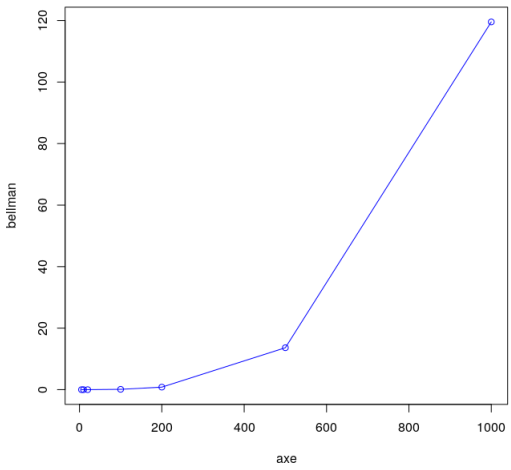


Fig. 2. Time for Bellman Ford for n nodes

### C. Djikstra's Algorithm

- Time complexity $O(EV + V^2)$
- Single source shortest path finding algorithm
- Greedy algorithm

Since Dijkstra's algorithm works on a singe source, we create a wrapper function for it. The wrapper calls the algorithm with each node in the graph as a source and constructs the routing table with the shortest paths from each node to all the other nodes of the graph. The algorithm uses adjacency list implementation. The vertex set is a 1-D array. For sparse matrices with less edges, the time complexity tends to be $O(V^2)$. For dense matrices, it tends to be $O(EV)$.

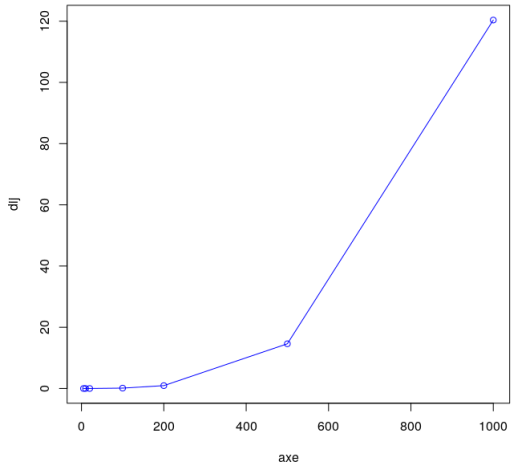


Fig. 3. Time for Dijkstra for n nodes

| Name of Algorithm | Adjacency Matrix | Adjacency List |
|---|---|---|
| Floyd-Warshall | $O(V^3)$ | - |
| Bellman-Ford | $O(V^2)$ | $O(EV)$ |
| Dijkstra | $O(V^2)$ | $O(EV + V^2)$ |

TABLE I
TIME COMPLEXITY OF LINK STATE ROUTING ALGORITHMS WHERE E IS NO OF EDGES AND V IS THE NO OF VERTICES

## IV. DECENTRALIZED ROUTING ALGORITHMS

In decentralized routing, the calculation of the least cost path is carried out in an iterative, distributed manner. Here, no node has complete information about the cost of all the other network links in the beginning. Instead, each node begins with information only about the costs of links to its own neighbouring nodes. Then, in a repeated manner of exchange of information with its neighbouring nodes, the node gradually calculates the least cost path to a destination or set of destinations. We have implemented the distance-vector(DV) routing algorithm. It is called distance-vector routing because each node maintains a vector of distances to all the other nodes.

## V. IMPLEMENTATION OF DISTANCE VECTOR ROUTING ALGORITHM

We have implemented two version of the distance vector routing algorithms- The threaded distance vector routing algorithm and the sequential distance vector routing algorithm

### A. Sequential Implementation

In sequential distance vector routing, we send updates from each node in an iterative manner to each of its neighbors. An update constitutes the routing table information of the node. When any node receives an update, it checks if there is a path from the sender node that sent the update to any of its neighboring nodes, and checks if the path from receiver to destination via sender is shorter than the original distance from receiver to destination. If this condition is satisfied, the cost of the edge between receiver and destination is updated and the sender node is the next hop of that edge. Each node in the network sends updates to all of its neighbors and the updates are sent sequentially.

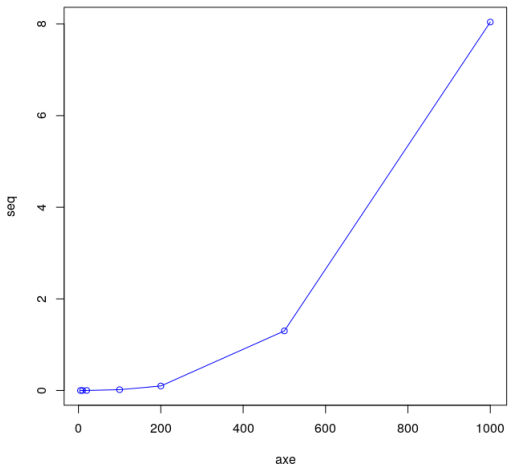


Fig. 4. Time for Sequential implementation for n nodes

### B. Threaded Implementation

In the threaded distance vector implementation, we send an update from each node in the network. Here, we make sure that updates are sent in separate threads. This means that the updates are sent in a truly distributed manner. The algorithm is implemented asynchronously and one thread does not depend on what another thread is operating on. There is no implicit order in thread execution and they run independently.

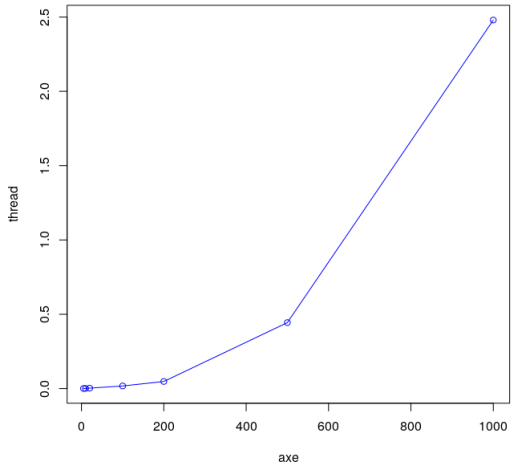


Fig. 5. Time for Threaded implementation for n nodes

## VI. Conclusion

We have found that as the number of nodes increases the following is the increasing order of time taken to build the routing table:

- Distance Vector (Threaded)
- Distance Vector (Threaded)
- Floyd- Warshall
- Bellman- Ford
- Dijkstra


## Acknowledgment

We would like to thank Mr. Channa Bankapur for his constant guidance. We would also like to thank Upasana S. and Sruthi V. for their valuable insight.


## References


[1] Global Internet Phenomena Report, https://www.sandvine.com/blog/global-internet-phenomena-rep

[2] The Bandwidth Bottleneck That Is Throttling the Internet, https://www.scientificamerican.com/article/the-bandwidth-bo

[3] Routing Algorithms, http://ecomputernotes.com/computernetworkingnotes/routing/r

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. *Introduction to Algorithms*. Third Edition,3rd, The MIT Press 2009.