# Understanding Network Routing Problem and Study of Network Routing Algorithms

# Introduction

Problem Statement:

*To identify, understand and compare various routing algorithms used in real world networks.*

# Introduction

Research Objectives:

Define and understand the concepts of routing

Identify the common routing algorithms used in real world networks

Compare existing routing algorithms, and identify which routing strategy Greedy or Dynamic Programming strategy algorithm is more efficient for routing

# Routing Algorithms

Types of Routing Algorithms:

- Link State
- Distance Vector

# Routing Algorithms

Common Routing Algorithms:

Dijkstra’s single-source, shortest path Algorithm

- Greedy
- Link state
- OSPF (Open Shortest Path First)

Bellman - Ford Algorithm

- Dynamic Programming
- Distance Vector
- RIP (Routing Information Protocol)

# Performance Metrics for Comparison

Throughput: The rate at which bits are sent into the network i.e. no of bits sent per unit time. The aim of a good routing algorithm is to maximise the throughput.

Delay:The delay determines the time taken for data to travel from source to destination. Aim is to minimise the delay.

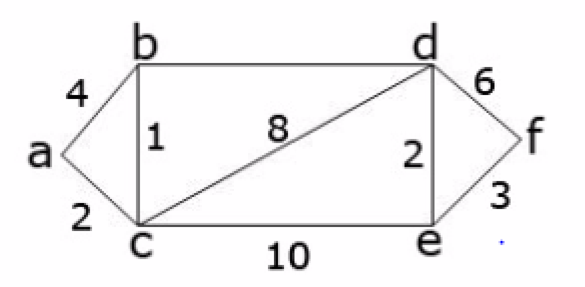

*The six evaluation criteria when determining the efficiency of an algorithm*

# Software and Testing Environment

The following software were used as part of testing and implementation:

- Network Simulator 2 (ns2)
- Dev C++
- GNS3

# Network Topology




| w(u,v) | a | b | c | d | d | f |
|---|---|---|---|---|---|---|
| a | 0 | 4 | 2 | INF | INF | INF |
| b | 4 | 0 | 1 | 5 | INF | INF |
| c | 2 | 1 | 0 | 8 | 10 | INF |
| d | INF | 5 | 8 | 0 | 2 | 6 |
| e | INF | INF | 10 | 2 | 0 | 3 |
| f | INF | INF | INF | 6 | 3 | 0 |

# Dijkstra’s Algorithm

Dijkstra’s algorithm is a single source shortest path algorithm that calculates shortest paths of all other nodes from the source vertex.

```
dist[s] ←o                                   (distance to source vertex is zero)
for  all v ∈ V–{s}
      do  dist[v] ←∞                         (set all other distances to infinity)
S←∅                                          (S, the set of visited vertices is initially empty)
Q←V                                          (Q, the queue initially contains all vertices)
while Q ≠∅                                   (while the queue is not empty)
do   u ← mindistance(Q,dist)                 (select the element of Q with the min. distance)
     S←S∪{u}                                 (add u to list of visited vertices)
      for all v ∈ neighbors[u]
            do  if   dist[v] > dist[u] + w(u, v)        (if new shortest path found)
                     then      d[v] ←d[u] + w(u, v)     (set new value of shortest path)
                                                        (if desired, add traceback code)

return dist
```

# Dijkstra’s Algorithm

Advantages :

- Quickly adjusts to link failures
- Does not propagate the entire routing protocol but it transmits information only about its link
- Suitable for large networks
- Fast for fault discovery and rerouting.

Disadvantages :

- Complicated to configure
- Consumes large bandwidth
- Requires higher processing and memory

# Bellman - Ford Algorithm

```
// Step 1: initialize graph
   for each vertex v in vertices:
       distance[v] := inf                  // At the beginning , all vertices have a weight of infinity
       predecessor[v] := null              // And a null predecesor
    distance[source] := 0                  // Except for the Source, where the Weight is zero
 // Step 2: relax edges repeatedly
   for i from 1 to size(vertices)-1:
       for each edge (u, v) with weight w in edges:
           if distance[u] + w < distance[v]:
               distance[v] := distance[u] + w
               predecessor[v] := u
  // Step 3: check for negative-weight cycles
   for each edge (u, v) with weight w in edges:
       if distance[u] + w < distance[v]:
           error "Graph contains a negative-weight cycle"
   return distance[], predecessor[]
```

# Bellman - Ford Algorithm

Advantages :

- Simple
- Easy to configure

Disadvantages :

- Not scalable in heterogeneous networks
- The periodic updating of routing table consumes bandwidth because the entire routing table is propagated to neighbours
- Slow convergence
- Not suitable for large networks

# Observations

From the comparison of performance of OSPFv2 Routing Protocol and RIPv2 in the IPv4 network:

- Every router within the same routing protocols build routing tables, based on information from neighbouring routers for sharing information between routers.
- Dijkstra’s algorithm performs better in terms of bandwidth utilization
- Dijkstra’s algorithm performs better in terms of speed

# Comparison of the two algorithms

| Features | RIP | | OSPF |
|---|---|---|---|
| | Version 1 | Version 2 | |
| Algorithm | Bellman-Ford | | Dijkstra |
| Path Selection | Hop based | | Shortest Path |
| Routing | Classful | Classless | Classless |
| Transmission | Broadcast | Multicast | Multicast |
| Administrative Distance | 120 | | *110* |
| Hop Count Limitation | 15 | | No Limitation |
| Authentication | No | MD5 | MD5 |
| Protocol | UDP | | IP |
| Convergence Time | RIP>OSPF | | |

# Conclusions

Dijkstra's Algorithm outperforms the Bellman - Ford algorithm in terms of:

- Average throughput
- Packet delay

For number of packets lost,

- In small networks, Dijkstra's algorithm is better
- Bellman-Ford is better in large networks

# Conclusions

The network implementation of Dijkstra’s Algorithm is better than that of the Bellman - Ford Algorithm because:

- Dijkstra’s Algorithm uses either bandwidth or delay as metric for shortest path and it does not use the number of hops as in the Bellman - Ford algorithm

- Dijkstra’s Algorithm can adjust the link more quickly than the Bellman - Ford algorithm

Thank you!