

# A FRAMEWORK FOR PERFORMANCE CHARACTERIZATION AND ENHANCEMENT OF THE OSPF ROUTING PROTOCOL

H. El-Sayed<sup>1</sup>, M. Ahmed<sup>2</sup>, M. Jaseemuddin<sup>3</sup>, D. Petriu<sup>4</sup>

<sup>1</sup>UAE University, Al-Ain P.O.Box 17555, UAE, [helsayed@uaeu.ac.ae](mailto:helsayed@uaeu.ac.ae), <sup>2</sup>CIENA Corporation, Atlanta, Georgia [mahmed@ciena.com](mailto:mahmed@ciena.com), <sup>3</sup>Ryerson University, Toronto, ON M5B 2K3 Canada, [jaseem@ee.ryerson.ca](mailto:jaseem@ee.ryerson.ca), <sup>4</sup>Carleton University, Ottawa ON K1S 5B6, [Dorina.Petriu@sce.carleton.ca](mailto:Dorina.Petriu@sce.carleton.ca)

## ABSTRACT

Open Shortest Path First (OSPF) is a popular Interior Gateway Protocol widely used inside large IP routing domains. Recent studies have shown that the time consumed by local SPF computations must be controlled to achieve millisecond convergence time. This paper presents the authors' experience in measuring and improving the performance of the OSPF routing protocol software. First, we propose a reusable performance characterization framework for routing performance study, which allowed us to perform reproducible experiments in a controlled environment with different network topologies and workloads. Then we present relative performance of several low-level optimizations suggested to optimize route computation code and data structures. Finally, we present the performance benefit of algorithm-level optimization using Incremental Shortest Path First algorithm (ISPF). We are able to achieve substantial gains in performance by using ISPF, more than what is possible by employing techniques for code optimization and using efficient data structures to implement Dijkstra's SPF (DSPF) algorithm.

## KEY WORDS

Performance evaluation, measurements, routing protocols, OSPF, and ISPF.

## 1. Introduction

The performance of routing protocol in laying down the data path is crucial to achieve high performance for data delivery within a network. Routing protocols update their routing tables in response to network changes. For example communication link or router failures in the network can change the optimal routes to certain destinations. It takes some time for any routing protocol to compute the new stable optimal routes after a network change, and the routes used in the interim might be sub-optimal or even nonfunctional. The process of finding the new optimal routes after the network changes is called *convergence* [6]. The convergence time of a routing protocol should be short to avoid packet losses due to transient routing blackholes, which can occur when a nonfunctional route is used during the time routing

converges to the new stable optimal routes. Routing protocols with short convergence time are important for building high-performance stable networks.

Open Shortest Path First (OSPF) is a link-state protocol, which is more reliable and less bandwidth-intensive than distance-vector routing protocols such as RIP [6] and is widely used inside large IP routing domains. But it also requires a great deal of CPU processing power and memory to keep the link-state database up to date. Recent studies [1] have shown that the time consumed by local SPF computations in a link-state protocol such as OSPF must be controlled to achieve millisecond convergence time.

In this paper, we present a framework to measure the performance of the OSPF routing protocol. Our main focus is on measuring the overheads involved in the routing table computation that is known to be one of the most CPU-intensive activities within OSPF-based routers [6]. Dijkstra's Shortest Path First (DSPF) algorithm [3] is a de-facto standard used in almost all commercial OSPF implementations for routing table computation. We have developed a set of tests to analyse and characterize the performance of the DSPF algorithm in the University of Maryland (UMD) OSPF routing software, in accord with the specifications of the IETF [8], [9]. Then we evaluate the relative benefit of employing some code optimization techniques together with using more efficient data structures to improve its performance. We outline the implementation of the Incremental SPF algorithm (ISPF) [5] in our framework and evaluate the benefit of ISPF. We clearly demonstrated that the algorithm-level optimization using ISPF outperforms the low-level optimizations.

The paper is organized as follows. Section 2 gives an overview of the OSPF protocol focusing on the DSPF computation and also describes the ISPF algorithm. Section 3 presents our framework for evaluating routing performance. Section 4 discusses and analyzes the performance of the DSPF computation using the low-level optimization techniques; and Section 5 presents the relative performance of the DSPF and ISPF algorithms. Finally Section 6 concludes the paper.

## 2. Background

In this section we give an overview of OSPF protocol, describing both SPF computation algorithms – Dijkstra and ISPF.

### 2.1 Overview of the OSPF Protocol

Open Shortest Path First (OSPF) is a dynamic, hierarchical, link-state routing protocol designed to be used inside an Autonomous System (AS). It employs DSPF algorithm to calculate the routing table. The OSPF divides a domain into a two-level hierarchy of areas, where the root is the backbone area connecting all other areas at the second level. Every router in an OSPF area maintains a synchronized topological database, called the *link-state database*, which contains the network topology as a map of interconnection of routers inside the area. The Link-State Advertisement (LSA) describes the state of the advertising router's interface to the link, which is the basic unit of information propagated through the network and stored in the link-state database. All the link-state databases in an area are synchronized via a procedure called reliable flooding [7].

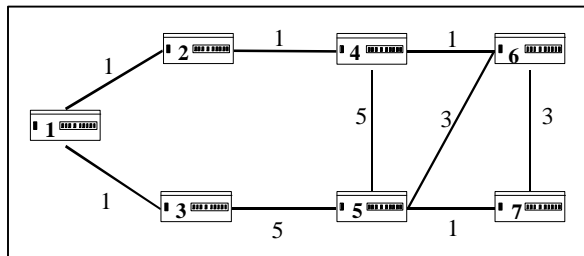


Figure 2.1 a sample physical view of an OSPF area

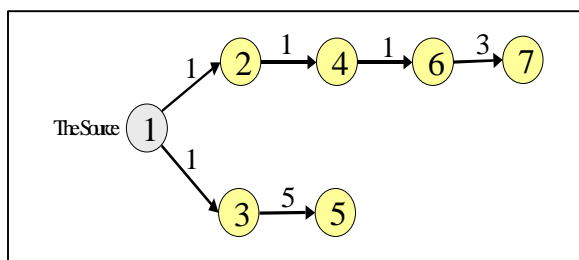


Figure 2.2 the DSPF Tree generated by Router 1

Each router builds the routing table from its link-state database by calculating the shortest paths to all possible destinations using SPF algorithm. The algorithm maintains a *Candidate List* of the nodes that are potential candidates to be added to the shortest path first tree (SPF-tree) in each iteration. It begins by adding the source router (the router performing the computation) to the SPF-tree, and all of the neighbours of the source router to the Candidate List including the cost of the links to the neighbours. A router in the Candidate List with the

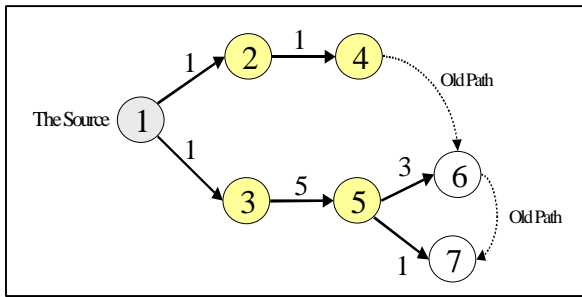
smallest distance to the source is removed from the list and added to the SPF-tree. Then all the neighbours of that router are examined for the possibility of including in the Candidate List if they are discovered for the first time or their shortest path costs are improved. The algorithm iterates over the Candidate List and terminates when it becomes empty. The key operations in the implementation of the SPF algorithm are: (1) to insert a new node into the Candidate List in a sorted order based on the node's distance to the source, and (2) to retrieve from the list the node with the minimum distance to the source. In this paper the terms routers and nodes are interchangeably used.

### 2.2 The ISPF algorithm

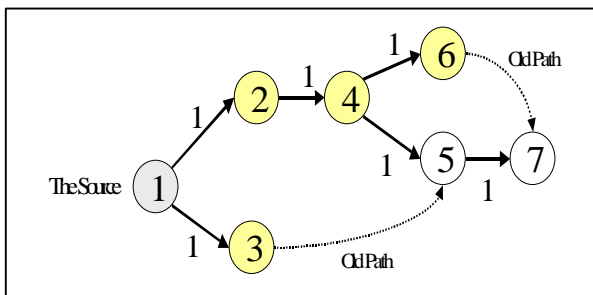
To avoid re-computing the whole routing table as a result of a change in the network topology McQuillan et. al. developed a more efficient variant of the DSPF algorithm, called Incremental SPF (ISPF) [5]. Before discussing the ISPF algorithm let us look at an example that demonstrates the inefficiency of the SPF algorithm. Figure 2.2 shows the SPF-tree computed by R1 for the OSPF area shown in Figure 2.1. Consider the case where the communication link between R4 and R5 goes down. Obviously the link is not in the SPF-tree; hence its failure does not change the routing table. Nevertheless, the DSPF algorithm requires re-computation of the whole routing table when the link failure is discovered regardless of the relevance or the impact of the change on the actual SPF tree. The waste of time and computational resource on such redundant computations slow down the routing convergence. The ISPF algorithm avoids redundant computation by determining and re-computing only those shortest paths that are affected by the change.

In this section we explain the ISPF algorithm by considering two major cases of link-state change. For detailed description of the algorithm see reference [580]. First, let us consider the case where the cost of the link from node A to node B increases by X. The ISPF does not recompute the routing table if the link were not in the SPF-tree, because the increased cost would not change the routing table. If the link were in the SPF-tree, then the new cost could potentially increase the cost of the shortest path to node B as well as the cost of the shortest paths to the nodes for which node B lie on their paths. Thus, only the nodes in the subtree rooted at B are candidates for the change of their positions in the SPF-tree. The ISPF algorithm first isolates all the nodes in the subtree rooted at B and increases the cost of their paths from the source by X. Then it tries to find a shorter path to each node in the subtree through an adjacent node that is not part of the subtree, and adds the node to the Candidate List if a new shorter path can be found. After examining all nodes in the subtree if the Candidate List is not empty, the DSPF algorithm is invoked, taking the new Candidate List and the previous SPF-tree as inputs, to find the best possible shortest paths for the nodes on the list and possibly other subtree nodes. Figure 2.3 shows the modifications to the

SPF-tree of Figure 2.2 that results when the cost of the communication link from router R4 to router R6 increases from 1 to 8.



**Figure 2.3. The new SPF tree after the topological change**



**Figure 2.4. The new SPF tree after the change**

Next, let us consider the case where the cost of the link from node A to node B decreases by X. There are two possible outcomes of the change:

1. If the link were in the SPF-tree, then the shortest paths to the nodes in the subtree rooted at B would remain unchanged but their costs would decrease by X. Moreover, if the cost of the path of any node C from the source is less than or equal to the cost of the path to B, the path to C will not change; because the path to C must reach B first in order to take advantage of the improved link cost. However, the nodes that are not in the subtree but are at a greater distance from the source than B may have a shorter path through one of the nodes in the subtree. The ISPF algorithm handles this case by first isolating all the nodes in the subtree rooted at B and decreasing the costs of their paths from the source by X. Then for each node S that is not in the subtree but is adjacent to a subtree node, it tries to find a shorter path to this node via the adjacent node in the subtree.
2. In the case where the improved link is not in the SPF-tree, but if the cost of the path from the source to B using the improved link is greater or equal than the cost of the original shortest path to B, then B cannot take advantage of this improvement and there will be no changes to the SPF-tree. On the other hand if the new path is shorter than the original path, then the

shortest route to reach B is through A, which would change the SPF-tree. The node B and its subtree are relocated and attached to node A via the improved link. The SPF-tree then becomes identical to the tree of the first case where the link from A to B was in the SPF-tree and its cost was decreased, hence the tree is manipulated in the same way. Figure 2.4 shows the modified SPF-tree of Figure 2.2 when the cost of the R4 to R5 link decreases from 5 to 1.

Finally, there are other types of topological changes that can happen in addition to the previous changes we discussed, such as the complete failure (or recovery) of a node or a link. The ISPF algorithm handles these cases in a similar fashion by isolating the affected nodes from the change and recalculating the shortest paths to those nodes.

### 3. Performance Evaluation Framework

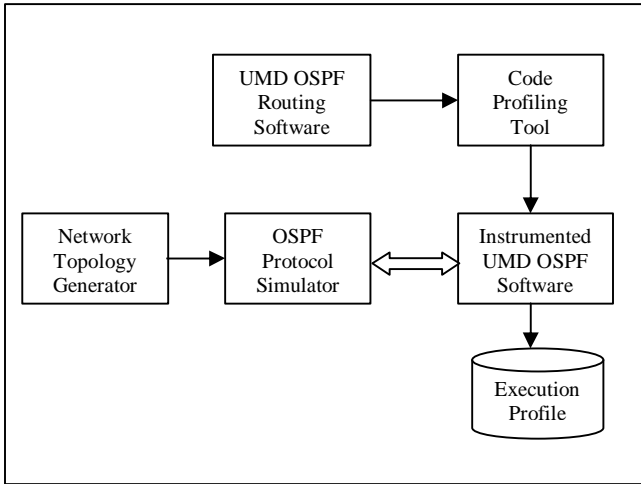
We developed an experimental performance characterization framework, which allows us to conduct reproducible experiments in a controlled environment with the ability to change network topologies, degrees of connectivity, and routing workloads. Figure 3.1 shows our performance evaluation framework. The framework is composed of the following components: (1) Network Topology Generator (TIERS) [2], (2) OSPF Routing Protocol Simulator (RPS), (3) Code Profiling Tool (QUANTIFY) [10], and (4) The Router under test (RUT), which is basically the instrumented UMD OSPF routing software.

In the proposed framework, TIERS is used to generate the network topologies that reflect the hierarchical domain structure and locality found in large networks and the Internet. We use the RPS to simulate logical OSPF clouds and exchange OSPF packets with the RUT. It acts as the boundary router between the simulated cloud and the RUT. The RPS can simulate multiple OSPF areas within several routing domains. Thus it makes the RUT believe that the RUT is connected to a real OSPF network. We link the OSPF code of the RUT with QUANTIFY to instrument the object code and monitor its execution. QUANTIFY collects the performance data at run time and produces an execution profile that shows, for example, the execution time of various functions and their frequency of calls. The execution profile can be used to find principal bottlenecks in the code and identify which portions need further tuning to improve the performance.

### 4. Performance Characterization of the SPF Algorithm

In this section we first characterize the performance of the DSPF algorithm to identify the operations mainly responsible for its inefficiency. Then, we evaluate the

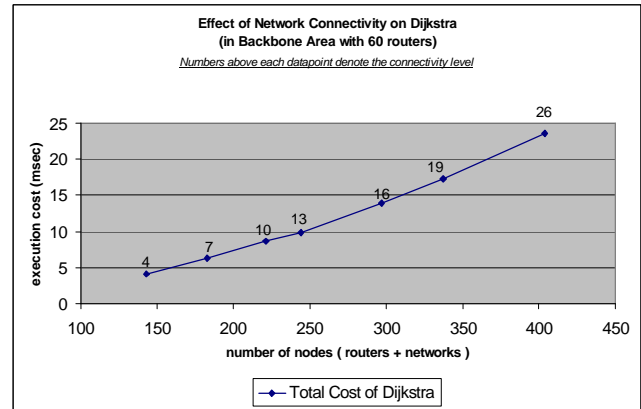
performance improvement of some suggested optimizations in the code and data structures. We performed two different experiments to investigate the impact of network topology on the execution time of the algorithm.



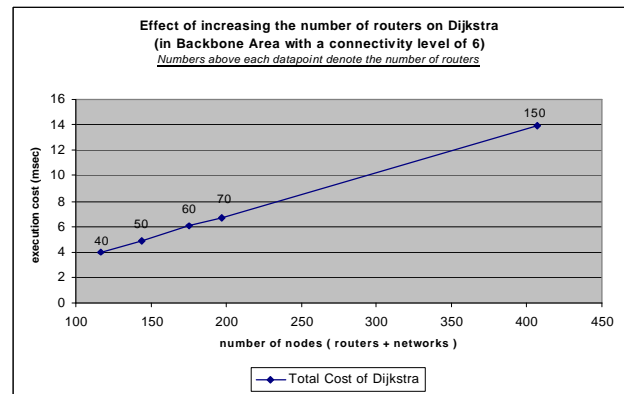
**Figure 3.1 Software Performance Evaluation Framework**

The convergence time is the main performance metric in this study, which is defined as the time taken by the RUT to recalculate its routing table as a result of a change in the area to which the RUT belongs. It is equivalent to the execution time of the DSPF algorithm. We can simulate a topological change by bringing a link up or down, which triggers recomputation of the whole routing table in DSPF. Figure 4.1(a) shows the execution time (execution cost) against the number of nodes, which is defined as the sum of the number of routers and networks inside an area. It shows the impact of connectivity level on the execution time. The average number of links per router determines the average connectivity of a router, which is shown in the graph corresponding to each data point. Figure 4.1(b) shows the impact of number of routers in an area on the execution time, where the number of routers is shown in the graph. We can make the following observations from the graphs in Figure 4.1 (a) , (b): (1) The execution time of DSPF is directly proportional to the number of nodes inside an area. It degrades as the connectivity level or the number of routers is increased. (2) The execution time of the algorithm varies significantly with the topology of the area with a fixed number of nodes. For example, the SPF computation takes 25 msec on a 400-node area with 60 routers and connectivity level of 26 (Figure 4.1(a)), while it takes less than 13 msec on a 400-node area with 150 routers and connectivity level of about 6 (Figure 4.1(b)). Hence, we can conclude that the connectivity level within an area has a greater effect on the convergence time than the number of routers in the same area. (3) Low connectivity level (sparse area) exhibits linear growth (Figure 4.1(b)), whereas high connectivity level (dense

area) exhibits quadratic growth in the execution time (Figure 4.1(a)). The result conforms to the complexity of Dijkstra's algorithm reported in the literature [7], which is of the order of  $n^2$  for dense networks and  $n \log n$  for sparse networks, where  $n$  is the number of nodes.



**Figure 4.1 (a): The effect of network connectivity on Dijkstra's algorithm**



**Figure 4.1 (b): The effect of the number of routers on Dijkstra's algorithm**

Figure 4.2 (a), (b) depict the execution time of the nine most expensive functions in the algorithm. It shows that *cl\_enq* and *findLSA* are the two most expensive functions, which can be optimized to reduce the convergence time. The *cl\_enq* is used to insert a new node in the Candidate List, whereas *findLSA* is used to find a node with a particular link state id in the link-state database. Through other experiments we find that the execution cost of inserting a new node into the Candidate List is highly variable, mainly because of using inefficient data structure (sorted link list) for the Candidate List. The *cl\_enq* function linearly searches the linked list when inserting a new node. We can optimize the *cl\_enq* function by using a more efficient data structure such as binary search tree or binary heap to implement the Candidate List.

Figure 4.3(a) shows the performance improvement of the DSPF algorithm as a result of using more efficient data structure. From the detailed analysis we also found that the high frequency of calls to *findLSA* function makes it one of the most expensive components within DSPF. The *findLSA* function is called several times to compute the pointers to the neighbor nodes. We implemented a more efficient data structure for nodes that facilitate caching of pointers to the neighbor nodes. With caching we can compute the pointers to the neighbor nodes once and cached them for subsequent accesses, which eventually reduces the frequency of calls to *findLSA*. We can observe that substantial performance gain is achieved by caching the pointers in Figure 4.3(b).

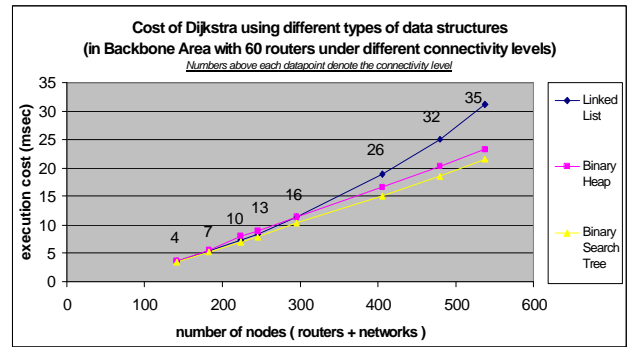


Figure 4.3 (a): Performance improvement by employing more efficient data structure

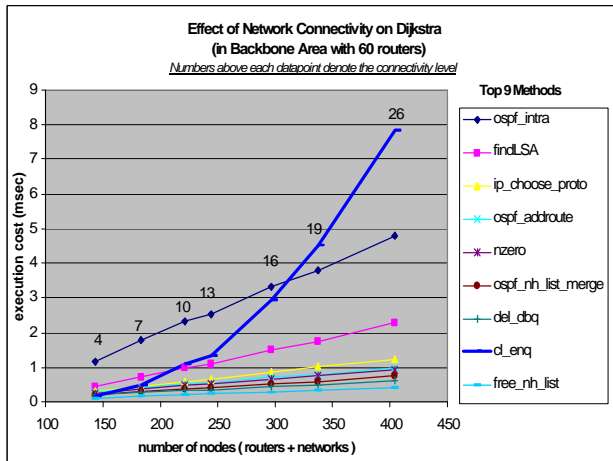


Figure 4.2 (a): The effect of network connectivity on the performance of SPF

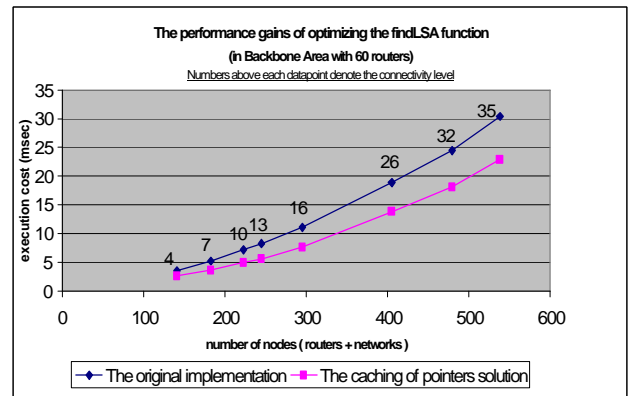


Figure 4.3 (b): Performance improvement by caching pointers

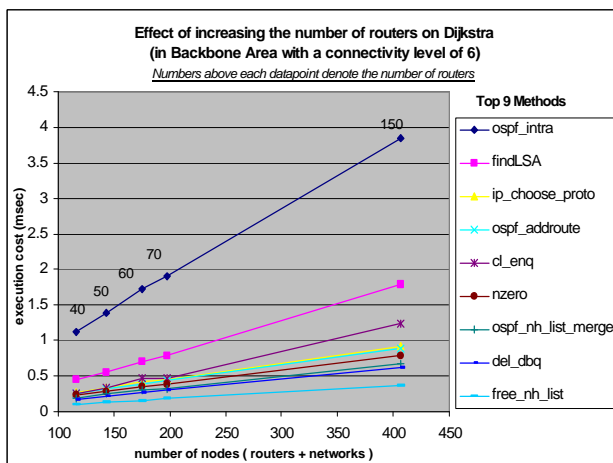


Figure 4.2 (b): The effect of number of routers on the performance of SPF

## 5. Optimization using ISPF Algorithm

This section gives an overview of the design of the ISPF algorithm that we implemented in the UMD OSPF software. The detailed discussion on the design can be found in [4]. And then we present the performance results obtained using the ISPF implementation in the performance evaluation framework discussed in Section 3. Figure 5.1 shows the data flow diagram of the ISPF algorithm. It also shows the major components of the ISPF implementation.

The ISPF implementation consists of four major building blocks, namely OSPF Reception, Topology Change Detector, the implementation of ISPF algorithm, and the modified DSPF implementation. The OSPF Reception implements the standard input processing and error checking logic for OSPF packets. The Topology Change Detector implements the logic that detects the changes in the area's topology. It analyses the incoming LSAs and compares them with the previously installed LSAs, if exist, in the link-state database originated by the same node. Once the Topology Change Detector determines the exact type of change, it generates a new change event and

adds that to the change event list. The ISPF algorithm traverses the change event list and examines each change event. It tries to identify and update only the shortest-path routes affected by the topological change. Table 5.1 shows the change events processed by the ISPF. The processing detail is discussed in [4].

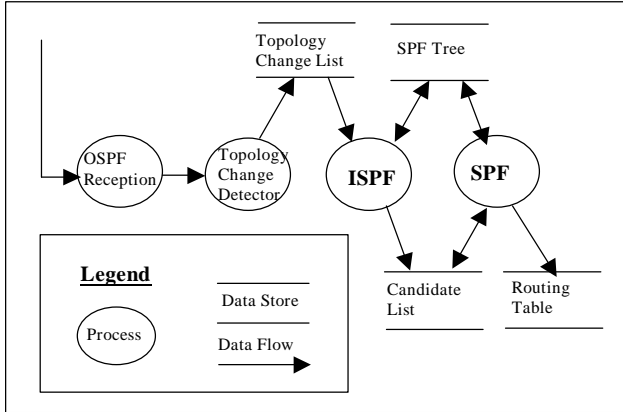


Figure 5.1: ISPF Implementation Components

Event Name	Description
NewNode	The arrival of a new LSA that has not been previously received
LinkDown	Failure of a communication link (or a node)
LinkUp	Recovery of a link (or a node)
LinkCostIncrease	Cost of a link increases
LinkCostDecrease	Cost of a link decreases
AgedLSA	The LSA of a node ages out (expiring)

Table 5.1: Types of topological change events

We slightly modified the implementation of the standard DSPF algorithm to update the routing table after the ISPF completes processing the topological change events. The standard DSPF algorithm builds the SPF-tree from an empty Candidate List, whereas the modified implementation takes as input the Candidate List formed by the ISPF algorithm. It then performs the steps of DSPF calculation for each of the nodes in the Candidate List.

Figures 5.2 shows the execution time of DSPF and ISPF algorithms for backbone and leaf areas. The execution time of the ISPF algorithm is much less than the execution time of the DSPF algorithm. The DSPF shows linear growth for low connectivity level in sparse area (as shown in Figure 5.3) while quadratic growth for high connectivity level in dense area (as shown in Figure 5.2). On the other hand, the execution time of ISPF shows much slower growth rate. For the same number of nodes,

the execution time of DSPF varies significantly depending upon the area topology, whereas the effects of topology on the execution time of ISPF is less obvious.

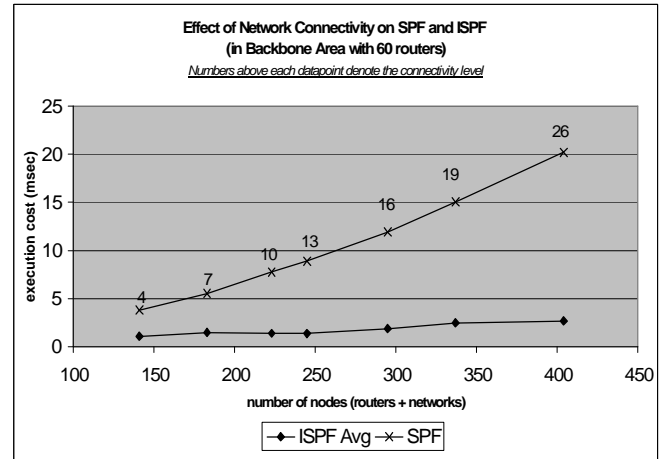


Figure 5.2: The effect of network connectivity in a backbone area

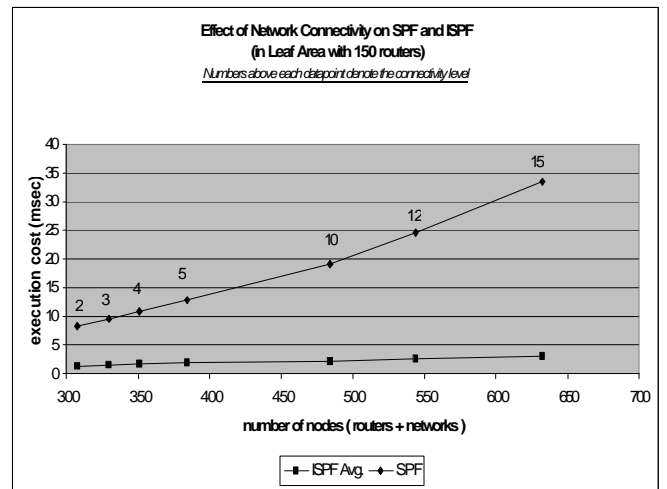


Figure 5.3: The effect of network connectivity in a leaf area

Although the above tendencies are not unexpected, to our knowledge the quantitative measure has never been presented before. Overall we observe that the DSPF algorithm takes a fixed amount of time to perform the routing table calculation in an area of a certain size and connectivity because every time it computes the full routing table. On the other hand, the execution time of ISPF varies in proportion to the number of shortest paths affected by the topological change.

## 6. Conclusion

OSPF is a link-state routing protocol that commonly uses Dijkstra's SPF (DSPF) algorithm to compute full routing table as a result of any link-state change in the network, which slows the routing convergence. It is known that the DSPF algorithm used in the routing table calculation has a considerable impact on the performance of OSPF routers. In this paper we presented three contributions. First, we presented our routing performance evaluation framework. We also demonstrated the viability of the framework by characterizing the performance of DSPF and evaluating the relative benefit of ISPF. Second, we measured the performance of DSPF under different topologies. We are also able to characterize the performance and identify the functions causing bottleneck. We showed that through code optimization and efficient data structures the execution time of DSPF can be reduced by 10-30%. Third, we study the impact of algorithm-level optimization on the convergence time. We showed through extensive simulation that the use of ISPF in optimizing the OSPF routing table calculation significantly improves the performance of OSPF routers. Optimizing the routing table calculation improves the responsiveness and the scalability of the routing engine by reducing the convergence time and by reducing the need for more CPU power as the network grows.

## References:

- [1] A. Shaikh, Experience in black-box OSPF measurement. *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, 2001.
- [2] K.L. Calvert, et al., Modeling Internet Topology. *IEEE Comm. Magazine*, 1997.
- [3] E.W. Dijkstra, A Note on Two Problems in Connection with Graphs. *Num. Math., Vol. 1*, 1959.
- [4] H. El-Sayed, M. Ahmed, M. Jaseemuddin, Routing Software Performance Evaluation Framework. *UAEU Tech Report*, 2004.
- [5] J. McQuillan, I. Richer and E. C. Rosen, The New Routing Algorithm for the ARPANET. *IEEE Transactions on Communications, COM-28(5)*; pp. 711-719, May 1980.
- [6] J. Moy, OSPF: Anatomy of an Internet Routing Protocol. *Addison-Wesley*, February 1998.
- [7] J. Moy, OSPF Version 2. *RFC 2328*, April 1998.
- [8] V. Manral, R. White, and A. Shaikh, OSPF Benchmarking Terminology and Concepts, *IETF Internet draft-ietf-bmwg-ospfconv-term-10.txt*, June 2004
- [9] V. Manral, R. White, and A. Shaikh, Considerations When Using Basic OSPF Convergence Benchmarks. *IETF Internet draft-ietf-bmwg-ospfconv-applicability-07.txt*, June 2004.
- [10] Rational PurifyPlus, IBM Rational suite.