Performance Comparison of Routing Protocols under Dynamic and Static File Transfer Connections*

A. Udaya Shankar†, Cengiz Alaettinoğlu, Klaudia Dussa-Zieger, Ibrahim Matta

Department of Computer Science
University of Maryland
College Park, MD 20742

Abstract

We compare the performance of two recently proposed distance–vector algorithms (Merlin–Segall and Extended Bellman–Ford) with a link–state algorithm (SPF), under varying file transfer workload. (Unlike the traditional distance–vector algorithms, these new distance–vector algorithms do not suffer from long–lived loops.) Our comparison is done using a recently developed network simulator, MaRS. We consider both dynamic and static file transfer connections, and both uniform and hotspot distributions of source–sink pairs. Our conclusion is that Extended Bellman–Ford performs as well as SPF in terms of delay and throughput. This suggests that distance–vector algorithms are appropriate for very large wide–area networks, since their space requirements are less than that of link-state algorithms.

1 Introduction

In wide–area store–and–forward computer networks, routing protocols are responsible for forwarding data packets over good routes which optimize real–time performance measures such as delay and throughput. The delay along a route depends on the traffic through its links, which depends on the external (time–varying) load. Consequently, a routing protocol must monitor link delays and adapt its routes to changes in link delays.

There are two basic kinds of routing algorithms: link–state and distance–vector. In the **link–state** approach, each node maintains a view of the network topology with a cost for each link. In the **distance–vector** approach, which is based on the Bellman–Ford algorithm [9], each node maintains for each destination a set of distances, one for each of its neighbors. The distance–vector approach has less space overhead. However, the straight–forward distributed implementation of the Bellman–Ford algorithm performs badly, because it can have long–lived loops giving rise to large dissemination times [2]. The ARPANET initially used this Distributed Bellman–Ford algorithm, but it was replaced in 1979 by a link state algorithm referred to as SPF (Shortest Path First) [19].

Since 1979, many new kinds of distance–vector algorithms have been proposed [20, 24, 15, 3, 11, 23] which achieve significantly reduced dissemination times by using node coordination mechanisms. We do not know whether the new distance–vector algorithms are good enough

*This work is supported in part by RADC and DARPA under contract F30602-90-C-0010 to UMIACS at the University of Maryland, and by National Science Foundation Grant No. NCR 89-04590. The views, opinions, and/or findings contained in this report are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency, RADC, or the U.S. Government. Computer facilities were provided in part by NSF grant CCR-8811954.

†Also with Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742.

to replace SPF. To answer this, we must compare these algorithms in the context of varying workload. To ignore the workload, as is done in all hop–count and message complexity analyses, is to ignore the critical delayed feedback between routing table changes and link cost changes. A testbed appears essential for understanding these issues.

We have developed such a testbed, called MaRS (Maryland Routing Simulator) [1]. In this paper, we use MaRS to evaluate the performance of three routing algorithms in terms of delay and throughput under varying file transfer workload. In all cases, we consider the NSFNET–backbone network and a hop–normalized link cost function. We compare **SPF** and two recently proposed distance–vector algorithms, namely Merlin–Segall [20, 24] henceforth called **MS**, and Extended Bellman–Ford [3] henceforth called **ExBF**. The file transfer workload we consider is specified in terms of source–sink pairs. We consider two kinds of geographic distributions for the source–sink pairs: **uniform**, where the pairs are distributed uniformly among the nodes of the network, and **hotspot–with–background**, where more source–sink pairs are allocated to a specified hotspot node. We consider two kinds of burstiness for the file transfer connections: dynamic and static. In the **dynamic** case, for each source–sink pair there is an unending succession of file transfer connections. In each connection, a finite number of data packets is produced by the source. The duration of the connection, i.e., the time to deliver the data packets to the sink, depends on the routing protocol performance. The poorer the performance, the longer the duration, resulting in more connections being active simultaneously. Thus, the number of active connections varies with time. In the **static** case, for each source–sink pair there is a single file transfer connection in which an unending succession of packets is produced by the source.

Our overall conclusion is that under hotspot–with–background workload, SPF and ExBF are about equivalent (with SPF being slightly better), while MS does significantly worse. Under uniform workload, MS and ExBF are about equivalent (with ExBF being slightly better), while SPF does significantly worse. We give more importance to hotspot workload than to uniform workload, because we believe that it is more representative of real–life network conditions.

Elsewhere [25], we have used MaRS to compare the three routing algorithms under static file transfer workload, for varying link cost functions, varying link failure and repair characteristics, and regular and high–speed versions of the NSFNET–backbone. Our conclusion there is that ExBF is as good as SPF for good link cost functions. Furthermore, ExBF is less sensitive to changes in the link cost function. MS performs significantly worse than SPF and ExBF except at very high and very low workloads.

The rest of the paper is organized as follows. In section 2, we review related work. In section 3, we describe SPF, MS, and ExBF. In section 4, we describe our simulation environment. In section 5, we summarize the results of the simulations and give our conclusions.

2 Previous Work

To our knowledge, there is no previous work comparing the new distance vector algorithms to SPF with respect to delay and throughput. There is previous work examining these algorithms individually, and we mention experimental, simulation, and analytic approaches.

Regarding experimental approaches, a series of tests was performed on the ARPANET to evaluate its new routing algorithm (i.e. SPF) [19] and its new link cost function (i.e. hop-normalized delay function) [17]. The tests showed that the new algorithm and cost function give better performance than the old ARPANET algorithm (i.e. Distributed Bellman-Ford) and the old delay function.

Regarding simulation approaches, reference [4] concluded that an adaptive strategy is needed in a skewed workload environment. For SPF, it investigated the effects of the link cost function parameters and demonstrated optimal settings. In [26], the original and new ARPANET algorithms (i.e. Distributed Bellman-Ford and SPF) were evaluated in a comparison with two hierarchical extensions to these algorithms. It was observed that a shorter link–cost update period should be used for higher data workload. Multi–path extensions to SPF were proposed in [22] and [27], and were shown to perform better. Reference [28] compared SPF, Distributed Bellman-Ford and DUAL algorithms with respect to measures such as number of paths with loops after a node/link change. However, it has no workload and assumes unit link delays and zero processing time at the nodes. Reference [21] compared SPF, Gallager's algorithm [10], and the CODEX algorithm [13, 14]. It was shown that SPF with the old delay link cost function performs poorly compared to the other two algorithms in terms of packet delay under moderate to heavy traffic conditions.

Regarding analytical approaches, one approach is to use a queueing network model where routing is based on delayed state information. Unfortunately, this is usually intractable except for extremely simple models [7, 8]. Another more tractable approach is to use *flow models* [2, 8]. However, this approach assumes that flows are static (or quasi-static) [2].

3 Overview of Routing Algorithms

In this paper, we are concerned with next–hop routing; that is, each data packet has its destination node id, and each node (other than the destination) maintains a neighboring node id, referred to as *next hop*, to forward the packets to. The next hop of a node can also be *nil*. In this case if the node receives a data packet, which needs to be forwarded, for that destination then the packet is dropped (and later retransmitted by the source).

Link–State Algorithms

In the link–state approach, each node maintains a view of the network topology with a cost for each link, and uses this view to obtain minimum cost routes for each destination. To keep these views up–to–date, each node regularly broadcasts the link costs of its outgoing links to all other nodes. Note that some of the link costs in a node's view can be old because of long propagation delays, partitioned network, etc. Such inconsistent views of network topology at different nodes might lead to loops in the next hops. However, these loops are *short–lived*, because they disappear in the time it takes a message to traverse the diameter of the network.

SPF is a link–state algorithm in which Dijkstra's shortest path algorithm [5] is applied to the view of the network topology to determine next hops.

Distance–Vector Algorithms

In the distance–vector approach, every node maintains for each destination a set of distances, one through each of its neighbors. The node chooses a neighbor with the minimum distance for a destination as the next hop towards that destination.

It is well known that the straight–forward implementation of the above approach, in addition to short–lived next hop loops, can have *long–lived* loops of duration proportional to link cost changes [23]. The new distance–vector algorithms have mechanisms to avoid long–lived loops.

MS is a distance–vector algorithm which avoids both short–lived and long–lived loops. For each destination, MS guarantees that the next hops on the nodes that can reach the destination form a tree rooted at the destination. MS attains loop–free paths by coordinating the next hop updates for each destination as a diffusion computation [6] which is started by the destination.

ExBF is another distance–vector algorithm which avoids long–lived loops but not short–lived loops. For each destination, every node maintains a distance and a prefinal node through each of its neighbors. A prefinal node is the last node before the destination on the next hop path towards that destination. Using the prefinal node information, ExBF avoids long–lived loops.

We point out that for all algorithms, link–costs are re–calculated not only in the case of a failure/repair but also periodically. This can cause routing table updates and the routing algorithm adapts if needed. These routing table updates in turn cause changes in the link costs. This delayed feedback between routing changes and link cost changes is very complicated to analyze and can only be understood via simulations/experiments.

4 Simulation Environment

In this section, we give a brief overview of MaRS. We then describe the range of parameters exercised in our simulations and the performance measures obtained. Assumptions and parameters were predetermined either consistently with those made in the literature, e.g. [19] [17] [12] [4], or from statistics provided by Merit/NSFNET Information Services, or by experimentation.

4.1 Overview of MaRS

Our simulation studies were done on a recently developed discrete–event simulator, MaRS (Maryland Routing Simulator). It provides a flexible platform for the evaluation and comparison of network routing algorithms. MaRS is implemented in C on a UNIX environment. It has an optional graphical (X Window System) interface. It is available publicly in the Internet by anonymous ftp from `ftp.cs.umd.edu` [1].

A network configuration consists of a *physical network*, a *routing algorithm* and a *workload*. The physical network consists of *nodes* and *links*. Each node i represents a store–and–forward entity, and is characterized by parameters such as buffer space, processing speed, packet queueing discipline, and failure and repair distributions. Each link (i, j) represents a transmission channel from node i to node j, and is characterized by parameters such as bandwidth, propagation delay, and failure and repair distributions. An operational link, i.e. one that is not in a failed state, behaves as an error–free FIFO channel.

The routing algorithm is implemented by *routing modules*, one at each node, which maintain the routing information, e.g. next hops, distances, link costs, etc. MaRS currently provides three routing algorithms, SPF, MS, ExBF (described in section 3). MaRS currently provides four types of link cost functions based on hop count, utilization, delay, and hop–normalized delay. The link cost function used in this paper is the hop normalized delay function of the ARPANET [19, 17]. The cost of a link is updated periodically, and reset whenever the link fails or recovers; at each update, a new cost is calculated based on the average packet delay for the link during the last update period, using exponential averaging, movement limits, etc. (see Appendix A for details).

The *workload* is defined in terms of *source–sink* pairs which are attached to nodes. In each pair i, source i produces packets destined for sink i and passes them to its node. The packets are forwarded across the network and consumed by sink i. MaRS currently implements three types of source–sink pairs: file transfer (FTP), remote login (TELNET), and a simple workload.

In FTP and TELNET, each source–sink pair is defined in terms of *connections*. A connection can be *finite* or *unending*. A finite connection is defined by a start time, inter–packet generation time, packet size, and total number of data packets to be produced. The connection ends when all data packets have been delivered to the sink. An unending connection is similar, except that the source never stops producing data packets (and the connection never ends). Each connection, whether finite or unending, incorporates a static send window–based flow control mechanism and an acknowledgment–with–retransmission mechanism using roundtrip time estimates. For each connection, the roundtrip time estimate is calculated with the help of token packets sent regularly. The roundtrip time estimate is calculated by exponentially averaging the roundtrip times of token packets with a factor of 0.5. Retransmissions can be due to node or link failures, buffer space limitation, or *nil* next hop. We only consider this simple flow control scheme to keep the interactions between routing and flow control algorithms minimal. With a more sophisticated flow control mechanism, such as slow–start in TCP, the interaction would be more complicated, and we believe it would be hard to draw conclusions about the performance of the routing algorithm itself.

4.2 Simulation Parameters

We consider the routing algorithms, SPF, MS and ExBF, on the topology of the NSFNET–T1–Backbone for varying workloads. Figure 1 illustrates the NSFNET topology. Link propagation delays in milliseconds are indicated. Also indicated is the node which will be a *hotspot* in some of our experiments.

Figure 1: NSFNET–Backbone: 14 nodes, 21 bidirectional links, average degree 3.

Physical Network

All links have the same bandwidth, 1.5Mbit/sec. There are no link or node failures. All nodes have adequate buffer space for buffering packets awaiting processing and forwarding. Workload packets can be processed (includes parsing the packet header, consulting the routing table, and adding the packet to the appropriate outgoing packet queue) in parallel. The processing time equals 1 msec. Routing packets have priority over workload packets in the nodes' outgoing packet queues.

Routing packets received at a node are processed sequentially. The size and processing time of routing packets depend on the algorithm. For SPF, a link–state packet sent by a node is $16 + 8e$ bytes long, where e is the number of neighbors of the node; it is processed in 6 msec. For MS, there are two types of packets: one conveying a distance, which is 20 bytes long and processed in 3 msec, and one carrying a request for initiating a new computation, which is 16 bytes long and processed in 2 msec. For ExBF, a distance–vector packet sent by a node is $24 + 12n$ bytes long, where n is the number of destinations for which a distance has been received; it is processed in 4.5 msec.

Link–Cost Function

The hop–normalized delay function is used with offset equal to 0, slope equal to 10, minimum cost limit equal to 1, maximum cost limit equal to 10, and movement limit equal to 1 (these terms are defined in Appendix A). The link–cost update period is uniformly distributed with mean 10 seconds and standard deviation 1 second. (These values were obtained from [19] [17], and experimentation [25].)

Workload

We use only FTP source–sink pairs (this is a major NSFNET application [16]). We consider two types of source–sink pair distributions: (i) *Uniform workload* with parameter U indicating the average number of FTP source–sink pairs between every two nodes; and (ii) *Hotspot–with–background workload* with parameters H and B, where H is the average number of FTP source–sink pairs from every node to the hotspot node, and B is the average number of FTP source–sink pairs between every pair of nodes excluding the hotspot node.

We consider two types of burstiness for the FTP connections: (i) *Dynamic workload*, where each source–sink pair is a succession of finite connections with average inter–start times for connections equal to 15 sec, average inter–packet generation time equal to 0 msec, and average number of packets per connection equal to 50 (thus, all packets of a connection are generated at the start of the connection); and (ii) *Static workload*, where each source–sink pair is a single unending connection that starts when the simulation begins and has inter–packet generation time equal to 150 msec. In both cases, the data packet size equals 512 bytes, the window size for flow control is 8 packets, and the acknowledgment packet size is 32 bytes.

4.3 Performance Measures

We consider average measures of throughput, delay and load. An *average measure* is based on statistics collected over a large *measurement interval*, which is the duration of the simulation except for an initial "startup interval" (to eliminate transient effects due to empty initial network). Thus:

- *Throughput.* Total number of data bytes acknowledged during the measurement interval divided by the length of the measurement interval.
- *Delay.* Total delay of all data packets acknowledged during the measurement interval divided by the number of data packets acknowledged in this interval, where *delay* of a data packet is defined to be the time difference between sending a packet for the first time and receiving the corresponding acknowledgment.
- *Data (Routing) Load.* Fraction of the network capacity, i.e. sum of all link capacities, used by workload (routing) packets during the measurement interval.

The length of the measurement interval is chosen long enough to ensure steady–state behavior. This behavior is detected when successively computed statistics stay within a specified percentage error bound.

5 Observations and Conclusions

We first consider the (steady–state) performance of the routing algorithms under dynamic workload, as the number of source-sink pairs is increased. Recall that each source–sink pair generates a succession of connections; the connection start times are separated by 15 sec on an average, and each connection generates an average of 50 packets at its start. Thus, the number of packets generated per second, also referred to as the **workload** in this section, grows linearly with the number of source-sink pairs.

We expect the throughput, delay and data load to be related to the workload in a manner typical of *open* queueing networks [18]. That is, the throughput should equal the workload as long as the workload is less than the system capacity (for workload higher than the system capacity the system is unstable). With increasing workload, the delay should increase at first slowly and then dramatically (with increasing rate) as the system becomes saturated. The data load should be proportional to the throughput divided by the system capacity.

Note that the delay is proportional to the average number of packets in the network, which is roughly proportional to the average number of active connections, since each active connection contributes roughly about a send window's worth to the packets in the network. As the workload is increased, connections take more time to finish.

We now present our simulation results for dynamic workload:

- **Dynamic uniform workload**. Figure 2 shows the number of active connections as a function of time. This illustrates the open nature of the system. Figure 3 shows the throughput, delay and data load, versus U in the range 1 to 6. Saturation begins around $U = 4$ (and instability around $U = 7$). The three routing algorithms differ only in the delay curves after onset of saturation. At $U = 6$, ExBF performs the best, MS is about 5% worse, and SPF is about 30% worse.

- **Dynamic hotspot–with–background workload**. Figure 4 shows the throughput, delay and data load, versus H in the range 4 to 15, for $B = 1$. Saturation begins around $H = 13$ (and instability around $H = 16$). Again, the three routing algorithms differ only in the delay curves after onset of saturation. However, this time SPF and ExBF perform about the same, while MS is about 30% worse.

  Figure 5 shows the corresponding curves for $B = 2$, and figure 6 for $B = 4$.

In the above figures, the throughput increases linearly with the workload, while the data load grows at first linearly and then slower. An explanation for this is that the routing algorithms become more efficient as the workload increases. That is, the average number of hops a packet travels becomes smaller (hence the slower increase in data load). This is not unlikely because the link costs (as indicated by the bounded discrete-valued link cost functions) become more representative of the link delays as the workload increases.

We now consider the (steady–state) performance of the routing algorithms under static workload, as the number of source-sink pairs is increased. Recall that in the static case each source–sink pair has a single unending connection that generates one packet every 150 msec on an average. Thus, as in the dynamic case, the *workload* grows linearly with the number of source-sink pairs. However, unlike the dynamic case, the number of active connections is constant over time. Thus the number of packets in the network is upper bounded, since each active connection cannot have more than a send window's worth of the packets in the network.

Therefore, we expect the throughput, delay and data load to be related to the workload in a manner typical of *closed* queueing networks [18]. That is, as the workload increases, the throughput (and data load) increase linearly at first and then slower (at which point packets are being generated at higher rate than they are being accepted into the network). The delay increases very slowly at first and then linearly. Therefore, the system should reach saturation at higher workload in the static case than in the dynamic case.

We now present our simulation results for static workload:

- **Static uniform workload**. Figure 7 shows the throughput, delay and data load, versus U in the range 0.5 to 6. Saturation begins around $U = 2.5$. The three routing algorithms differ only in the delay curves after onset of saturation. At $U = 3$, MS and ExBF perform about the same, and SPF is about 75% worse.

  To compare static workload with dynamic workload, we show in Figure 8 the delay versus throughput for the static uniform workload and for the dynamic uniform workload. (The static throughput range is truncated to be the same as the dynamic throughput range.) As expected, the dynamic workload has higher delay and enters saturation sooner.

- **Static hotspot–with–background workload**. Due to lack of space, we do not show graphs of throughput, delay and data load, versus H, for various B. Instead, we directly compare the delay–throughput curves for static and dynamic cases. Figure 9 shows the delay versus throughput graphs for static hotspot workload and for dynamic hotspot workload. Both cases have equal background workload corresponding to $B = 1$ for the dynamic case. As expected, the dynamic workload has higher delay and enters saturation sooner. Also, in the static case, around saturation SPF and ExBF perform about the same, while MS is about 30% worse.

  Figures 10 and 11 show the corresponding graphs for background workload of $B = 2$ and $B = 4$, respectively. Note that for fixed H, as B increases the delay decreases, because the proportion of hotspot workload decreases.

Our general conclusions are as follows. As expected, the network behaves as an open system under dynamic workload, and more or less as a closed system under static workload. In every scenario, the three routing algorithms are practically equivalent with respect to throughput and data load. Before the onset of saturation, they are also practically equivalent with respect to delay. However, they differ significantly in delay around the saturation point, although which algorithm is better depends on the geographic distribution of the workload. Under a hotspot–with–background workload, SPF and ExBF are about equivalent (with SPF being slightly better), while MS does significantly worse. Under uniform workload, MS and ExBF are about equivalent (with ExBF being slightly better), while SPF does significantly worse.

The explanation we have is the following: in general we have observed that SPF adapts faster in response to traffic burst and link failure/recovery. We think that this leads to oscillations in routes[3] which degrade performance under uniform workload [4], while under skewed workload, these route changes would cause the use of under–utilized links.

We give more importance to hotspot workload than to uniform workload, because we believe that it is more representative of real–life network conditions. In this case, in spite of short–lived loops, SPF and ExBF perform better than MS in terms of delay and throughput. This means that short–lived loops do not cause congestion. And the diffusion computation mechanism used

[3] We have observed that SPF has higher oscillations in instantaneous delay compared to ExBF and MS under uniform workload around saturation [25]. We have not checked for oscillations in routes.

by MS causes MS to adapt too slowly. (These conclusions held even in the high–speed network we studied in [25].)

ExBF has a worst–case exponential message complexity ($O(2^N)$), which is much higher than that of SPF ($O(E)$) and that of MS ($O(H \times E)$), where N is the number of network nodes, E is the number of network links, and H is the length of a maximum length shortest path between any two nodes. However, our simulations show that routing load is generally very small compared to the data load and a negligible fraction of the overall network capacity. We also do not find any correlation between routing load and system performance. Hence, reducing message complexity and routing load should not be a primary criteria in proposing new routing algorithms.

Our experiments were limited to only one network topology, that too of only 14 nodes. Experiments on more topologies are required to further analyze the routing algorithms, and to see how the performance scales with network parameters such as the number of network nodes, network diameter, average nodal degree, etc. Other factors such as memory requirements should also be considered when the network becomes large.

References

[1] C. Alaettinoğlu, K. Dussa-Zieger, I. Matta, and A. U. Shankar. MaRS (Maryland Routing Simulator) – Version 1.0 User's Manual. Technical Report UMIACS-TR-91-80, CS-TR-2687, Department of Computer Science, University of Maryland, College Park, MD 20742, June 1991.

[2] D. Bertsekas and R. Gallager. *Data Networks*, pages 297–333. Prentice-Hall, Inc., 1987.

[3] C. Cheng, R. Riley, S. P. R. Kumar, and J. J. Garcia-Luna-Aceves. A Loop-free Bellman-Ford Routing Protocol Without Bouncing Effect. In Proc. *ACM SIGCOMM '89*, pages 224–237, September 1989.

[4] Wushow Chou, Arnold Bragg, and Arne Nilsson. The Need for Adaptive Routing in the Chaotic and Unbalanced Traffic. *IEEE Transactions on Communications*, 1981.

[5] E. Dijkstra. A Note on Two Problems in Connection with Graphs. *Numer. Math.*, 1:269–271, 1959.

[6] E. Dijkstra and C. Scholten. Termination Detection for Diffusing Computations. *Information Processing Letters*, 11(1):1–4, 1980.

[7] A. Ephremides, P. Varaiya, and J. Walrand. A Simple Dynamic Routing Problem. *IEEE Transactions on Automatic Control*, 25(4):690–693, August 1980.

[8] A. Ephremides and S. Verdu. Control and Optimization Methods in Communication Network Problems. *IEEE Transactions on Automatic Control*, 34(9):930–942, September 1989.

[9] L. Ford and D. Fulkerson. *Flows in Networks*, pages 297–333. Prentice-Hall, Inc., 1962.

[10] R. Gallager. A Minimum Delay Routing Algorithm Using Distributed Computation. *IEEE Transactions on Communications*, COM-25:73–84, Jan 1977.

[11] J.J. Garcia-Luna-Aceves. A Unified Approach to Loop Free Routing Using Distance Vectors or Link States. In Proc. *ACM SIGCOMM '89*, pages 212–223, September 1989.

[12] S. A. Heimlich. Traffic Characterization of the NSFNET National Backbone. In Proc. *USENIX '90*, pages 207–227, Washington, D.C., January 1990.

[13] P.A. Humblet and S.R. Soloway. Algorithms for Data Communication Networks–Part 1. Technical report, Codex Corp., 1986.

[14] P.A. Humblet, S.R. Soloway, and B. Steinka. Algorithms for Data Communication Networks–Part 2. Technical report, Codex Corp., 1986.

[15] J. M. Jaffe and F.H. Moss. A Responsive Distributed Routing Algorithm for Computer Networks. *IEEE Transactions on Communications*, COM-30(7):1758–1762, July 1982.

[16] Dale Johnson. NSFnet Report. In Proc. *of the Nineteenth Internet Engineering Task Force*, pages 377–382, University of Colorado, National Center for Atmospheric Research, December 1990.

[17] A. Khanna and J. Zinky. A Revised ARPANET Routing Metric. In Proc. *ACM SIGCOMM '89*, pages 45–56, September 1989.

[18] L. Kleinrock. *Queueing Systems, Volume II: Computer Applications.* John Wiley and Sons, Inc., 1976.

[19] J. M. McQuillan, I. Richer, and E. C. Rosen. The New Routing Algorithm for the ARPANET. *IEEE Transactions on Communications*, COM-28(5):711–719, May 1980.

[20] P. M. Merlin and A. Segall. A Failsafe Distributed Routing Protocol. *IEEE Transactions on Communications*, COM-27(9):1280–1287, September 1979.

[21] B. P. Mohanty, C. G. Cassandras, and D. Towsley. Performance Comparison of Routing Algorithms in Packet Switched Networks. In Proc. *IEEE Globecom '90*, San Diego, California, 1990.

[22] D. J. Nelson, K. Sayood, and H. Chang. An Extended Least-Hop Distributed Routing Algorithm. *IEEE Transactions on Communications*, 38(4):520–528, April 1990.

[23] B. Rajagopalan and M. Faiman. A New Responsive Distributed Shortest-Path Routing Algorithm. In Proc. *ACM SIGCOMM '89*, pages 237–246, September 1989.

[24] A. Segal. Advances in Verifiable Fail-Safe Routing Procedures. *IEEE Transactions on Communications*, COM-29(4):491–497, April 1981.

[25] A. Udaya Shankar, Cengiz Alaettinoğlu, Ibrahim Matta, and Klaudia Dussa-Zieger. Performance Comparison of Routing Protocols using MaRS: Distance-Vector versus Link-State. In Proc. *ACM SIGMETRICS and PERFORMANCE*, pages 181–192, Newport, Rhode Island, June 1992.

[26] W.T. Tsai, C. Ramamoorthy, W.K. Tsai, and O. Nishiguchi. An Adaptive Hierarchical Routing Protocol. *IEEE Transactions on Computers*, 38:1059–1075, August 1989.

[27] Z. Wang and J. Crowcroft. Shortest Path First with Emergency Exits. In Proc. *ACM SIGCOMM '90*, pages 166–176, September 1990.

[28] W. Zaumen and J.J. Garcia-Luna-Aceves. Dynamics of Distributed Shortest-path Routing Algorithms. In Proc. *ACM SIGCOMM '91*, September 1991.

A Link Cost Function

The cost of a link[4] is maintained by its transmitting node. The cost is updated at time instants $t_0, t_1, \ldots$, where for all $i > 0$, either $t_i - t_{i-1}$ equals an update interval size T, or at time t_i the link fails or recovers. To compute the cost, the transmitting node maintains the following variables (the comments apply just before t_i):

- $RawCost$: A real-valued statistic reflecting the delay over interval $[t_{i-1}, t_i)$, e.g. utilization, delay, etc.
- $AverageRawCost$: A real-valued statistic. The exponential average of the raw cost over time interval $[t_k, t_{i-1})$, where t_k is the time of the last repair of the link.
- $LinkCost$: $\{MinLimit, \ldots, MaxLimit\} \cup \{\infty\}$. Link cost calculated at time t_{i-1}.
- $MovementLimit$: An integer-valued parameter. Maximum possible change of the link-cost in one update period.
- $Slope, Offset$: Real–valued parameters. Characterize the shape of the function mapping $AvgRawCost$ to $NormRawCost$.

The new link cost at time t_i is computed as follows:

- If the link fails, $LinkCost$ is set to ∞.

[4] The cost of a link is usually discrete–valued instead of real–valued because it can be encoded in fewer bits.

- If the link becomes operational, $LinkCost$ is set to $MinLimit$, and $RawCost$ and $AverageRawCost$ are set to their minimum values.
- Otherwise, the link cost is calculated as follows:
  $AverageRawCost := 0.5 \times (RawCost + AverageRawCost)$
  $NormalizedRawCost := max(min(AverageRawCost \times Slope + Offset,\ MaxLimit),\ MinLimit)$
  if $|NormalizedRawCost - LinkCost| > MovementLimit$ then
  $LinkCost := LinkCost + MovementLimit \times sign(NormalizedRawCost - LinkCost)$
  else $LinkCost := NormalizedRawCost$
  The first line does exponential averaging, the second line bounds the cost, and the remaining lines bound the change in the cost.

The difference between different link cost functions lies in how they compute the $RawCost$. In this paper, we use the hop normalized delay function of the ARPANET [17]. Here the transmitting node monitors the *average packet delay* (queueing and transmission) and *average packet transmission time* for the link during the last update period. From these, assuming an $M/M/1$ model, it calculates the utilization, which it uses as the raw cost. That is:

$$RawCost(=\ utilization) = 1 - \frac{average\ packet\ transmission\ time}{average\ packet\ delay}$$

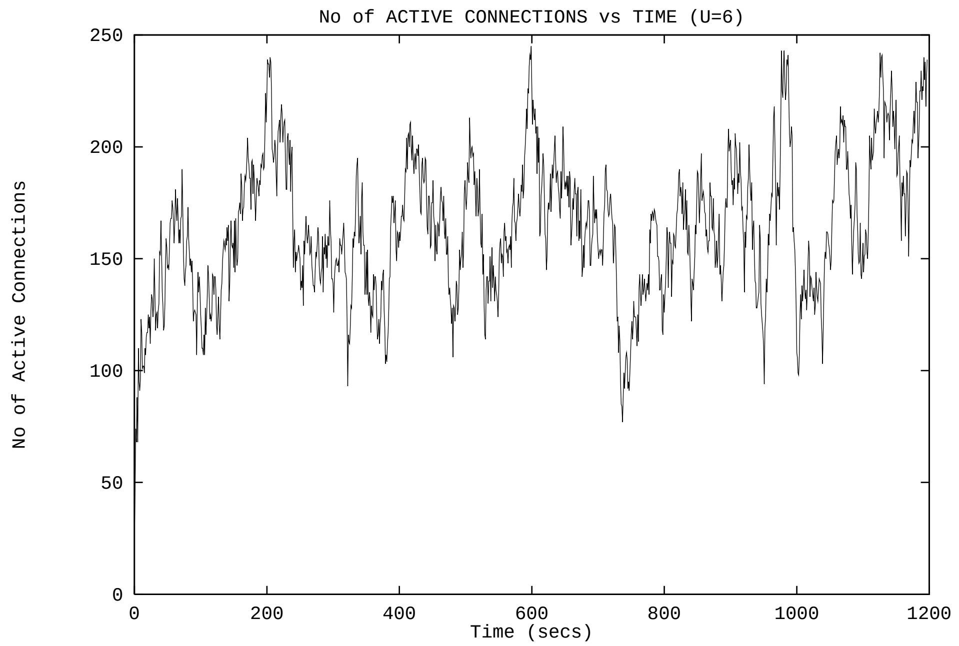

Figure 2: Number of active connections vs time under dynamic uniform workload.

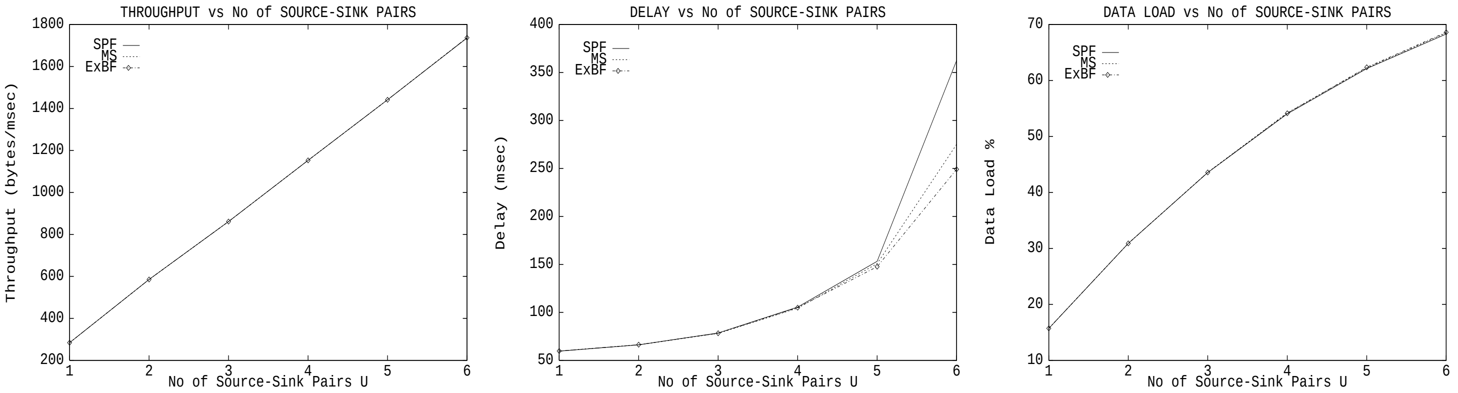

Figure 3: Varying the average number of source–sink pairs in dynamic uniform workload.

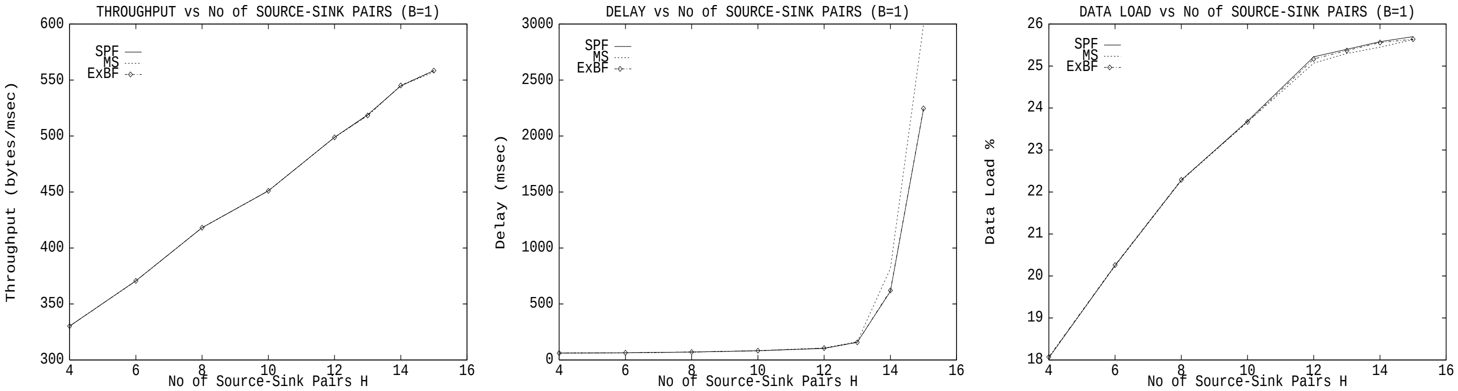

Figure 4: Varying the average number of hotspot source–sink pairs for background $B = 1$.

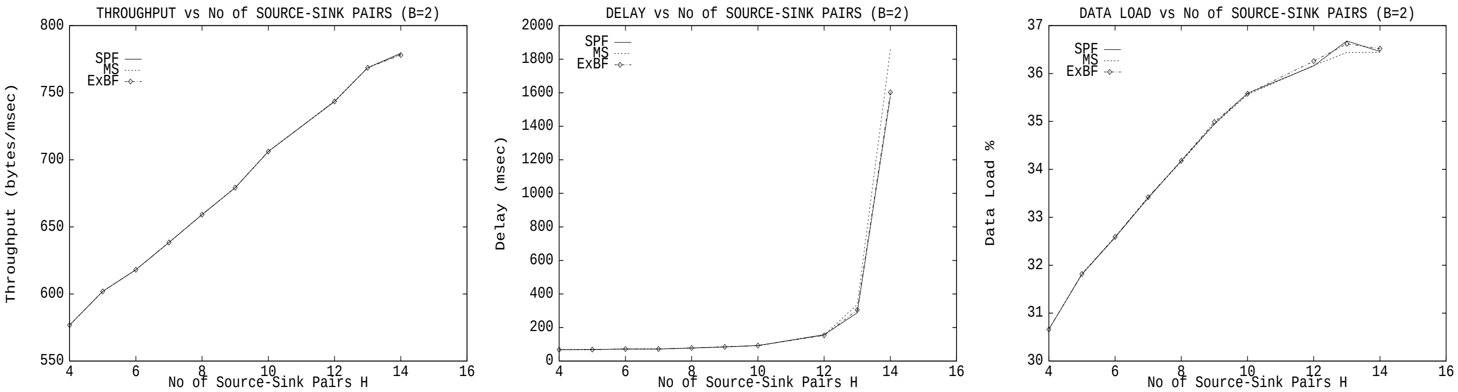

Figure 5: Varying the average number of hotspot source–sink pairs for background $B = 2$.

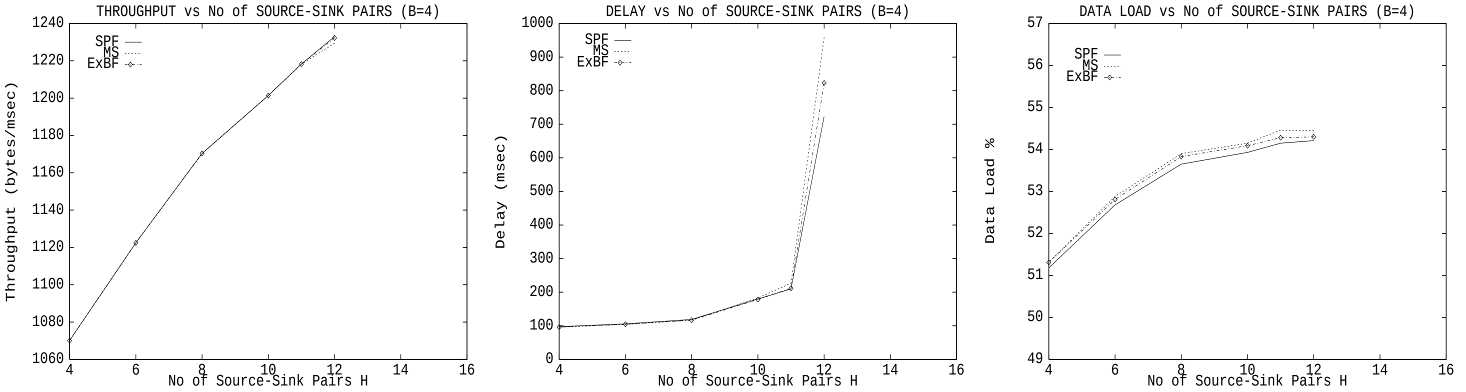

Figure 6: Varying the average number of hotspot source–sink pairs for background $B = 4$.

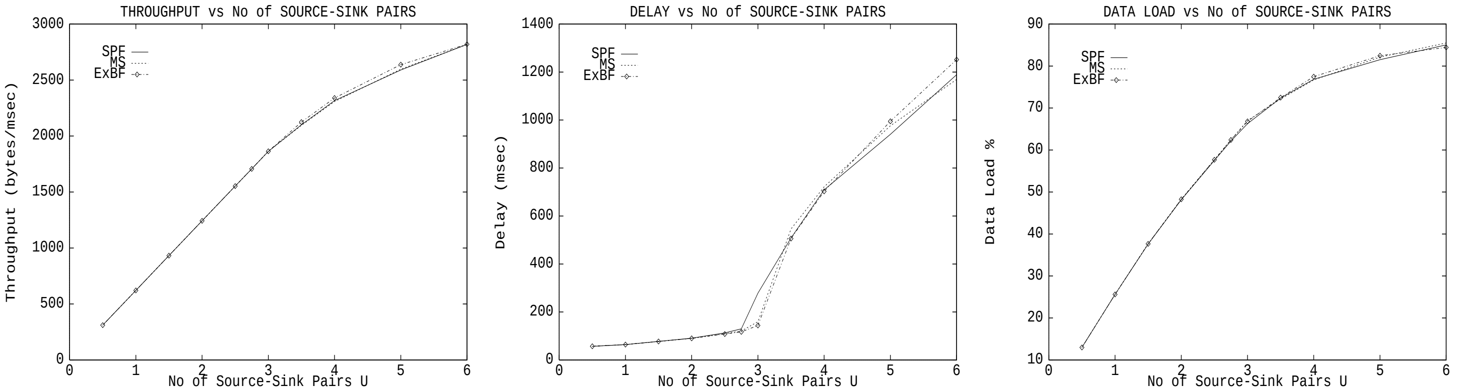

Figure 7: Varying the average number of source–sink pairs in static uniform workload.

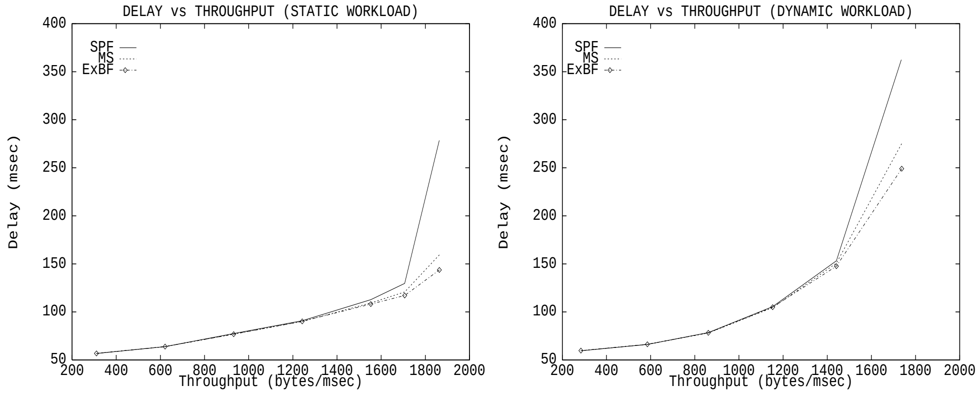

Figure 8: Comparing static uniform workload vs dynamic uniform workload.

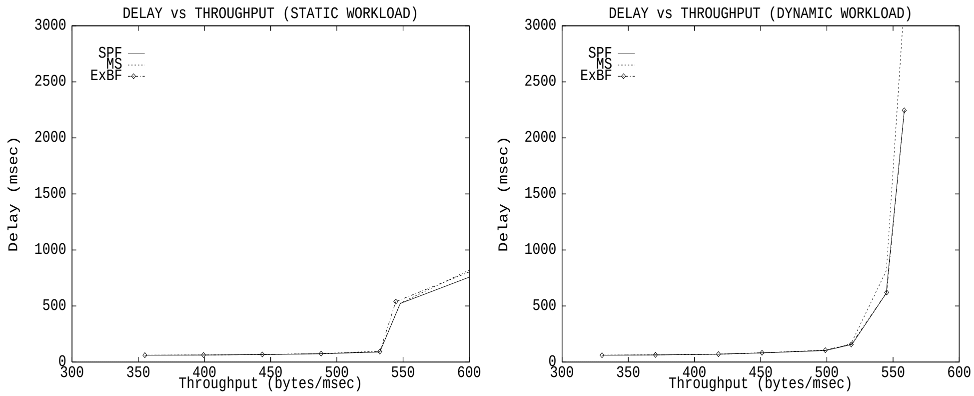

Figure 9: Comparing static hotspot workload vs dynamic hotspot workload for background $B = 1$.

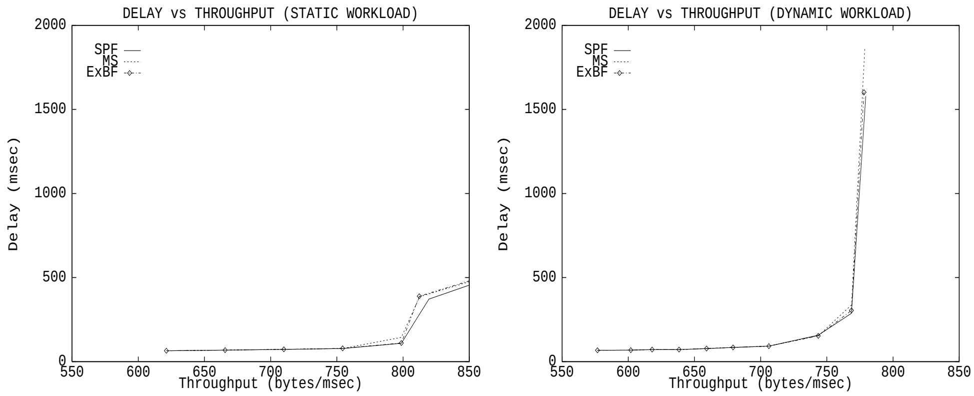

Figure 10: Comparing static hotspot workload vs dynamic hotspot workload for background $B = 2$.

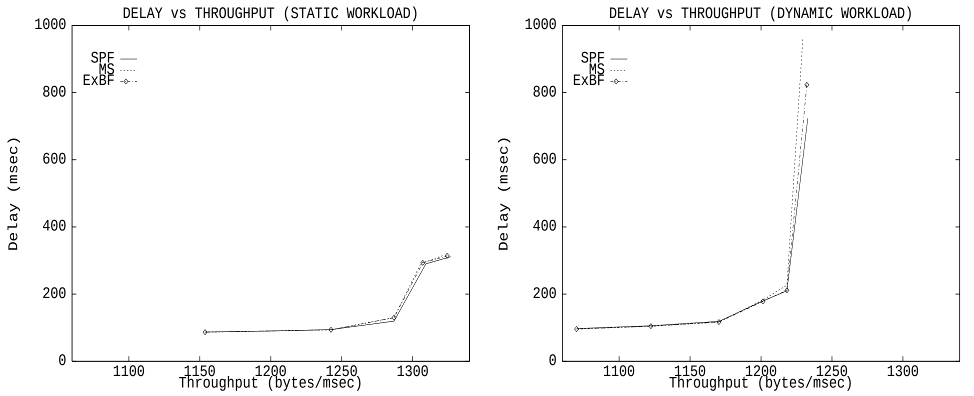

Figure 11: Comparing static hotspot workload vs dynamic hotspot workload for background $B = 4$.