Transient and Steady-State Performance of Routing Protocols: Distance-Vector versus Link-State*

A. Udaya Shankar†, Cengiz Alaettinoğlu, Klaudia Dussa-Zieger, Ibrahim Matta
Institute for Advanced Computer Studies and
Department of Computer Science
University of Maryland
College Park, Maryland 20742

UMIACS-TR-92-87
CS-TR-2940

August 1992

Abstract

We examine two approaches to adaptive routing protocols for wide-area store-and-forward networks, namely, distance-vector and link-state. Distance-vector algorithms have less storage requirements than link-state algorithms. The ARPANET started with a distance-vector algorithm (Distributed Bellman-Ford), but because of long-lived loops, changed to a link-state algorithm (SPF). We evaluate, using a recently developed network simulator, MaRS, the transient and steady-state performance of SPF and two newly proposed distance-vector algorithms (ExBF and MS). Overall, SPF and ExBF have comparable performance and MS is worse.

Categories and Subject Descriptors: C.2.1 [Computer-Communication Networks]: Network Architecture and Design—*packet networks; store and forward networks*; C.2.2 [Computer-Communication Networks]: Network Protocols—*protocol architecture*; C.2.m [Routing Protocols]; C.4 [Performance of Systems]: *measurement techniques; modeling techniques; performance attributes;* F.2.m [Computer Network Routing Protocols]; I.6.3 [Simulation and Modeling]: Applications.

*This work is supported in part by RomeLabs 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.

Contents

1 Introduction

Routing protocols are among the most important protocols in wide-area store-and-forward computer networks. They are responsible for forwarding data packets from source nodes to destination nodes 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 time-varying external load. Consequently, a routing protocol must monitor link delays and adapt its routes to changes in link delays.

In this paper, we are concerned with next-hop routing. Each node maintains for each destination a neighbor node id, referred to as the **next-hop**. Each data packet has its destination node id in its header. When a node receives a data packet, it forwards the packet to the next-hop for its destination. The objective of the routing protocol is to choose the next-hops so that the resulting routes are optimal.

Next-hop routing is a simple and robust way to do adaptive routing. A routing protocol does the following functions: (a) each node maintains for each of its outgoing links a dynamic **link cost** which is updated regularly according to the traffic flowing through the link; (b) link cost information is disseminated regularly to nodes of the network; (c) based on received link cost information, each node maintains and regularly updates a next-hop for each destination.

The dynamics and performance of such systems can be very complex due to the distributed nature of the network and the delayed feedback between route changes and link cost changes. Let us consider a simplified situation where the link cost updates at different nodes are synchronized. Let $t_1, t_2, ...$ be the times at which the nodes of the network update their outgoing link costs. Between successive updates, nodes exchange their link cost information and adjust their next-hops. Note that the link cost information exchanged between nodes during (t_i, t_{i+1}) is based on the new link costs at t_i and reflects the network traffic pattern before t_i. Assume that the user workload between some source-destination pairs changes during (t_{i-1}, t_i). Then the routes at t_i are not optimal. The next-hops are adjusted during (t_i, t_{i+1}) to achieve optimal routes using the new link costs at t_i. But changing the routes changes the network traffic pattern which results in different link costs at t_{i+1}. Thus, it is unlikely that the new routes at t_{i+1} are optimal with respect to the new link costs at time t_{i+1}. Hence, the process is repeated, yielding new routes and new link costs at time t_{i+2}, t_{i+3}, and so on.

Assuming that the user workload does not change, the routes converge to an optimal configuration with respect to the link costs, and then in general oscillate[1] around the optimal. The rate of convergence and the magnitude of oscillations depend on how rapidly the nodes attempt to adapt and how old their link cost information is at each update interval. Oscillations in routes give rise to oscillations in link costs and end-to-end user performance. Ideally, we want routes to adapt quickly, and we want the oscillations to have small magnitude[2] and be centered around an optimal steady-state. But in practice one has to trade off between responsiveness and steady-state performance.

Clearly, the link cost function must reflect the performance measure of interest to the user, such as delay. However, because packet-switched traffic is inherently bursty, a link's new cost at time t_i cannot be based only on the instantaneous traffic at t_i. For stability, it must involve some averaging over the interval (t_{i-1}, t_i). And this is why the performance depends on the routing algorithm used to disseminate link cost information and adjust next-hops. Although different routing algorithms may achieve the same next-hops at t_{i+1} starting from the same costs at t_i, they will differ in the order and the times at which the next-hops are updated. Therefore the new link costs at t_{i+1} depend on the routing algorithm. (One way to become independent of the routing algorithm is to have $t_{i+1} - t_i$ much larger that the time needed by the routing algorithm. But this is not feasible in general because node updates are asynchronous. Also, having large $t_{i+1} - t_i$ makes the routing protocol unresponsive.)

There are two basic approaches to next-hop routing algorithms: **link-state** and **distance-vector**. The link-state approach is a straight-forward brute-force 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. Each node requires $O(N^2)$ space in order to store the view of the entire network topology, where N is the number of nodes in the network[3]. The dissemination time is small and proportional to D where D is the network diameter in hops.

The distance-vector approach, which is based on the Bellman-Ford algorithm [12], is more subtle: each node maintains for each destination a set of distances, one for each of its neighbors.

[1]Oscillations are caused by the delayed feedback between route changes and link cost changes. For example, suppose the network has two similar paths between a source and a destination. If traffic is routed on one path during an update interval, the cost of the other path will be less; thus, the other path will be chosen in the next update interval.

[2] The latter is because performance measures, such as delay, usually degrade faster than linearly with increasing traffic.

[3] Using adjacency list representation, this storage can be reduced to $O(N \times e)$, where e is the average degree of a node. However, this increases the processing time for determining shortest routes. And in a high-speed network, it is the node processing time that is often the bottleneck.

It routes packets through a neighbor with the minimum distance. Each node requires $O(N \times e)$ space, where e is the average degree of a node, a substantial saving over the link-state approach[4]. However it is well known that the straight-forward distributed implementation of the Bellman-Ford algorithm can have long-lived loops giving rise to large dissemination times, i.e. of the order of distances [3].

The ARPANET initially used this Distributed Bellman-Ford algorithm. Because of its long dissemination time, it was replaced in 1979 by a link state algorithm referred to as SPF (Shortest Path First) [24]. At that time, the link cost function was link delay. Experience showed that this leads to unstable oscillations, and in 1987 it was replaced by a more slowly changing hop-normalized-delay function, which uses exponential averaging, movement limit, etc [21].

Since 1979, many new kinds of distance-vector algorithms have been proposed [25, 30, 19, 6, 15, 29] which achieve significantly reduced dissemination times by using node coordination mechanisms. The Merlin-Segall algorithm [25, 30] has a worst-case dissemination time of $O(H^2)$ hop count, where H is the length of a maximum length shortest path between any two nodes. The worst-case dissemination time is $O(H)$ for the Jaffe-Mosse algorithm [19] and the DUAL algorithm by Garcia-Luna-Aceves [15], and is $O(N)$ for the Extended Bellman-Ford algorithm by Cheng *et al.* [6]. However, in terms of real time they may have successively smaller dissemination times because they impose successively weaker coordination constraints.

The obvious question is: Are the new distance-vector algorithms good enough to replace SPF? To answer this, we must compare these algorithms in the context of varying data packet workloads and link cost functions. To ignore the workload and link cost function, as is done in all hop-count and message complexity analyses, is to ignore the crucial dynamics of the delayed feedback. Thus, a related question is: How critical is the link cost function?

We examine both these questions in this paper using simulation studies. Given the current understanding of next-hop routing algorithms, we feel that simulations are probably the only way to develop further insight into their performance. We use a recently developed simulator called MaRS (Maryland Routing Simulator) [2]. We consider three algorithms: SPF; Merlin-Segall, henceforth called MS; and Extended Bellman-Ford, henceforth called ExBF. We consider the NSFNET-T1-backbone network with T1 hardware as well as a high-speed hardware, with varying uniform and hotspot-with-background workloads, and varying failure patterns. We examine both steady-state

[4] Usually a node broadcasts its minimum distance only if there is a change. However, if this broadcast is done periodically whether or not there is a change, this storage can be reduced to $O(N)$ at the expense of increasing dissemination time.

and transient behaviors.

Our overall conclusion is as follows. Regarding steady-state performance, SPF performs well under hotspot-with-background workload and failures. ExBF performs well under both uniform workload and hotspot-with-background workload, and is the least sensitive to changes in the link cost function. MS performs well under uniform workload. Regarding transient performance, SPF performs the best in general.

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 the link cost function. In section 5, we give a brief overview of MaRS, and describe the workload. In section 6, we describe the simulation parameters, and in section 7, the performance measures. In section 8, we present our observations about the simulation results. We give our conclusions in section 9. Details of the simulated scenarios and plots are given in Appendix I.

2 Related 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 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) [24] and its new link cost function (i.e. hop-normalized-delay function) [21]. The tests showed that the new algorithm and link cost function give better performance than the old ARPANET algorithm (i.e. Distributed Bellman-Ford) and the old delay function.

Regarding simulation approaches, reference [7] 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 [35], 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. In [36], an extension to SPF was proposed where alternate paths are used in case of congestion and failures. It was concluded that the proposed algorithm gives higher throughput than SPF. In [27], multi-path extensions to SPF were shown to perform better. Here, each node finds the min hop and min plus 1 hop paths for every destination, estimates their delays and chooses between them accordingly. Reference [37] 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 [26] compared SPF, Gallager's algorithm [13], and the CODEX algorithm [17, 18]. 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.

The obvious analytical approach is to use a queueing network model where routing is based on delayed state information. Unfortunately, this is usually intractable. The usual approach is to ignore the delay (and averaging) in the feedback, leading to queueing control models formulated as Markovian decision process problems [10, 11]. Even then only simple topologies can be considered. For example, reference [10] considers a queueing system with two identical independent exponential servers and determines an optimal way to route incoming packets to one of the two servers based on their "instantaneous" queue sizes.

Imposing more assumptions results in more tractable *flow models* [3, 11]. Here, each link is modeled by a capacity and propagation time, and the message traffic on the link is modeled by a continuous-valued flow, representing the average rate of the message traffic. The routing problem is then formulated as an optimization problem. In general, some cost function $D(f_{ik})$ is optimized with respect to variables f_{ik}, where f_{ik} indicates the flow on link (i, k) and is subject to constraints such as conservation of flow and nonnegativity of flows [5]. Typically, the solution procedure is iterative. Flow models have been used to study both link-state algorithms [4] and distance-vector algorithms [14]. A serious limitation of flow models is the assumption that flows are static (or quasi-static) [3].

3 Overview of Routing Algorithms

This section describes the routing algorithms examined in our simulations. All of them use next-hop routing. At a node, the next-hop for a destination can be *nil* (e.g. because the destination is not reachable from the node). In this case if the node receives a data packet for that destination, the packet is dropped[5].

[5]It will eventually be 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. To keep these views up-to-date, each node regularly broadcasts the link costs of its outgoing links to all other nodes using a protocol such as flooding (i.e. it sends link cost information to its neighbors, they forward this information to their neighbors and so on). As a node receives this information, it updates its view of the network topology and applies a shortest path algorithm to choose its next-hop for each destination. 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 where (1) each node calculates and broadcasts the costs of its outgoing links periodically or whenever a failure/repair occurs, and (2) Dijkstra's shortest path algorithm [8] is applied to the view of the network topology to determine next-hops.

Distance-Vector Algorithms

In the distance-vector approach, for each destination d, every node i maintains a set of distances $\{D_i(j)\}$ where j ranges over the neighbors of i. Node i treats neighbor k as a next-hop for a packet destined for d if $D_i(k)$ equals $\min_j\{D_i(j)\}$. The quantity $\min_j\{D_i(j)\}$ is referred to as the minimum distance to d. The intention is that from any node i the succession of next-hops should lead to d; furthermore $D_i(j)$ should equal the sum of the costs of the links on the path from i to d where the first hop is to j, and the rest of the path is the succession of the next-hops. To keep these distances up-to-date, each node regularly updates the cost of its outgoing links, and hence its distances and next-hops. If this causes a change in a minimum distance, it sends the new minimum distance to its neighbors; if as a result a minimum distance of a neighbor changes, it repeats the process.

The above distance-vector algorithm is the classical Distributed Bellman-Ford algorithm [3, 12]. It is well known that this algorithm, in addition to short-lived next-hop loops, can have **long-lived** loops of duration proportional to link cost changes [29]. 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 [9] which is started by the destination. When a failure/repair occurs, a request message is sent to each destination. Periodically or upon receiving a request message, the destination starts a new diffusion computation. (The diffusion computation is as follows. The destination node starts the computation by sending the distance zero to all of its neighbors, and waits for distance messages from all its neighbors, at which point the computation is over. Every node i, other than the destination, does the following: upon receiving a distance message for the destination from its next-hop, it calculates its distance, and sends its distance to all of its neighbors except its next hop. After receiving distance messages from all its neighbors, node i chooses its new next-hop, and sends the new distance to its old next-hop. If node i's next-hop is nil, it chooses the first neighbor that reports a finite distance as its next-hop.)

ExBF is another distance-vector algorithm which avoids long-lived loops but not short-lived loops. For each destination d, every node i maintains, in addition to the set of distances $\{D_i(j)\}$, a set of prefinal nodes $\{P_i(j)\}$, where j ranges over the neighbors of i. The intention is that $P_i(j)$ is the last node before d on the next-hop path from i to d through j. Because node i has the prefinal node for every destination, it can obtain a complete path $s_0, s_1, \ldots s_{k-1}, s_k$ to d via j, where $s_0 = j$, $s_k = d$, and for all l, s_l is the prefinal node for destination s_{l+1}. Furthermore, it turns out that after short-lived inconsistencies die out, this "prefinal node" path equals the "next-hop" path, and $D_i(j)$ is its cost. ExBF avoids long-lived loops by the following trick: When node i disseminates distance changes for destination d, it avoids sending to any neighbors in the path $s_0, s_1, \ldots, s_k$, where s_0 is i's next hop to d. Periodically or whenever a failure/repair occurs, link costs are re-calculated and if there is a change in the cost of its outgoing links, a node starts a computation in which its distances are updated to reflect that change.

4 Link Cost Function

The cost of a link[6] 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 period P, 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):

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

- $RawCost$: A real-valued statistic reflecting the delay over interval $[t_{i-1}, t_i)$, e.g. utilization, delay, etc.
- $AvgRawCost$: 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.
- $NormalizedRawCost$: An real-valued statistic. The normalized value of $AvgRawCost$.
- $LinkCost$: $\{MinLimit, \ldots, MaxLimit\} \cup \{\infty\}$. Link cost calculated at time t_{i-1}.
- $MovementLimit$: An integer-valued parameter. Maximum possible change of the link cost in one update period.
- $Slope, Offset$: Integer-valued parameters. Characterize the shape of the function mapping $AvgRawCost$ to $NormalizedRawCost$.

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

- If the link fails, $LinkCost$ is set to ∞.
- If the link becomes operational, $LinkCost$ is set to $MinLimit$, and $RawCost$ and $AvgRawCost$ are set to their minimum values.
- Otherwise, the link cost is calculated as follows:

```
$AvgRawCost := 0.5 \times (RawCost + AvgRawCost)$
$NormalizedRawCost := max(min(AvgRawCost \times Slope + Offset,\ MaxLimit),\ MinLimit)$
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 [21]. 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:

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

5 Overview of MaRS

Our simulation studies were done with a recently developed discrete-event simulator, MaRS (Maryland Routing Simulator). MaRS provides a flexible platform for the evaluation and comparison of network routing algorithms. It allows the user to define a network configuration, control its simulation, log the values of selected parameters, and save, load and modify network configurations. The scheduling of events can be done using random number generators, a trace file, or both. 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` [2].

A network configuration consists of a physical network, a routing algorithm, and a workload. These are described in the following paragraphs.

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, 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), and 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 [24, 21] (described in section 4).

The *workload* is defined in terms of connections. Each connection is associated with a source node and a sink node. The source produces packets destined for the sink. The packets are forwarded across the network and consumed by the sink. MaRS currently implements three types of connections: file transfer (FTP), remote login (TELNET), and a simple workload. In this paper, we use FTP connections. An FTP connection is characterized by a start time, inter-packet generation time, and packet size. The connection is unending, i.e. the source never stops producing data packets[7].

Each FTP connection incorporates a static send-window flow control mechanism and an acknowledgment-with-retransmission mechanism. Retransmissions are needed because packets can

[7] MaRS also allows an FTP connection to be characterized by the total number of data packets to be produced, in which case the connection ends when all data packets have been delivered to the sink.

be lost due to node or link failures, buffer space limitation, or *nil* next-hop. A flow control scheme is needed because otherwise the routing system may be subjected to unrealistic workloads[8]. A roundtrip time estimate (for retransmissions) is calculated by regularly sending token packets and exponentially averaging the roundtrip times of the token packets with a factor of 0.5.

Because of flow control, it is possible for a source to produce packets that cannot be introduced immediately into the send window. Such packets are temporarily buffered in a "produce window".

6 Simulation Parameters

In this section, we describe the range of parameters exercised in our simulations. Assumptions and parameters were predetermined either consistently with those made in the literature, e.g. [24] [21] [16] [7], or from statistics provided by Merit/NSFNET Information Services, or by simulations with MaRS. We consider the routing algorithms, SPF, MS and ExBF, on the topology of the NSFNET-T1-Backbone, for varying workloads, varying link cost functions, varying link failures, and for two types of hardware – a high-speed network and a low-speed network (with NSFNET-T1 parameters). 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 simulations, and an incident link which will be used in our transient failure simulations.

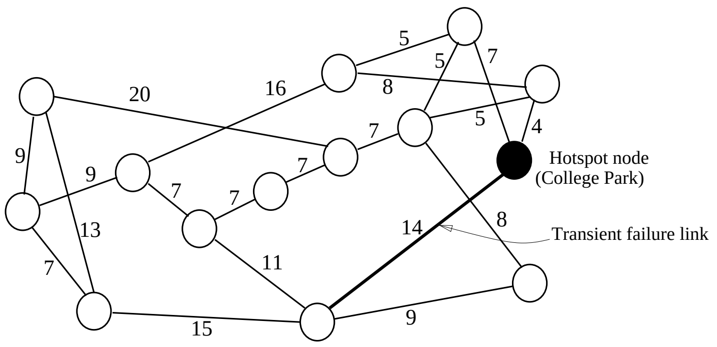

Figure 1: NSFNET-T1-Backbone (14 nodes, 21 bidirectional links, average degree 3).

[8] We have used a simple flow control scheme in an attempt to minimize the interactions between the routing and flow control algorithms. With a more sophisticated flow control mechanism, such as slow-start in TCP [38, 28], the interaction would be more complicated, and we believe it would be hard to draw conclusions about the performance of the routing algorithm itself.

Physical Network

All links have the same bandwidth, 1.5Mbit/sec for the low-speed case, and 100Mbit/sec for the high-speed case. All nodes have adequate buffer space for buffering packets awaiting processing and forwarding. Workload packets[9] 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 in the low-speed case and 400 μsec in the high-speed case. 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 $Offset = 0$, $Slope = 10$, $MinLimit = 1$, $MaxLimit = 10$, and $MovementLimit = 1$. The link-cost update period is uniformly distributed with mean 10 seconds and standard deviation 1 second. In some simulations, we vary the movement limit, slope, and the link-cost update period.

Workload

We use only FTP connections (file tranfers constitute a major NSFNET application [20]). We consider two types of source-sink pair distributions:

- *Uniform workload* with parameter U indicating the average number of FTP connections between every two nodes, and

- *Hotspot-with-background workload* with parameters H and B, where H is the average number of FTP connections from every node to the hotspot node, and B is the average number of FTP connections between every pair of nodes excluding the hotspot node.

For the low-speed network, each connection has data packet size equal to 512 bytes, inter-packet generation time equal to 150 msec, and window size for flow control equal to 8 packets. For the

[9] Workload packets include data, acknowledgment and token packets.

high-speed network, data packet size equals 5000 bytes, inter-packet generation time equals 20 msec, and window size equals 200 packets.

In simulations examining *steady-state performance*, all connections start when the simulation begins. In simulations examining *transient performance*, we start with a uniform workload and apply a step increase to the uniform workload at 150 seconds, i.e. the number of connections increases at time 150 seconds. In all cases, the connections are unending.

Failure Model

We consider link failures under uniform workload and hotspot-with-background workload. In the uniform workload case, every link in the network is subject to failure. In the hotspot-with-background workload case, only the links incident to the hotspot node are subject to failure.

In simulations examining steady-state performance, the failures of a link are characterized by an average *uptime* and an average *downtime*; the uptimes and downtimes are independent and exponentially distributed. All links in the network which are subject to failure have the same average uptime and average downtime. The failures of the network are characterized by two dependent parameters, the *up+down time*, and the *failure intensity*. *Up+down time* is the sum of the average link uptime and average link downtime; thus, a small *up+down time* means a high frequency of topology changes. *Failure intensity* is the average number of failed links in the network at any instant. These dependent parameters are varied by varying the average link uptime and average link downtime. For example, if all 21 links in the network are subject to failure, then an average link uptime of 4 seconds and an average link downtime of 3 seconds yield *up+down time* equal to 7 $(= 4 + 3)$ and *failure intensity* equal to 9 $(= (3/7) \times 21)$.

In simulations examining transient performance, we use hotspot-with-background workload and subject only one link to failure (the transient failure link indicated in Figure 1). It is failed at 100 seconds and repaired at 200 seconds.

7 Performance Measures

For steady-state behavior, we consider performance measures of throughput, delay, data load and routing load. A *steady-state 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 acknowledgement.
- *Data (Routing) Load.* Fraction of the network capacity used by workload (routing) packets during the measurement interval. (The network capacity is the sum of all link capacities.)

Our transient performance measures are overshoots and recovery times of various instantaneous measures after a step change in the workload, or a link failure, or a link repair. An *instantaneous measure* is a function of time updated periodically every 500 msec. An instantaneous measure at time t is based on statistics collected during the 500 msec period preceding t. We consider the following instantaneous measures:

- *Instantaneous Throughput at time t.* Total number of data bytes acknowledged in the period preceding t divided by the length of the period.
- *Instantaneous Delay at time t.* Total delay of data packets acknowledged in the period preceding t divided by the number of data packets acknowledged in this period.
- *Instantaneous Data (Routing) Load at time t.* Fraction of the network capacity used by workload (routing) packets in the period preceding t.

We define *recovery time* of an instantaneous measure to be the time needed for the measure to reach its steady-state value after a step change in the workload, or a link failure, or a link repair. The overshoot (undershoot) of an instantaneous measure is its maximum (minimum) value above (below) the steady-state value during the recovery time.

8 Observations

In this section, we present general observations about the simulation results. A brief summary appears in Table 1. Detailed descriptions of the scenarios simulated (parameters settings, etc.) and plots of the observed performance measures are given in Appendix I.

In every scenario examining steady-state performance, the number of packets in the network is upper bounded because each connection can have at most one send window's worth of packets in the network. Therefore, as the workload increases, the network behaves more as a closed queueing network [22]. That is, the data load and throughput first increase linearly, and then level off as the system becomes saturated. The delay increases very slowly at first, and then very rapidly after the saturation point.

Note that at high workload when the system is saturated, packets are being produced at a higher rate than they are admitted into the network because of flow control. Thus, the produce window of some sources is expanding without bound. However, the network itself (including the send windows) reaches steady-state.

Our observations are grouped into six groups (A) through (F). Except for group (E), all groups are for the low-speed network.

(A) *Varying workload with no failures.*

$(A.1)$ *Steady-state measures versus varying workload.*

The three routing algorithms are practically equivalent with respect to data load and throughput. Far from the saturation point, the algorithms are also practically equivalent with respect to delay. However, they can differ in delay around the saturation point. (Hence in many scenarios, we show delay plots only.) Under uniform workload, all three algorithms are about equivalent except that SPF is much worse near the saturation point (has higher delay by about 75% in the low-speed case and 1500% in the high-speed case). Under hotspot-with-background workload, SPF and ExBF are about equivalent in delay (with SPF being slightly better), while MS does significantly worse over a large range surrounding the saturation point.

The explanation we have is the following: in general we have observed that SPF recovers faster in response to step changes in the workload and link failure/repair. We think that this leads to oscillations in routes[10] which degrade performance under uniform workload [7], while under skewed workload, these route changes would cause the use of under-utilized links.

$(A.2)$ *Transient measures versus varying step changes in uniform workload.*

In response to a step change, SPF recovers faster than ExBF which recovers faster than MS. In addition, SPF has the lowest delays during recovery, followed by ExBF, and then MS.

[10]We have observed that SPF has higher oscillations in instantaneous delay compared to ExBF and MS under uniform workload around saturation [33]. We have not checked for oscillations in routes.

(*B*) *Varying failures and uniform workload.*

The three algorithms are equivalent with respect to throughput. ExBF and SPF are equivalent with respect to data load, while MS has slightly higher data load (by less than 10%). The algorithms differ highly in delay as described in *B.1* and *B.2*.

(*B.1*) *Delay versus varying uniform workload U for fixed up+down time and failure intensity.*

SPF has the smallest delay, followed by ExBF, and then MS. The difference in delay is more significant for small U and around saturation (MS being 60% worse than SPF, and ExBF being 20% worse than SPF).

(*B.2*) *Delay versus varying up+down time and failure intensity for fixed uniform workload* $U = 1$.

SPF and ExBF have approximately equivalent delays, while MS has much higher delay (upto 70% more) except for very high failure intensity values. For high failure intensity values, the three algorithms have approximately equivalent delays.

(*C*) *Varying failures and hotspot-with-background workload.*

ExBF and SPF are equivalent with respect to throughput and data load. MS has lower throughput and higher data load, except for very high up+down time and very low failure intensity values (upto 20% less throughput). Note that this is the first scenario with significant difference in throughput between the algorithms. The algorithms differ highly in delay as described in *C.1* and *C.2*.

(*C.1*) *Delay versus varying hotspot workload for fixed background workload, up+down time and failure intensity.*

MS has the smallest delay, followed by SPF, and then ExBF. The delay for SPF and ExBF can be twice as high as for MS. However, this is misleading because for the same hotspot workload, MS has lower throughput. By looking at delay versus throughput (rather than delay versus workload), we observe that SPF is the best, followed by ExBF, and then MS.

(*C.2*) *Delay versus varying up+down time and failure intensity for fixed hotspot-with-background workload.*

MS has the smallest delay, followed by SPF, and then ExBF. The difference between SPF and ExBF is much more significant for small up+down time and also for small failure intensity values (SPF being upto 100% better than ExBF). Again, examining delay versus throughput shows that SPF is the best.

Our explanation for MS having the lowest delay, while having lower throughput and higher data load, is that in response to a link failure, MS sets next-hop entries for some destinations to *nil* to avoid loops entirely (while the other algorithms allow short-lived loops). While a next-hop entry for a destination is *nil*, packets received for the destination are dropped, which in turn causes retransmissions.

(*C.3*) *Transient measures for a single link failure for fixed hotspot-with-background workload.*

With respect to instantaneous throughput and delay after a link failure, the three algorithms are equivalent in recovery time and overshoot. With respect to instantaneous routing load, SPF and MS took almost no time to recover, whereas ExBF had much longer recovery time. In addition, ExBF has the highest routing load during recovery.

(*C.4*) *Transient measures for a single link repair for fixed hotspot-with-background workload.*

With respect to instantaneous throughput and delay after a link repair, far from saturation, SPF and ExBF have similar recovery times, while MS has much longer recovery time. Around saturation, ExBF and SPF differed significantly, SPF having the smaller recovery time. With respect to overshoots, MS has the smallest overshoot in delay, followed by SPF, and then ExBF. The high overshoot and large recovery time after link repair is because of routing updates and the fact that the repaired link becomes very attractive to many source-destination pairs since it reports a low cost. With respect to instantaneous routing load, the same observation made above for a link failure holds. We believe the recovery time for instantaneous routing load reflects the complexity measures such as "message" and "hop-count" complexities. Note that the recovery behavior for instantaneous routing load is not related to the recovery time for delay and throughput.

(D) *Varying link cost function parameters with no failures.*

ExBF has the best overall performance and is the least sensitive.

(E) *Varying workload with no failures for a high-speed network.*

In general, the behavior for the high-speed case is an exaggerated version of the behavior for the low-speed case. That is, there are no differences in throughput, data load, and delay far from saturation; and the differences in delay around saturation are greater in proportion.

(F) *Other observations.*

In every scenario, the routing load is a negligible fraction of the overall network capacity (hence we do not display plots of it except in *C.3* and *C.4*). ExBF drops more data packets than the other two algorithms. Regardless of that, we observe no degradation in its performance in terms of throughput and delay.

	Varying workload (uniform and hotspot) with no failures	Varying failures and hotspot workload	Varying failures and uniform workload
Throughput	MS $\approx$ ExBF $\approx$ SPF	SPF $\approx$ ExBF $>$ MS	SPF $\approx$ ExBF $\approx$ MS
Data load	MS $\approx$ ExBF $\approx$ SPF	SPF $\approx$ ExBF $<$ MS	SPF $\approx$ ExBF $\leq$ MS
Delay	MS $\approx$ ExBF $\approx$ SPF (far from saturation) ExBF $\approx$ MS $<$ SPF (around saturation, uniform) ExBF $\approx$ SPF $<$ MS (around saturation, hotspot)	MS $<$ SPF $<$ ExBF	SPF $\leq$ ExBF $<$ MS
Overshoot	SPF $<$ ExBF $\leq$ MS	MS $<$ SPF $<$ ExBF	
Recovery time	SPF $<$ ExBF $\ll$ MS	SPF $\leq$ ExBF $\ll$ MS	

Table 1: Summary of the observations (if two algorithms are separated by a $<$ (or $>$), the algorithm on the left of the $<$ (or $>$) is better).

9 Conclusions

We summarize our observations as follows. Regarding steady-state performance, SPF performs well under hotspot-with-background workload and failures. ExBF performs well under both uniform workload and hotspot-with-background workload, and is the least sensitive to changes in the link cost function. MS performs well under uniform workload. Regarding transient performance, SPF performs the best in general. Overall, in spite of short-lived loops, SPF and ExBF perform better than MS. This implies that short-lived loops do not degrade performance. And the diffusion computation mechanism used by MS causes MS to adapt too slowly. (These conclusions held even in the high-speed network we studied.)

The simulations in this paper typically considered a constant number of unending connections. Elsewhere [31], we considered a more dynamic workload, where each source initiates a succession of finite file transfer connections to its sink. The duration of a connection, i.e. the time to deliver the data packets of a file, 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 connections varies with time, and the system behavior is similar to that of an open queueing network [22]. Our conclusions in [31] agree with those in this paper. The only difference is that the dynamic workload results in higher delay and earlier saturation, as expected.

It is interesting to compare our results with the usual metric for comparing routing algorithms, namely the number of messages needed to adapt to a link cost change. 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 ExBF has a comparable routing load to SPF, and that the overall routing load is 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 simulations were limited to only one network topology, which although realistic was not very large. Simulations 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. Further work is needed to study the sensitivity of the results to different traffic characteristics.

Through these simulations, we have gained valuable insight into the behavior of routing algorithms. We are now engaged in developing accurate mathematical models for analyzing the performance of routing algorithms in very large networks.

References

[1] C. Alaettinoğlu, K. Dussa, A. U. Shankar, and J. Bolot. Routing Testbed: Initial Design. Technical Report UMIACS-TR-90-71, CS-TR-2475, Department of Computer Science, University of Maryland, College Park, MD 20742, May 1990.

[2] 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.

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

[4] D.P. Bertsekas. Dynamic Behavior of Shortest Path Routing Algorithms for Communication Networks. *IEEE Transactions on Automatic Control*, 27(1):60–74, February 1982.

[5] C. Cassandras, M.V. Abidi, and D. Towsley. Distributed Routing with On–line Marginal Delay Estimation. *IEEE Transactions on Communications*, 38(3):348–359, March 1990.

[6] 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.

[7] W. Chou, A.W. Bragg, and A.A. Nilson. The Need for Adaptive Routing in the Chaotic and Unbalanced Traffic Environment. *IEEE Transactions on Communications*, 29:481–490, April 1981.

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

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

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

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

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

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

[14] R. Gallager. A Minimum Delay Routing Algorithm Using Distributed Computation. *IEEE Transactions on Communications*, COM-25(1):73–85, January 1977.

[15] 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.

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

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

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

[19] 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.

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

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

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

[23] Stephen S. Lavenberg. *Computer Performance Modeling Handbook.* Academic Press, 1983.

[24] 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.

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

[26] 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.

[27] 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.

[28] J. Postel. Transmission Control Protocol: DARPA internet program protocol specification. Request for Comment RFC-793, Network Information Center, SRI International, 1981.

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

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

[31] A. U. Shankar, C. Alaettinoğlu, K. Dussa-Zieger, and I. Matta. Performance Comparison of Routing Protocols under Dynamic and Static File Transfer Connections. *ACM SIGCOMM Computer Communication Review*, 22(5):39–52, October 1992.

[32] A. U. Shankar, C. Alaettinoğlu, K. Dussa-Zieger, and I. Matta. Transient and Steady-State Performance of Routing Protocols: Distance-Vector versus Link-State. Technical Report UMIACS-TR-92-87, CS-TR-2940, Institute for Advanced Computer Studies and Department of Computer Science, University of Maryland, College Park, MD 20742, August 1992. Submitted for publication in *IEEE/ACM Transactions on Networking.*

[33] A.U. Shankar, C. Alaettinoğlu, I. Matta, and K. Dussa-Zieger. Performance Comparison of Routing Protocols using MaRS: Distance-Vector versus Link-State. In Proc. *ACM SIGMETRICS/PERFORMANCE*, volume 20, pages 181–192, Newport, Rhode Island, June 1992.

[34] H. A. Taha. *Operations Research : An Introduction.* Macmillan publishing Co., second edition, 1976.

[35] 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.

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

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

[38] L. Zhang and D. D. Clark. Oscillation Behaviour of Network Traffic : A Case Study Simulation. *Internetworking Research and Experience*, 1990.

Appendix I

In this Appendix, we present details of the scenarios simulated along with plots of the observed performance measures. The scenarios are divided into groups corresponding to the groups in section 8. Thus, for each group, the discussion here supports the corresponding observation made in section 8.

In our simulations, 95% confidence intervals were computed using either the method of batch means [23] or the method of independent replications [23, 34]. The latter method was used for all transient measures. In particular, an instantaneous measure, say $x(t)$, is obtained as $\frac{x_1(t)+x_2(t)+\cdots+x_n(t)}{n}$, where the $x_1(t), x_2(t), \ldots, x_n(t)$ are the instantaneous measures obtained using the different independent simulation runs. In almost all cases, the size of the confidence intervals is less than 10% of the mean. Only in cases *(A.2)*, *(C.3)* and *(C.4)*, this is not so at some points. Even when the size of the confidence interval is highest, it does not affect our conclusions.

(A) Varying workload with no failures.

(A.1) Steady-state measures versus varying workload.

The first graph in Figure 2 shows the throughput versus uniform workload U in the range 0.5 to 6. The second and third graphs show the data load and delay versus throughput, respectively. The data load varies from a low of 13% to a high of 85%. The saturation point is around $U = 3$, which corresponds to a throughput of almost 1864 bytes/msec. The algorithms differ only in delay. MS and ExBF perform about the same while SPF has higher delay, about 75%, near the saturation point.

The first graph in Figure 3 shows the throughput versus H in the range from 2 to 12, for $B = 1$. The second and third graphs show the data load and delay versus throughput, respectively. The data load varies between 25% to 34%. The saturation point is around 30% data load ($H = 6$), which corresponds to a throughput of almost 800 bytes/msec. Saturation occurs earlier than in the uniform case (see Figure 2), because the incoming links to the hotspot node become a bottleneck. SPF, MS and ExBF are similar except around the saturation point where MS performs 50% worse in delay than the others.

Figure 4 shows the delay versus throughput for $B = 0.5, 1.5, 2$. As the background load increases, saturation occurs earlier (i.e. smaller H). Note that with an increasing background load, the delay decreases. This behavior results from the fact that with increasing background load, the weight of the hotspot load decreases, resulting in lower average delays.

(A.2) Transient measures for varying step changes in uniform workload.

In this scenario, a set of connections ($U = U1$) starts at time zero, while a second set of connections ($U = U2$) starts at simulated time 150 seconds. The simulation is stopped at simulated time 500 seconds. Figure 5 shows the instantaneous delay versus time in the range 100 to 470 seconds for $U1 = 0.2$ and $U2 = 2.5, 3, 3.5$.

For $U2 = 2.5$, the overshoot in the instantaneous delay for MS and ExBF is significantly higher than for SPF, about 50%, in response to the step change in the workload at 150 seconds. The recovery time for SPF is almost negligible, for ExBF is 14 seconds whereas the recovery time for MS is 40 seconds. Although SPF recovers quickly, it exhibits more oscillations and has a slightly higher average delay at steady-state.

For $U2 = 3$, MS and ExBF have an overshoot in the instantaneous delay resulting in higher delays during recovery. MS has about 50% higher delay than SPF, whereas ExBF is about 30% worse. However, SPF settles in a higher average delay and with less oscillations. On the other hand, ExBF recovers to the lowest average delay but exhibits higher oscillations. Here, the recovery time for all algorithms is almost the same, about 50 seconds.

For $U2 = 3.5$, all algorithms have almost the same overshoot in the instantaneous delay during recovery. The recovery time for SPF and ExBF is about the same (100 seconds). On the other hand, MS recovers in about 150 seconds. All algorithms have almost the same steady-state performance.

(B) Varying failures and uniform workload.

(B.1) Delay versus varying uniform workload for fixed up+down time and failure intensity.

The first three graphs in Figure 6 show data load, throughput, and delay versus uniform workload U in the range 0.25 to 5, for up+down time $UDT = 21$ and failure intensity $FI = 1$. The fourth curve shows the delay versus throughput. For U in the range 0.5 to 2, MS performs much worse than the other two algorithms, more than 50% worse in delay for $U = 1$ and $U = 1.5$. SPF performs better than the other two algorithms for all values of U. ExBF is worse by at most 20% (for $U = 2$) than SPF.

(B.2) Delay versus varying up+down time and failure intensity for fixed uniform workload.

Figure 7 shows data load and delay versus UDT in the range 5.25 to 105, for $U = 1$ and $FI = 1$. MS performs much worse than the other two algorithms both in delay and data load (more than 70% worse in delay for $UDT = 10.5$). SPF performs better than the other two algorithms for all values of UDT. ExBF is very close to SPF.

Figure 8 shows data load and delay versus FI in the range 0.5 to 8, for $U = 1$ and $UDT = 21$. MS has higher data load values for all values of FI. For FI in the range 0.5 to 6, SPF and ExBF have approximately equivalent delays, while MS has much higher delays (around 70% higher for $FI = 2, 3$). For $FI = 7$ and 8, the algorithms have approximately equivalent delays, ExBF being slightly better.

(C) Varying failures and hotspot-with-background workload.

(C.1) Delay versus varying hotspot-with-background workload.

The first three graphs in Figure 9 show data load, throughput, and delay versus H in the range 2 to 8, for $B = 1$, $UDT = 6$, and $FI = 1$. The fourth curve shows the delay versus throughput. For all values of H, MS has higher data load and lower throughput than SPF and ExBF which have approximately equivalent data load, throughput, and delay. For most values of H, MS has much lower delay than the other two algorithms. For example for $H = 4$, delay for MS is almost half the delay for SPF.

(C.2) Delay versus varying up+down time and failure intensity for fixed hotspot-with-background workload.

Figure 10 shows data load, throughput, and delay versus UDT in the range 1.5 to 15, for $B = 1$, $H = 4$ and $FI = 1$. For all values of UDT, MS has higher data load, lower throughput, and lower delay than the other two algorithms. SPF and ExBF have approximately equivalent data load and throughput. SPF has lower delay than ExBF (e.g. ExBF has 40% higher delay than SPF for $UDT = 3$).

Figure 11 shows data load, throughput, and delay versus FI in the range 0.25 to 2.5, for $B = 1$, $H = 4$ and $UDT = 6$. For most values of FI, MS has higher data load, lower throughput, and lower delay than the other two algorithms. SPF and ExBF have approximately equivalent data load and throughput. For small FI, SPF has lower delay than ExBF (e.g. ExBF has 100% higher delay than SPF for $FI = 0.25$).

(C.3, C.4) Transient measures for a single link failure/repair for fixed hotspot-with-background workload.

In these scenarios, we are concerned with the recovery time and overshoot in the instantaneous measures in response to a link failure and repair. We only consider hotspot-with-background workload. One of the links incident on the hotspot node (indicated in Figure 1) is failed at 100 seconds and repaired at 200 seconds.

Figure 12 shows the instantaneous delay versus time, for $B = 1$ and H in the range 2 to 8 (except $H = 4$). Figure 13 shows the instantaneous delay, throughput, and routing load versus time, for $B = 1$ and $H = 4$ (similar results have been obtained for other values of H [32]).

Our general observations are as follows: (1) with increasing H, overshoots increase; (2) with increasing H, the recovery time first increases and then decreases, with maximum recovery time achieved around saturation (i.e. around 20–35 seconds for $H = 2$, around 90–116 seconds for $H = 5$, around 45–65 seconds for $H = 8$); (3) with respect to instantaneous routing load after the failure and the repair, ExBF takes much longer to recover (e.g. around 40 seconds after the link failure for $H = 4$) than SPF and MS which take very small time to recover (e.g. less than 5 seconds for $H = 4$); (4) with respect to instantaneous delay and throughput after a link failure, the three algorithms take the same time to recover; (5) with respect to instantaneous delay and throughput after a link repair, MS takes much longer to recover, SPF recovers slightly faster than ExBF, except for $H = 5$ and $H = 6$ where SPF recovers much faster than ExBF (e.g. 90 seconds for SPF, 99 seconds for ExBF, and 116 seconds for MS for $H = 5$); (6) with respect to instantaneous delay after a link repair, in most cases MS has the smallest overshoots, followed by SPF, and then ExBF (e.g. 200 msecs for MS, 220 msecs for SPF, 275 msecs for ExBF for $H = 3$).

(D) Varying link cost function parameters with no failures.

Recall that the default values for the link cost function parameters are $Offset = 0$, $Slope = 10$, $MinLimit = 1$, $MaxLimit = 10$, $MovementLimit = 1$. The link-cost update period is uniformly distributed with mean 10 seconds and standard deviation 1 second. Figure 14 shows the delay versus the link-cost update period P in the range 1 to 100 seconds for $U = 2, 3, 4$, keeping the slope and movement limit at their default values. For $U = 3$, SPF has higher delays than MS and ExBF over the whole range of P.

The first graph in Figure 15 shows the delay versus movement limit in the range 1 to 10 for $U = 3$, keeping P and the slope at their default values. The second graph in Figure 15 shows the delay versus slope in the range 2 to 15, keeping P and the movement limit at their default values. ExBF is the least influenced by changes in the link cost function.

(E) Varying workload with no failures for a high-speed network.

The first graph in Figure 16 shows the throughput versus U in the range 0.5 to 4. The second and third graphs show the data load and delay versus throughput, respectively. The data load varies from a low of 10% to a high of 70%. The saturation point is around $U = 3$, which corresponds to a throughput of almost 136500 bytes/msec. MS and ExBF perform about the same while SPF has higher delay by about 1500% near the saturation point. Note that this happens at the same data load as in the low-speed case.

The first graph in Figure 17 shows the throughput versus H in the range 6 to 14, for $B = 2$. The second and third graphs show the data load and delay versus throughput, respectively. The data load varies between 39% to 42%. The saturation point is around $H = 10$, which corresponds to a throughput of almost 100000 bytes/msec. SPF and ExBF perform about the same while MS performs poorly around the saturation point and has higher delay by about 50%.

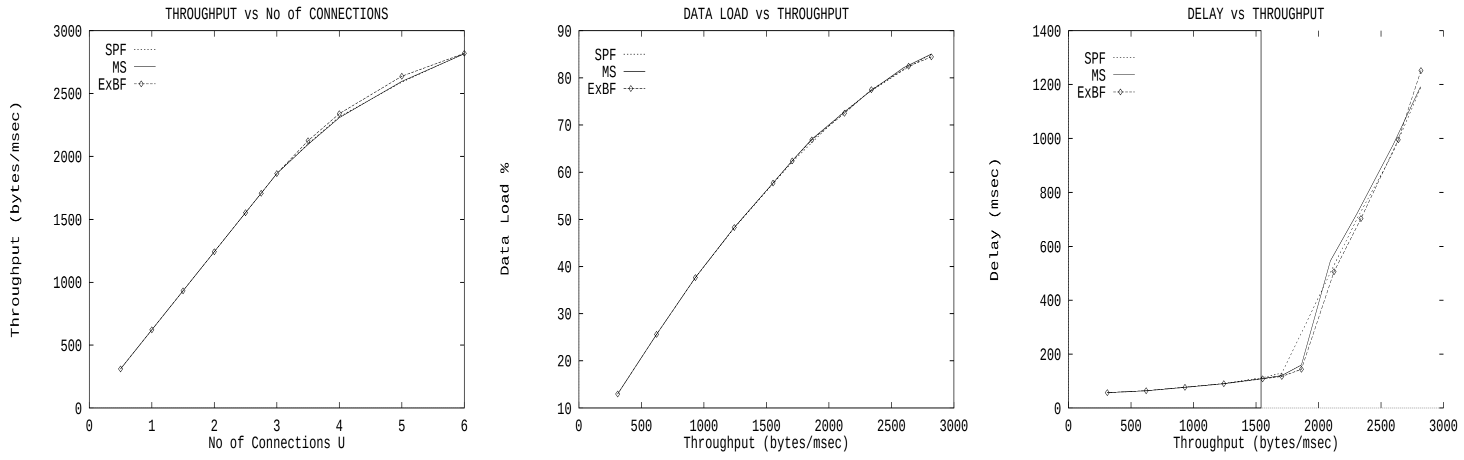

Figure 2: Varying the average number of connections between node pairs.

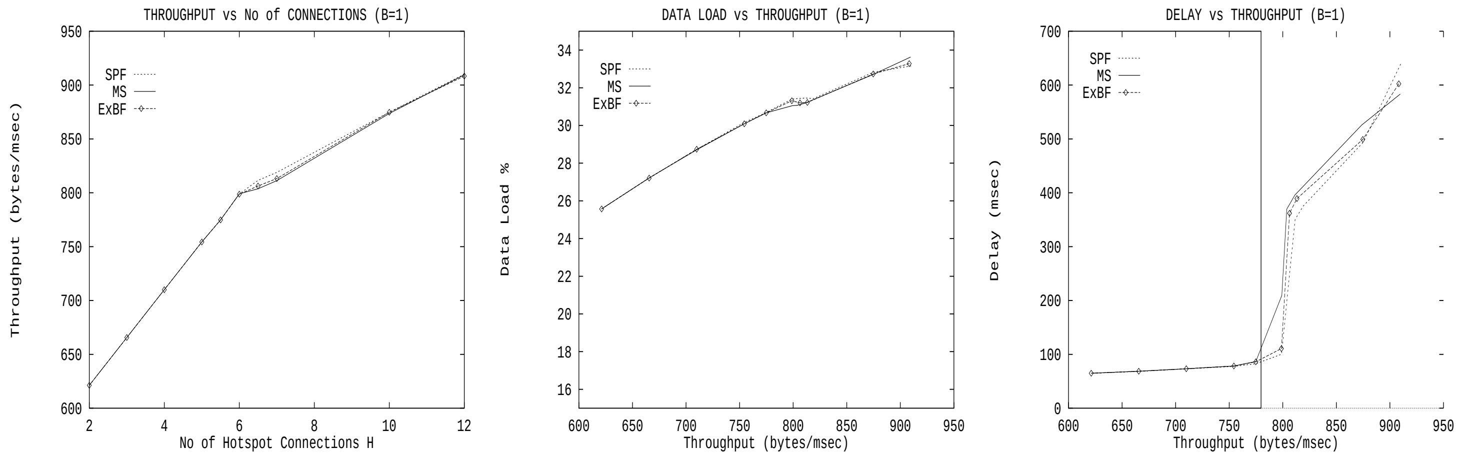

Figure 3: Varying the number of hotspot connections.

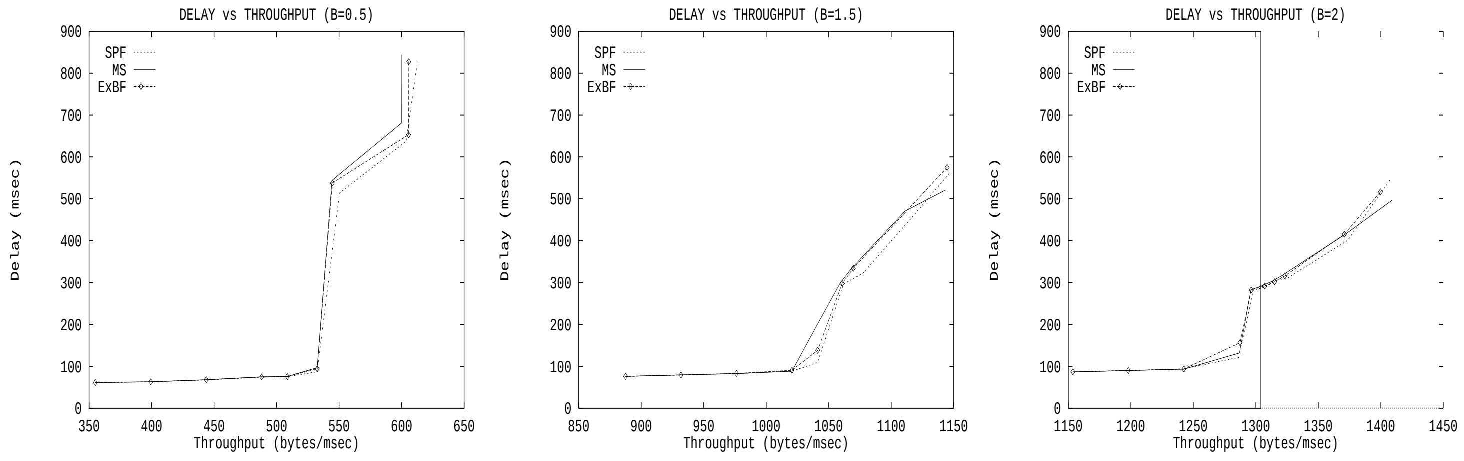

Figure 4: Varying the number of background and hotspot connections.

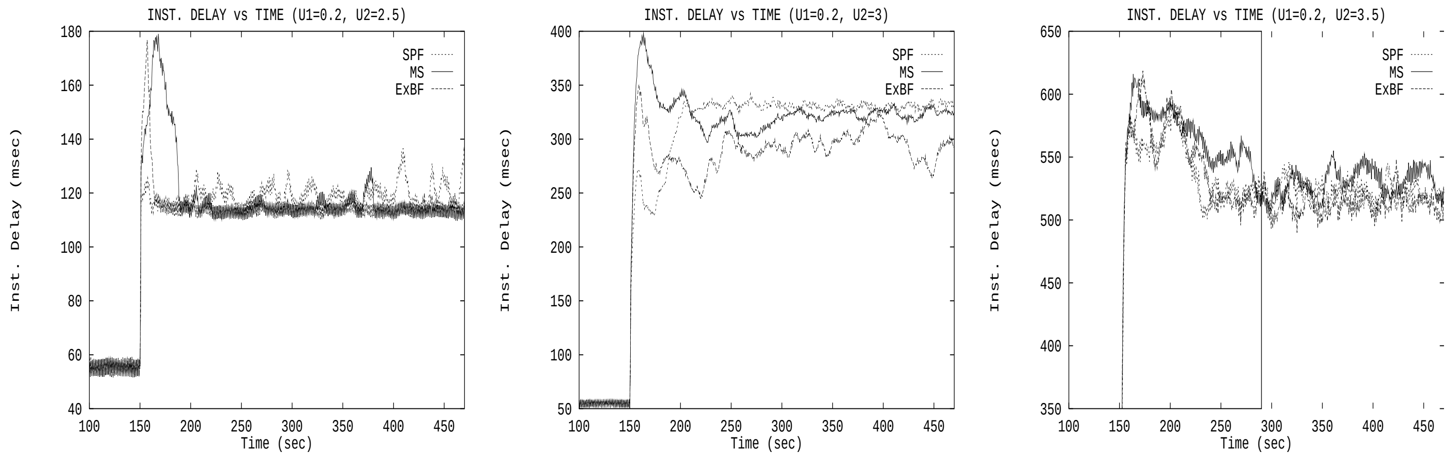

Figure 5: Step change in uniform workload.

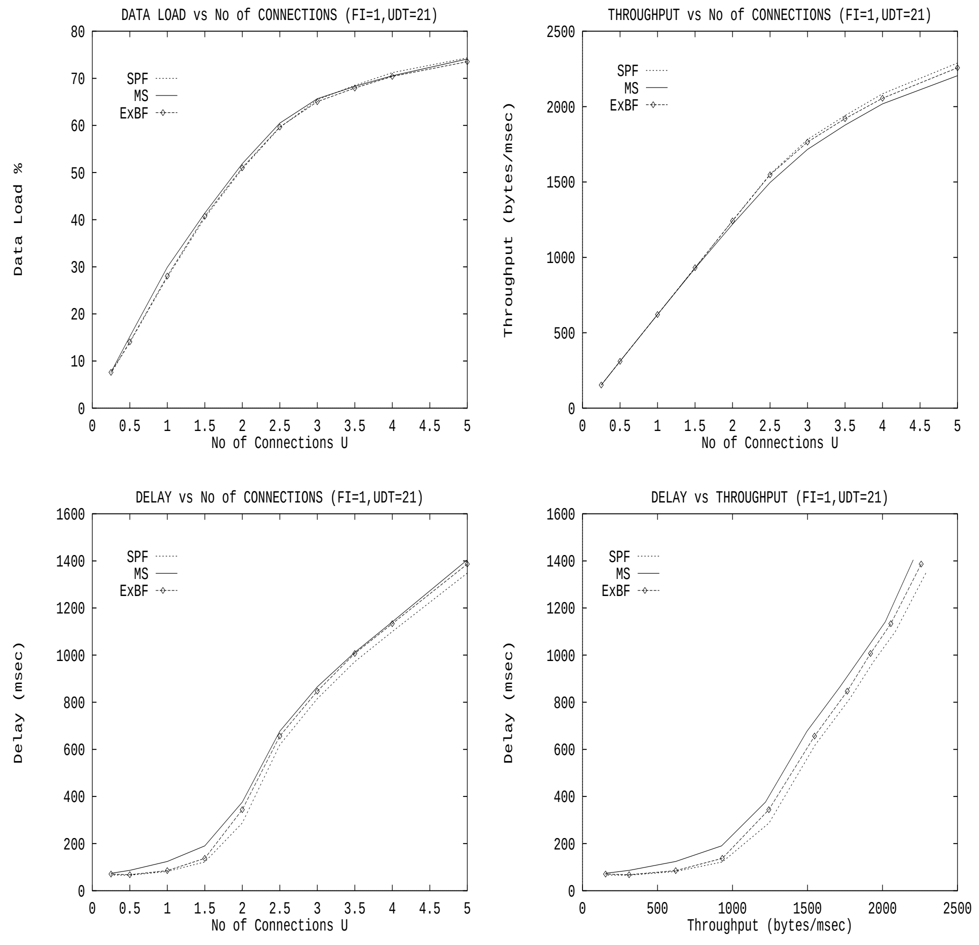

Figure 6: Link failures and repairs, uniform workload, varying U.

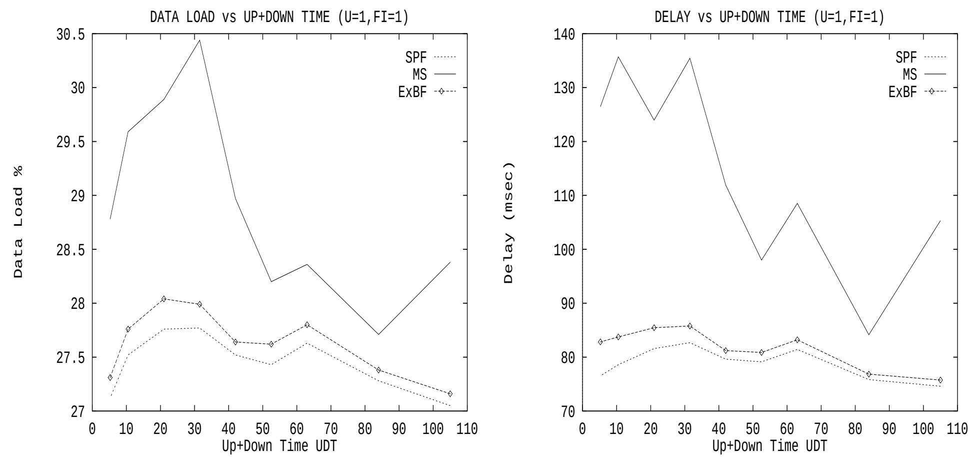

Figure 7: Link failures and repairs, uniform workload, varying UDT.

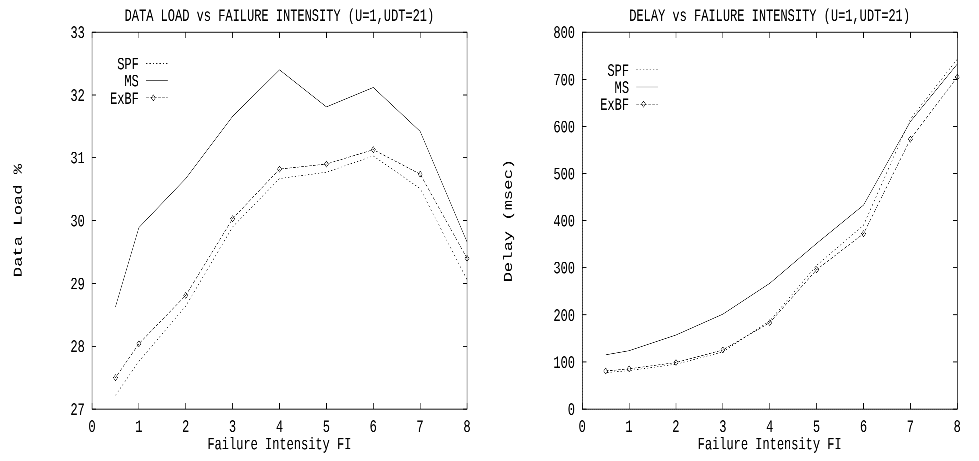

Figure 8: Link failures and repairs, uniform workload, varying FI.

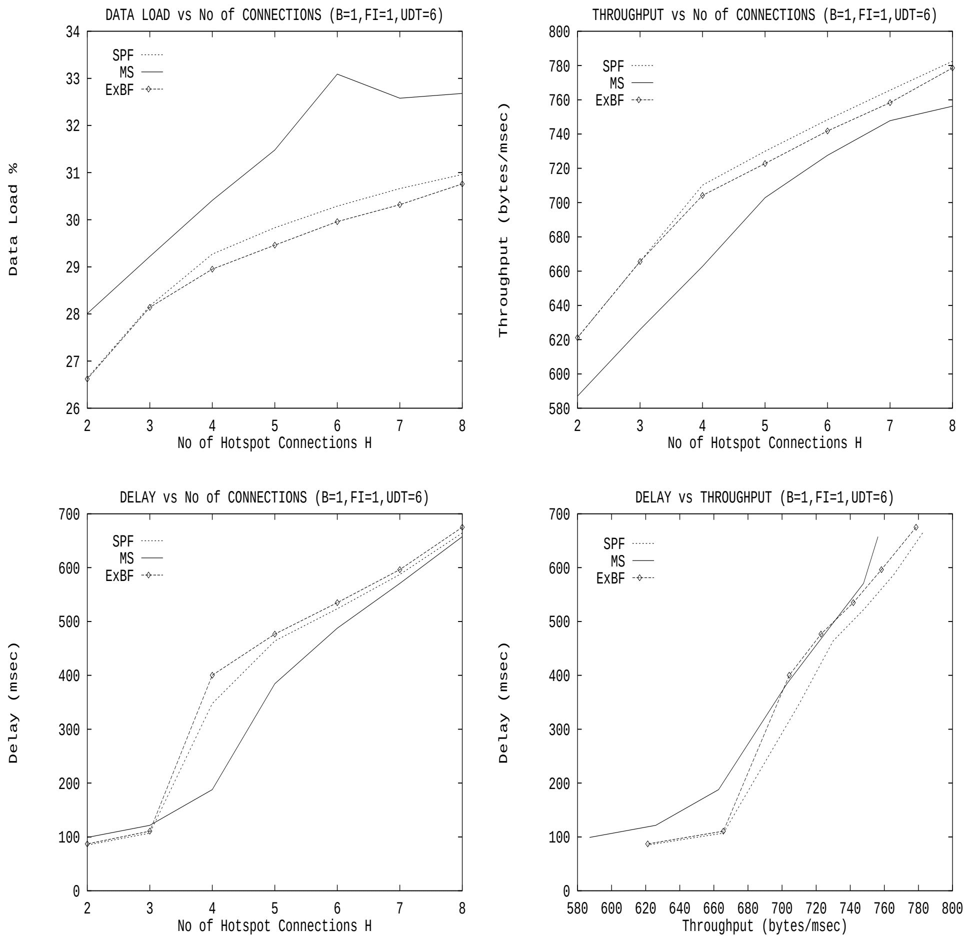

Figure 9: Link failures and repairs, hotspot-with-background workload, varying H.

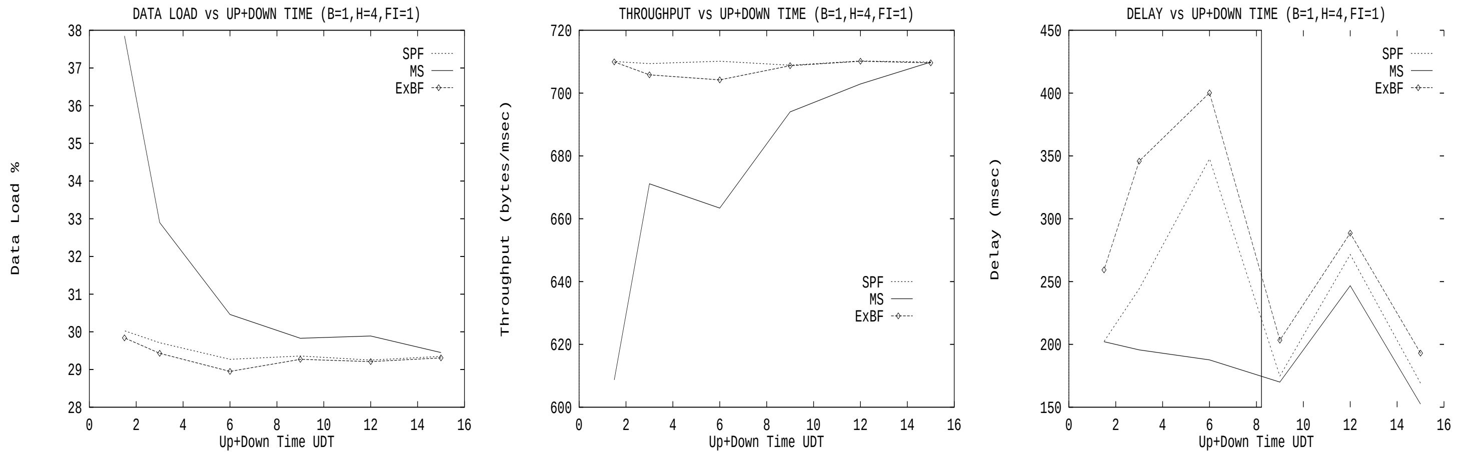

Figure 10: Link failures and repairs, hotspot-with-background workload, varying UDT.

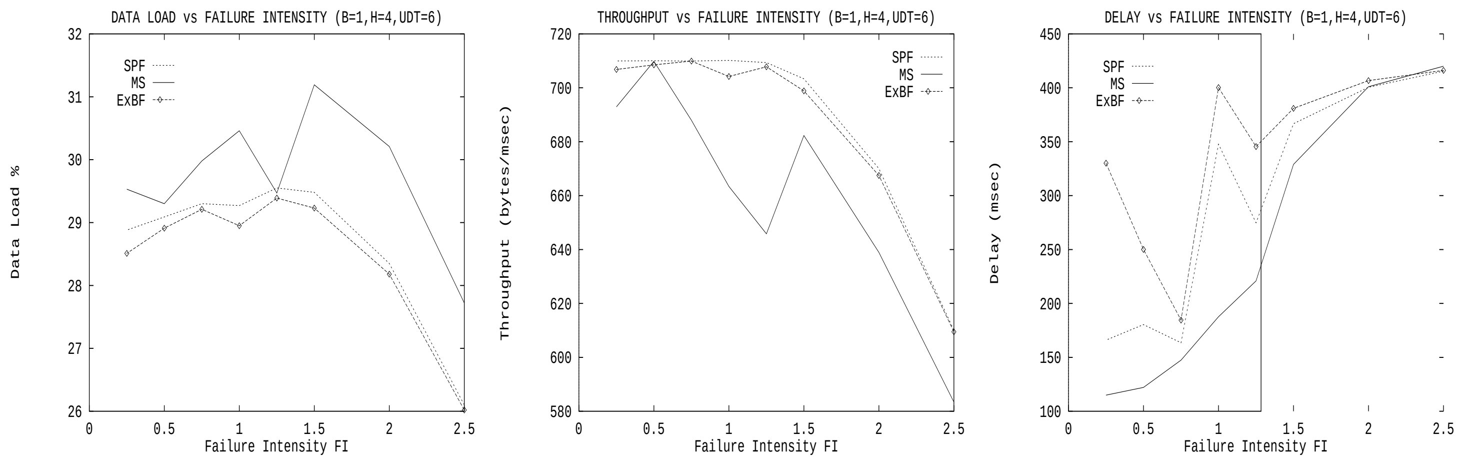

Figure 11: Link failures and repairs, hotspot-with-background workload, varying FI.

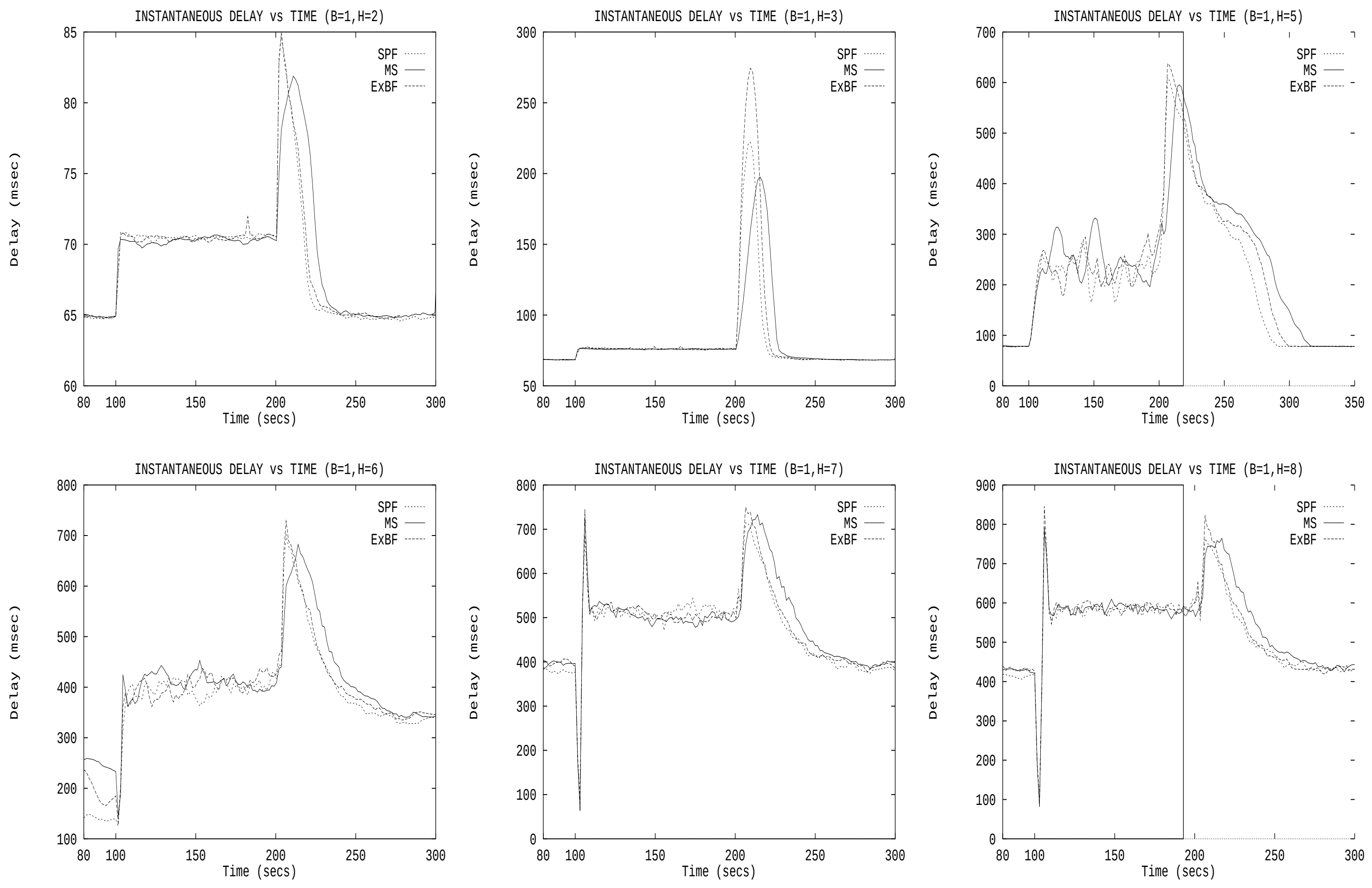

Figure 12: Link failure and repair, recovery behavior in instantaneous delay, varying H for $B = 1$.

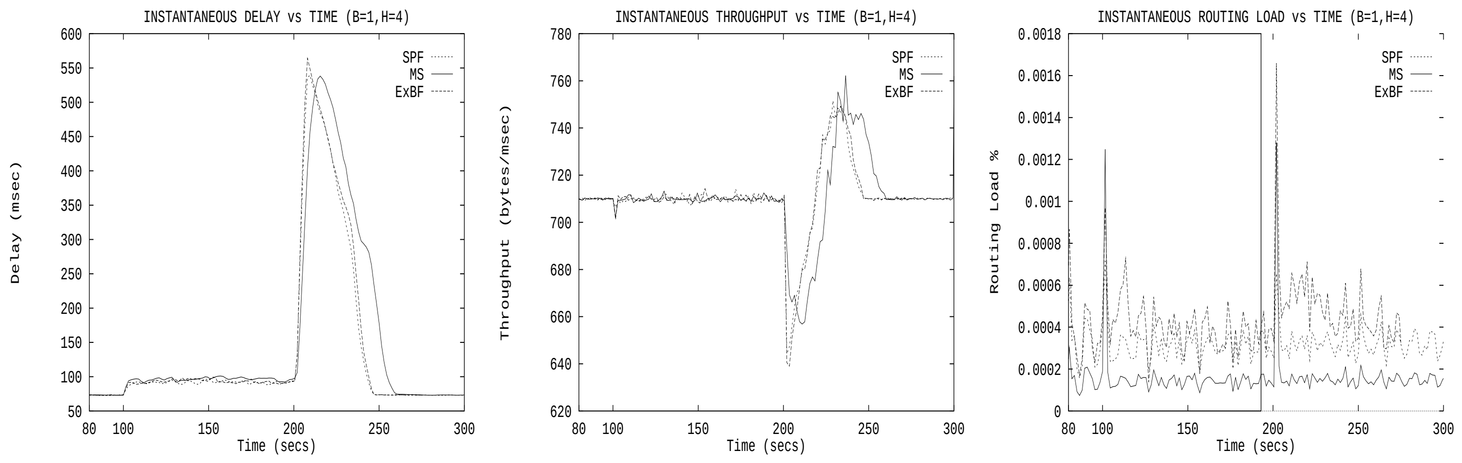

Figure 13: Link failure and repair, recovery behavior for $B = 1$ and $H = 4$.

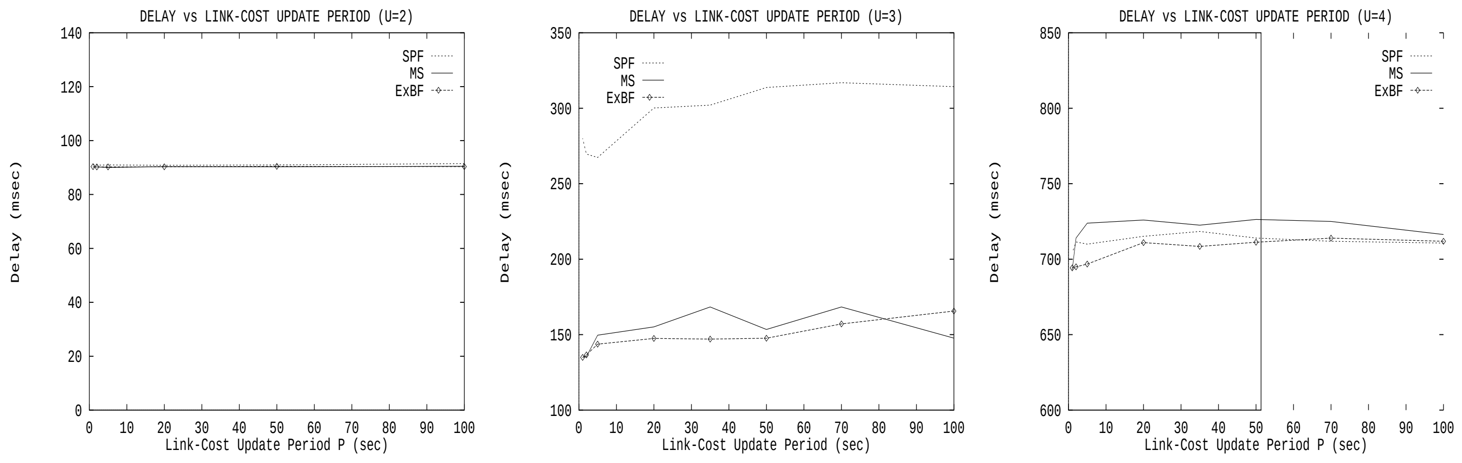

Figure 14: Varying the link-cost update period for different U.

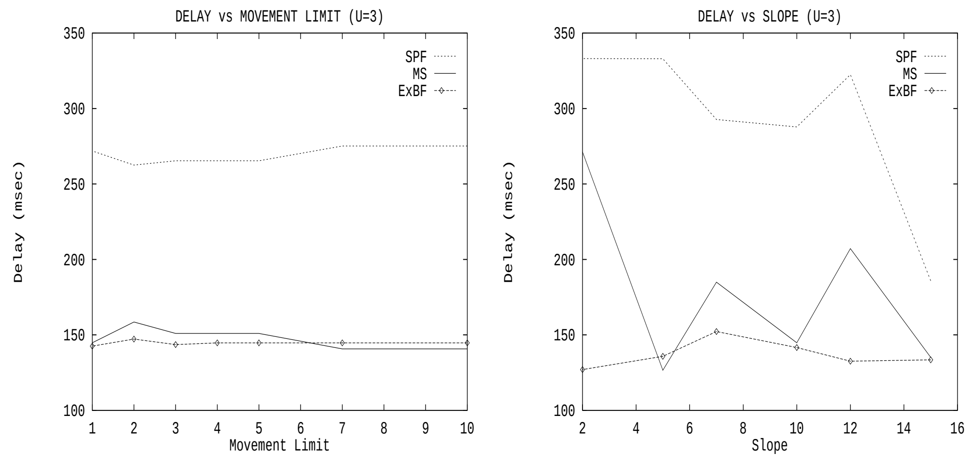

Figure 15: Varying the movement limit and slope at the saturation point.

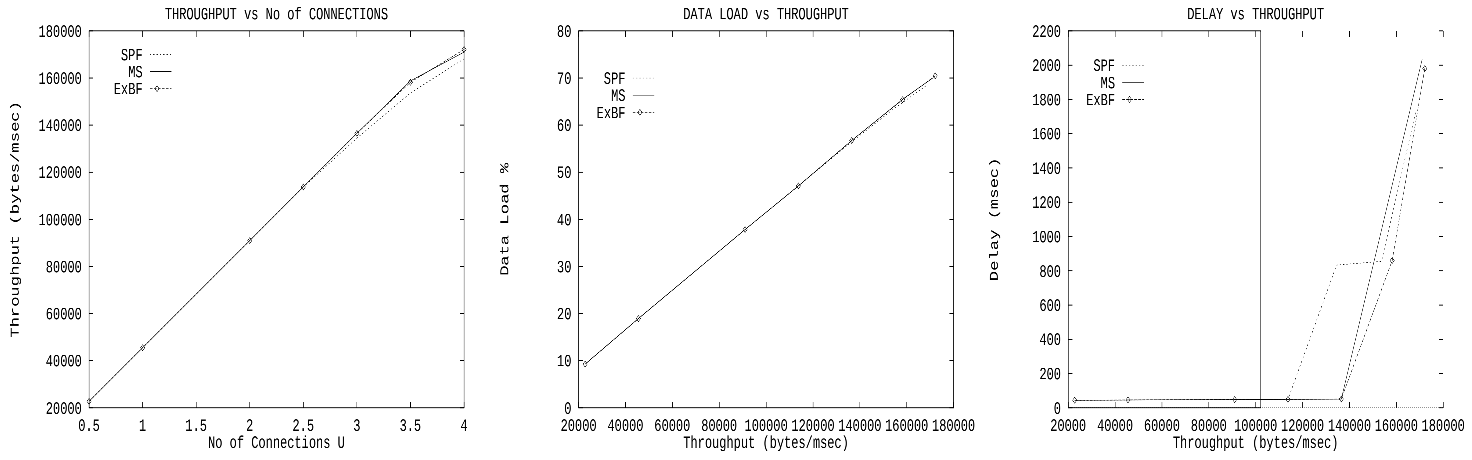

Figure 16: High-speed, uniform workload.

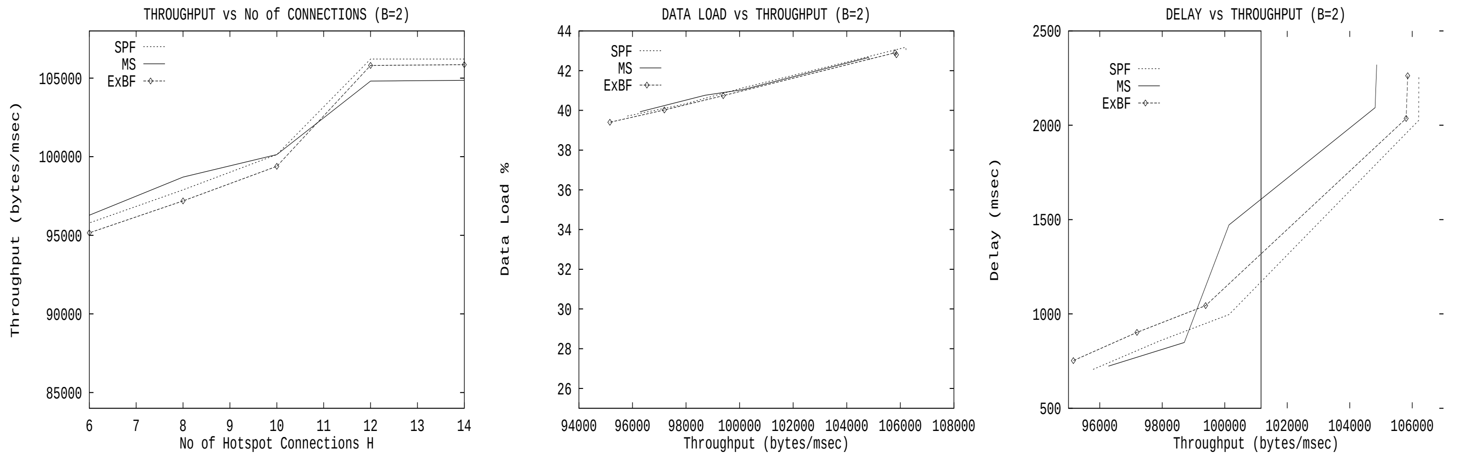

Figure 17: High-speed, hotspot-with-background workload.