Performance Comparison of Routing Protocols using MaRS: Distance-Vector versus Link-State*

A. Udaya Shankar† shankar@cs.umd.edu
Cengiz Alaettinoğlu ca@cs.umd.edu
Ibrahim Matta imm@cs.umd.edu
Klaudia Dussa-Zieger kmd@cs.umd.edu

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

Abstract

There are two approaches to adaptive routing protocols for wide-area store-and-forward networks: distance-vector and link-state. Distance-vector algorithms use $O(N \times e)$ storage at each node, whereas link-state algorithms use $O(N^2)$, where N is the number of nodes in the network and e is the average degree of a node. 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 show, using a recently developed network simulator, MaRS, that a newly proposed distance-vector algorithm (ExBF) performs as well as SPF. This suggests that distance-vector algorithms are appropriate for very large wide-area networks.

*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 paper 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 appear in *ACM SIGMETRICS/PERFORMANCE '92*, Newport, Rhode Island, June 1992.

1 Introduction

Routing protocols are one of the most important protocols in wide-area store-and-forward computer networks such as the Internet. They 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 load that can change. Consequently, a routing protocol must monitor link delays and adapt its routes to changes in link delays.

Being adaptive in a distributed environment brings up two related issues: (a) How to disseminate link delay changes to nodes in the network, so that nodes can change their routes if needed, and (b) How to control oscillations when nodes do change their routes. Oscillations arise naturally because of the delayed feedback between routing changes and link delay changes; if they are not controlled, the performance benefits of adaptive control can be lost (and in extreme cases the system can break down).

It is generally accepted that oscillations are reduced if nodes across the network have a consistent view of link delays, and do not overcompensate (or react "too quickly") to changes in link delays. Thus, routing protocol designers usually do the following: (1) represent the delay on a link by a more slowly changing measure, referred to as a **link cost**, and (2) try for rapid dissemination of link cost information across the network, i.e. a small **dissemination time**. Typically, a link's cost is maintained by its transmitting node, which monitors the link traffic and regularly updates the link cost using some **link cost function**.

There are two basic approaches to 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, where N is the number of nodes in the network. The dissemination time is small and proportional to D where D is the network diameter in hops (this requires that routing packets are given priority over data packets, which is usually the case, e.g. [20]).

The distance–vector approach which is based on the Bellman-Ford algorithm [11] is more subtle: each node maintains for each destination a set of distances, one for each of its neighbors. 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. 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, e.g. of the order of distances [2].

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) [20]. At that time, the link cost was the same as the 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 [18].

Since 1979, many new kinds of distance–vector algorithms have been proposed [21, 15, 5, 13, 23] which achieve significantly reduced dissemination times by using node coordination mechanisms. The Merlin-Segall algorithm [21, 24] has a worst-case dissemination time of $O(D^2)$ hop count. The Jaffe–Mosse algorithm [15], the DUAL algorithm by Garcia-Luna-Aceves [13], and the Extended Bellman–Ford algorithm by Cheng *et al.* [5] all have worst-case dissemination time of $O(D)$ hop count. Although the last three algorithms have the same order of dissemination times in terms of hop count, in terms of real time they 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. We use a recently developed simulator called MaRS (Maryland Routing Simulator) [1]. We consider three algorithms: SPF; Merlin-Segall, henceforth called MS; and Extended Bellman-Ford, henceforth called ExBF. We consider the NSFNET-backbone network with today's hardware as well as a future high-speed hardware. We consider varying uniform and skewed workloads and failure distributions. Our overall conclusion is that ExBF is as good as SPF for good link cost functions. Furthermore, ExBF is far less sensitive to changes in the link cost function. MS performs significantly worse than SPF and ExBF except at very high and very low loads, and is very sensitive to changes in link cost function.

The paper is organized as follows. In section 2, we review related work. In section 3, we describe SPF, MS, and ExBF. In section ??, we give a brief overview of MaRS. In section 4, we describe the parameters and performance measures of the simulations. In section 5, we summarize the results of the simulations and give our conclusions. Details of the simulated scenarios and plots are given in Appendix A.

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) [20] and its new link cost function (i.e. hop-normalized delay function) [18]. 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 [6] 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 [25], 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 [26], 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 [22], 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 [27] 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.

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 [9, 10]. Even then only simple topologies can be considered. For example, reference [9] 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* [2, 10]. 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 [4]. Typically, the solution procedure is iterative. Flow models have been used to study both link–state algorithms [3] and distance–vector algorithms [12]. A serious limitation of flow models is the assumption that flows are static (or quasi-static) [2].

3 Overview of Routing Algorithms and Link Cost Function

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 which case the node does not know where to forward data packets for that destination.

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 [7] 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 cost 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 [2, 11]. 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 [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 [8] which is started by the destination[1]. 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 computation.

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, $s_l = P_i(s_{l+1})$. Furthermore, $D_i(j)$ is the cost of this path. 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.

We point out that the reader should be aware that even in the absence of failures, routing table updates can occur as link costs are periodically re–calculated 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.

Link Cost Function

The cost of a link[2] 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.
- $AvgRawCost$: 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.
- $LinkCost$: $\{MinLimit, \ldots, MaxLimit\} \cup \{\infty\}$. 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 $AvgRawCost$

[1]Upon receiving a distance message for a destination from its next hop, a node calculates its distance, and sends its distance to all of its neighbors but its next hop. After receiving a distance message from all its neighbors, the node chooses its new next hop, and sends the new distance to its old next hop. A node with *null* next hop chooses the first neighbor that reports a finite distance as its next hop.

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

to $NormRawCost$.

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:

  $AvgRawCost := 0.5 \times (RawCost + AvgRawCost)$
  $NormRawCost := max(min(AvgRawCost \times Slope + Offset,\ MaxLimit),\ MinLimit)$
  if $|NormRawCost - LinkCost| > MovementLimit$ then
  $LinkCost := LinkCost + MovementLimit \times sign(NormRawCost - LinkCost)$
  else $LinkCost := NormRawCost$

  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 [18]. 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:

$$RawCost = 1 - \frac{\textit{average packet transmission time}}{\textit{average packet delay}}$$

4 Simulation Parameters and Performance Measures

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. It allows the user to define a network configuration, control its simulation, log the values of selected parameters, calculate average and instantaneous performance measures, 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 in public domain by anonymous ftp from `ftp.cs.umd.edu` [1].

MaRS was verified through a series of tests. We conducted simple experiments on small network topologies. We examined the simulation logs. Snapshots of different network parameters, such as routing tables, link costs and status, etc. were taken to verify the correct operation of the simulator. The correct computation of the performance measures was verified by carrying out the calculations in different ways.

A network configuration consists of a *physical network* (nodes and links), a *routing algorithm* and a *workload*. In this paper, the physical network is the NSFNET–Backbone with two types of hardware – a *high-speed* network and a *low-speed* network (with today's NSFNET parameters). The routing algorithms are SPF, MS and ExBF. The workload is file transfer (FTP). (MaRS also implements remote login (TELNET), and a simple workload. However, we use only FTP workload because experience indicates that this is the major NSFNET application [17]).

The workload is defined in terms of *source-sink* pairs which are attached to nodes. An FTP source produces data packets according to a packet-train model [16]; intuitively, each train corresponds to a connection. This workload model incorporates a static send window-based flow control mechanism and an acknowledgment–with–retransmission mechanism using roundtrip time estimates. Retransmissions can be due to node or link failures or buffer space limitation. 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 it would be hard to draw conclusions about the performance of the routing algorithm itself.

In the rest of this section, we 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. [20] [18] [14] [6], or from statistics provided by Merit/NSFNET Information Services, or by experimentation.

Physical Network

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.

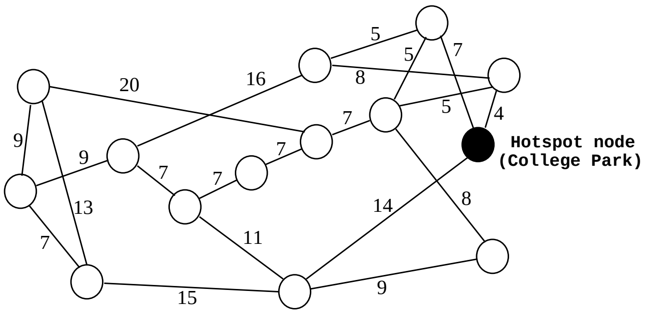

Figure 1: NSFNET–Backbone: 14 nodes, 21 bidirectional links, average degree 3.

All links have the same bandwidth, 1.5Mbit/sec for the low-speed case, and 100Mbit/sec for the high-speed case. There are no link or node failures unless stated otherwise. 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 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 $16 + 8n$ bytes long, where n 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. (These values were obtained from [20] [18], and experimentation.) In some experiments, we vary the movement limit, slope, and the link–cost update period.

Workload

We consider two types of FTP workload: (i) *Uniform workload* with parameter U indicating the average number of FTP connections between every two nodes; and (ii) *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.

All FTP connections have the following parameters: for the low-speed case, the data packet length equals 512 bytes, the inter–packet generation time is 150 msec, and the window size is 8 packets; for the high-speed case, the data packet length is 5000 bytes, the inter–packet generation time is 20 msec, and the window size is 200 packets. Unless stated otherwise, connections start when the simulation begins and they are never ending. In some cases, we have a hotspot traffic *burst* in the middle of the simulation.

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 acknowledgement.
- *Data (Routing) Load.* Fraction of the network capacity, i.e. sum of all link capacities, used by workload (routing) packets during the measurement interval.

To examine recovery time, we also consider instantaneous measures of throughput and delay. 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.

- *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.

5 Observations and Conclusions

Detailed description of the scenarios simulated (parameters settings, etc.) and plots of the observed performance measures are given in Appendix A. Each scenario was simulated with several seeds. The length of the simulation runs was chosen long enough to ensure steady–state behavior. This behavior is detected when successively computed statistics stay within a specified percentage error bound. In this section, we first make general observations about the simulation results and then present our conclusions.

- We note that 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, the network behaves more or less as a closed system and its behavior is typical of closed queueing networks [19]. Thus, in every scenario, as the workload is increased, the data load, throughput and delay follow the classical behavior. The data load and throughput first increase linearly, then level off as the system becomes saturated. The delay increases very slowly at first, and then very rapidly after the saturation point.
- In every scenario, 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 only show delay plots.)
- In every scenario, the routing load is a negligible fraction of the overall network capacity, and we do not display any plot of it. This is consistent with the previous observation.
- Under a *skewed* workload (i.e. hotspot-with-background, or failures, or hotspot burst), SPF and ExBF are about equivalent (with SPF being slightly better), while MS does significantly worse over a large range surrounding the saturation point.
  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). We give more importance to a skewed workload since it is more representative of real–life network condition than the uniform workload.
- In general, the behavior for the high–speed case is an exaggerated version of the behavior for the low–speed case.
- Under link failures and repairs, SPF and ExBF are practically equivalent (in terms of delay). MS is equivalent to them at low load or low frequency of topology changes (i.e. small number of link failures and repairs per second). For high frequency of topology changes, the delay for MS is much bigger than SPF and ExBF.
  If instead of delay we look at *recovery time* defined as the time needed to reach steady-state delay after a link failure or repair, then SPF recovers faster than ExBF which recovers faster than MS.
- For changing link–cost parameters, ExBF has the best overall performance and is the least sensitive.
- 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.

Overall, 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, even in the high-speed network we studied. And the diffusion computation mechanism used by MS causes MS to adapt too slowly.

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 ($O(N \times E)$), where E is the number of network links. However, our experiments show that ExBF has a comparable routing load to SPF, and that the overall routing load is very small compared to the data load. 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. 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. Further work is needed to study the sensitivity of the results to different traffic characteristics.

Acknowledgements

We would like to thank the anonymous referees for their helpful comments.

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] D.P. Bertsekas. Dynamic behavior of shortest path routing algorithms for communication networks. *IEEE Transactions on Automatic Control*, 27(1):60–74, February 1982.

[4] C. Cassandras, M. V. Abidi, and D. Towsley. Distributed routing with on–line marginal delay estimation. 38(3):348–359, March 1990.

[5] 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 *ACM SIGCOMM '89*, pages 224–237, September 1989.

[6] Wushow Chou, Arnold Bragg, and Arne Nilsson. The need for adaptive routing in the chaotic and unbalanced traffic. *IEEE Transactions on Communications*, 1981.

[7] E. Dijkstra. A note on two problems in connection with graphs. *Numer. Math.*, 1:269–271, 1959.

[8] E. Dijkstra and C. Scholten. Termination detection for diffusing computations. *Information Processing Letters*, 11(1):1–4, 1980.

[9] A. Ephremides, P. Varaiya, and J. Walrand. A simple dynamic routing problem. *IEEE Transactions on Automatic Control*, 25(4):690–693, August 1980.

[10] A. Ephremides and S. Verdu. Control and optimization methods in communication network problems. *IEEE Transactions on Automatic Control*, 34(9):930–942, September 1989.

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

[12] R. Gallager. A minimum delay routing algorithm using distributed computation. *IEEE Transactions on Communications*, COM-25(1):73–85, January 1977.

[13] J.J. Garcia-Luna-Aceves. A unified approach to loop free routing using distance vectors or link states. In *ACM SIGCOMM '89*, pages 212–223, September 1989.

[14] S. A. Heimlich. Traffic characterization of the NSFNET national backbone. In *Proc. USENIX '90*, pages 207–227, Washington, D.C., January 1990.

[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] R. Jain and S.A. Routhier. Packet trains - measurements and a new model for computer network traffic. *IEEE Journal on Selected Areas in Communication*, SAC-4(6):986–995, September 1986.

[17] 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.

[18] A. Khanna and J. Zinky. A revised ARPANET routing metric. In *ACM SIGCOMM '89*, pages 45–56, September 1989.

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

[20] 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.

[21] P. M. Merlin and A. Segall. A failsafe distributed routing protocol. *IEEE Transactions on Communications*, COM-27(9):1280–1287, September 1979.

[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 *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] 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.

[26] Z. Wang and J. Crowcroft. Shortest path first with emergency exits. In *ACM SIGCOMM '90*, pages 166–176, September 1990.

[27] W. Zaumen and J.J. Garcia-Luna-Aceves. Dynamics of distributed shortest-path routing algorithms. In *ACM SIGCOMM '91*, September 1991.

A Details of Simulations

A.1 Low–speed, uniform workload

Varying the average number of connections between node pairs

Figure 2 shows the data load, throughput and delay versus U in the range 0.5 to 6. The data load varies from a low of 13% to a high of 85%. The saturation point is around $U = 3$. The algorithms differ only in the delay curves. MS and ExBF perform about the same while SPF has higher delay, about 75%, near the saturation point.

Varying link–cost parameters at the saturation point

The first graph in Figure 3 shows the delay versus the link–cost update period P in the range 1 to 100 seconds for $U = 3$, keeping the slope and movement limit at their default values. SPF has higher delays than MS and ExBF over the whole range of P. The second graph in Figure 3 shows the delay versus movement limit in the range 1 to 5 for $U = 3$, keeping P and the slope at their default values. The third graph in Figure 3 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.

A.2 Low–speed, hotspot-with-background workload

Varying the number of hotspot connections

Figure 4 shows the data load, throughput and delay versus H in the range from 2 to 15, for $B = 1$. The data load varies between 25% to 34%. The saturation point is around 30% data load ($H = 6$). 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 than the others.

Varying the number of background and hotspot connections

Figure 5 shows the delay versus H for $B = 0.5, 1.5, 2$. As the background load increases, saturation occurs earlier. 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.

Varying the hotspot burst

In this scenario hotspot connections are initially passive and become active at simulated time 150 sec. Each hotspot connection sends 750 packets. When all hotspot connections end, the simulation is stopped. Figure 6 shows the data load, throughput and delay versus H in the range 2 to 12 for $B = 1$. SPF, MS and ExBF perform similar except around the saturation point ($H = 6$) where MS performs slightly worse than the others.

A.3 Low–speed, link failures and repairs

Link failures and repairs, uniform workload

In this scenario, the frequency F of topology change, and the average number I of failed links in the network, are varied by adjusting the exponentially–distributed duration of the operational and failed periods of each link.

The first graph in Figure 7 shows delay versus U in the range 0.25 to 5, for $F = 2$ and $I = 1$ (achieved by having for each link, an average operational period of 20 seconds and failed period of 1 second). For U in the range 0.5 to 2, MS performs much worse than the other two algorithms, about 100% worse for $U = 0.5$ and $U = 1$. SPF performs better than the other two algorithms for all values of U. ExBF is very close to SPF.

The second graph in Figure 7 shows delay versus F in the range 0.4 to 2, for $U = 1$ and $I = 1$. For small F the algorithms perform about the same, and as F increases MS starts performing badly.

The third graph in Figure 7 shows delay versus I in the range 1 to 5 for $U = 1$ and $F = 2$. As I increases, the algorithms use longer paths, as a result delay increases. The results show that MS starts using longer paths earlier.

Recovery time, hotspot-with-background workload

In this scenario, one of the links incident on the hot spot node is failed at time 100 seconds and repaired at time 200 seconds. Figure 8 shows the instantaneous throughput and delay versus time in the range 50 to 300 seconds for $B = 1$, and $H = 4$.

As a result of the link failure the delay increases. After link repair, the delay increases sharply 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. The recovery-time for SPF is 35 seconds whereas the recovery time for ExBF is 50 seconds and for MS it is 55 seconds. The difference in recovery time is also apparent in the instantaneous throughput versus time graph.

A.4 High–speed

High–speed, uniform workload

Figure 9 shows the data load, throughput and delay versus U in the range 0.5 to 4. The data load varies from a low of 10% to a high of 70%. The saturation point is around $U = 3$. 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.

High–speed, background-with-hotspot workload

Figure 10 shows the data load, throughput and delay versus H in the range 6 to 15, for $B = 2$. The data load varies between 39% to 42%. The saturation point is around $H = 10$. SPF and ExBF perform about the same while MS performs poorly around the saturation point and has higher delay by about 50%.

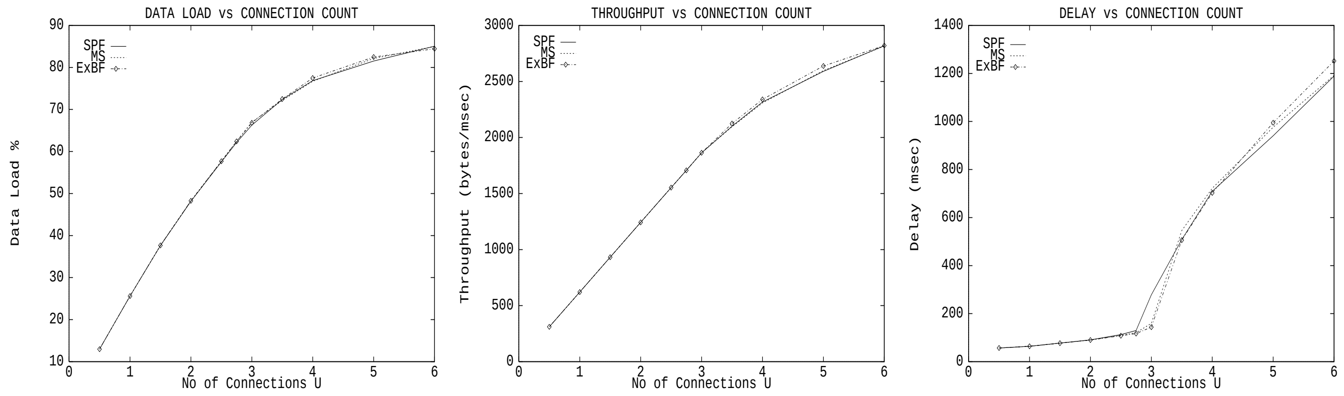

Figure 2: Varying the average number of connections between node pairs.

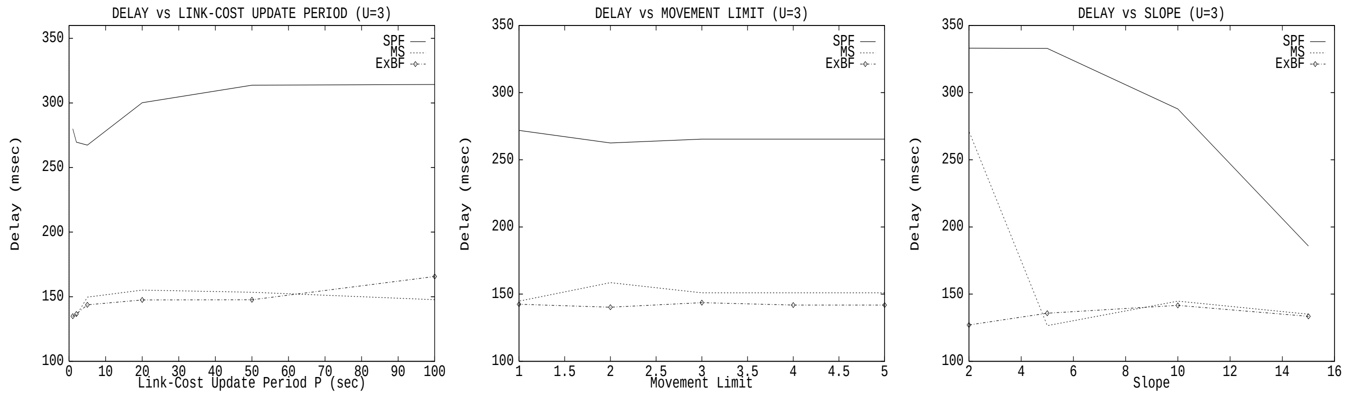

Figure 3: Varying link–cost parameters at the saturation point.

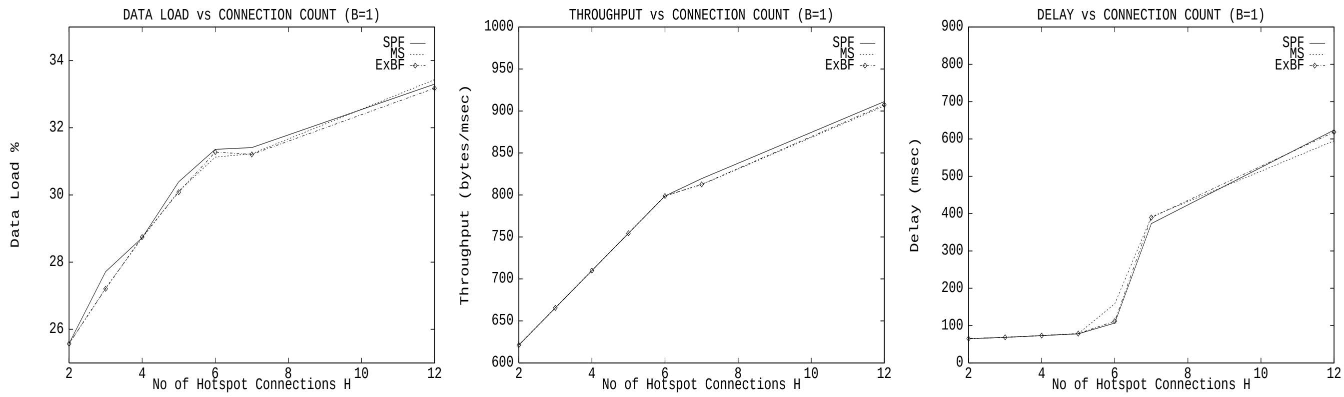

Figure 4: Varying the number of hotspot connections with fixed background connections.

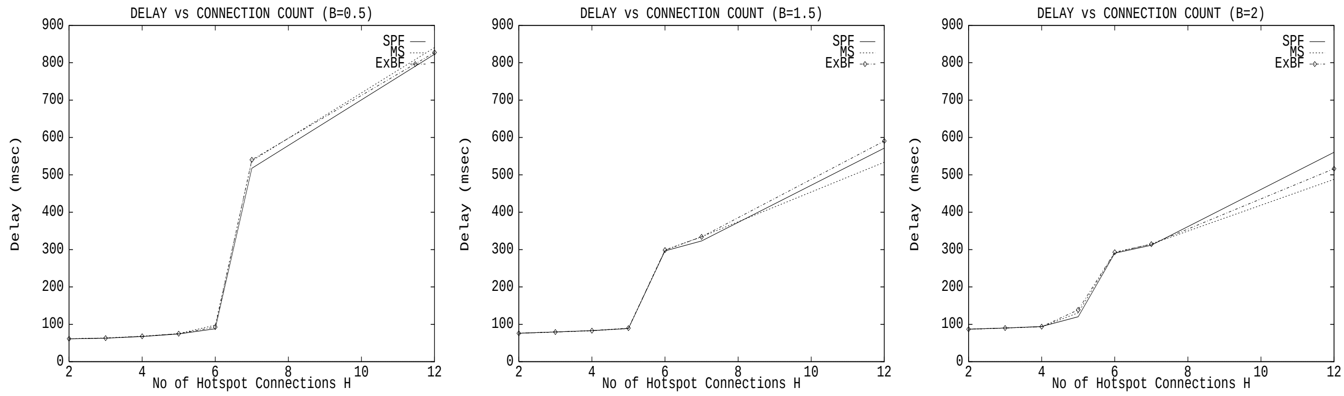

Figure 5: Varying the number of background and hotspot connections.

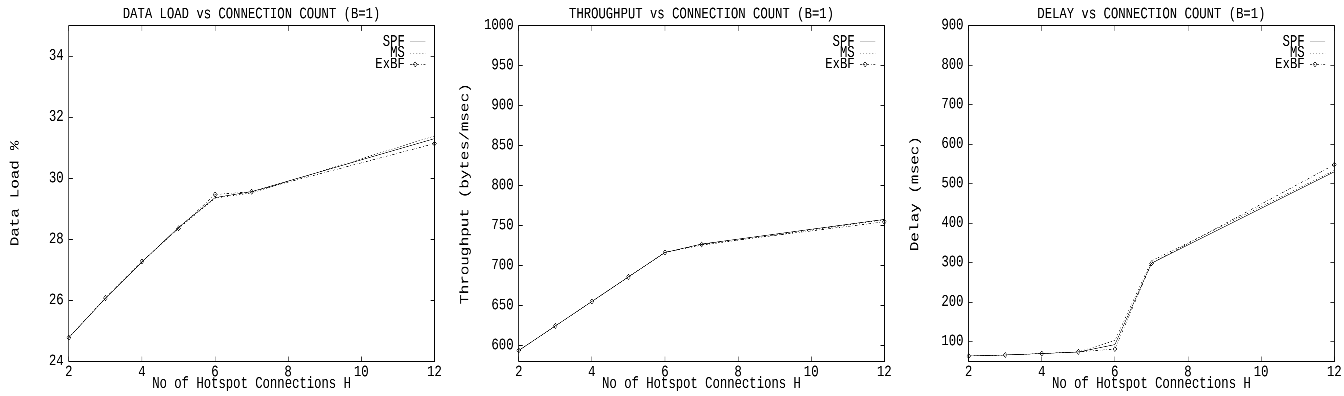

Figure 6: Hotspot Burst.

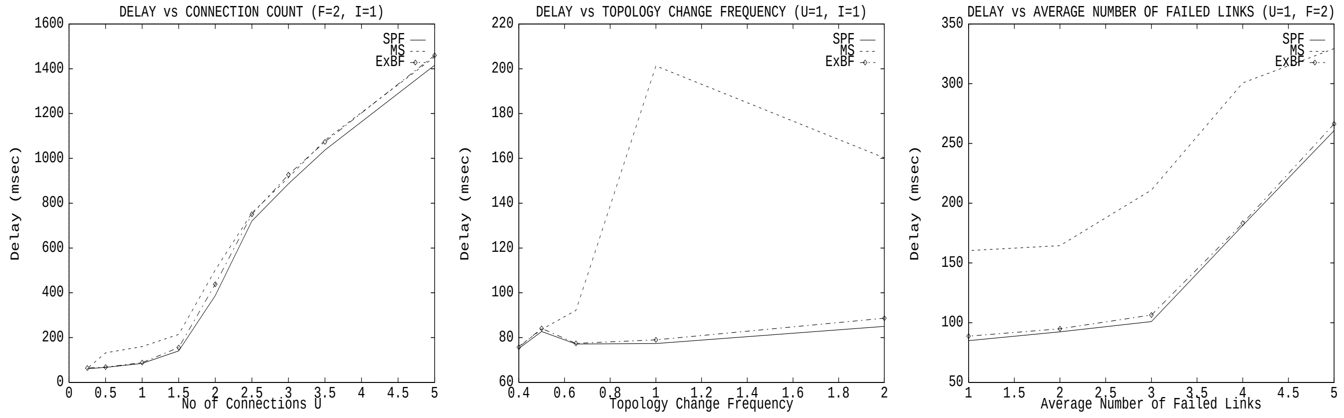

Figure 7: Link Failures and Repairs.

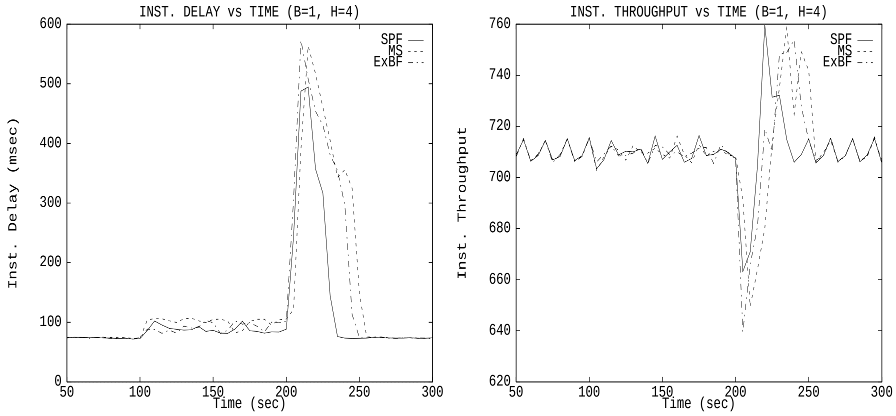

Figure 8: Recovery Time.

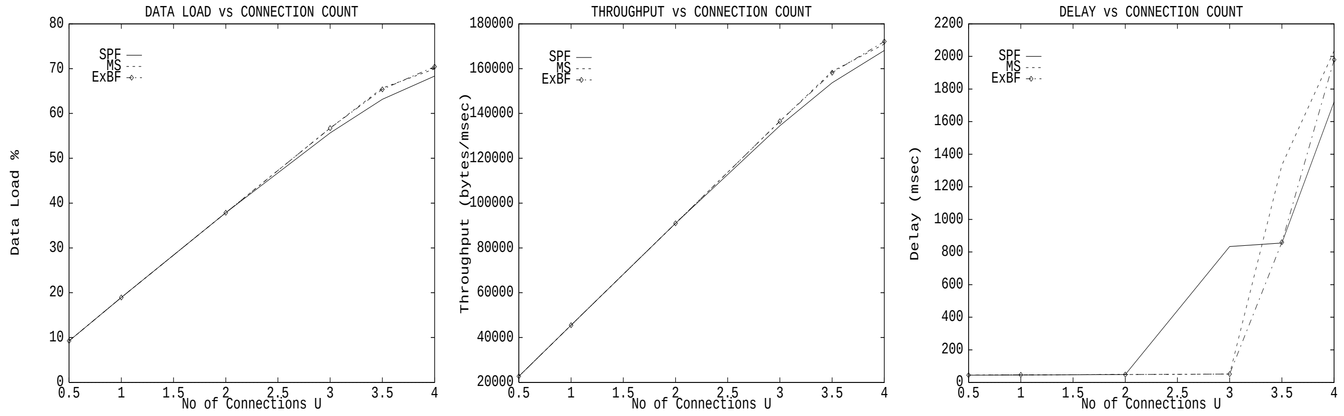

Figure 9: High–speed, uniform workload.

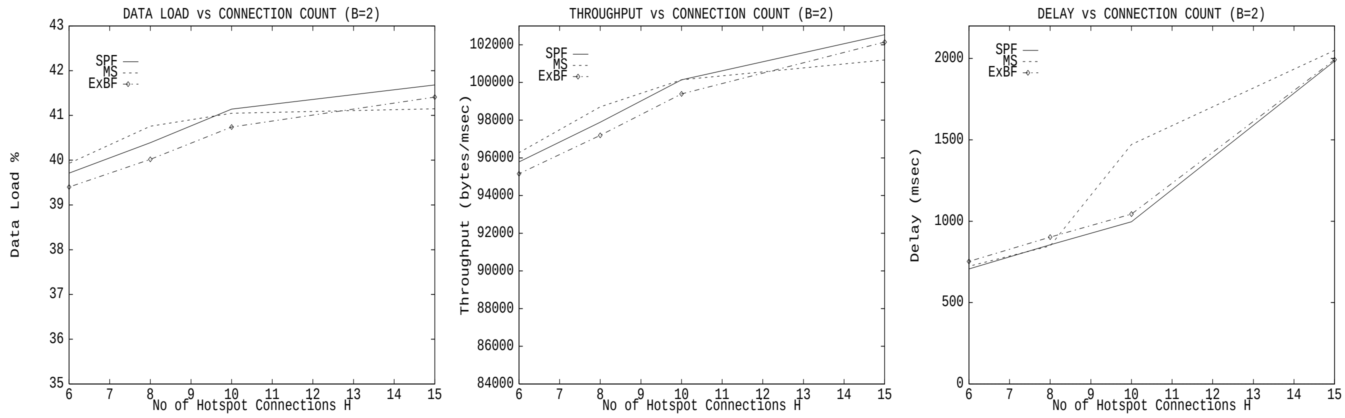

Figure 10: High–speed, background-with-hotspot workload.