Achieving Faster Failure Detection in OSPF Networks

Mukul Goyal
CIS Department
The Ohio State University
Columbus, OH, USA
mukul@cis.ohio-state.edu

K. K. Ramakrishnan
Networking Research
AT&T Labs - Research
Florham Park, NJ, USA
kkrama@research.att.com

Wu-chi Feng
Dept of Computer Science & Engg
Oregon Institute of Technology
Beaverton, OR, USA
wuchi@cse.ogi.edu

***Abstract*— With the current default settings of the OSPF parameters, the network takes several tens of seconds before recovering from a failure. The main component in this delay is the time required to detect the failure using *Hello* protocol. Failure detection time can be speeded up by reducing the value of *HelloInterval*. However, too small a value of *HelloInterval* will result in an increased chance of network congestion causing loss of several consecutive Hellos, thus leading to false breakdown of adjacency between routers. Such false alarms not only disrupt network traffic by causing unnecessary routing changes but also increase the processing load on the routers which may potentially lead to routing instability. In this paper, we investigate the following question - What is the optimal value for the *HelloInterval* that will lead to fast failure detection in the network while keeping the false alarm occurrence within acceptable limits? We examine the impact of both network congestion and the network topology on the optimal *HelloInterval* value. Additionally, we investigate the effectiveness of faster failure detection in achieving faster failure recovery in OSPF networks.** (*Abstract*)

Keywords—Failure Recovery; OSPF (*key words*)

I. INTRODUCTION

Link state protocols, such as OSPF [1] and IS-IS [2] using shortest path first forwarding are the most commonly used Interior Gateway Protocols in the Internet today. Each router knows the topology of the network, and the associated weights, and uses this information to determine the shortest paths to different destinations. However, when there is a failure in the network (link or node failure), these protocols take some time to detect the failure and re-establish a consistent view of the new topology. During this transient, the data traffic forwarded towards the failed device will be dropped. Additionally, routing loops might emerge leading to artificial congestion in the network.

In a carrier's network, service level assurances (SLA) provided to customers potentially limit the number of packets that may be lost or are delayed excessively. To ensure that SLAs are met, a carrier network often uses lower layer transport (or data link) layer failure detection and restoration techniques, so that service is not excessively impacted. Depending on just the routing layer (and hence OSPF or IS-IS) for recovery from failures has been typically considered unacceptable, because it takes too long to recover from failures. However, incorporating protection against failures at the transport layer is expensive as it requires significant redundant capacity. This motivates us to examine how we can optimize the time to recover from failures at the routing layer, by examining the mechanisms used within OSPF.

OSPF has been designed to be generally applicable, and the timers and protocols have been designed so that it can be deployed in a network of reasonably large scale. However, we have observed in practice that service providers generally limit the number of routers in a single OSPF area, for a variety of reasons. This naturally begs the question of whether we could adapt the parameters in the OSPF protocol to achieve faster recovery from failures, especially when we know the topology of the network. Minimizing the failure recovery time has the benefit of a reduced need to depend on transport/data link layer recovery and the possibility that a more complete, network layer failure recovery mechanism could be put in place.

II. FAILURE DETECTION AND RECOVERY IN OSPF

In OSPF, two adjacent routers in the same area periodically exchange *Hello* messages to maintain the link adjacency. If a router does not receive a *Hello* message from its neighbor within a *RouterDeadInterval (*typically 40 seconds or 4 *HelloInterval*s), it assumes the link between itself and the neighbor to be down and generates a new *Router LSA* to reflect the changed topology. All such LSAs, generated by the routers affected by the failure, are flooded throughout the network and cause the routers in the network to redo the *shortest path first* (SPF) calculation and update the next hop information in the forwarding table. Thus, the time required to recover from the failure consists of: (1) the failure detection time (2) LSA flooding time (3) the time to complete the new SPF calculations and update the forwarding tables. With *HelloInterval* value 10 seconds and *RouterDeadInterval* value 40 seconds, the failure detection can take anywhere between 30 to 40 seconds. The LSA flooding times consist of the propagation delays and any pacing delays resulting from the rate-limiting of *LSUpdate* packets sent down an interface. Once a router receives a new LSA, it schedules an SPF

calculation. Since SPF calculation using Dijkstra's algorithm [3] constitutes significant processing load, the router waits for some time (*spfDelay* - typically 5 seconds) to let other LSAs arrive before doing an SPF calculation. Moreover, the routers place a limit on the frequency of SPF calculations (governed by *spfHoldTime*, typically 10 seconds, between successive SPF calculations) which can introduce further delays. In Table 1, we list different standard and vendor introduced delays that affect the OSPF operation in networks of popular commercial routers.

In this paper, we focus on reducing the failure detection time which is clearly the main component of the overall failure recovery time in OSPF based networks. While the availability of link layer notifications can help achieve fast failure detection, such mechanisms are often not available. Hence, the routers use the *Hello* protocol to detect the loss of adjacency with a neighbor. The *Hello* protocol operates via periodic exchange of *Hello* messages between neighbor routers. A router declares its adjacency with a neighbor to be down if it does not receive a *Hello* from the neighbor for more than *RouterDeadInterval*. This can happen if the link between the router and the neighbor is down or the neighbor router is no longer functional. To avoid a false breakdown of adjacency because of congestion related loss of *Hello* messages, the *RouterDeadInterval* is usually set to be four times the *HelloInterval* – the interval between successive *Hello* messages sent by a router to its neighbor. The failure detection via *Hello* protocol can be substantially speeded up by reducing the *HelloInterval*. However, there is a limit up to which the *HelloInterval* can be safely reduced. As the *HelloInterval* becomes smaller, there is an increased chance that the network congestion will lead to loss of several consecutive *Hello* messages and thereby cause false breakdown of adjacency between routers even though the routers and the link between them are functioning perfectly well. The LSAs generated because of a false alarm will lead to new path calculations, avoiding the supposedly down link, by all the routers in the network. A false alarm is soon corrected by successful *Hello* exchange between the affected routers which causes new set of LSAs to be generated and possibly new path calculations to be done by the routers in the network. Thus, false alarms cause unnecessary processing load on the routers and some times lead to temporary changes in the network traffic's path which can have a serious impact on the QOS levels in the network. If the false alarms are too frequent, the routers will have to spend a lot of time doing unnecessary LSA processing and SPF calculations which may significantly delay important tasks such as *Hello* processing, thereby leading to more false alarms. Persistent overload on router CPUs will ultimately result in complete meltdown of routing operation in the network.

In this paper, our objective is to make a realistic assessment regarding how small the *HelloInterval* can be, to achieve faster detection and recovery from network failures while limiting the occurrence of false alarms. This assessment is done via simulations on the network topologies of commercial ISPs [4] using a detailed implementation of OSPF protocol in NS2 simulator [5] which models all the protocol features as well as various standard and vendor-introduced delays in the functioning of the protocol (Table 1). We examine the network wide impact of reducing the *HelloInterval* in terms of number of false alarms under a realistic model of network congestion. We quantify the detrimental effect of these false alarms in terms of unnecessary SPF calculations done by the routers. We examine how the network topology influences the occurrence of false alarms. Finally, we evaluate how much does the faster detection of network failures help in achieving faster recovery from these failures in the operation of OSPF networks.

TABLE I. VARIOUS DELAYS AFFECTING THE OPERATION OF OSPF PROTOCOL

Standard Configurable Delays	
RxmtInterval	The time delay before an un-acked LSA is retransmitted. Usually 5 seconds.
Hello Interval	The time delay between successive Hello packets. Usually 10 seconds.
Router Dead Interval	The time delay since the last Hello before a neighbor is declared to be down. Usually 4 times the HelloInterval.
Vendor-introduced Configurable Delays	
Pacing delay	The minimum delay enforced between two successive Link State Update packets sent down an interface. Observed to be 33ms. Not always configurable.
spfDelay	The delay between the shortest path calculation and the first topology change that triggered the calculation. Used to avoid frequent shortest path calculations. Usually 5 seconds.
spfHoldTime	The minimum delay between successive shortest path calculations. Usually 10 seconds.
Standard Fixed Delays	
LSRefreshTime	The maximum time interval before an LSA needs to be reflooded. Set to 30 minutes.
MinLSInterval	The minimum time interval before an LSA can be reflooded. Set to 5 seconds.
MinLSArrival	The minimum time interval that should elapse before a new instance of an LSA can be accepted. Set to 1 second.
Router-specific Delays	
Route install delay	The delay between shortest path calculation and update of forwarding table. Observed to be 0.2 seconds.
LSA generation delay	The delay before the generation of an LSA after all the conditions for the LSA generation have been met. Observed to be around 0.5 seconds.
LSA processing delay	The time required to process an LSA including the time required to process the Link State Update packet before forwarding the LSA to the OSPF pocess. Observed to be less than 1 ms.
SPF calculation delay	The time required to do shortest path calculation. Observed to be $0.00000247x^2 + 0.000978$ seconds on Cisco 3600 series routers; x being the number of nodes in the topology.

III. RELATED WORK

In this section, we briefly survey the existing literature on speeding the recovery from network failures in the operation of OSPF and IS-IS protocols. First, we discuss the previous work on reducing the *HelloInterval* and the impact of congestion in causing false alarms. Alaettinoglu et al. [6] proposed reducing the *HelloInterval* to millisecond range to achieve sub-second recovery from network failures but did not consider any side effects of *HelloInterval* reduction. Shaikh et al. [7] used Markov Chain based analysis of a simple network topology to obtain the expected times before high packet drop rates cause a healthy adjacency to be declared down and then

back up again. However, this work did not study the network wide generation of false alarms caused by congestion as the *HelloInterval* is reduced. Basu and Riecke [8] have also examined using sub-second *HelloInterval*s to achieve faster recovery from network failures. This work is similar to ours in the sense that it also considered the tradeoff between faster failure detection and increased frequency of false alarms. It reports 275ms to be an optimal value for *HelloInterval* providing fast failure detection while not resulting in too many false alarms. However, this work did not consider the impact of different levels of network congestion and topology characteristics on the optimal *HelloInterval* value. We believe these factors impact the setting of the *HelloInterval* substantially, as we illustrate in this paper.

False alarms can also be generated if the *Hello* message gets queued behind a huge burst of LSAs and can not be processed in time. The possibility of such an event increases with reduction in *RouterDeadInterval*. Large LSA bursts can be caused by a number of factors such as simultaneous refresh of a large number of LSAs or several routers going down/coming up simultaneously. Choudhury et al. [9] studied this issue and observed that reducing the *HelloInterval* lowers the threshold (in terms of number of LSAs) at which an LSA burst will lead to generation of false alarms. However, the probability of such events is quite low. In our experiments with more probable events such as the failure of a single router, the resulting LSA burst was too small to cause false alarms. Similarly, we investigated if frequent update of *Traffic Engineering* LSAs [10] leads to large enough LSA bursts to cause false alarms. However, we did not observe any such effect even for reasonably high update frequency of such LSAs.

Since the loss and/or delayed processing of *Hello* messages can result in false alarms, recently there have been proposals to give such packets prioritized treatment at the router interface as well as in the CPU processing queue [9][11]. An additional proposal is to consider the receipt of any OSPF packet (e.g. an LSA) from a neighbor as an indication of the good health of the router's adjacency with the neighbor [11]. This provision can help avoid false loss of adjacency in the scenarios where *Hello* packets get dropped because of congestion, caused by a large LSA burst, on the control link between two routers. Such mechanisms should help mitigate the false alarm problem significantly. However, it will take some time before these mechanisms are standardized and widely deployed.

Since SPF calculation using Dijkstra's algorithm imposes significant processing load on the routers, vendors have introduced delays (*spfDelay* and *spfHoldTime*) that limit the frequency of such operations. These delays ultimately result in slowing down the failure recovery process. Alaettinoglu et al. [12] propose eliminating any restrictions on SPF calculations. They argue that the frequency of SPF calculations can be reduced by careful filtering of status changes in the links/routers and the processing time of an SPF calculation can be reduced by using modern algorithms (such as [13][14][15]) instead of Dijkstra's algorithm. In a related work, Thorup [16] proposes the use of data structures that will help routers make a constant time determination of the next hop on the shortest path to a destination avoiding a given failed link. This will help in avoiding the routing loops while the routers recalculate shortest paths after a link failure.

IV. Experimentation Methodology

We implemented substantial extensions to the OSPF routing model [17] currently available in NS2 simulator such as the *Hello* protocol, LSA generation and flooding, shortest path calculation and adjacency establishment. Our emphasis has been to include in the simulation model various standard (i.e., as per the OSPF specification [1]) and vendor-implemented delays and timers, listed in Table 1, that affect the functioning of OSPF protocol in operational networks of commercial routers. Some of these delays are configurable, some have a fixed value and some depend on the architecture and processing capability of the routers. Values for the delays that depend on the architecture and processing capability of the routers were obtained after extensive experimentation with commercial routers [18][19]. In our experimentation, we used the standard or the typical values for the different delay parameters (except *HelloInterval* and *RouterDeadInterval*) as listed in Table 1. This enables us to have a higher degree of confidence in the applicability of our simulation results to real operational networks.

Rather than using actual packet flows to create congestion, we used realistic models to achieve the same effects. This choice was driven by the lack of information about realistic traffic loads as well as a desire to keep the processing and running time of the simulations reasonable. The congestion model used in our simulations tries to emulate the behavior of *Random Early Drop (RED)* [20] and *droptail* buffer management policies. In RED, the packet drop probability (*p)* at a router interface increases linearly from a value 0 to *max_p* as the average buffer occupancy *qlen* (the ratio of the average queue length to the total buffer size) increases from *min_th* to *max_th.* The packet drop probability remains 0 for *qlen* values less than *min_th* and remains equal to *max_p* as the *qlen* becomes more than *max_th.* If *qlen* exceeds 1, the packet drop probability becomes 1, i.e., all the incoming packets are dropped. We simulate congestion by assigning random *qlen* values between 0 and *max_q* to the router interfaces. The assigned *qlen* value determines the packet loss probability for the OSPF packets arriving at the interface. The *qlen* value assigned to an interface persists for a random duration with in the range {*min_pers, max_pers*}. This is to emulate the slowly varying average queue length, an exponential moving average, in RED buffers. The *min_th, max_th* and *max_p* values used in the RED simulations are 0.25, 0.75 and 0.1 respectively. The congestion level in the simulation is controlled by parameter *max_q* and range {*min_pers, max_pers*}. As the value of *max_q* is increased, higher packet drop rates in the network become possible. The range {*min_pers, max_pers*} will determine how long high (and low) packet drop rates persist on an interface.

A similar technique is used to emulate the behavior of droptail buffer management. For droptail buffers, *qlen* represents the instantaneous buffer occupancy. A new value is assigned to the *qlen* associated with each router interface every time a new OSPF packet arrives. The packet drop probability remains 0 unless *qlen is* greater than 1 in which

case the packet drop probability is 1. Note that in the droptail simulations, *max_q* value needs to be greater than 1 for packet loss to occur and a given *max_q* value corresponds to the packet drop rate of *(max_q -1)/max_q*.

The simulations were conducted on a number of topologies obtained from [4]. These topologies correspond to real IP backbones for several commercial ISPs. Table 2 lists some characteristics of these topologies. While most of the topologies are irregular, topology *A* is a pure mesh and topology *B* has a star-like structure.

V. Simulation Results

The first set of simulation results examines how reducing the *HelloInterval* causes more false alarms to take place and how increase in network congestion exacerbates the problem. Figure 1 shows the total number of false alarms observed on topology C during 1 hour of failure-free operation for different *HelloInterval* values. These numbers were obtained from RED simulations assuming that average buffer occupancy persists for 100ms to 500ms. Different curves in the figure correspond to different congestion levels (modeled by parameter *maxQ_*). As expected, false alarms become more frequent with decrease in the *HelloInterval* value and increase in network congestion levels. Further, the impact of increased congestion levels seems to be more severe for lower *HelloInterval* values. Clearly, the optimal value of *HelloInterval* depends on the expected congestion levels in the network and an understanding of what constitutes an acceptable limit on false alarm frequency. Assuming that no more than 20 false alarms in an hour can be tolerated and if the average buffer occupancy in the router interfaces will rarely rise above 0.5, the *HelloInterval* for topology C can be set to be 250ms. However, if the buffer overflows are not uncommon, it will be prudent not to reduce *HelloInterval* below 1.5 seconds. As shown in figure 2, if the congestion persists for longer durations (200ms to 2s, rather than 100ms to 500ms as in figure 1), the number of false alarms observed for a given *HelloInterval* increase further. Again, the increase in false alarms is more severe for lower *HelloInterval*s; hence there is a need to be conservative while setting *HelloInterval* value. The results for droptail simulations are shown in figure 3. The different curves in figure 3 show results for *maxQ_* values 1.02, 1.05 and 1.1 which correspond to packet drop rates of 1.96%, 4.76% and 9.09% respectively. Note that with around 10% overload on the system, any *HelloInterval* value less than 10s leads to unacceptable number of false alarms.

False alarms disrupt traffic in the network and cause unnecessary processing load on the routers. The LSAs generated as the result of a false alarm will be flooded throughout the network and lead to new SPF calculation by each router in the network. As the frequency of false alarms increases, routers spend more and more time doing unnecessary SPF calculations; generally one SPF calculation for each false alarm. Some times, for large *HelloInterval* values, a false alarm causes two SPF calculations to be done in each router; first one in response to adjacency breakdown and second one in response to re-establishment of adjacency following successful exchange of *Hello* messages between the routers affected by the false alarm. For smaller *HelloInterval* values, a broken adjacency is generally re-established soon enough so that only one SPF calculation (scheduled 5 seconds, the *spfDelay*, after receiving the false alarm) is required to take care of both changes. Thus, for smaller *HelloInterval*s, since false alarms are corrected soon enough, they may not always lead to changes in routing tables and hence re-routing of network traffic. Nevertheless, smaller *HelloInterval* values do result in frequent false alarms and thus the processing load on the routers because of SPF calculations can become significant. Persistent overload on router CPUs can potentially lead to total meltdown of routing operation in the network. When the frequency of false alarms in the network becomes very high, *spfDelay* and *spfHoldTime* limit the frequency of SPF calculations. This and other previously mentioned effects can be seen in figure 4 which shows the average number of SPF calculations done by a router in topology C in response to false alarms in the simulations whose results regarding false alarms were previously shown in figure 1. The LSAs generated because of false alarms also impose unnecessary processing load on every router since each router may have to send and receive an LSA on each one of its interfaces as part of flooding of such LSAs.

TABLE II. Network Topologies used in Simulations

Topology	Nodes	Links	Topology	Nodes	Links
A	9	72	D	37	88
B	27	58	E	51	176
C	27	116	F	116	476

Next, we examine the impact of topology characteristics on the optimal value of *HelloInterval* for a network. The probability of a false alarm occurring in the network increases with the number of links in the network. This trend is clear from figures 5 and 6 which show the false alarm occurrence during 1 hour for different topologies for congestion levels created by *maxQ_* values 0.75 and 1 respectively. In figure 7, we plot the optimal *HelloInterval* value for different topologies assuming that no more than 20 false alarms per hour can be tolerated. It can be seen that the optimal *HelloInterval* value increases with the number of links in the topology. Further, as observed earlier, expected congestion level in the network plays a significant part in determining the optimal value.

Finally, we examine the impact of lower *HelloInterval* values on the failure detection and recovery times. For this purpose, we caused a particular router in topology C to fail and observed the failure detection time i.e. the time by when all the neighbors of the failed router have detected the failure and the failure recovery time i.e. the time by when all the routers in the network have finished SPF calculation and forwarding table update in response to the failure. The simulations used RED packet drop model with *maxQ_* value 1 and average buffer persistency in the range 0.2s to 2s. The simulations were conducted for several different seed values for the random number generation. In Table 3, we report some typical and interesting cases. As expected, the failure detection time is within the range 3 to 4 times the *HelloInterval*. Once a neighbor detects the router failure, it generates a new LSA about 0.5 seconds after the failure detection. The new LSA is flooded throughout the network and will lead to scheduling of

SPF calculation 5 seconds (*spfDelay*) after the LSA receipt. This is done to allow one SPF calculation to take care of several new LSAs. Once the SPF calculation is done, the router takes about 200ms more to update the forwarding table. After including the LSA propagation and pacing delays, we can expect the failure recovery to take place about 6 seconds after the 'earliest' failure detection by a neighbor router.

Notice that many entries in Table 3 show the recovery to take place much sooner than 6 seconds. This is mainly because the reported failure detection times are the 'latest' ones rather than the 'earliest'. In one interesting case (seed 2, *HelloInterval* 0.75s), the failure recovery takes place about 2 seconds after the 'latest' failure detection. This happens because the SPF calculation scheduled by an earlier false alarm takes care of the LSAs generated because of router failure. Many times, the failure recovery can be noticed to take place much later than 6 seconds after the failure detection (notice entries for *HelloInterval* 0.25s, seeds 1 and 3). Failure recovery can be delayed because of several factors. The SPF calculation frequency of the routers is limited by *spfHoldTime* (typically 10s) which can delay the new SPF calculation in response to the router failure. The delay caused by *spfDelay* has already been explained. Finally, the routers with low connectivity may not get the LSAs in the first try because of loss due to congestion. Such routers may have to wait for 5 seconds (*RxmtInterval*) for the LSAs to be retransmitted.

The results in Table 3 indicate that a smaller value of *HelloInterval* speeds up the failure detection but is not effective in reducing the failure recovery times beyond a limit because of other delays like *spfDelay*, *spfHoldTime* and *RxmtInterval*. While it may be possible to further speed up the failure recovery by reducing the values of these delays, eliminating such delays altogether may not be prudent. Eliminating *spfDelay* and *spfHoldTime* will result in several SPF calculations to take place in a router in response to a single failure (or false alarm) as different LSAs generated because of the failure arrive one by one at the router. The resulting overload on the router CPUs may have serious consequences for routing stability especially when there are several simultaneous changes in the network topology. Analyzing how to achieve still faster failure recovery, without compromising on routing stability, when failure detection is no longer an issue constitutes the logical next step to the work presented in this paper.

VI. CONCLUSION

With the current default settings of the OSPF parameters, the network takes several tens of seconds before recovering from a failure. The main component in this delay is the time required to detect the failure using *Hello* protocol. Failure detection time can be speeded up by reducing the value of *HelloInterval*. However, too small a value of *HelloInterval* will lead to too many false alarms in the network which cause unnecessary routing changes and may even lead to routing instability. In this paper, we explored the optimal value for the *HelloInterval* that will lead to fast failure detection in the network while keeping the false alarm occurrence within acceptable limits. Our simulation results indicate that the optimal value for *HelloInterval* for a network is strongly influenced by the expected congestion levels and the number of links in the topology. While the *HelloInterval* can be much lower than current default value of 10s, it is not advisable to reduce it to millisecond range as it will lead to too many false alarms. Further, it is difficult to prescribe a single *HelloInterval* value that will perform optimally in all cases. The network operator should set the *HelloInterval* conservatively taking in account both the expected congestion levels as well as the number of links in the network topology.

REFERENCES

[1] J. Moy, "OSPF version 2," *IETF Request for Comments 2328*, April 1998.

[2] D. Oran, "OSI IS-IS intra-domain routing protocol," *IETF Request for Comments 1142*, February 1990.

[3] E. Dijkstra, "A note on two problems in connection with graphs," *Numerische mathematik*, 1:269-271, 1959.

[4] Mapnet: Macroscopic Internet Visualization and Measurement Tool, http://www.caida.org/tools/visualization/mapnet/

[5] The Network Simulator – ns-2 , http://www.isi.edu/nsnam/ns/.

[6] C. Alaettinoglu, V. Jacobson, and H. Yu, "Toward millisecond IGP convergence," *NANOG 20*, October 2000.

[7] A. Shaikh, L. Kalampoukas, R. Dube, and A. Varma, "Routing Stability in Congested Networks: Experimentation and Analysis," *Proc. ACM SIGCOMM*, August 2000.

[8] A. Basu, and J. Riecke, "Stability issues in OSPF routing," *Proc. ACM SIGCOMM*, August 2001.

[9] G. Choudhury, V. Sapozhnikova, A. Maunder, V. Manral, "Explicit marking and proritized treatment of specific IGP packets for faster IGP convergence and improved network scalability and stability," *IETF Internet Draft draft-ietf-ospf-scalability-01.txt*, Work in progress, April 2002.

[10] D. Katz, D. Yeung, and K. Kompella, "Traffic engineering extensions to OSPF," *IETF Internet Draft draft-katz-yeung-ospf-traffic-06.txt*, Work in progress, October 2001.

[11] J. Ash, G. Choudhury, V. Sapozhnikova, M. Sherif, V. Manral, and A. Maunder, "Congestion avoidance and control for OSPF networks," *IETF Internet Draft draft-ash-manral-ospf-congestion-control-00.txt*, Work in progress, April 2002.

[12] C. Alaettinoglu, and S. Casner, "Detailed analysis of IS-IS routing protocol on the Qwest backbone," *NANOG 24*, February 2002.

[13] P. Fraciosa, D. Frigioni, and M. Giaccio, "Semi-dynamic shortest paths and breadth-first search in digraphs," *Proc. 14th Symp. Theoratical Aspects of Computer Science*, 113-124, 1997.

[14] D. Frigioni, M. Ioffreda, U. Nanni, and G. Pasqualone, "Experimental analysis of dynamic algorithms for the single-source shortest path problem," *ACM Journal of Experimental Algorithmics*, 3:5, 1998.

[15] G. Ramalingam, and T. Reps, "An incremental algorithm for a generalization of the shortest-path problem," *Journal of Algorithms*, 21(2):267-305, 1996.

[16] M. Thorup, "Fortifying OSPF/IS-IS against link-failure," Unpublished.

[17] http://networks.ecse.rpi.edu/~sunmin/rtProtoLS/

[18] A. Shaikh, M. Goyal, A. Greenberg, R. Rajan, and K. K. Ramakrishnan, "An OSPF topology server: design and evaluation," *IEEE Journal on Selected Areas in Communications (JSAC)*, Vol. 20, no 4, May 2002.

[19] A. Shaikh, and A. Greenberg, "Experience in black-box OSPF measurement," *Proc. ACM SIGCOMM Internet Measurement Workshop (IMW)*, November 2001.

[20] S. Floyd, and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, Vol. 1, no 4, August 1993.

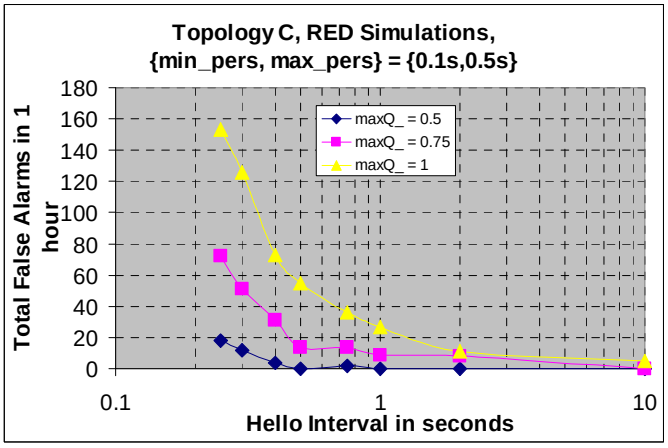

Figure 1 False Alarm Occurence in Topology C For Different *HelloInterval* Values and Congestion Levels.

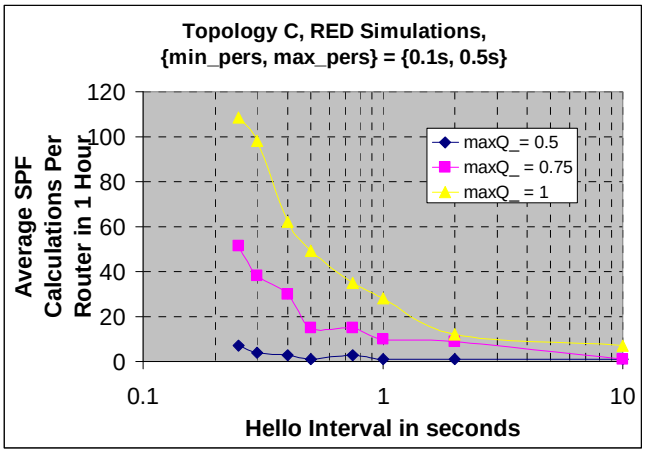

Figure 4 Average Number of SPF Calculations on a Router in Topology C Due to False Alarms Shown in Figure 1.

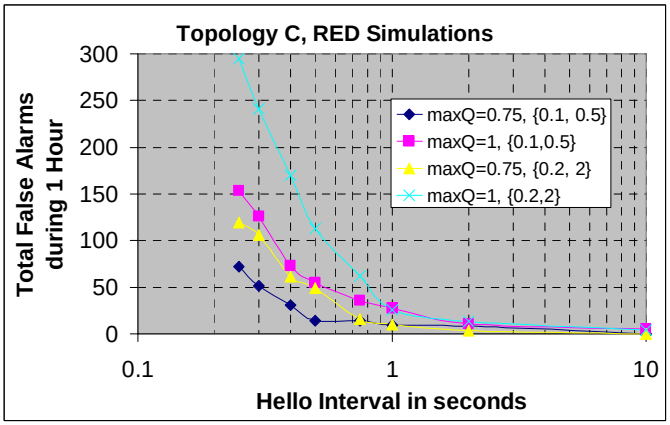

Figure 2 Change in False Alarm Frequency As High Drop Rates Persist for Longer Durations.

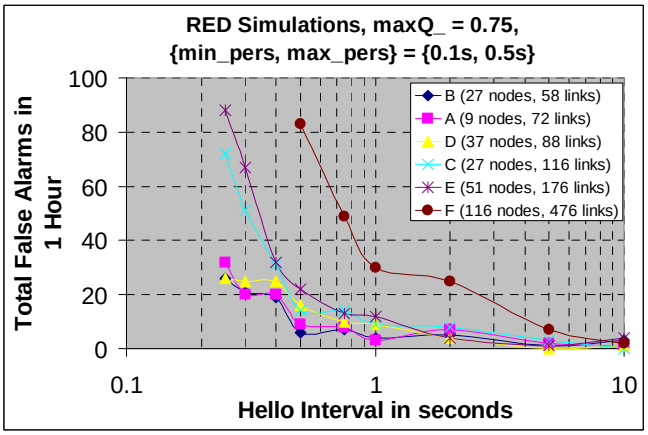

Figure 5 False Alarm Occurrence in Different Topologies For Different *HelloInterval* Values; RED Simulations with *maxQ_*=0.75.

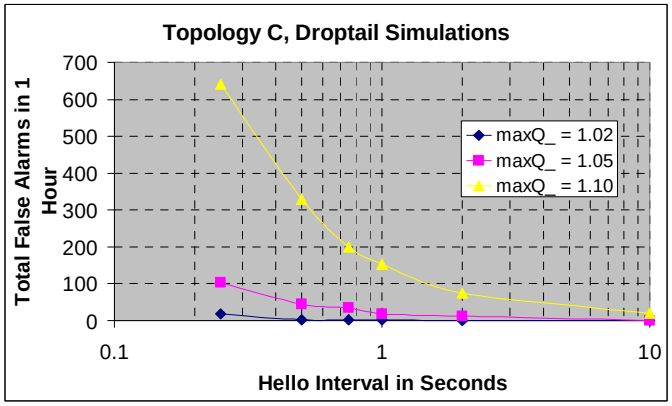

Figure 3 False Alarm Occurrence in Topology C in Droptail Simulations

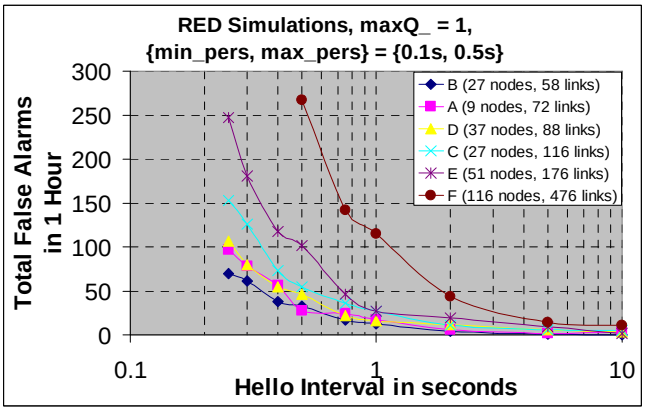

Figure 6 False Alarm Occurrence in Different Topologies For Different *HelloInterval* Values; RED Simulations with *maxQ_*= 1.

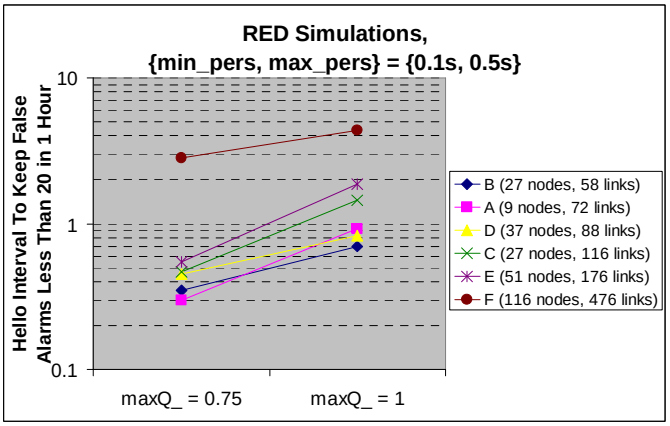

Figure 7 Optimal *HelloInterval* Values for Different Topologies for Different Congestion Levels.

TABLE III. FAILURE DETECTION TIME (FDT) AND FAILURE RECOVERY TIME (FRT) FOR A ROUTER FAILURE ON TOPOLOGY C WITH DIFFERENT *HELLOINTERVAL* VALUES. (RED SIMULATIONS WITH *MAXQ_*=1 AND {*MIN_PERS, MAX_PERS*} = {0.2S, 2S})

Hello	Seed 1		Seed 2		Seed 3	
	FDT	***FRT***	***FDT***	***FRT***	***FDT***	***FRT***
10s	32.08s	36.60s	39.84s	46.37s	33.02s	38.07s
2s	7.82s	11.68s	7.63s	12.18s	7.79s	12.02s
1s	3.81s	9.02s	3.80s	8.31s	3.84s	10.11s
0.75s	2.63s	7.84s	2.97s	5.08s	2.81s	7.82s
0.5s	1.88s	6.98s	1.82s	6.89s	1.79s	6.85s
0.25s	0.95s	10.24s	0.84s	6.08s	0.99s	13.41s