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— **A network running OSPF takes several tens of seconds to recover from a failure, using the current default parameter settings. The main component of this delay is the time required to detect a failure using the *Hello* protocol. Reducing the value of the *HelloInterval* can speed up the failure detection time. However, too small a value of the *HelloInterval* can result in an increase in network congestion, potentially causing multiple consecutive Hellos to be lost. This can lead to a false breakdown of adjacencies 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 occurrences of false alarms within acceptable limits? We examine the impact of both network congestion and the network topology on the optimal value for the *HelloInterval*. Additionally, we investigate the effectiveness of faster failure detection in achieving fast failure recovery in OSPF networks.**

Keywords—Failure Recovery; OSPF, routing stability

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 several tens of seconds to detect the failure and re-establish a consistent view of the new topology. During the transient period, delivery of packets to the destinations may be disrupted (packets may be lost or delayed.) The disruption may not be acceptable to delay or loss-sensitive applications. This motivates us to examine how to reduce the time that OSPF takes to recover from failures. OSPF has been designed with a view to limit the protocol's processing requirements and ensure stable operation over networks of reasonably large scale. However, the processing power of routers has improved considerably over past several years. Furthermore, service providers generally limit the size of a single OSPF area. Hence, the OSPF protocol specifications, designed for large networks and slow routers, may seem overly conservative for current networks. This naturally begs the question of how do we adapt the parameters of the OSPF protocol to achieve faster failure recovery.

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 to its neighbor is 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 their respective forwarding tables. 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 a *HelloInterval* value 10 seconds and a *RouterDeadInterval* value 40 seconds, the failure detection time 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 (Link State Update) *LSUpdate* packets sent down an interface. Once a router receives a new LSA, it schedules an SPF calculation. Since the SPF calculation using Dijkstra's algorithm [3] constitutes a significant processing load, a router typically waits for some time (*spfDelay* - typically 5 seconds) to let any additional LSAs arrive, before doing the SPF calculation on a batch of LSAs. Moreover, routers place a limit on the frequency of SPF calculations (governed by a *spfHoldTime*, typically 10 seconds, between successive SPF calculations), which can introduce further delays.

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 not always available. Hence, routers use the *Hello* protocol within the OSPF protocol itself to detect the loss of adjacency with a neighbor. The *Hello* protocol operates through a periodic exchange of

Hello messages between neighboring routers. A router declares its adjacency with a neighbor to be down if it does not receive a *Hello* message from its neighbor for more than the *RouterDeadInterval*. This can happen if the link between the router and the neighbor is down, or the neighboring router is no longer functional. To avoid a false indication that an adjacency is down 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. Reducing the *HelloInterval* can substantially reduce the Hello protocol's failure detection time. 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 network congestion will lead to loss of several consecutive *Hello* messages and thereby cause a false alarm that the adjacency between routers is lost, 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 SPF calculations by all the routers in the network, to avoid the supposedly down link. This false alarm would soon be corrected by a successful *Hello* exchange between the affected routers, which then causes a new set of LSAs to be generated, and possibly new path calculations by the routers in the network. Thus, false alarms cause an unnecessary processing load on routers and sometimes lead to temporary changes in the path taken by network traffic. This can have a serious impact on the QOS achieved in the network. If false alarms are too frequent, routers have to spend considerable 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 overloads on router CPUs will ultimately result in complete meltdown of the routing function 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 failure recovery from network, while limiting the occurrence of false alarms. This assessment is done through simulations on network topologies of commercial ISPs [4] using a detailed implementation of the 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 OSPF protocol. 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 performed by the routers. We examine how the network topology influences the occurrence of false alarms. Finally, we evaluate the benefit of faster detection of network failures in achieving faster failure recovery in a network using OSPF.

We briefly survey the existing literature on speeding up the failure recovery in OSPF and IS-IS protocols. There has been previous work on reducing the *HelloInterval,* and the impact of network congestion in causing false alarms. Alaettinoglu et al. [6] proposed reducing the *HelloInterval* to the millisecond range, to achieve sub-second recovery from network failures but did not consider the side effects of reducing the *HelloInterval*. Shaikh et al. [7] used a 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 falsely declared down, and then back up again. However, this work did not study the network wide generation of false alarms due to 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. They also considered the tradeoff between faster failure detection and increased frequency of false alarms. They found 275milliseconds to be an optimal value for the *HelloInterval,* providing fast failure detection while not resulting in too many false alarms. However, their work did not consider the impact of different levels of network congestion and the network topology characteristics on the optimal *HelloInterval* value, which we believe to be critical. 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 large burst of LSAs, and cannot 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 the 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 the . These include generation of false alarms.

II. Experimentation Methodology

We implemented substantial extensions to the OSPF routing model [10] currently available in NS2 simulator. These include incorporation of 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 that affect the functioning of the OSPF protocol in operational networks. Some of these delays are configurable while some have a fixed value. Others 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 [11][12]. In our experiments, we used the standard or the typical values for the different delay parameters (except *HelloInterval* and *RouterDeadInterval*).

Rather than using actual packet flows to create congestion, we used realistic models to achieve the same effects. This a was based on a need to keep the processing and running time of the simulations reasonable, and the lack of availability of the actual link traffic loads on the individual topologies. The congestion model used in our simulations emulates the behavior of *Random Early Drop (RED)* [13] and *droptail* buffer management policies of routers. With 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* when the *qlen* is larger than *max_th.* If *qlen* exceeds

1, the packet drop probability = 1, i.e., all 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 within the range {*min_pers, max_pers*}. This is to emulate the slow variation of *qlen*, which is an exponential moving average. The values used for RED in the simulations are *min_th* = 0.25, *max_th* = 0.75 and *max_p* = 0.1. The congestion level in the simulation is controlled by the parameter *max_q* and the range {*min_pers, max_pers*}. As *max_q* is increased, higher packet drop rates in the network are 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 ratio of the instantaneous buffer occupancy to the buffer size. A new value is assigned to the *qlen* associated with each router interface every time a new OSPF packet arrives. The packet drop probability = 0, unless *qlen is* greater than 1 (i.e., buffer is full) in which case the packet drop probability is 1. For the droptail simulations, the packet drop rate = *(max_q -1)/max_q*, (with the constraint, *max_q > 1*).

The simulations were done using a number of topologies corresponding to real IP backbones of several commercial ISPs, obtained from [4]. TABLE I lists some of the characteristics of these topologies. Most of the topologies are irregular. However, topology *A* is a pure mesh and topology *B* has a star-like structure.

III. SIMULATION RESULTS

We first examine how reducing the *HelloInterval* causes more false alarms and how an 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 an interval ranging from 100ms to 500ms. Different curves in the figure correspond to different congestion levels (modeled by the parameter *max_q*). As expected, false alarms become more frequent with a decrease in the *HelloInterval* value and an increase in network congestion level. Further, the impact of an increase in the congestion level is more severe for lower *HelloInterval* values. Clearly, the optimal value of the *HelloInterval* depends on the expected congestion level in the network, and an understanding of what constitutes an acceptable limit on the frequency of false alarms. Assuming that no more than 20 false alarms in an hour can be tolerated network-wide, and if the average buffer occupancy (*qlen)* at a router interface 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* increases further. Again, the increase in false alarms is more severe for lower *HelloInterval*s; hence there is a need to be conservative while reducing *HelloInterval* value. The droptail simulations results are shown in **Figure 3**. The different curves in **Figure 3** show results for *max_q* values of 1.02, 1.05 and 1.1, which correspond to packet drop rates of 1.96%, 4.76% and 9.09% respectively. With around a 10% overload on the system, any *HelloInterval* value less than 10 seconds leads to an unacceptable number of false alarms.

LSAs generated as the result of a false alarm will be flooded throughout the network and lead to new SPF calculations 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 by a router for each false alarm. Some times, for large *HelloInterval* values, a false alarm causes two SPF calculations to be done at each router; the first one in response to an adjacency breakdown, and second one in response to the re-establishment of the adjacency following the successful exchange of *Hello* messages between the routers affected by the false alarm.

TABLE I. 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

For smaller *HelloInterval* values, a broken adjacency is generally re-established soon enough, so that only one SPF calculation (scheduled every *spfDelay* = 5 seconds, after receiving the false alarm) is required to take care of both changes. Thus, for smaller *HelloInterval*s, since the 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. 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 for one particular topology (topology C), in response to false alarms. The false alarms generated for this topology 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.

Next, we examine the impact of the 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 **Figure 5** and **Figure 6**, which show the false alarm occurrence during a 1 hour period for different topologies for congestion levels created by *max_q* values of 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, the 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 when all the neighbors of the failed router have detected the failure. We also observed the failure recovery time, i.e., the time when all the routers in the network have finished their SPF calculation and forwarding table update in response to the failure. The simulations used the RED packet drop model with $max\_q = 1$ and average buffer persistency, $\{min\_pers,\ max\_pers\} = \{0.2s,\ 2s\}$. Multiple simulations with different seed values were run. Table 2 shows some typical and interesting cases. As expected, the failure detection time is within a range of 3 to 4 times the *HelloInterval*. Once a neighbor detects a 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 receipt of the LSA. 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 2 show the recovery to take place much sooner than 6 seconds. This is mainly because the reported failure detection times are the 'latest' (most recent) 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 an actual router failure. Often, the failure recovery takes 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 frequency of SPF calculation by the routers is limited by *spfHoldTime* (typically 10s), which can delay the new SPF calculation in response to a router failure. There is the delay caused by *spfDelay.* Finally, routers with a poor connectivity may not get the LSAs in the first attempt because of a loss due to congestion. Such routers may have to wait for 5 seconds (*RxmtInterval*) for the LSAs to be retransmitted.

The results in Table 2 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 speed up the failure recovery further 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 taking 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 routing stability, when failure detection is no longer an issue constitutes the logical next step to the work presented in this paper.

IV. Conclusion

With the current default settings of the OSPF parameters, a network may take several tens of seconds before recovering from a failure. The main component in this delay is the time required to detect a failure using the *Hello* protocol. Reducing the value of the *HelloInterval* can speed up failure detection time. However, too small a value for the *HelloInterval* will lead to excessive false alarms in the network, 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 the *HelloInterval* in 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 may not be advisable to reduce it to the millisecond range. 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] http://networks.ecse.rpi.edu/~sunmin/rtProtoLS/

[11] 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.

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

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

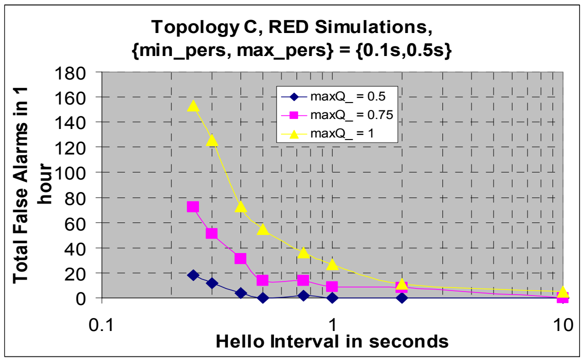

Figure 1 False Alarm Occurrence in Topology C for Different HelloIntervals & Congestion Levels.

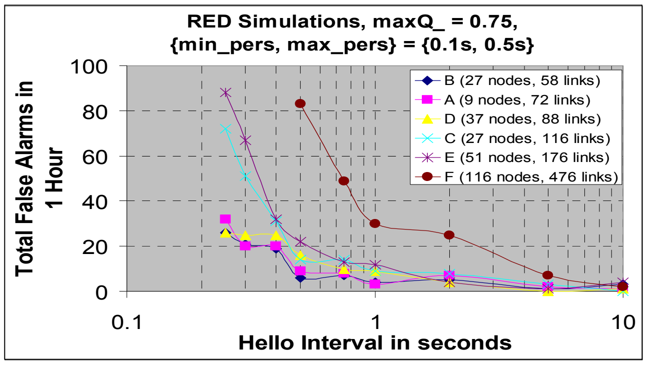

Figure 5 False Alarm Occurrence in Different Topologies For Different HelloIntervals; RED Simulations with max_q=0.75.

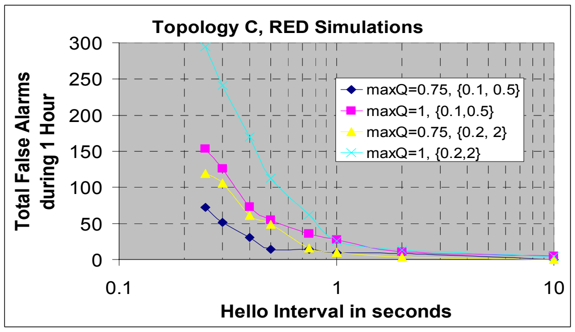

Figure 2 Change in False Alarm Frequency As High Drop Rates Persist for Longer Durations.

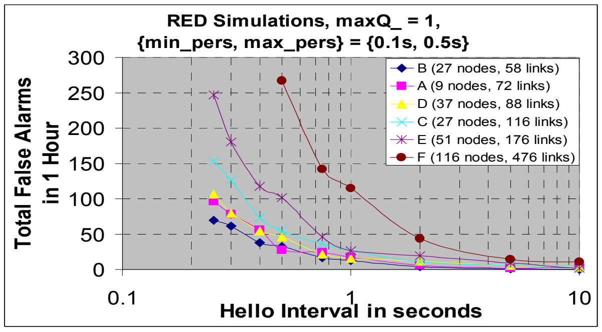

Figure 6 False Alarm Occurrence in Different Topologies For Different HelloIntervals; RED Simulations with max_q= 1.

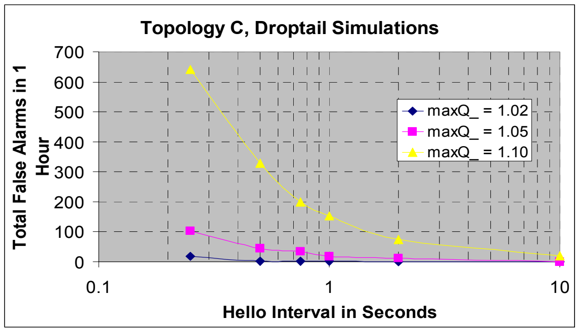

Figure 3 False Alarm Occurrence in Topology C in Droptail Simulations.

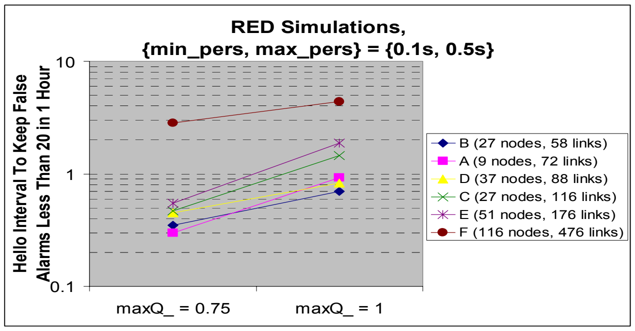

Figure 7 Optimal HelloInterval for Different Topologies.

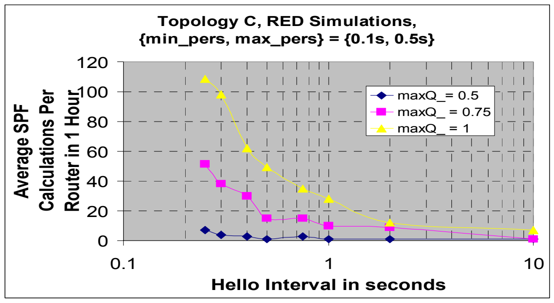

Figure 4 Average SPF Calculations per Router in Topology C Due to False Alarms Shown in Figure 1.

TABLE II. Failure Detection & Recovery Times (FDT,FRT) For a Router Failure on Topology C. (RED,max_q=1 , {min_pers,max_pers}={0.2S, 2S}).

Hello Intvl	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