Stability Issues in OSPF Routing

Anindya Basu

Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974

basu@research.bell-labs.com

Jon G. Riecke

Aleri Inc.
41 West 12th Street
New York, NY 10011

riecke@home.com

Abstract

We study the stability of the OSPF protocol under steady state and perturbed conditions. We look at three indicators of stability, namely, (a) network convergence times, (b) routing load on processors, and (c) the number of route flaps. We study these statistics under three different scenarios: (a) on networks that deploy OSPF with TE extensions, (b) on networks that use subsecond HELLO timers, and (c) on networks that use alternative strategies for refreshing link-state information. Our results are based on a very detailed simulation of a real ISP network with 292 nodes and 765 links.

1. Introduction

A number of new uses for Interior Gateway Protocols (IGPs) have led to protocol extensions and enhancements. For example, new methods for fast switching and traffic engineering (TE) in IP networks (such as Multi Protocol Label Switching (MPLS) [21]) need to know what resources are available in the network. This enables them to compute traffic-engineered paths with QoS guarantees. In order to address these requirements, various extensions have been proposed for IGP protocols such as OSPF [18] and IS-IS [19]. Each such IGP protocol is a link-state protocol that maintains consistency of the link-state database at every node by exchanging information about the state of the network in the form of Link State Advertisements (LSAs). This information is exchanged both periodically as well as when a network-state change is detected. It is well known that link-state protocols, in their pure state (i.e., without TE extensions) are stable. In other words, if there is a change in the network state (e.g., a link goes down), all the nodes in the network are guaranteed to converge to the new network topology in finite time (in the absence of any other events). This is an important property since it prevents route flaps and other undesirable occurrences.

However, adding TE extensions to IGPs complicates the issue. In the pure state, IGP protocols send out state-change update messages only when a failure occurs in the network. In the case of IGPs with TE extensions, state-change update messages are also sent out when there is a change in the resources (e.g., link bandwidth) available in the network. Furthermore, when network failures occur, traffic-engineered paths may need to be rerouted. As the amount of control information exchanged and the frequency with which it is exchanged increases, it becomes increasingly likely that the network will be driven to unstable operating regimes. For example, there could be scenarios where a combination of network failures and resource changes lead to bursts of update messages that prevent the network from converging to a stable state.

In this paper, we focus on the OSPF protocol and study some of these stability-related issues. We use simulation as our primary tool. Our simulations incorporate a detailed OSPF implementation based on RFC 2328 [18] and a realistic processor model based on a commercially available router. The simulations were run on a real ISP network topology with 292 nodes and 765 links.[1]

In order to study network stability, we attempt to answer three specific questions, namely:

- What effect do TE extensions have on the stability of a network that is running OSPF? In particular, what are the processor loads like when the network is perturbed, how many route flaps occur, and how long does it take for the network to converge to a steady state?
- How can the network convergence time be speeded up? More specifically, we look at current proposals involving subsecond HELLO timers and study their effect on routing processor loads and convergence times.
- How does OSPF behave under normal conditions? In particular, we study issues that arise from synchronization of LSA refresh times (i.e., the periodic exchange of LSAs) and suggest alternatives to evenly distribute the LSA refresh traffic over time.

Our findings include the following. First, the OSPF-TE protocol does converge to steady state after (multiple) failure(s), despite the added complexities (and overheads) of rerouting traffic-engineered paths. On the other hand, multiple concurrent failures may result in high processor loads and increased number of route flaps for

[1] Some minor changes have been made to the topology for reasons of confidentiality.

SIGCOMM'01, August 27-31, 2001, San Diego, California, USA

short time periods. In general, OSPF-TE appears to be quite robust. Second, current proposals for subsecond HELLO timers do result in significant improvements in convergence times — at the same time, the processor loads stay within reasonable ranges. Finally, we suggest a way of avoiding LSA refresh synchronization using randomization and show its effectiveness using simulations.

The contributions of this paper are as follows. To our knowledge, this is the first work that attempts to study the stability of an IGP protocol that incorporates TE extensions. In order to ensure the transferability of our results to real life networks, we use an implementation that duplicates the specifications in RFC 2328 and run our simulations over a real network topology consisting of 292 nodes and 765 links. While there have been simulation studies of a related nature in the past, they have typically used much smaller network topologies consisting of 10s of nodes. Finally, as mentioned earlier, we incorporate a very detailed processor model based on a real life commercial router. This provides very accurate estimates of queueing and processor delays related to scheduling as well as processor utilization times. Thus, we can assert with reasonable confidence that our simulations are as true-to-life as possible. We believe that our results will provide adequate guidelines for network operators deploying OSPF with TE extensions in their networks.

The rest of this paper is organized as follows. We first provide (Section 2) a brief overview of OSPF-TE.[2] In Section 3, we discuss in greater detail what we mean by network stability and the scenarios that we want to study. In Section 4, we describe the experimental methodology, including the simulator that we use, the processor model, and the network topology. This is followed by Sections 5, 6, and 7, where we describe some of the experimental results and their implications. In Section 8, we present a summary of our findings. We discuss related work in Section 9, followed by conclusions and future work in Section 10.

2. A Brief Overview of OSPF-TE

OSPF-TE [13] stands for the Open Shortest Path First protocol with Traffic Engineering extensions. Traffic Engineering (TE) refers to techniques for optimizing the performance of operational networks. In particular, traffic engineering attempts to ameliorate congestion-control problems by allocating resources (e.g., bandwidth) efficiently.

The TE extensions to OSPF are described in detail in [13]. We summarize the aspects that are relevant for our purposes. In essence, the OSPF-TE protocol uses opaque Link State Advertisements (LSAs) to disseminate traffic-engineering information. Such a (traffic-engineering) LSA carries a special type of TLV (type-length-value structure) called a link TLV that encodes information about a specific link. This includes (among other things) the maximum bandwidth on the link, the maximum reservable bandwidth on the link (which could be different from the maximum bandwidth if links are oversubscribed) and the unreserved bandwidth for 8 different priority levels. Flooding is used to distribute this information, which is required by the nodes to compute traffic-engineered paths (e.g., end-to-end MPLS paths with QoS guarantees).

A key *implementation* component of OSPF-TE (though not part of the specification in [13]) is the triggering module. The triggering module decides when to advertise changes in link states, especially when the unreserved bandwidth on a link changes. Advertising changes helps to maintain the consistency of the link-state database on each router. In our work, we use two types of triggering: periodic and threshold-based. Periodic triggering is used to refresh self-originated LSAs whose age has reached the LSRefreshTime [18] limit — this kind of refreshing is done to maintain soft state in the routers.

Threshold-based triggering is used when the network state changes. The network state changes when one of the following events occur: the state of a node changes (node failure or repair), or the state of a link changes (link failure, repair, or change in unreserved bandwidth by more than a pre-determined amount). The function of threshold-based triggering is to generate LSAs when the magnitude of a state change crosses a pre-determined threshold. For example, a threshold-based triggering policy could generate LSAs when the unreserved bandwidth on a link changes by more than 10%.

Finally, we briefly describe the signaling mechanisms used in conjunction with OSPF-TE to set up traffic-engineered paths. Thus far, two protocols have been proposed for this purpose, RSVP with Traffic Engineering extensions (RSVP-TE) [6] and Constrained-based Routing Label Distribution Protocol (CR-LDP) [3]. RSVP-TE is a soft-state signaling protocol that uses the RESV and PATH messages in RSVP for a two stage path-setup process. CR-LDP is a hard-state signaling protocol that runs over TCP and uses LDP request and response messages for setting up traffic-engineered paths. In either case, the signaling protocols use the traffic-engineering information disseminated by OSPF-TE for making path-setup decisions. For the simulations in this paper, we use a soft-state signaling protocol that is similar to RSVP-TE but uses a slightly different message format.

In summary, the TE extensions to OSPF provide mechanisms to ensure that all the nodes in the network have a consistent view of the traffic-engineering parameters associated with the network. This is accomplished by advertising special traffic-engineering parameters associated with each link periodically, as well as when certain critical events occur. The traffic-engineering parameters are used by the signaling protocols for making path-setup decisions.

3. Issues in Network Stability

In this section, we describe our notion of network stability and how we measure it in our experiments. We study three indicators of network stability: the network convergence time, the routing load on processors, and the number of route flaps caused by failures.

3.1 Network Convergence Time

The network convergence time is the time taken by all the OSPF routers in the network to go back to steady state operations after there is a change in the network state (in the absence of any further failures). For example, when a link failure occurs, the network convergence time is the total time taken for all the routers to update their link-state databases to reflect the fact that the failed link is down and to reroute all the traffic-engineered paths (if any) around the failure. The network convergence time is determined by the diameter of the network (maximum number of hops between any two routers), congestion, routing load on processors, and several other factors. A low convergence time indicates a stable network. This is because the network can quickly come back to steady state when perturbed.

[2] The informed reader can skip this section.

There are two components to the network convergence time when TE extensions are used. First, we have the propagation component, which is the time for the new information to be flooded across the network after a node/link failure/repair occurs or there is a change in the unreserved bandwidth on a link. This determines the time for all the routers to update their link-state databases.

Second, we have the reroute component, which is the time taken to reroute all the traffic-engineered paths when a node or a link fails. Even if alternate routes have been precomputed, there are overheads involved in tearing down the old route and setting up the new one. Clearly, a smaller reroute time indicates a more stable network since it can resume forwarding data traffic with less disruption.

In addition, we also evaluate proposed techniques that aim to speed up network convergence. More specifically, we study the effects of subsecond HELLO timers [1] on network convergence times and routing processor loads. Using subsecond HELLO timers would make it possible to detect link/node failure/repairs faster than is currently possible (at Layer 3) and therefore reduce network convergence time. However, this imposes an additional cost on the network because more packets are now sent out per time unit. The aim is to find the right balance in setting the HELLO timers.

3.2 Routing Load on Processors

The routing load on processors is a measure of how much time a router spends in processing control packets from the routing protocol. A high routing load is a possible indication of incipient network instability. This is because a high routing load is either caused by frequent changes in the network state (that generates a heavy volume of control traffic) or due to slow processing of control packets. In either case, ingress queues get filled and incoming packets get dropped. In particular, incoming control packets get dropped which causes timeouts, which in turn generates more control packets and higher routing loads. Clearly, such scenarios can ultimately lead to a network meltdown. Note that even high-end routers that have separate data and control paths are also subject to this kind of meltdown. This is because higher routing loads can lead to packet drops from the internal queues that are used to store incoming control packets before they are processed by the route control processor.

We look at the routing load under two different conditions. First, we study the routing load under steady state (i.e., when there are no changes in the network state). Under these conditions, there are two kinds of packets that are sent out: periodic HELLO packets (to indicate to a neighbor that a router is alive), and periodic refresh LSAs (to refresh the soft state on other routers). We look at the routing load generated by periodic refresh LSAs. Each router refreshes its self-originated LSAs whenever the age of the LSA reaches LSRefreshTime [18]. Since the LSA age is incremented by one every second, an LSA is refreshed every LSRefreshTime seconds. A major concern in this scenario is the synchronization of the refresh intervals of multiple LSAs on multiple routers. This causes all the synchronized routers to refresh (and therefore flood) their LSAs at the same time. Consequently, the routing load on the network experiences periodic spikes. We attempt to formulate and explore strategies to ensure that this kind of routing load is evenly distributed over time.

Second, we study the routing load when the network is perturbed, i.e., when the network state has changed in some way. A network state change causes fresh LSAs to be flooded from the point of origin of the change so that the link-state databases at the various routers can be brought up-to-date. Clearly, this kind of flooding adds to the routing load and may cause a network to become unstable, especially if multiple state changes occur within a short time interval.

3.3 Route Flaps

The third parameter we study is the number of route flaps caused by a network failure. Route flaps refer to routing table changes in a router, usually in response to a network failure or a recovery. In some sense, the number of route flaps characterizes the intensity of the perturbation in the network. This is in contrast to the duration of the perturbation, which is characterized by the convergence time. For obvious reasons, a large number of route flaps in a short time interval adversely affects network stability.

4. Experimental Methodology

In this section, we describe the methodology for our study of OSPF stability. Ideally, we would have liked to do an analysis using a control-theoretic model of a network running OSPF. Indeed, an early work [14] does mention a control-theoretic approach. However, given the size of the network that we are proposing to study, and the added complexities of TE extensions, the problem becomes intractable very fast. Therefore, we decided to use simulation tools to perform these studies. In the rest of this section, we first describe the simulation tool, and then describe the network topology that we used.

4.1 The VENUS Simulator

We used the VENUS simulator for our experiments with OSPF-TE. The VENUS simulator is an event-driven, packet-level simulation tool based on the MaRS simulator from the University of Maryland [2]. VENUS retains the basic event and packet-handling mechanisms from the MaRS simulator. To this, an implementation of OSPF based on RFC 2328 [18] has been added along with TE extensions. VENUS also incorporates a detailed processor model based on a commercially available OSPF router. This enabled us to accurately model queueing and message processing delays as well as processor utilizations. The entire effort resulted in about 8500 lines of C code (over and above the original MaRS code).

The structure of the VENUS simulator is very modular and consists of objects. The two main classes of objects relevant to our simulation study are the routing objects and the processor model objects. Each of these objects is associated with a router node in the simulated network. We describe these objects in the next two subsections.

4.1.1 Routing Objects

We use two main types of routing objects: OSPF objects and TE objects. The OSPF objects implement the OSPF protocol for single OSPF areas in great detail, including node startup, adjacency establishment by exchange of database packets and maintenance of adjacencies by periodic HELLO packet exchange. For a detailed description of each of these processes, see RFC 2328 [18].

A TE object implements the TE extensions to OSPF (described earlier in Section 2). In addition, a TE object also implements signaling for path setup. The signaling in this case uses a soft-state

approach similar to RSVP-TE [6] (though the actual packets we send have a slightly different format from that of RSVP-TE packets). Periodic keepalive messages are used to maintain path-specific state on nodes. If the path-specific state has not been refreshed for some (predetermined) time, the path is torn down. Furthermore, when a network failure is detected on a path, the signaling component on the source of the path recomputes and reroutes the path around the failure.

4.1.2 Processor Model

Since we want to measure processor utilization, we have built in a fairly accurate model of the routing processor and its operating system. Each router in the simulated network has an associated processor, and the simulator records the amount of time that each processor is busy or idle.

The processor model incorporates the basic aspects of a non-preemptive, real-time operating system. There are four basic task types, each with its own priority, namely, OSPF-TE packet processing, Dijkstra calculation, expiration of timers, and path setup and clear. The priority levels for the different tasks are as follows (in increasing order): OSPF-TE packet processing (lowest), path setup and clear, Dijkstra calculation, and timer expiration (highest).

Each processor has a task queue associated with each task type, i.e., four queues in all. Once a task begins to run, it runs to completion with full utilization of the processor — running tasks cannot be pre-empted even by higher priority tasks. When complete, the system time is updated using the time it takes to complete the current task, and the next task with the highest priority is scheduled; if there is no task, the processor becomes quiescent.

We now describe each of the task types. As its name implies, an OSPF-TE packet processing task is invoked when an OSPF packet (possibly carrying TE information) arrives at a routing node. In particular, such a task is responsible for processing HELLO packets, database exchange packets, link-state request packets, link-state update packets and link-state acknowledgment packets. In addition, if any OSPF packets need to be sent in response to a received packet, this task does that as well. For example, if a newly received LSA is to be flooded, the OSPF-TE packet processing tasks builds a packet containing the LSA and schedules it for sending.

The path setup and clear task deals with all the signaling functions. This includes sending path-setup request messages in response to client demands, receiving and processing reservation request and response messages as well as path teardown requests. Furthermore, when a network element (node or link) fails, this task handles the rerouting of any paths around the failed network element.

Note that timers are implemented by scheduling a timer expiration. Timer expirations are handled by a special timer expiration task that runs with the highest priority. This task executes periodically (the period is a configurable parameter in our simulations) and looks at all the unexpired timers associated with the current processor. Each timer whose expiration time is less than or equal to the current time is then fired, and the timer expiration task itself is rescheduled for execution after the fixed time period.

Finally, the Dijkstra calculation task is solely responsible for doing Dijkstra calculations. This task is scheduled when a new or updated LSA is inserted in the link-state database. In our implementation, a Dijkstra task is scheduled at most once every 5 seconds.

We now briefly describe how we estimated the times taken by some of the tasks mentioned above. In particular, we show how the processing and sending times for OSPF-TE packets were calculated. The time taken to process an OSPF-TE packet (t_r) is given by

$$t_r = \begin{cases} t_{ph} + n * t_{up} & \text{if packet is an LSA update packet} \\ t_{ph} & \text{otherwise} \end{cases}$$

where t_{ph} denotes that fixed overhead for any type of OSPF-TE packet (HELLO, Database Exchange, Link-State Request, Link-State Update and Link-State Ack). The t_{up} parameter denotes the overhead per LSA in an update packet and n denotes the number of LSAs in the update packet. The parameter t_{up} for a router LSA is given by

$$t_{up} = \begin{cases} t_c + t_{rc} + k * t_{rv} & \text{if LSA not present in database} \\ t_{rc} + k * t_{rv} & \text{if LSA present but not duplicate} \\ t_{rd} & \text{if duplicate LSA} \end{cases}$$

where t_c is the time taken to insert an LSA in the Link-State Database, t_{rc} is the fixed processing time required per router LSA, t_{rv} is the cost per advertised link in a router LSA, k is the number of advertised links in the LSA, and t_{rd} is the time taken to process a duplicate router LSA.

Similarly, if the LSA is an opaque LSA containing traffic-engineering information, t_{up} is given by

$$t_{up} = \begin{cases} t_c + t_{tc} & \text{if LSA not presennt in database} \\ t_{tc} & \text{if LSA present but not duplicate} \\ t_{td} & \text{if duplicate LSA} \end{cases}$$

where t_c is time taken to insert the LSA in the Link-State Database, t_{tc} is the fixed processing time per opaque LSA, and t_{td} is the time taken to process a duplicate opaque LSA.

The time t_s to send an OSPF-TE packet on an interface is given by

$$t_s = \begin{cases} t_{hf} + n_r * t_{rf} + n_t * t_{tf} & \text{if update packet} \\ t_{af} & \text{if ack packet} \\ 0 & \text{otherwise} \end{cases}$$

where n_r is the number of router LSAs in the update packet, n_t is the number of opaque LSAs in the update packet, t_{rf} is the time taken to add a single router LSA to the update packet, t_{tf} is the time taken to add a single opaque LSA to the update packet, t_{hf} is the fixed overhead to send an update packet on an interface, and t_{af} is the time taken to send an acknowledgment packet. Note that we assume that the time to send other packet types (e.g. HELLO packets) is negligible and approximate it by 0.

We have attempted to be realistic about the time that each task requires. For this purpose, we used timing measurements on a commercially available router to obtain values for the following parameters: t_c, t_{rc}, t_{rv}, t_{tc}, t_{td}, t_{af}, t_{hf}, t_{rf} and t_{tf}. The actual values ranged from 10 microseconds to 1101 microseconds. Individual parameter values have not been shown here due to privacy reasons.

Finally, our processor model assumes that data packets are forwarded by line cards in hardware and control packets are handled by the routing control processor. This is not an unrealistic assumption since most high-end routers are configured in this manner.

The model is not complete. It does not include, for instance, times for computing paths when servicing path-setup requests; we expect that an implementation would calculate many paths at once,

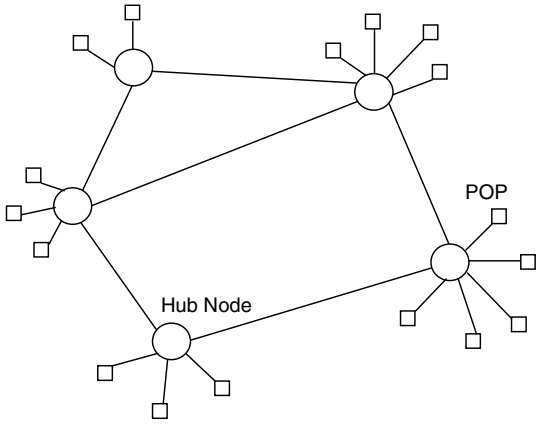


Figure 1: Sample Hub and Spoke Network Topology

Hub Nodes	59
POPs	233
Total Nodes	292
Hub-to-Hub Links	393
POP-to-Hub Links	124
POP-to-POP Links	248
Total Links	765
Max Node Degree	11
Min Node Degree	1
Max Link BW	622 Mbps
Min Link BW	512 Kbps
Max Latency	113ms
Min Latency	10μs

Figure 2: Network Topology Information Summary

and cache these paths for servicing (multiple) future requests. Furthermore, some of the processor parameters may be quite different in products from other vendors. Nevertheless, we believe that the processor model produces results that are applicable in other settings, because the model reflects a common implementation strategy.

4.2 Network Topology

We used a 292 node, 765 link network based on a real ISP network that is spread worldwide. The network essentially consists of a collection of hubs that correspond to major cities. Connected to each hub are a set of nodes that represent local points of presence (POPs), in the form of a hub and spoke configuration. (A sample hub and spoke configuration with 5 hubs is shown in Figure 1.)

In the simulated network, there are a total of 59 hub nodes and 233 POPs. The POPs are at the edge of the network and aggregate traffic from multiple customers. The hubs constitute the network core — they act as aggregators of traffic originating from the POPs that are connected to them. The entire network spans 77 cities worldwide, with multiple hub nodes in major cities (up to 3), and none in the smaller ones. Figure 2 summarizes the network topology information.

The hubs are typically connected by high-speed links with speeds varying between 45Mbps (T3) to 622Mbps (OC-12). POPs are connected to the local hub node and to each other (if they are in the same geographic area) with low speed links. Bandwidths in this case can vary from as low as 256Kbps to 45Mbps (T3). The link latencies were determined by dividing the geographic distance between the endpoints with the speed of light. The latencies varied from 10 microseconds to as high as 113 milliseconds for some slow speed links. For some of the links, the latencies were small enough that we approximated them by 0. The entire network was structured as a single OSPF area with point-to-point links only.

Note that the networks of today are of much higher speeds — they mostly have OC-48 (2.5Gbps) and OC-192 (10Gbps) links in the core. However, the data we had access to is somewhat older and represented the network state then. One way to get around this problem would have been to scale up the link speeds in the network. This would have meant scaling up the bandwidth demands in the path-setup requests as well. We chose to use the data as is since the qualitative nature of the results would have remained unchanged even with scaling.

Furthermore, for privacy reasons, some of the adjacencies in the actual network have been altered such that the simulated network is somewhat different from the real ISP network. Once again, we felt that the nature of the results would not be substantially different if some minor modifications were made to the network topology.

5. Experiments with OSPF-TE

We designed the first set of experiments to answer the first question posed in the Introduction, i.e., what is the effect of TE extensions on processor loads, network convergence times, and route flaps? Looking at processor loads enables us to understand if network state changes (e.g., failures) can cause meltdowns on functioning routers. When looking at network convergence times, we study two parameters: the propagation time and the reroute time (see Section 3.1). Finally, we look at the number of route flaps to get an indication of how disruptive a network failure can be.

We ran the simulation on the topology described in Section 4.2 for a duration of 10,000 simulated seconds (about 2.8 hours). The network nodes were brought up randomly (with a uniform distribution) during the first 1000 seconds. The network was allowed to stabilize for another 1000 seconds before starting the path-setup requests.

For our experiments, we used a set of 10,000 path-setup requests obtained from the same ISP whose network topology we simulated. Each path-setup request consisted of a source, a destination, and bandwidth values for both the forward and the backward directions (i.e., source-to-destination and destination-to-source directions). All the requests were symmetric, i.e., the bandwidths requested in the forward and the backward directions were the same. The range of bandwidth values used in the path-setup requests varied from 604 bps to 4.6Mbps. The table in Figure 3 shows the distribution of path-setup requests.

Furthermore, the path-setup requests were uniformly distributed in time starting from 2000 seconds after the beginning of the experiments. All the path-setup requests had the same priority. At 6000 seconds, one or more node failure(s) were caused. In our simulations, a failed node stopped sending or receiving packets, and all the packets scheduled for processing at the failed node were dropped. Once a node failed, all the traffic-engineered paths with one end-

Bandwidth Range	% of requests
$\leq$ 16K	25.49
$>$ 16K and $\leq$ 32K	25.62
$>$ 32K and $\leq$ 64K	22.82
$>$ 64K and $\leq$ 128K	11.94
$>$ 128K and $\leq$ 256K	6.43
$>$ 256K and $\leq$ 512K	3.95
$>$ 512K and $\leq$ 1M	2.21
$>$ 1M	1.54

Figure 3: Path-Setup Request Statistics

point at the failed node were shut down. Furthermore, all those paths that were using the failed node as an intermediate node were rerouted along a different path.

Note that we could have chosen link failures or changes in available link-bandwidth as the source of the network-state change. We chose node failures since they subsume link failures and are more disruptive than changes in available link-bandwidth.

We studied the effects of three different parameters on network stability, namely, the number of node failures, the message loss rate on each link, and the triggering mechanism. We describe the results for each of these parameters in the following subsections.

5.1 Number of Node Failures

To explore the effect of multiple node failures, we ran experiments where the top 1, 5, and 10 nodes in terms of the number of adjacencies failed concurrently at 6000 seconds. Figure 4(a) shows the processor utilizations (for the most loaded node) for the three cases. Note that for 1 failure, the processor utilization does not show any spikes — however for 5 failures, the processor utilization crosses 50% at the time of failures and for 10 failures, the processor utilization is close to 100% at the time of failures. We also see that for the 5 and 10 node failure cases, the high processor loads repeat at 1800 second intervals due to the synchronization of refresh LSAs. We discuss possible solutions to this problem in Section 7.

Figure 4(b) shows the number of route flaps over time for each failure scenario. Again, for the 1-node failure case, the number of route flaps is an order of magnitude lower than the multiple node failure case(s). (It is almost impossible to see the route flaps for the 1-node failure case because of the scale.) In fact, even for the 2-node failure case (not shown here), we found that the number of route flaps at the time of failure quadruples from the 1-node failure case. Clearly, these figures indicate that even though OSPF-TE is robust with respect to single failures, multiple concurrent node failures interact in subtle ways to cause much more instability in the network. We have been unable thus far to pin-point the nature of this interaction, and have left it as the subject for further study. We also note that despite the large number of route flaps and the high processor loads, OSPF-TE does converge, in the sense that the routes do not continue to oscillate indefinitely. As we see in the next paragraph, there is a finite time after which the network converges (and hence the route flaps die down).

Figure 4(c) shows the effect of multiple failures on convergence times. Although both the propagation and the reroute components show a slight increase with the number of failures (which is to be expected), it is not substantial. In this respect, OSPF-TE appears to be robust as well.

The more interesting case is the comparison with pure OSPF without TE extensions. The bar marked NTE is the propagation component for pure OSPF in the 1-node failure case. We see that adding TE extensions does not significantly add to the propagation times — only about 6 seconds (i.e., about 18% in this case). The three main components of the propagation component are (a) the failure detection time, (b) the flooding time for the failure information, and (c) the overhead due to TE extensions, if any. Clearly, (a) is bounded above by the HELLO timeout period (40 seconds in most implementations, including ours), and (b) is determined by the diameter of the network. These two quantities are approximately the same whether we use TE extensions or not (if the same network is used in both experiments). The quantity (c) is determined by the overheads involved in sending keepalive messages for all the active paths, the rerouting messages for those paths that contain failed node(s) and/or link(s), and the path-setup messages for new paths. Thus, the extra overhead due to the components in (c) is about 18% from our experiments.

On the other hand, the reroute times are in the region of 80 to 84 seconds (depending on the number of failures). This is because the timeout period for the path teardown and reroute process (if the path specific state on a node has not been refreshed in that time) is 80 seconds in our implementations.

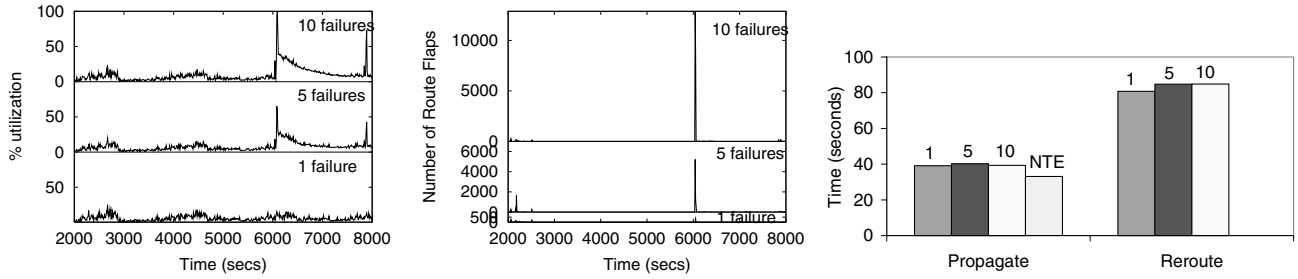
5.2 Message Loss

The second parameter whose effect we studied was the message-loss rate on each link. We used a uniform distribution to generate message drops on each link at the rates of 2, 5 and 10%. At 6000 seconds, the node with the highest adjacency was taken down. Figure 5(a) shows the processor utilization for the most loaded node in each of the three loss scenarios. Interestingly enough, the processor utilization goes up as the loss increases from 2 to 5% and then comes down for 10%. The reason is as follows: as the loss rate goes up from 5 to 10%, more adjacencies are lost due to HELLO timeouts. Consequently, each node sends out fewer messages since it thinks that it has fewer OSPF neighbors. This reduction in messages compensates for the increase in the number of retransmissions. However, when the loss rate goes up from 2 to 5%, the decrease in the number of adjacencies is not enough to balance the increase in the number of retransmissions.

The number of route flaps are shown in Figure 5(b). We find that as the loss rate goes up, the number of route flaps increases. Furthermore, route flaps begin to occur more often — in the 10% loss case, they occur almost all the time. This is because the higher message-loss rates cause adjacencies to be taken down and re-established more often. Thus, higher message-loss rates have a particularly debilitating effect on stability, although a 10% message-loss rate is unrealistically high. For a more realistic 2% message-loss rate, the network appears to be fairly stable. Note here that we do not show any results for convergence times. When messages are lost, it is impossible to distinguish between those LSAs that are sent out as a direct result of a node failure and those that are sent out because of loss-induced timeouts. The same holds for path reroutes.

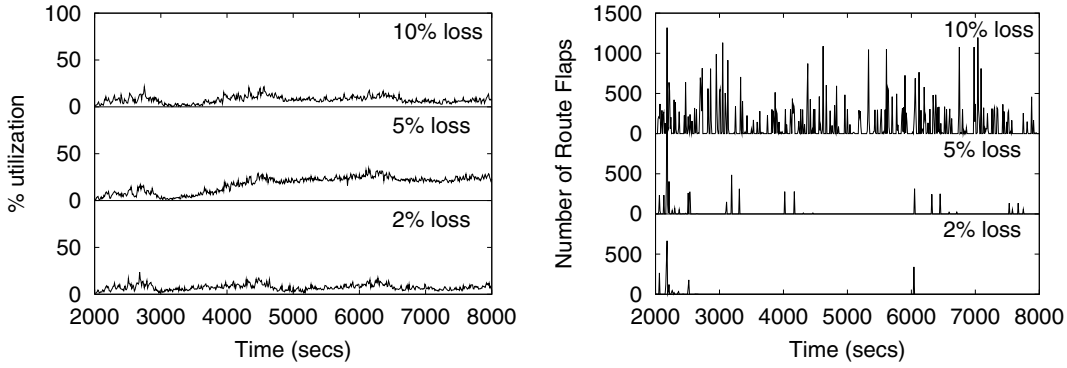
5.3 Triggering Thresholds

The final parameter was the triggering threshold mechanism, of



(a) Processor Utilization (b) Route Flaps (c) Convergence Times

Figure 4: Effect of Multiple Node Failures



(a) Processor Utilization (b) Route Flaps

Figure 5: Effect of Message Loss

which we used two variants. The *relative* threshold mechanism works as follows. Let the bandwidth available on a link l at the time of the last update for l be $b(u)$, and the current available bandwidth on l be $b(c)$. Then an LSA is generated when $\frac{|b(u)-b(c)|}{b(u)} * 100 \geq t$ where t is the threshold percentage. In other words, the change in bandwidth relative to the *previously available bandwidth* has to be higher than the threshold. This implies that as the available bandwidth on a link grows smaller, the bandwidth changes required to produce update LSAs also grow smaller in magnitude.

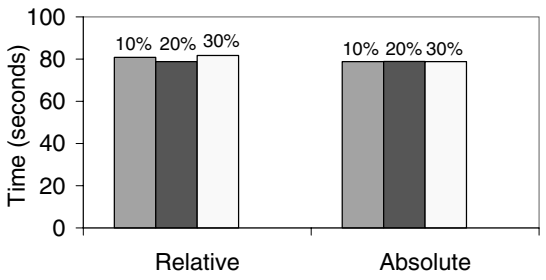
In contrast, the *absolute* threshold mechanism works as follows. Let the total link bandwidth on a link l be B, the bandwidth available on l at the time of the last update for l be $b(u)$, and the current available bandwidth on l be $b(c)$. Then an LSA is generated when $\frac{|b(u)-b(c)|}{B} * 100 \geq t$ where t is the threshold percentage. In other words, the relative change in bandwidth with respect to the *total bandwidth* has to be higher than the threshold. In this method, the number of LSAs sent out grows smaller as the available link bandwidth decreases. In our experiments, we used thresholds of 10, 20 and 30%.

Figure 6(a) shows that changing the triggering threshold or the nature of the threshold mechanism (relative vs absolute) does not affect the reroute times. This is to be expected since a failure immediately triggers an LSA send irrespective of what the triggering mechanism is.[3] However, the triggering mechanism does have an effect on path setups. Figure 6(b) shows that the relative threshold mechanism has fewer rejects than the absolute threshold mechanism. This is because in comparison to the absolute threshold mechanism, the relative threshold mechanism sends out more frequent updates as the available link bandwidths grow smaller, and therefore allows nodes to maintain a more accurate network snapshot. On the other hand, the actual value of the triggering threshold (in the 10 to 30% range) does not appear to have any effect on the number of rejects. Furthermore, the nature or the value of the triggering threshold does not have any significant effect on the processor load. This is shown in Figures 6(c) and (d). (The graphs are for the most loaded processor). Thus, using a relative triggering threshold is preferable since it produces fewer rejects (and more accurate network snapshots) without significantly increasing the processor loads.

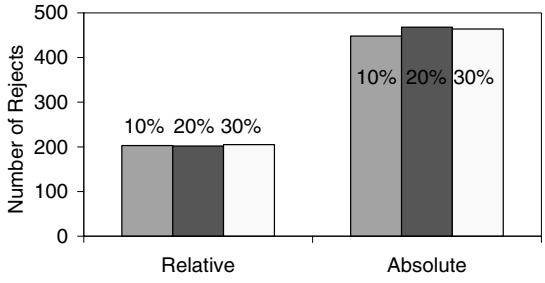
6. Experiments with Convergence Times

The experiments in this section were used to answer the second question posed in the Introduction, namely, can the network con-

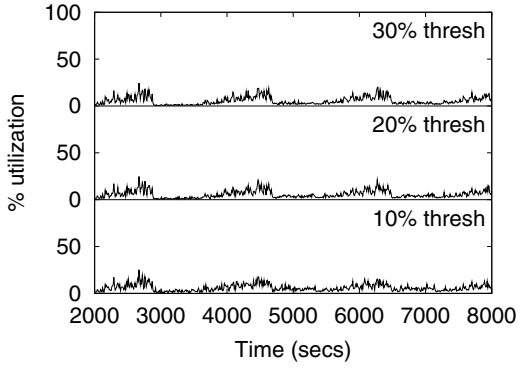
[3]More precisely, it is the detection of a failure that immediately triggers the LSA send, and not the occurrence of the failure. There is a time lag between the occurrence of a failure and its detection that is determined by the HELLO timeout period.



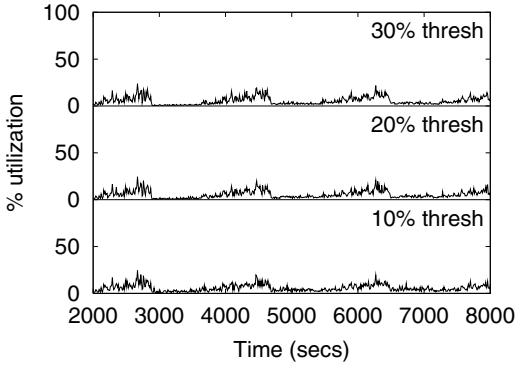
(a) Reroute Times



(b) Path Rejects



(c) Processor Load: Relative



(d) Processor Load: Absolute

Figure 6: Effect of Triggering Thresholds

vergence time (when failures occur) be speeded up? The current OSPF protocol depends on the HELLO timeout mechanism [18] for detecting network failures, which in effect determines the convergence time for the network. The rate at which HELLO messages are sent is configurable, as is the timeout period. In current implementations of OSPF, it is recommended that the routers be configured to send HELLO messages every 10 seconds, and that an adjacency be dropped if no HELLO messages have been seen within the last 40 seconds (i.e., 4 HELLO intervals). A recent proposal [1] has advocated setting the HELLO intervals to subsecond ranges to achieve faster convergence times. Experiments described in [1] show that such modifications to the IS-IS protocol work reasonably well without unduly loading the routers.

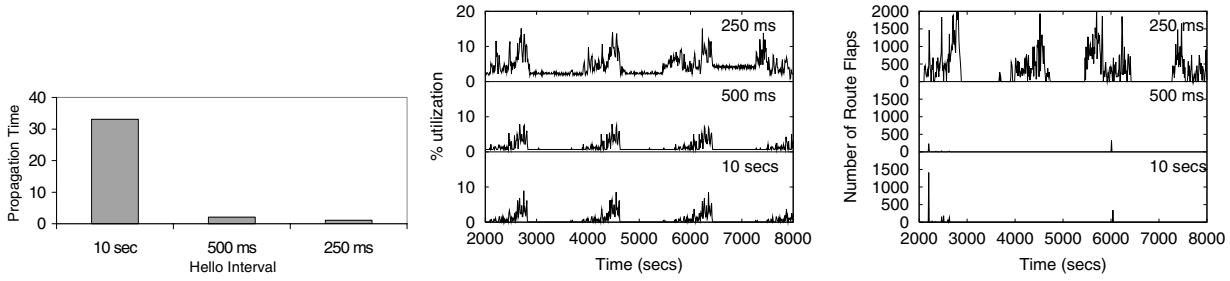
In this section, we explore whether OSPF can also benefit from a similar technique. We ran our experiments with two different values of the HELLO intervals — 250ms and 500ms, with a failure (the node with the highest adjacency) occurring at 6000 seconds. No TE extensions were used.[4] Figure 7(a) shows the convergence time comparisons. We see that compared to the 33 second propagation time for a HELLO interval of 10 seconds (i.e., a 40 second timeout period), a 500 ms HELLO interval yields a 2 second propagation time and a 250 ms HELLO interval yields a 1 second propagation time. In other words, there are substantial improvements in the convergence times if HELLO intervals are in the subsecond range.

The added cost for sending the extra HELLO messages, in terms of processor utilization, is not very high. In Figure 7(b), which shows the processor utilization graphs for the most loaded processor, we see that on an average, the extra HELLO messages add about a 1% extra load for the 500ms case and about a 2% extra load for the 250ms case. The processor load at the time of the node failure (6000 seconds) for the 250ms case does show a spike of about 15% utilization, but that is small enough (in our opinion) to be tolerable for short time periods. These experiments also indicate that the processing of HELLO messages does not account for a large part of the overheads — indeed, most of the overheads come from processing update LSAs.

However, setting the HELLO timeout period to be too low may cause other problems. We find that as we go from a 500ms HELLO interval to a 250ms HELLO interval, the number of route flaps increases about sixfold (see Figure 7(c)). A major reason for this is an increase in the number of HELLO timeouts, on account of setting the HELLO interval timer to be too small. This indicates the existence of an optimal operating point for setting the timers.

In order to narrow down the range for the optimal operating point, we ran further simulations, setting the HELLO interval to 200, 225, 275, and 300ms. We found that the route flap behavior remains the same in the 200 – 250ms range, but resembles the behavior in the 500ms case in the 275 – 300ms range. Thus, setting the HELLO interval in the neighborhood of 275ms gives us a huge

[4] We did try running with TE extensions, but our simulator ran out of memory.



(a) Propagation Times (b) Processor Utilization (c) Route Flaps

Figure 7: Effect of Subsecond HELLO Timers

decrease in the propagation time (from 33 seconds to about 1.09 seconds), without acquiring much overhead in terms of processor utilizations or route flaps.

These results complement the findings for the IS-IS protocol in [1]. However, the IS-IS findings indicate that the number of HELLO timeouts do not increase significantly even for very low timeout values. In the absence of actual figures[5], it is hard to determine what these values are. In contrast, our findings indicate that the number of timeouts (as indicated by the route flap numbers) increases significantly even for modest values of the HELLO intervals (in the 200 – 250ms range). The implications of our findings is that although the convergence times can be reduced by an order of magnitude (from 10s of seconds to seconds) using sub-second HELLO timers, they cannot be reduced to millisecond ranges without substantially increasing the number of route flaps. Without more details about the experiments in [1], it is difficult to determine whether this is an artifact of the protocols (OSPF vs. IS-IS) or of the processor model that we use.

7. Experiments with Refresh Synchronization

In this section, we attempt to answer the last question posed in the Introduction. We first outline the LSA Refresh Synchronization problem, discuss possible solutions, and then describe our experimental results. During steady state operation, each OSPF router refreshes all the LSAs that it originates after every LSRefreshTime [18] seconds of stability. The recommended value for LSRefreshTime is 1800 seconds (or 30 minutes). Periodic refreshment of LSAs guards against errors in the detection of node failures, since LSAs that have not been refreshed within a certain period of time are not included in routing table calculations.

LSA refreshments can cause periodic bursts of traffic due to synchronization [10]. This kind of synchronization occurs because all the routers in the system attempt to refresh their self-originated LSAs over a short time interval. This is an artifact of the periodic LSA refresh scheme — if all the routers in the system come up during a short time interval and start sending out LSAs, the refresh times for these LSAs will also lie within a short time interval. This phenomenon is especially damaging in large networks since a large proportion of routers refreshing LSAs close together in time causes the routing load on the network to spike sharply.

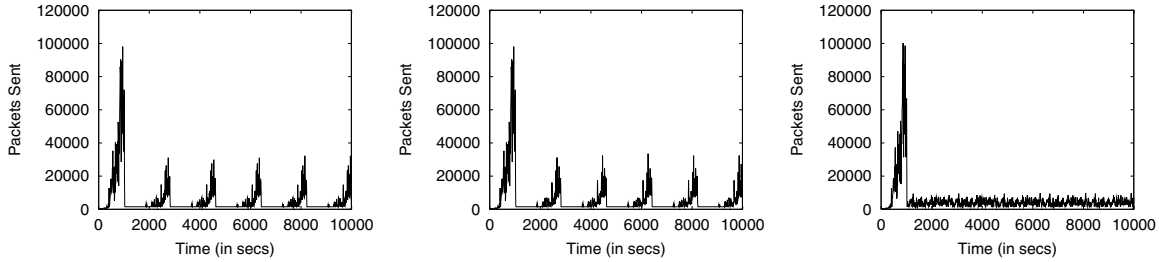
[5] The HELLO intervals in their study is measured in terms of what percentage of the link bandwidth is used by the HELLO packets.

One way to solve this problem is to bring up the routers randomly (instead of all at once) over some (reasonably large) startup time interval. However, a little analysis shows that this technique is not very effective. Consider a network of routers where each router has adjacency k. Furthermore, assume that these routers are brought up randomly over a period of r seconds. The probability that all the k routers that are neighbors of a given router R are up in n seconds ($r \geq n$) is given by $\left(\frac{n}{r}\right)^k$. For example, if n is 500 seconds, r is 1000 seconds, and k is 3, then the probability that all the k neighbors are up in 500 seconds is 0.125. If we want a 0.5 probability that all the k neighbors are up, we must set n to 794 seconds. In a network with higher adjacencies, the number for n (i.e., the time taken for all neighbors to be up with high probability) is even closer to 1000 seconds. This implies that OSPF LSAs stabilize (i.e. stop changing) at the end of the startup phase. This is because it is at this time that all the routers are up and have communicated their presence to every other router.

The crucial observation here is that whenever an LSA changes, its age is reset to zero. In other words, the refresh period is restarted. Thus, it is only when the LSA stabilizes (i.e. stops changing) that the refresh period progresses (in the absence of further changes in the network state). Hence, if most of the LSAs in the system stabilize at around the same time (towards the end of the startup period), the starting points of the refresh periods for all of these LSAs get synchronized. This leads to subsequent synchronization of LSA refreshments.

Note that this observation implies that the synchronization phenomenon is independent of the length of the startup period. Essentially, it depends on when the LSAs originated by a router stabilize, which is determined in turn by how many adjacencies the router has (as we pointed out earlier). Thus, changing the length of the startup period has no effect on LSA refresh synchronization, but reducing the number of OSPF adjacencies that a router has could ameliorate the situation.

A recent proposal by Zinin [26] incorporates a variant of the randomization technique. Zinin advocates a two-fold solution to the problem. First, self-originated LSAs are organized into fixed size groups. All the LSAs in a group are sent out back-to-back, and the groups themselves are spaced out in time. This helps to avoid bursts of traffic initiated by the same node. Second, the refresh time of the initial version of each LSA (i.e., that with sequence number 0) is randomized. For each subsequent version, a small, random "jitter"



(a) Standard OSPF (b) Zinin (c) Randomized

Figure 8: OSPF Refresh Packet Performance

component is added to the normal LSRefreshTime refresh period. The idea here is to prevent the synchronization of LSA refreshes from multiple routers.

In this section, we focus on studying the second part of the solution. The performance of the standard OSPF refresh strategy and that of Zinin's strategy (minus the temporal grouping of LSAs) are shown in Figures 8(a) and (b), respectively. We used the same network topology as before and ran our simulations without TE extensions and without any failures for 10,000 seconds. Since the routers were running OSPF without TE extensions, each router originated a single router-LSA (that corresponding to itself). The nodes were brought up randomly over the first 1000 seconds. The figures show that in terms of the number of packets sent over time, Zinin's algorithm also tends to bunch packets every 1800 seconds.

The reason is the following: even though the refresh period for the *initial* version of every LSA is randomized, it does not have much effect since the initial version of an LSA is typically unstable. In general, the initial version of every LSA is reoriginated multiple times during the router startup phase mainly due to neighbor discovery. Therefore, the router-LSA refresh times at the end of the startup interval (1000 seconds in our experiments) are more or less synchronized. The small random jitter that Zinin's algorithm adds to the refresh periods for subsequent versions still keeps the refresh periods of these LSAs in a very narrow range. This is because the jitter value (as suggested in [26]) is typically small (between 0 and 10 seconds), and hence does not disperse the subsequent refresh times enough.

Our proposal advocates randomizing the refresh time of any LSA that has been modified and needs to be reoriginated. LSAs that have been stable (i.e., have been refreshed at least once) are refreshed at the end of the usual LSRefreshTime seconds. For LSAs that have been modified (e.g., due to the loss of an adjacency), we choose a random number between MinLSInterval and LSRefreshTime and schedule the refreshment after that period of time. Here, MinLSInterval denotes the minimum time that must elapse between two successive originations of an LSA, and is set to the suggested default value of 5 seconds in our implementation. The performance of this algorithm is shown in Figure 8(c). It is clear that this technique is able to disperse the LSA refreshes uniformly over time and spread the router loads (except in the initial startup phase). We believe that using this technique in conjunction with Zinin's technique of grouping LSAs will lead to even better performance.

Finally, we point out that random link failures (and restorations) after the startup phase tend to reduce the synchronization effect that we have observed. A random link failure (or restoration) causes a LSA to be generated when the failure (or restoration) is detected (say, at time t). Subsequently, the LSA is refreshed periodically starting from this time t. Since these times t are randomly distributed for different LSAs, the refreshment periods for the LSAs are no longer synchronized as before. In other words, random changes in link states after the startup period reduce the effect of LSA refresh synchronization.

Once again, note that the random link-state changes must occur *after* the startup period has occurred. Otherwise, the effect of these changes will get nullified by the effect of the network state changes caused by new routers that are coming up during the startup period. Consequently, the LSAs will again stabilize at the end of the startup period and the LSA refreshments will remain synchronized.

8. Result Summary

Based on the results of our simulations, we can make the following recommendations. First, the OSPF-TE protocol appears to be reasonably stable. Our simulations show that adding TE extensions does not change the convergence times significantly, even in the presence of multiple failures. A major potential problem stems from multiple concurrent failures — we saw that the interactions between such failures could lead to very high processor loads and large numbers of route flaps in the short term. In the presence of message losses, the OSPF-TE protocol also proved quite robust — the processor loads do not spike in the presence of a realistic 2% message loss.

Regarding triggering thresholds, our experiments show that relative threshold-based schemes have a clear advantage over absolute threshold-based schemes (fewer rejects, no significant change in processor utilization). Another interesting find was the fact that the triggering threshold does not appear to have any significant effect on the number of path rejects. This could, of course, be an artifact of the network, or the kinds of demands that we used. However, given that we used a real network topology, along with a real demand schedule from that network, we feel reasonably confident that in the 10 to 30% threshold range, we would see similar performance in other scenarios. Thus, it would be advantageous to set the thresholds at the high end of the (10 to 30%) range to reduce the number of messages.

Second, setting HELLO timers to the subsecond range appears to be a good idea. If the HELLO timers are set properly, this tech-

nique does reduce the convergence times by an order of magnitude, without adding significantly to the processor loads. However, making the HELLO interval too low causes too many route flaps due to frequent timeouts, hence the HELLO interval should be chosen carefully to be near the optimal operating point.

Finally, carefully designed randomization goes a long way in evenly distributing the routing load due to refresh synchronization. We advocate the use of our randomization technique in conjunction with Zinin's technique for grouping self-originated LSAs and spacing out the LSA groups in time.

9. Related Work

One of the earliest works on routing stability was done by Khanna and Zinky [14]. This work looked at routing oscillations in the ARPANET and devised changes to the routing metrics that resulted in better stability. The authors also attempted some control-theoretic modeling of the network as a feedback loop in order to understand the stability behavior of the network. Other works have also looked at the design of routing protocols as a feedback control problem. In particular, work by Boel [8] showed that in order to improve both short- and long-term routing performance, it is necessary to disseminate failure information quickly and repair information slowly.

Some later work has used statistics gathered from the Internet to provide quantitative information about the stability of Internet routes. One of the first attempts in this area was [9] which studied the dynamics of Internet routing information. In particular, it looked at the nature of routing updates (what percentage actually provided new information), the size and distribution of routing fluctuations, unreachability cycles, update propagation times, and connectivity transitions. More recently, [16] studied the effect of Internet backbone failures on stability. In particular, this work used data collected at the time of several Internet failures and analyzed their probable origins. This study also looked at the frequency of route flaps and the mean time between the network failures that cause these route flaps. Finally, work in [15] has looked at the latencies in Internet path failovers and repairs and how they are affected by the convergence properties of inter-domain routing.

Separately, QoS-based routing has also been the subject of various studies. In [5], the authors examined the cost of QoS routing. They proposed three parameters that affect the overheads of QoS routing, namely, the triggering update policy, sensitivity of the policy, and the clamp-down timers that limit the update rates. The main focus in this work was to study how the number of successful path setups vary as these parameters are changed. The effect of different triggering thresholds on path setup was also studied by [22]. This work pointed out that the rate of connection blocking (or path-setup failure) is insensitive to the value of the triggering threshold over a wide range of values. Our work extends all of these results to a more realistic setting using a real OSPF implementation, a real ISP network and an accurate processor model based on a commercial router. Furthermore, we also explore the tradeoffs between routing loads on the processor and the accuracy of the triggering mechanisms.

In [4], the authors studied the overheads of QoS routing using a combination of simulations and an OSPF implementation with QoS extensions based on `gated` [17] daemon. They concluded that the cost of QoS routing is well within the limits of the capabilities of modern router technology.

Another work [12] has focused on scalability issues in QoS routing, especially with respect to topology aggregation. Here, the authors used network simulations to show that topology aggregation can reduce routing fluctuation and increase stability.

Link-state protocols (which includes OSPF) have also been studied extensively. In particular, simulations were used in [24] to study the performance of the OSPF Election Protocol and how it affects routing stability. The work in [7] looked at a new way of combining link-state protocols and distance-vector protocols to form a new class of protocols called Link Vector protocols. The authors showed that such protocols have lower overheads and are more amenable to aggregation. Another work [25] modeled and compared the performance of the Bellman Ford algorithm used in OSPF to that of a loop-free distance vector algorithm. In particular, this work looked at stability issues such as paths affected by routing loops when a link or a node changes state.

In [23], the authors looked at the performance of OSPF and BGP (the Border Gateway Protocol [20]) in the presence of traffic overload. This work developed an analytical model of route flap times and adjacency recovery times based on Markov Chains and validated the model using data from a real 3-node network. The main finding in this work was that both route flap times and adjacency recovery times decrease with an increase in traffic overload (and hence message loss). Our results also confirm these findings. Additionally, we show that while routes flap more frequently as message loss increases, the routing load on processors does not change very significantly (in a large network).

Calculation of optimal traffic-engineered paths has also been a subject of research. Most recently, Fortz and Thorup [11] have examined how OSPF link costs can be set such that path calculations yield end-to-end connections that minimize the maximum link utilization. They show that this problem is NP-Complete and propose heuristics that yield solutions reasonably close to the ideal solution.

Floyd and Jacobson [10] were the one of the first to observe the synchronization of periodic routing messages. They used simulations and (Markov Chain based) analysis to show that the transition from unsynchronized to synchronized traffic is very abrupt and suggested that randomization techniques may be used to avoid this kind of synchronization. More recently, Zinin [26] has proposed a scheme for randomizing refresh LSAs in OSPF that we have described earlier in Section 7.

Finally, there has recently been some work in the area of sub-second IGP convergence. In [1], the authors proposed the idea of reducing the HELLO intervals in IGPs to the millisecond range. The goal was to achieve convergence in milliseconds (as opposed to seconds) when the network state changes. They showed, using a combination of simulations and measurements, that this technique works well for the IS-IS [19] protocol.

10. Conclusions and Future Work

In this paper, we have presented a detailed experimental study of stability issues for OSPF routing. Our experiments use a simulation tool based on a realistic implementation of OSPF and a detailed processor model from a commercially available router. The network topology that we have used corresponds (with minor modifications) to a real ISP network with 292 nodes and 765 links. We find that adding Traffic Engineering (TE) extensions to OSPF does not substantially change the stability properties of OSPF. However,

multiple failures that are close together in time can lead to potential instabilities. We also find that using subsecond HELLO timers considerably improves the convergence times for OSPF without significantly adding to processor loads. Finally, we study the LSA refresh synchronization problem and suggest a technique to avoid such synchronized LSA refreshes.

In the future, we would like to extend this study in multiple ways. First, we would like to explore in more detail the interactions between multiple node failures and how they affect stability, and (if possible) suggest modifications to OSPF to handle these scenarios gracefully. This is important since as networks grow in size, multiple concurrent failures become more likely. Second, we would like to study stability issues using a realistic data traffic model that uses bursty traffic to load the network. We did attempt some studies using CBR traffic sources to load the network. However, due to the nature of CBR sources, the queues do not build up enough to affect convergence times. Given that most Internet traffic today is of a bursty nature, we feel that it would be more appropriate to use a traffic source modeled on some heavy-tailed distribution. Finally, we would like to extend our simulations to multiple OSPF areas and also explore interactions with an Exterior Gateway Protocol such as BGP.

11. Acknowledgments

We would like to thank Suvo Mittra and the anonymous referees for their valuable comments.

12. References

[1] C. Alaettinoglu, V. Jacobson, and H. Yu. Towards Milli-Second IGP Convergence. Internet Draft draft-alaettinoglu-isis-convergence-00.txt, IETF, November 2000.

[2] C. Alaettinoglu, A. U. Shankar, K. Dussa-Zieger, and I. Matta. Design and Implementation of MaRS: A Routing Testbed. *Journal of Internetworking: Research & Experience*, 5(1):17–41, 1994.

[3] O. Aboul-Magd et. al. Constraint-Based LSP Setup using LDP. Internet Draft draft-ietf-mpls-cr-ldp-05.txt, IETF, February 2001.

[4] G. Apostolopoulos, R. Guérin, and S. Kamat. Implementation and Performance Measurements of QoS Routing Extensions to OSPF. In *Proceedings of Infocom '99*, New York, NY, March 1999.

[5] G. Apostolopoulos, R. Guérin, S. Kamat, and S. K. Tripathi. Quality of Service Based Routing: A Performance Perspective. In *Proceedings of SIGCOMM '98*, pages 29–40, Vancouver, Canada, August–September 1998.

[6] D. O. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow. RSVP-TE: Extensions to RSVP for LSP Tunnels. Internet Draft draft-ietf-mpls-rsvp-lsp-tunnel-08.txt, IETF, February 2001.

[7] J. Behrens and J. J. Garcia-Luna Aceves. Distributed, Scalable Routing Based on Link-State Vectors. In *Proceedings of SIGCOMM '94*, pages 136–147, London, England, August–September 1994.

[8] R. K. Boel. Dynamic Routing in Delay Networks: Ergodicity, Transients and Large Excursions. In *Proceedings of the 30th Conference on Decision and Control*, Brighton, England, December 1991.

[9] B. Chinoy. Dynamics of Internet Routing Information. In *Proceedings of SIGCOMM '93*, pages 45–52, San Francisco, CA, September 1993.

[10] S. Floyd and V. Jacobson. The Synchronization of Periodic Routing Messages. In *Proceedings of SIGCOMM '93*, pages 33–44, San Francisco, CA, September 1993.

[11] B. Fortz and M. Thorup. Internet Traffic Engineering by Optimizing OSPF Weights. In *Proceedings of IEEE Infocom 2000*, Tel Aviv, Israel, March 2000.

[12] F. Hao and E. Zegura. On Scalable QoS Routing: Performance Evaluation of Topology Aggregation. In *Proceedings of IEEE Infocom 2000*, Tel Aviv, Israel, March 2000.

[13] D. Katz and D. Yeung. Traffic Engineering Extensions to OSPF. Internet Draft draft-katz-yeung-ospf-traffic-03.txt, IETF, September 2000.

[14] A. Khanna and J. Zinky. The Revised ARPANET Routing Metric. In *Proceedings of SIGCOMM '89*, pages 45–56, Austin, TX, September 1989.

[15] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed Internet Routing Convergence. In *Proceedings of SIGCOMM '00*, pages 175–187, Stockholm, Sweden, August–September 2000.

[16] C. Labovitz, A. Ahuja, and F. Jahanian. Experimental Study of Internet Stability and Wide-Area Backbone Failures. In *Proceedings of the Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing*, Madison, WI, June 1999.

[17] The GateDaemon (GateD) Project. Merit GateD Consortium. `http://www.gated.org`.

[18] J. Moy. OSPF Version 2. RFC 2328, IETF, April 1998.

[19] D. Oran. OSI IS-IS Intra-domain Routing Protocol. RFC 1142, IETF, February 1990.

[20] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). RFC 1771, IETF, March 1995.

[21] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. RFC 3031, IETF, January 2001.

[22] A. Shaikh, J. Rexford, and K. G. Shin. Evaluating the Overheads of Source-Directed Quality-of-Service Routing. In *Proceedings of the 6th IEEE International Conference on Network Protocols*, Austin, TX, October 1998.

[23] A. Shaikh, A. Varma, L. Kalampoukas, and R. Dube. Routing Stability in Congested Networks: Experimentation and Analysis. In *Proceedings of SIGCOMM '00*, pages 163–174, Stockholm, Sweden, August–September 2000.

[24] D. Sidhu, T. Fu, S. Abdallah, and R. Nair. Open Shortest Path First (OSPF) Routing Protocol Simulation. In *Proceedings of SIGCOMM '93*, pages 53–62, San Francisco, CA, September 1993.

[25] W. T. Zaumen and J. J. Garcia-Luna Aceves. Dynamics of Distributed Shortest-Path Routing Algorithms. In *Proceedings of SIGCOMM '91*, pages 31–42, Zurich, Switzerland, September 1991.

[26] A. Zinin. Guidelines for Efficient LSA Refreshment in OSPF. Internet Draft draft-ietf-ospf-refresh-guide-01.txt, IETF, July 2000.