Experience in Black-box OSPF Measurement

Aman Shaikh, Albert Greenberg

Abstract— **OSPF (Open Shortest Path First) is a widely used intra-domain routing protocol in IP networks. Internal processing delays in OSPF implementations impact the speed at which updates propagate in the network, the load on individual routers, and the time needed for both intra-domain and inter-domain routing to reconverge following an internal topology or a configuration change. An OSPF user, such as an Internet Service Provider, typically has no access to the software implementation, and no way to estimate these delays directly. In this paper, we present black-box methods (i.e., measurements that rely only on external observations) for estimating and trending delays for key internal tasks in OSPF: processing Link State Advertisements (LSAs), performing Shortest Path First calculations, updating the Forwarding Information Base, and flooding LSAs. Corresponding measurements are reported for production routers from Cisco Systems. To help validate the methodology, black-box and white-box (i.e., measurements that rely on internal instrumentation) are reported for a open source OSPF implementation, GateD.**

Keywords— **Routing, OSPF, black-box measurements, SPF calculation**

I. Introduction

OSPF is used widely as an intra-domain routing protocol [1][2] in IP networks today. Overall, OSPF implementations are now robust and high quality. Still, the *behavior* of these implementations in large operational IP networks, especially under transient stress, is not very well understood. Any sort of service level agreement or quality assurance depends on routing stability. Any internal topological or OSPF configuration change will, in general, alter traffic flows throughout the network, following a transient period during which route calculation has yet to converge. In general, such an event triggers not only intra-domain routing changes, but also inter-domain routing changes, since BGP (Border Gateway Protocol) uses intra-domain (OSPF) distance calculations to break ties between candidates for traffic egress points. Thus, a very large number of flows and a very large number of customers are potentially impacted by OSPF events.

A number of key tasks internal to OSPF implementations affect the speed at which updates propagate in the network, the load on individual routers, and the time needed to reconverge. The delays associated with some of these tasks depend on scaling factors such as the number of routers and links in the network. To understand these delays and their dependencies, we could imagine gathering data by instrumenting OSPF implementations deployed in the network. However, users, such as Internet Service Providers, have limited opportunities for this. First, commercial implementations are proprietary. Second, even if appropriate access was provided, the necessary instrumentation to measure certain tasks (e.g., updates to the Forwarding Information Base) may be difficult to achieve. The instrumentation may involve kernel level measurement, grappling with various platform dependencies, and reverse engineering complex code written and debugged over the years by numerous developers.

In this paper, we present black-box measurement techniques for estimating key internal delays in OSPF implementations, and our experience in applying these techniques to production routers from Cisco Systems. We tested a variety of Cisco platforms including the 12012 (GSR), 7513 and 3660. By *black-box*, we mean the internal task times are estimated using only external observations of the behavior of the box under test. Table I summarizes internal tasks considered: processing Link-State Advertisements (LSAs), performing Shortest Past First (SPF) calculations, updating the Forwarding Information Base (FIB), and flooding LSAs to neighboring routers. As described in Section II, these are the key tasks OSPF goes through upon receiving an update. As Table I indicates, we examine the dependencies of these tasks on scaling factors such as the network size. The methods presented are effective in estimating delays across a wide range, from a few hundred microseconds to a few hundred milliseconds, and in correctly capturing dependencies with scaling parameters. For example, the black-box SPF measurements agree with white-box counterparts and scale quadratically with the network size.

The idea behind the black-box measurements is straight-

Aman Shaikh is at the University of California, Santa Cruz, CA 95064, E-mail: aman@cse.ucsc.edu

Albert Greenberg is with AT&T Research, Florham Park, NJ 07932, E-mail: albert@research.att.com

TABLE I

Summary of processing delays for which measurements are presented in this paper.

Task	Scaling Factors	Type of Measurement
LSA Processing	Number of links at a router	black-box
	Number of LSAs per LS Update packet	black-box
LSA Flooding	Number of links at a router	black-box
Shortest Path First Calculation	Number of routers	black-box, white-box
Forwarding Information Base Update	Number of routers	black-box

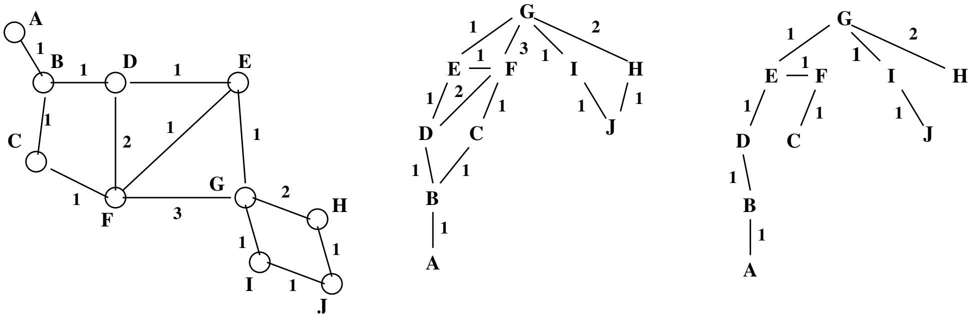


Fig. 1. From left to right, the figure depicts an example OSPF topology, router G's view of the topology and the shortest path tree calculated at G. For simplicity, we depict the topology as an undirected graph (implicitly assuming symmetry in connectivity and weight assignment).

forward. To estimate a task delay, we need to determine when the task starts and ends. We found that the difficulty of determining the start and finish times of a given task depends on that specific task. As described in Section IV, we bracket the start and finish times, and then subtract out time intervals that precede or exceed these times. We combine three techniques to design experiments for this purpose:

- Using an OSPF emulator for generating specific patterns of OSPF or ICMP ping messages.
- Exploiting features mandated by the OSPF specification [1]. One such feature turned out be extremely useful in providing time-stamps: duplicate LSAs must be acknowledged immediately.
- Setting vendor-specific configuration parameters so as to force tasks to occur in an order that allows for measurement.

We believe that in general a judicious mix of techniques that rely on behavior mandated by protocol standards and behavior configurable by vendor-specific commands are necessary for black-box protocol measurement.

Simple empirical models of routing behavior, or simulators aiming for higher fidelity, require sound measurements to guide parameterization. The measurements presented here on internal OSPF task delays could be used to investigate, in large testlabs or in simulations, scenarios that cause OSPF to meltdown or routing in general to break. There has been a recent interest in studying OSPF stability, convergence and scalability via simulation [3][4]. We know of few studies on routing protocol measurement methodology or results. A notable exception is the compelling analysis of Alaettinoglu et al. [5] on the factors that impact the convergence of IS-IS in detail, based on white-box measurement techniques applied to Cisco, Juniper and customized IS-IS implementations. IS-IS is a link-state protocol, similar to OSPF. Our numerical results for SPF calculation delays in OSPF on Cisco routers are comparable to those reported by Alaettinoglu et al. for SPF calculation on Cisco and Juniper routers. We are unaware of other work in black-box protocol measurement. Frameworks for router benchmarking have been proposed in the IETF [6]. It is worth mentioning that router software can typically be run in *debug* mode, which provides information related to many of the OSPF internal tasks considered here. However, the extra processing and I/O distorts associated measurements in a difficult to predict fashion.

The paper is organized as follows. Section II provides a brief overview of OSPF and the workflow modeling of OSPF processing. Section III describes the testbed we used for black-box measurements. A key component of the testbed is the OSPF emulator which is also described in the section. Section IV describes the experimental design and the results. Finally, Section V presents the conclusions.

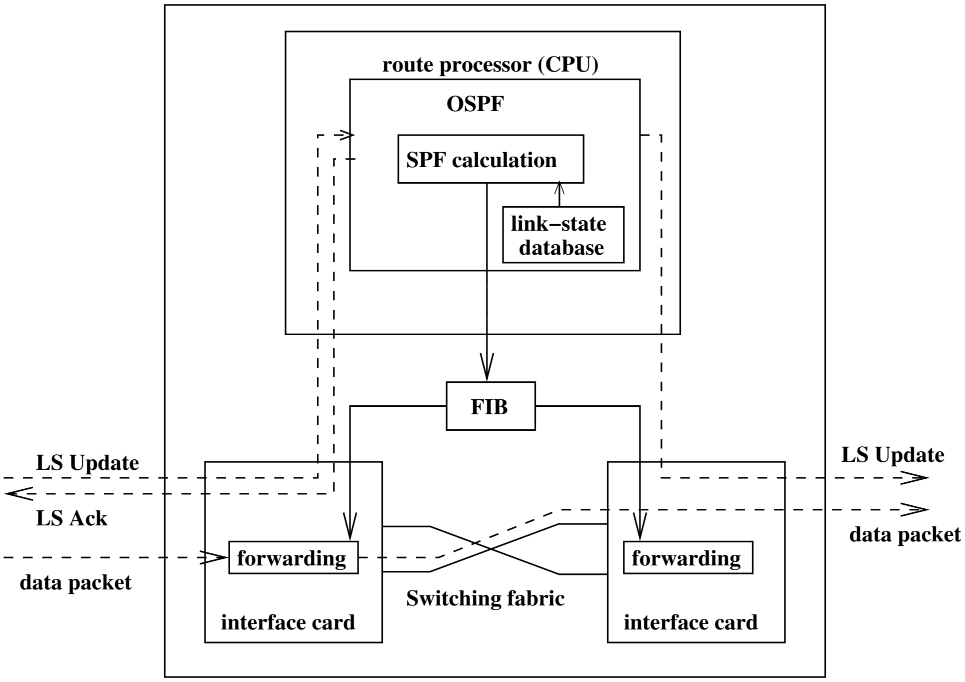


Fig. 2. Model of a router.

II. OSPF PROCESSING

Overview

OSPF is a link-state routing protocol, meaning each router (i) discovers and maintains a complete view of the network topology (within the the domain controlled by OSPF), and (ii) uses this view to calculate paths to all destinations in the network [2]. In essence, the topology is a directed graph, where routers correspond to vertices and links between neighboring routers correspond to unidirectional edges. All links are administratively assigned fixed numerical weights. Each router independently computes a shortest path tree with itself as the root, and applies the results to build its *Forwarding Information Base* (FIB). We refer to the computation of the shortest path tree as the *SPF computation*, and the tree itself as the *SPF tree*. Figure 1 provides an example. OSPF allows a network to be divided into one or more areas for scalability.

In OSPF, each router describes a certain part of the network in a message termed a *Link-State Advertisement (LSA)*. LSAs are *flooded* reliably to other routers in the network, so that all routers can build a consistent view of the network topology. Each router stores a current set of LSAs as a local *link-state database*; LSAs contained in the database determine the topology visible from the router. When sending LSAs to a neighbor, a router bundles them together in a *Link-State Update (LS Update)* packet. To acknowledge receipt of each LSA, the receiving router bundles individual LSA acks into *Link-State Acknowledgment (LS Ack)* packets, and sends them to the appropriate neighbor.

Figure 2 depicts a simplified model of a router. Routing protocols like OSPF run on a route processor. OSPF receives LSAs bundled in LS Update packets as shown in the figure and processes these to build the link-state database. OSPF then uses the link-state database to perform an SPF calculation and applies the result to build the FIB. In most modern routers, the FIB is maintained in specialized memory to maximize forwarding performance. Data packets do not consume CPU cycles of the route processor. Once a data packet arrives on an interface card, the card consults the FIB to determine the next hop and forwards the packet to the outgoing interface through a switching fabric as shown in the figure.

Processing

Consider the processing tasks initiated by receipt of an LS Update packet (Figure 3). Although the OSPF specification clearly describes the tasks to be performed upon receiving an LS Update packet, it gives implementors a lot of leeway in how and when these tasks are scheduled. The flow chart in Figure 3 follows the specification while capturing the scheduling choices available to implementors. As can be seen from the figure, upon receiving an LS Update packet, OSPF processes all the LSAs contained in the packet. For each LSA, OSPF classifies the LSA as *new* or *duplicate* based on the sequence number contained in the LSA. An LSA is deemed a duplicate if the

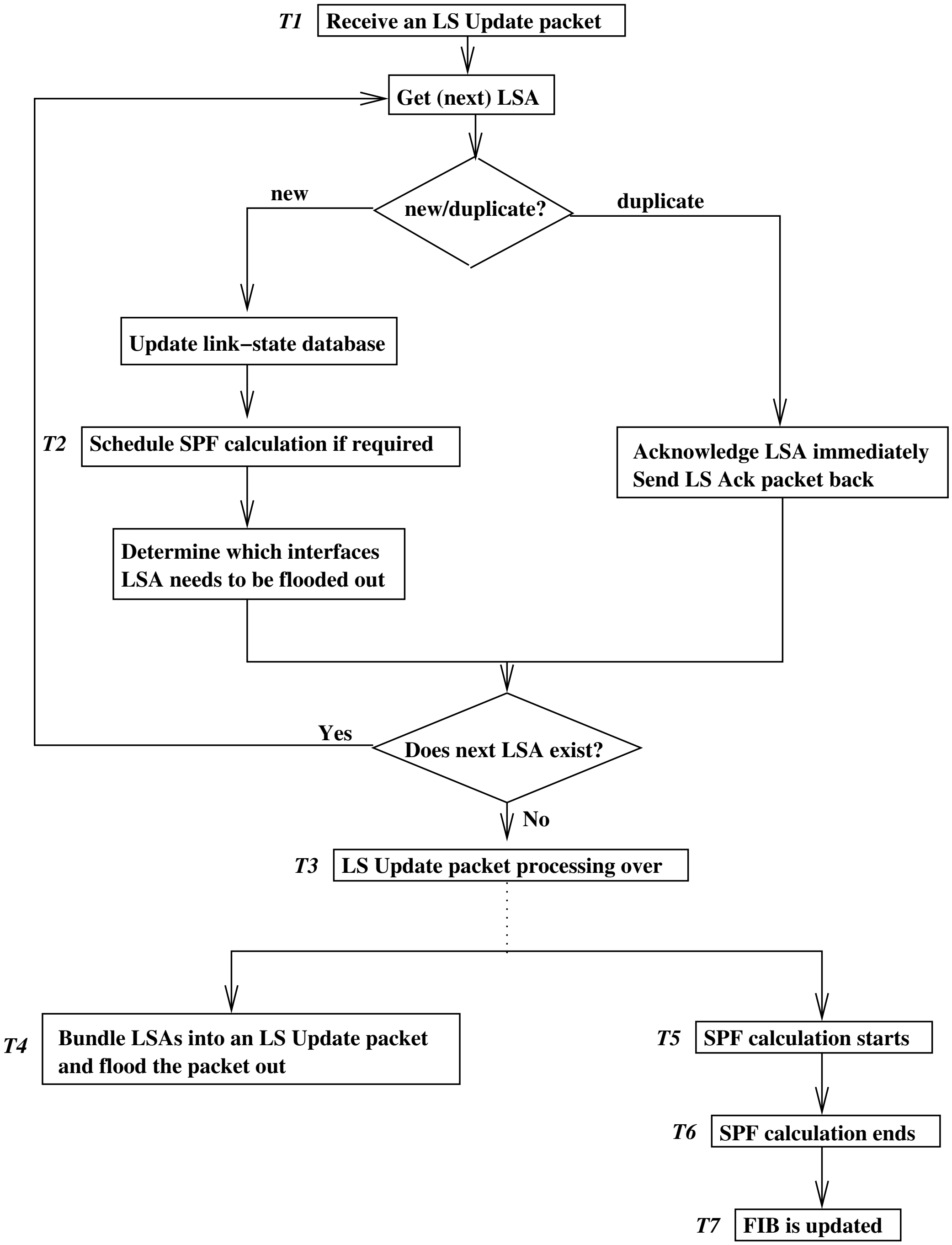

T1
Receive an LS Update packet
Get (next) LSA
new
new/duplicate?
duplicate
Update link−state database
T2
Schedule SPF calculation if required
Acknowledge LSA immediately
Send LS Ack packet back
Determine which interfaces
LSA needs to be flooded out
Yes
Does next LSA exist?
No
T3
LS Update packet processing over
T4
Bundle LSAs into an LS Update packet
and flood the packet out
T5
SPF calculation starts
T6
SPF calculation ends
T7
FIB is updated

Fig. 3. Flow chart depicting OSPF processing initiated by the receipt of an LS Update packet.

sequence number is the same as that of a matching LSA instance in the router's link-state database. OSPF typically receives duplicate LSAs because of flooding redundancy, i.e., when all the neighbors of the router send the same LSA to the router. An LSA is new if its sequence number is higher than that of the matching LSA instance in the router's database. As the figure shows, LSA processing depends on the new or duplicate distinction. For every new LSA, OSPF has to update its link-state database, schedule an SPF calculation and determine which interfaces the LSA needs to be flooded out.

Actual flooding of the LSA may or may not happen immediately after LSA processing. Modern routers employ pacing mechanisms as a form of flow control while sending out OSPF packets [7]. LSA flooding is driven by a timer off the path of LSA processing. In other words, while processing the LSAs, OSPF merely determines which interfaces the LSA needs to be flooded out according to [1], but does not actually send the LSA. The LSA is sent out on an interface along with other "to-be-flooded" LSAs when the timer associated with the interface fires.

In this paper, our objective is to determine two parameters relevant to LSA processing: t_{lsa_proc} which is the time for processing LSAs ($T_3 - T_1$ in Figure 3), and t_{lsa_flood} which indicates how long it takes to flood an LSA after it is received ($T_4 - T_1$).

When processing an LSA, OSPF also has to determine whether to carry out an SPF calculation. Not all LSAs indicate a change in topology; OSPF requires periodic refreshing of LSAs even when the topology has not changed. In addition, since SPF calculation is a CPU-intensive task, modern routers merely schedule an SPF calculation when they receive an LSA indicating a change. This gives the router a chance to receive more LSAs that may indicate changes in the topology and amortize the cost of an SPF calculation over a number of LSAs requiring such calculation. We assume that SPF calculation is non-preemptable in the sense that the router completes the calculation before doing any other OSPF processing. This makes sense since the router has already waited for a certain time period before undertaking the calculation. It makes little sense to delay its completion further by preempting it. Moreover, data collected for SPF calculation time (Section IV-C) validate the assumption. We have also verified these assumptions with vendors.

Once a router has done its SPF calculation, it has to install all the routes in its FIB, which introduces an additional delay. Accordingly, we estimate two parameters relevant to the routing calculation: the time t_{spf} taken by a router to perform an SPF calculation ($T_6 - T_5$ in Figure 3), and the time t_{fib_update} to update the FIB ($T_7 - T_5$ in Figure 3).

Table II summarizes the four internal task delays we wish to measure, in terms of the start and finish times of Figure 3. We measured these four parameters on Cisco Systems 12012 (GSR) and 7513 routers running IOS 12.0(7). (Some of the experiments were repeated on the Cisco Systems 3600, with very similar results.) We also measured one of these parameters (t_{spf_time}) on a Linux PC running GateD version 4.0.6, and compared the measurements with corresponding white-box measurements.

TABLE II
SUMMARY OF FOUR PROCESSING DELAYS WE MEASURE.

Processing task	Symbol	Start time	Finish time
LSA Processing	t_{lsa_proc}	T_1	T_3
LSA Flooding	t_{lsa_flood}	T_1	T_4
SPF Calculation	t_{spf}	T_5	T_6
FIB Update	t_{fib_update}	T_5	T_7

III. Testbed Setup

Figure 4 depicts the physical testbed setup. We refer to the router whose OSPF implementation is under test as the *target router*. To understand the OSPF behavior of the target router, and investigate the impact of scaling parameters, we developed an OSPF topology emulator, termed *TopTracker*, which runs on PC1. TopTracker is derived from Moy's OSPF implementation and simulator [8]. An Ethernet switch provides VLAN connectivity between the boxes.

A (Linux) PC, PC1, plays multiple key roles. First, it runs TopTracker, which is capable of emulating any desired OSPF topology, making the target router behave as if the emulated topology exists "behind" a TopTracker interface. Specifically, TopTracker generates LSAs for all nodes in a given emulated topology and floods the LSAs to the target router. In addition, we applied TopTracker to generate specific patterns of LSAs required in the experimental designs described in Section IV.

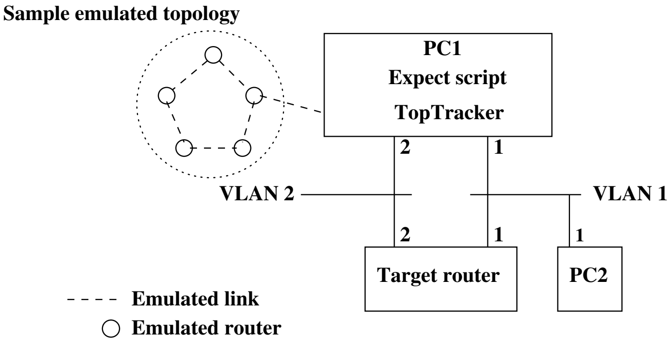


Fig. 4. Testbed for measuring OSPF processing on a target router. PC1 runs the TopTracker OSPF emulator, as well as an Expect script that controls an overall experiment. PC2 is used as a ping generator if required.

Second, PC1 is used to control each experiment, through the use of Expect scripts [9]. Third, PC1 sources, sinks and time-stamps OSPF packets and applies the time-stamps to estimate internal OSPF processing delays on the target router. An advantage of using a single PC in this role is of course that we avoid clock synchronization problems.

We vary the logical connectivity from experiment to experiment. Table III provides the configuration details. For the most part, the VLANs simply provide one or two links between the target router and TopTracker. In the FIB Update experiment, VLAN 1 also provides connectivity to PC2, which functions as a ping generator (Section IV-D).

IV. Experimental Design and Results

In this section, we provide the details of the methods for estimating the four OSPF internal task delays described

TABLE III

TESTBED CONFIGURATION FOR FOUR PROCESSING DELAYS

Task	Logical Connectivity
LSA Processing	PC1 - target router on VLAN 1
LSA Flooding	PC1 - target router on VLANS 1 and 2
SPF Calculation	PC1 - target router on VLAN 1
FIB Update	PC1 - target router on VLAN 1 and PC2 acts as ping generator on VLAN 1.

in Section II, and provide associated results obtained for Cisco and GateD routers. For each task, we estimate the associated start and finish times by a black-box technique. In each case, we bracket the start and finish times by measurable events t_s occurring before the start time and t_r occurring after the finish time. We then compute an estimate $t_{overhead}$ that accounts for the time from t_s to the task start time, and from the task finish time to t_r. The estimated task delay is then $(t_r - t_s) - t_{overhead}$.

For each of the four tasks, we first describe the testbed configuration (Figure 4) used to measure the task delay. Then we describe how we determine the bracketing start and finish times, t_s and t_r respectively. Next we characterize the overhead ($t_{overhead}$) and describe how we measure it. Finally we present the results.

We make use of two configurable parameters provided by Cisco routers to pace SPF calculation [7]:

1. *spf-delay*, which specifies how long OSPF waits between receiving a topology change and starting an SPF computation.
2. *spf-holdtime*, which enforces a lag of *spf-holdtime* between two consecutive SPF computations.

In practice, these parameters can be set to help ensure that the SPF calculations act on LSAs in batch and create only moderate load on the route processor. In addition, Cisco OSPF uses a *pacing-timer* [7] to control the rate at which LS Update packets are transmitted out an interface. This timer is non-configurable and expires every 33 milliseconds. Specifically, Cisco routers send out one LS Update packet (if present) every 33 milliseconds to every neighbor, helping to ensure that the neighbors are not overwhelmed with bursts of LS Update packets.

A. LSA Processing Time

Of the four tasks, the measurement of LSA processing delays is the most complex. As described in Section II, OSPF bundles LSAs into LS Update packets. Thus, we focus on the time needed to process an LS Update packet, and then examine how this delay varies with the number of LSAs within the packet.

The testbed configuration is simple. TopTracker establishes an adjacency with the target router on VLAN 1 (Figure 4).

To measure the LS update processing delay, TopTracker sends two LS Update packets back to back: the first containing legitimate LSAs, which we term *probe* LSAs, and the second containing a duplicate LSA. As noted in Section I, OSPF mandates that duplicate LSAs be acknowledged immediately, via an LS Acknowledgment (ack) [1]. By design, the duplicate LSA is different from any of the probe LSAs, allowing TopTracker to unambiguously identify the duplicate LSA ack from probe LSA acks. We assume that LS Update packets are processed consecutively, with a negligible intervening gap.

TopTracker logs the time t_s at which it sends out the LS Update packet containing the probe LSAs, and the time t_r at which it receives the ack for the duplicate LSA. Figure 5 describes the sequence of events, marked at times t_s, t_1, t'_1, t_2, t'_2, t_3 and t_r. By the assumption that the LS update packets are processed back to back, t'_2 is less than or equal to t_2.

From Figure 5, we can see that

$$\begin{aligned}
t_{lsa\_proc} &= t_2 - t_1 \\
&= (t_r - t_s) - [(t_r - t_3) + (t_3 - t_2) + (t_1 - t_s)] \\
&= (t_r - t_s) - \\
&\quad [\text{(Duplicate LSA processing time)} + \text{RTT}] \\
&= (t_r - t_s) - [t_{dup\_lsa} + \text{RTT}] \\
&= (t_r - t_s) - t_{overhead} \qquad (1) \\
&\quad \text{where } t_{overhead} = t_{dup\_lsa} + \text{RTT}
\end{aligned}$$

Here RTT denotes the round trip propagation delay between PC1 and the target router (Figure 4) and t_{dup_lsa} denotes the processing delay for the duplicate LSA on the target router.

It remains to estimate $t_{overhead}$. To this end, TopTracker sends a single LS update packet containing the duplicate LSA, and logs the time between the LS Update transmission and LS ack receipt, as an estimate of $t_{overhead}$.

Let us now present some of the results we obtained.

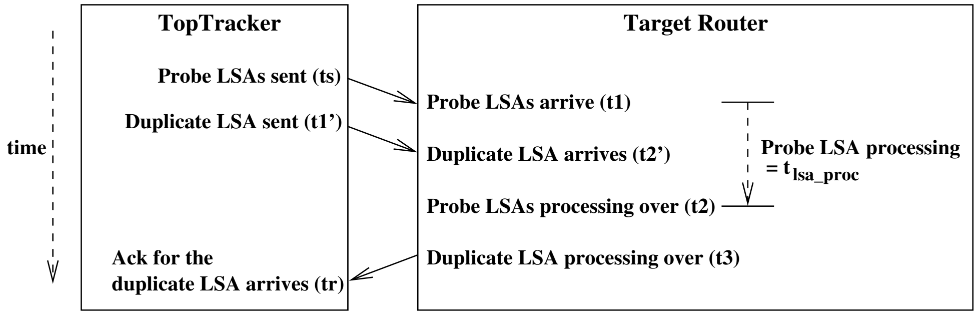
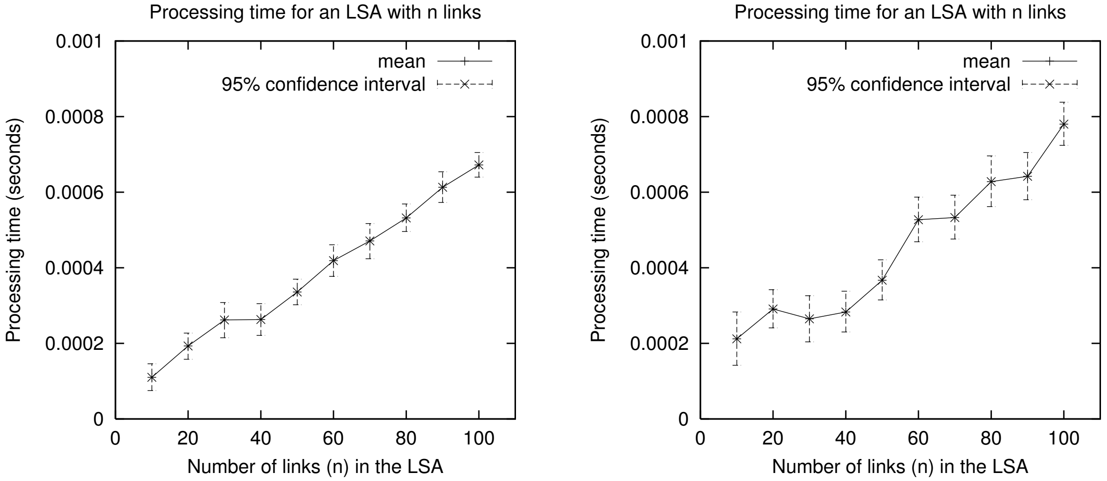
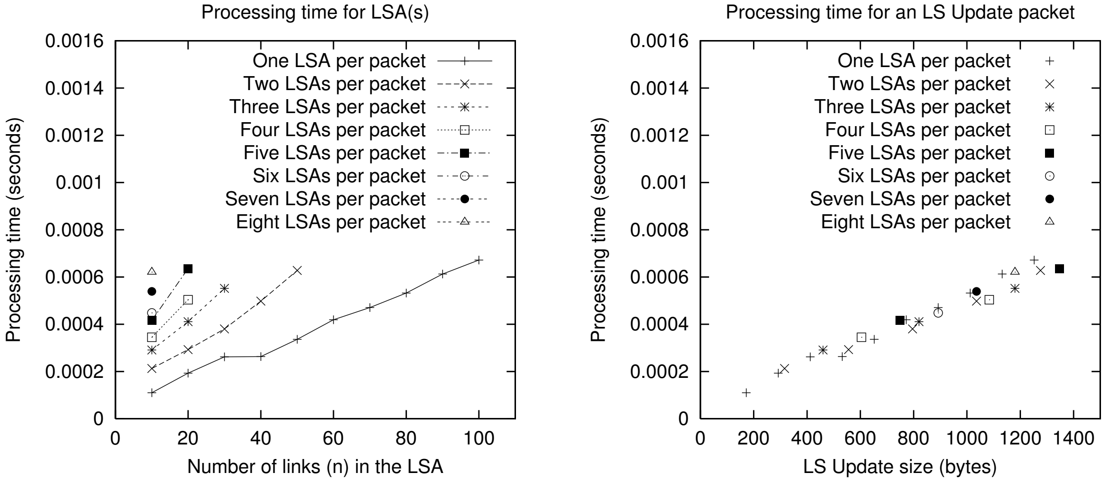

TopTracker
Target Router
time
Probe LSAs sent (ts)
Duplicate LSA sent (t1')
Ack for the duplicate LSA arrives (tr)
Probe LSAs arrive (t1)
Duplicate LSA arrives (t2')
Probe LSAs processing over (t2)
Duplicate LSA processing over (t3)
Probe LSA processing = t_{lsa_proc}

Fig. 5. Sequence of events during the measurement of LSA processing delays on the target router.



(a) Cisco GSR (b) Cisco 7513

Fig. 6. Processing time for an LSA (t_{lsa_proc}) containing n links, as a function of n.



(a) Number of links (b) LS Update packet size

Fig. 7. Processing time for LS Update packets having one or more LSAs. The figure shows how the processing time is effected as the number of LSAs in the LS Update packets increases. The results are for the Cisco GSR.

First consider the case where an LS Update packet contains a single LSA. The processing time for the packet depends on three factors: the number of links the LSA describes, the number of interfaces on which the LSA must (later) be flooded, and the size of the link-state database. We performed experiments to examine the effect of all three factors on t_{lsa_proc}. Due to space constraints, we present results showing only the first.

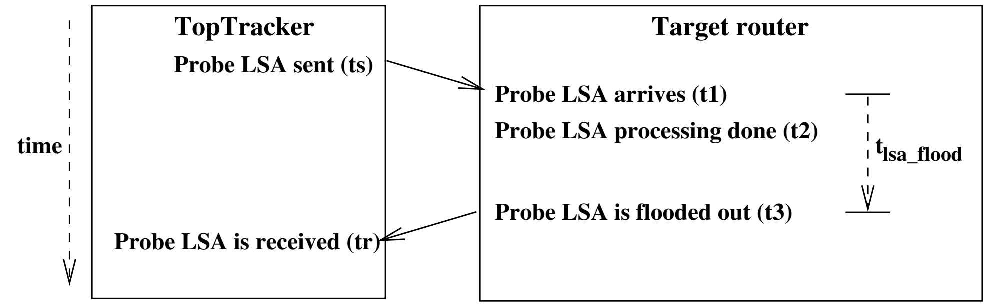
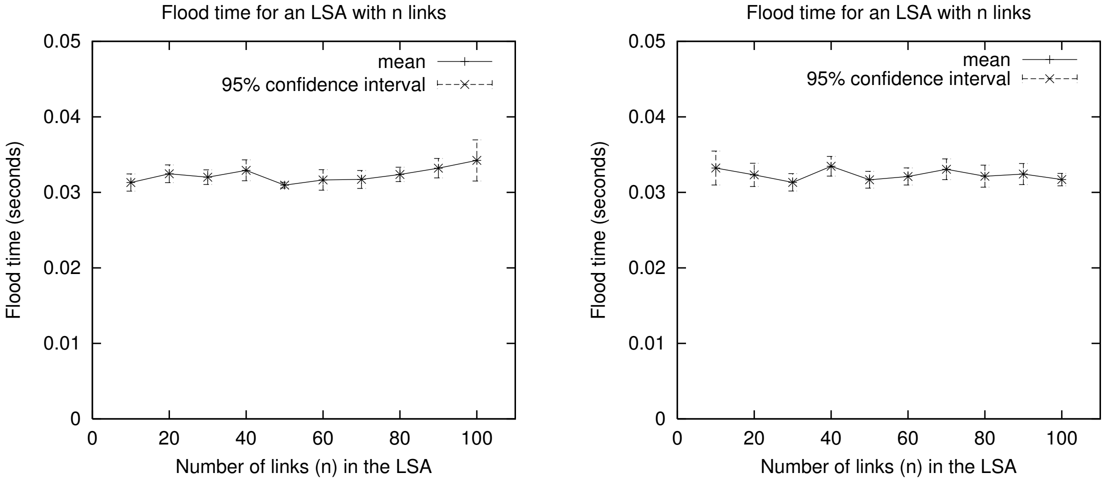

time
TopTracker
Probe LSA sent (ts)
Probe LSA is received (tr)
Target router
Probe LSA arrives (t1)
Probe LSA processing done (t2)
Probe LSA is flooded out (t3)
t_{lsa_flood}

Fig. 8. Sequence of events in experiment designed to estimate the time the target router takes to flood out an LSA.



(a) Cisco GSR

(b) Cisco 7513

Fig. 9. Time taken by the target router for flooding an LSA (t_{lsa_flood}) describing n links, for varying n.

To gauge the effect of number of links on t_{lsa_proc}, we used probe LSAs each describing n links, and varied n in the range 10, 20, ..., 100. Figure 6 shows how t_{lsa_proc} varies as n increases for the two Cisco target routers.

Next consider the case where the LS Update packet contains multiple LSAs. Recall from Figure 3 that a router first has to process the interrupt and copy the LS Update packet into its memory before it can process the LSAs. The time spent in performing this step is amortized over all the LSAs in the packet. How much time is spent in packet copying as compared with how much is spent in processing individual LSAs? To investigate this question, we increased the number of probe LSAs (each LSA having n links) from one to the maximum number of such LSAs that can fit into a single LS Update packet, and measured the processing time for the LS Update packet. The maximum number of LSAs that can fit into a single LS Update packet is limited by the maximum size an LS Update packet can grow to. In our case, the maximum size was equal to the Ethernet MTU (1500 bytes) including the IP and Ethernet headers. With this limit, consider a probe LSA with 100 links ($n = 100$). Each such LSA is 1224 bytes long. Hence, we cannot fit more than one such LSAs into a single LS Update packet. On the other hand, a probe LSA with 50 links ($n = 50$) is 624 bytes long, hence we can fit up to two such LSAs in a single LS Update packet.

Figure 7(a) provides the results obtained for the Cisco GSR. First, note that if we keep the number of links n constant, the increase in t_{lsa_proc} for every additional LSA per LS Update packet is smaller than t_{lsa_proc} for a single LSA. This indicates that the time taken in interrupt handling and copying the LS Update packet dominates the time taken to process individual LSAs. In order to make this point more clear, we show in Figure 7(b) how t_{lsa_proc} for the same experiment varies with the size of the LS Update packet. Note that the processing time remains roughly the same for LS Update packets of a given size, irrespective of the number of LSAs contained in the packet or the number of links described in the LSAs. An important implication is that for the router in question, the Cisco GSR, we can safely characterize the processing time of an LS Update packet simply as a function of the packet's size, ignoring the number of LSAs contained in the packet.

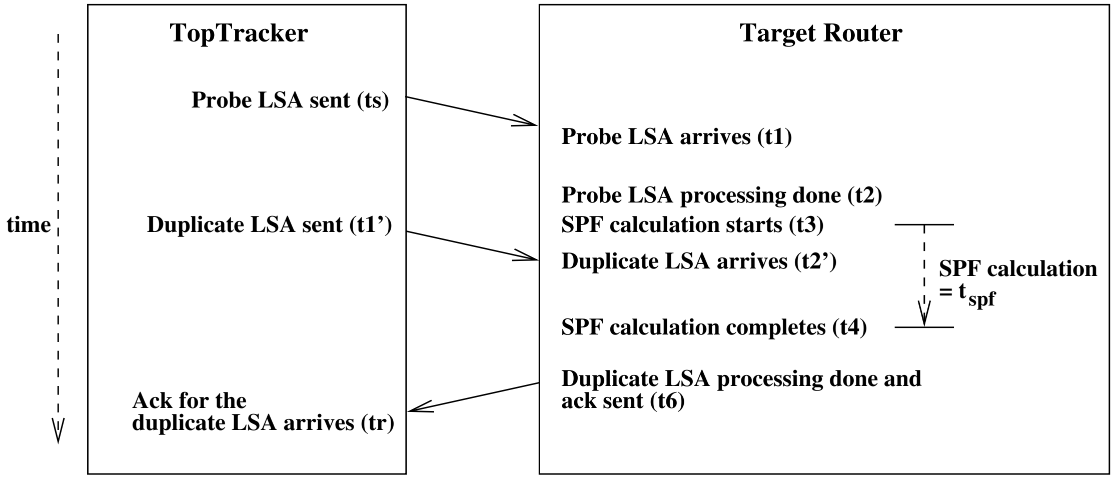
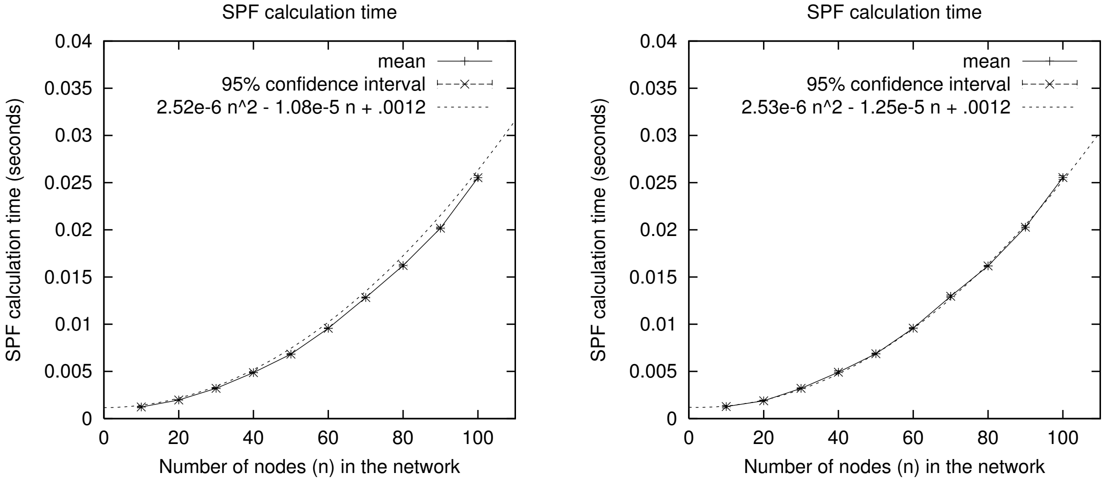

time
TopTracker
Target Router
Probe LSA sent (ts)
Probe LSA arrives (t1)
Probe LSA processing done (t2)
SPF calculation starts (t3)
Duplicate LSA sent (t1')
Duplicate LSA arrives (t2')
SPF calculation = t_spf
SPF calculation completes (t4)
Duplicate LSA processing done and ack sent (t6)
Ack for the duplicate LSA arrives (tr)

Fig. 10. Sequence of events in measuring the SPF calculation time on the target router.



(a) Cisco GSR (b) Cisco 7513

Fig. 11. SPF calculation time (t_{spf}) for a fully connected n node topology, for varying n.

B. LSA Flood Time

We next consider the time the target router takes to flood an LSA after receiving it. On Cisco routers, as we will see, the dominant effect is the *pacing-timer* mentioned earlier.

Estimating the time needed to flood a newly received LSA to neighbors is relatively straightforward. We configure the testbed (Figure 4) with TopTracker having two adjacencies with the target router on two interfaces.

Figure 8 describes the setup and the sequence of events. TopTracker sends a probe LSA on one interface to the target router, and receives the LSA flooded in response by the target router on the other interface. TopTracker logs the time t_s at which it sends the probe LSA, and the time t_r at which it receives the flooded copy.

$$\begin{aligned} t_{lsa\_flood} &= (t_3 - t_1) \\ &= (t_r - t_s) - [(t_r - t_3) + (t_1 - t_s)] \\ &= (t_r - t_s) - \text{RTT} \\ &= (t_r - t_s) - t_{overhead} \qquad (2) \\ & \quad \text{where } t_{overhead} = \text{RTT} \end{aligned}$$

and RTT is the round trip time to forward a packet between the two TopTracker interfaces via the target router, which we estimate using ICMP ping from TopTracker.

To investigate how the number of links (n) affects t_{lsa_flood}, we performed measurements using a single LSA containing n links as the probe LSA. Figure 9 shows that t_{lsa_flood} has little dependence on the size of the LSA. Indeed the size dependence is apparently dominated by the 33 millisecond *pacing-timer* controlling LSA flooding.

C. Shortest Path First (SPF) Calculation Time

Next consider SPF calculation delay estimation. As mentioned earlier, Cisco routers provide two configurable parameters that influence the scheduling of the SPF calculation: *spf-delay* and *spf-holdtime*. We set both these parameters to 0. The idea of the experiment is to send a probe LSA whose receipt immediately initiates SPF calculation on the target router, and then send a duplicate LSA whose role is to bracket the finish time of the SPF calculation. By removing the "overhead" between the transmission of the probe LSA and the receipt of the LS Ack from the duplicate LSA, we extract the SPF processing time.

We configure the testbed with TopTracker having one adjacency with the target router on VLAN 1. To begin the experiment, TopTracker sends to the target router a probe LSA indicating a change in the network topology. Upon receiving the probe LSA, the target router processes it and schedules an SPF calculation. Next, TopTracker sends a duplicate LSA in a separate LS Update packet to the target router. TopTracker then receives an ack as soon as the target router is finished processing the duplicate LSA.

Since *spf-delay* and *spf-holdtime* are set to 0, SPF computation starts as soon as the target router receives the probe LSA. We would like to use the LS Ack for the duplicate LSA to closely bracket the finish time for the SPF calculation. Specifically, we want the duplicate LSA to arrive to the target router while its route processor is engaged in the SPF calculation. We achieved this by introducing a certain delay (determined by sweeping this delay parameter) between the transmission of two LS Update packets.

TopTracker logs the time t_s at which it sends out the LS Update packet containing the probe LSA, and the time t_r at which it receives the ack for the duplicate LSA. From Figure 10, we then see that

$$\begin{aligned} t_{spf} &= t_4 - t_3 \\ &= (t_r - t_s) - [(t_r - t_6) + (t_6 - t_4) \\ &\quad +(t_3 - t_2) + (t_2 - t_1) + (t_1 - t_s)] \end{aligned}$$

We can safely assume that $t_3 - t_2$ is negligible. The quantity $(t_r - t_6) + (t_1 - t_s)$ is equal to the RTT between TopTracker and the target router. Hence,

$$\begin{aligned} t_{spf} &= (t_r - t_s) - [\text{RTT} + (t_6 - t_4) + (t_2 - t_1)] \\ &= (t_r - t_s) - [\text{RTT} + t_{dup\_lsa} + t_{probe\_lsa}] \\ &= (t_r - t_s) - t_{overhead} \qquad (3) \\ &\quad \text{where } t_{overhead} = \text{RTT} + t_{dup\_lsa} + t_{probe\_lsa} \end{aligned}$$

In order to determine $t_{overhead}$, we repeat the experiments with two changes: First, we set *spf-delay* to a very large value (nominally 60 seconds). Second, TopTracker sends the probe LSA and the duplicate LSA back-to-back in two separate LS Update packets. These choices ensure that the SPF calculation is removed from the interval between the probe and the duplicate LSA processing tasks.

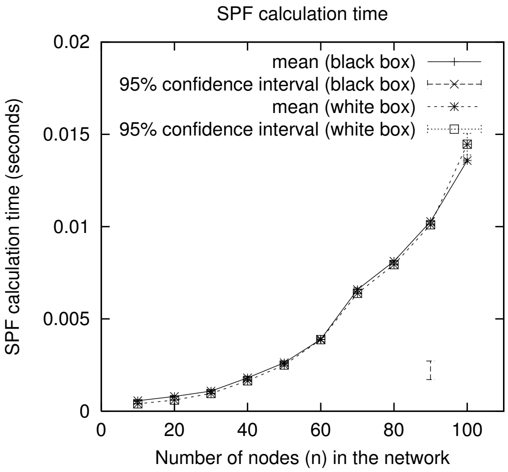


Fig. 12. Comparison of black-box and white-box measurements for SPF calculation time (t_{spf}) for GateD, with varying n, the number of nodes in a fully connected topology.

TopTracker logs the time at which it sends out the probe LSA and the time at which it receives the ack for the duplicate LSA. The difference between these two time values provides an estimate of $t_{overhead}$.

To investigate the effect of network size on SPF calculation time, we collected t_{spf} values for fully connected emulated topologies with varying number of nodes n. Figure 11 presents the results. Note that the results for the GSR and 7513 are essentially identical. Though these routers have vastly different forwarding capabilities, their route processors are similar, and SPF calculation is a CPU intensive task. A closer examination of the data also validates the assumption that the SPF calculation is non-preemptable.

To help validate the above methodology, we repeated the same experiments with GateD and compared the results with white-box measurements obtained by instrumenting GateD's SPF calculation. Figure 12 shows that the black-box measurements closely track the white-box measurements.

D. Forwarding Information Base (FIB) Update Time

Next we consider the time needed for the target router to update its Forwarding Information Base (FIB) after it starts the SPF calculation. The idea behind the experiment is (i) to configure the testbed initially with a given address d unreachable on the target router, (ii) to inject an LSA that makes d reachable on the target router, and (iii) closely bracket the time that the target router takes to install d in its FIB. To achieve step (iii), we ping d on the target router at high rate. Until the FIB is updated, all pings are dropped. Thus, the first ping forwarded closely brackets the time the FIB update completes.

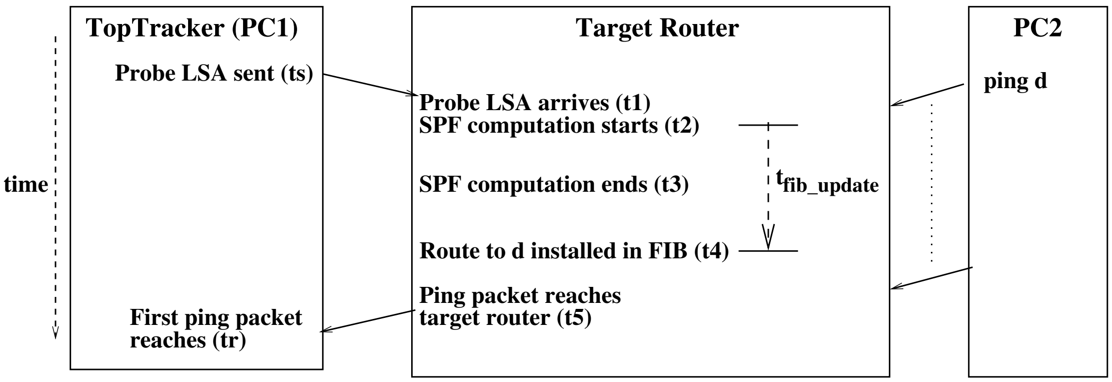
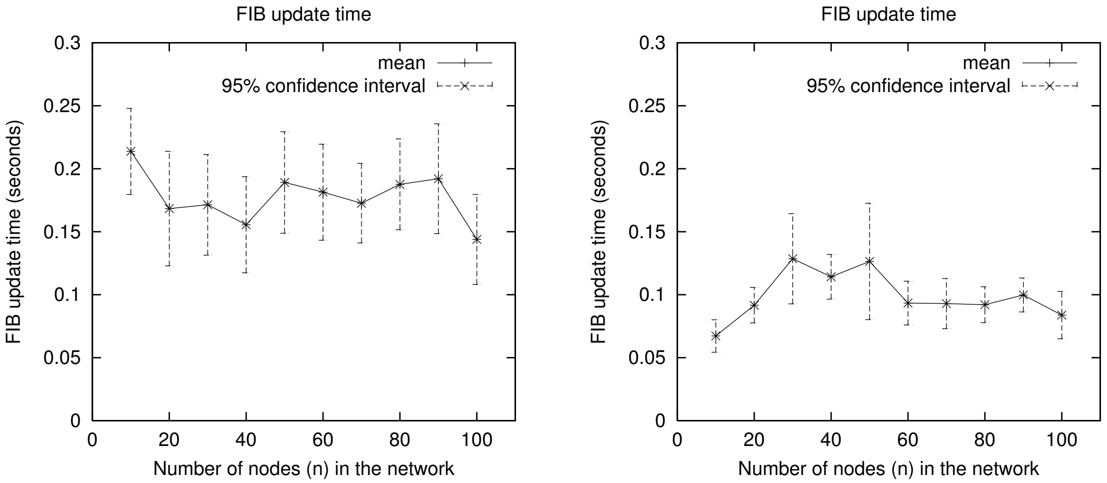

TopTracker (PC1)
Probe LSA sent (ts)
time
First ping packet reaches (tr)
Target Router
Probe LSA arrives (t1)
SPF computation starts (t2)
SPF computation ends (t3)
t_{fib_update}
Route to d installed in FIB (t4)
Ping packet reaches target router (t5)
PC2
ping d

Fig. 13. Sequence of events in measuring the FIB update time on the target router.



(a) Cisco GSR

(b) Cisco 7513

Fig. 14. FIB update time (t_{fib_update}) for a fully connected topology with n nodes, for varying n.

We configure the testbed (Figure 4) using PC2 as the ping generator. PC2 pings destination d at a constant rate of one ping every 0.01 second (the maximal rate available on Linux). Since PC2 does not run OSPF, we install a static route on PC2 for destination d with the target router as the next hop. Thus, PC2 forwards all ping packets (ICMP request packets to be more specific) to the target router on VLAN 1. At this point, the target router does not know how to reach d and hence drops the pings.

Figure 13 describes the sequence of events that occur in the course of the experiment, in some detail. TopTracker generates a probe LSA designed to make d reachable from the target router's perspective. As soon as this change is reflected in the FIB, the target router starts forwarding ping packets to TopTracker (PC1). TopTracker computes a time-stamp t_s when it sends out the probe LSA. The Expect script running on PC1 computes a time-stamp t_r as soon as it receives the first ping packet destined to d. From Figure 13, we can see that

$$\begin{aligned} t_{fib\_update} &= t_4 - t_2 \\ &= (t_r - t_s) - [(t_r - t_5) + (t_5 - t_4) \\ &\quad +(t_2 - t_1) + (t_1 - t_s)] \end{aligned}$$

As we did for SPF calculation delay, we can safely assume that $t_2 - t_1$ is negligible. Similarly,

$$\begin{aligned} t_{fib\_update} &= (t_r - t_s) - \Bigg[\frac{\text{RTT(PC1-router)}}{2} \\ &\quad +(t_5 - t_4) + \frac{\text{RTT(PC2-router)}}{2}\Bigg] \\ &= (t_r - t_s) - t_{overhead} \qquad (4) \\ &\quad \text{where } t_{overhead} = \frac{\text{RTT(PC1-router)}}{2} \\ &\quad +(t_5 - t_4) + \frac{\text{RTT(PC2-router)}}{2} \end{aligned}$$

As Eq. 4 indicates, the value of $t_{overhead}$ is a sum of three quantities. Two of these quantities depend on RTT values, which we estimated by using pings. The value of the third quantity, $t_5 - t_4$, depends on when the first packet arrives at the target router after it has updated its FIB. Since FIB update occurs essentially at a random time with respect to the

TABLE IV
SUMMARY OF RESULTS

Task	Range of task delay	Dependence
LSA processing	100-800 microseconds	Depends on the size of the LS Update packet; Individual LSA in the packet contribute little to the overall processing time of the packet
LSA flooding	30-40 milliseconds	Depends on the pacing time
SPF calculation	1-40 milliseconds	Depends on the number of nodes in the network
FIB update	100-300 milliseconds	Depends on router architecture; Little dependence on the number of nodes in the network

pings from PC2 occurring every 0.01 seconds, we assume that quantity $t_5 - t_4$ is a uniformly distributed variable in the interval [0.00, 0.01] seconds and hence has a mean of 0.005 seconds.

Figure 14 plots the FIB update time, as it varies with the size of the network, which we assumed for simplicity to be fully connected. The figure clearly demonstrates that t_{fib_update} has little dependence the size of the network or the duration of the SPF calculation. It is worth noting that FIB update time is significantly higher than the SPF calculation time. Thus, it takes a significant amount of time for the target router to update its FIB even after it is done with the SPF calculation. FIB update time depends heavily on the router architecture. As pointed out earlier, the two Cisco routers have different architectures, which explains the significantly different FIB update behaviors.

V. CONCLUSIONS

In this paper, we presented methods and results for estimating internal OSPF processing delays. The methods are black-box, i.e., they are based on external observations rather than internal instrumentations. We applied the methods to production Cisco Systems routers. Table IV summarizes the results. Delay associated with LSA processing is relatively small, on the order of 100 microseconds. Moreover, the major contribution of the delay comes from time spent in copying the LS Update packet that contains the LSAs. Delay associated with LSA flooding depends on the pacing timer employed by the router, and is on the order of tens of milliseconds for the Cisco routers under test. On the other hand, delays associated with SPF calculation and FIB update are relatively large, on the order of 10 and 100 milliseconds, respectively. The delay associated with SPF calculation (Dijkstra's algorithm) scales quadratically with the number of nodes in fully connected topologies. FIB update shows no correlation with the network size, but is much larger than the SPF calculation time. Though FIBs greatly speed packet forwarding, update is relatively costly.

We presented results for Cisco GSR and 7513 routers. Though these routers vary considerably in hardware architecture and forwarding capabilities, their route processors are similar. As a result, the routers behaved similarly for CPU-bound tasks such as SPF calculation, LSA processing and flooding. On the other hand, the routers showed widely varying behavior for FIB update, which is natural since FIB update performance depends heavily on router architecture.

We applied the methodology to an open source OSPF implementation, GateD 4.0.6, running on a Linux PC, for SPF calculation time. In this case, white-box and black-box measurements were found to be quite close.

The basic idea behind the black-box method is straightforward: design experiments that allow us to bracket the start and the finish times of the task in question, and related experiments that allow us to estimate "overhead" in the bracket accounting for time spent before the task start time and after the task finish time. Our experimental designs made use of:

- A protocol emulator, used to generate specific patterns of control messages, and to investigate the impact of scaling parameters.
- Attributes of the protocol standard that help provide time-stamps. In the paper, we introduced a time-stamping trick that works for any protocol packet that requires immediate processing with an externally observable response. Applied to OSPF using duplicate LSAs, we were able to estimate the finish time for two tasks: LSA processing and SPF calculation. The same trick can be used to measure the processing time for other OSPF tasks (for example, processing OSPF Hellos).
- Vendor specific configuration parameters. For example, to help measure SPF calculation time, we set the parameter *spf-delay* to 0, and to help measure LSA processing time we set the same parameter to a large quantity (60 seconds).

In future work, we plan to further investigate LSA processing and SPF calculation delays. For LSA processing, we plan to explore how the delay varies with: the size of the link-state database, the number of adjacencies that the LSA needs to be flooded out, and the way LSAs are distributed across LS Update packets. For SPF calculation, we plan to see how the delay varies with different kinds of network topologies. It is also of interest to adapt the experimental designs to routers from other vendors. Another avenue of future research is developing black-box techniques for other routing protocols, such as BGP [10] and IS-IS [11].

ACKNOWLEDGMENT

We are grateful to Mukul Goyal, K. K. Ramakrishnan and Raju Rajan for several helpful discussions during the course of the experiments, and to Jennifer Rexford, Timothy Griffin, Randy Bush and Nick Feamster for comments on the paper. We are also grateful to David Katz of Juniper Networks for answers to several questions related to commercial OSPF implementations, to Brian Colbry and Steven Gao for help with Cisco router configuration and testbed setup, and to Matthew Roughan for help with curve-fitting software. Finally, we would like to thank the anonymous reviewers for their comments and feedback.

REFERENCES

[1] John Moy, "OSPF Version 2," Request for Comments 2328, April 1998.

[2] J. T. Moy, *OSPF : Anatomy of an Internet Routing Protocol*, Addison-Wesley, January 1998.

[3] Anindya Basu and Jon G. Riecke, "Stability issues in ospf routing," in *Proc. ACM SIGCOMM*, August 2001.

[4] A. S. Maunder and G. Choudhury, "Explicit Marking and Prioritized Treatment of Specific IGP Packets for IGP Convergence and Improved Network Stability and Scalability," Internet Draft draft-ietf-scalability-00.txt, work in progress, March 2001.

[5] "Toward milli-second IGP convergence," Internet Draft draft-alaettinoglu-isis-convergence-00.txt, work in progress, November 2000.

[6] J. H. Dunn and C. E. Martin, "Framework for Router Benchmarking," Internet Draft draft-ietf-bmwg-rtr-framework-00.txt, work in progress, July 2000.

[7] "http://www.cisco.com," .

[8] J. T. Moy, *OSPF : Complete Implementation*, Addison-Wesley, September 2000.

[9] Don Libes, *Exploring Expect*, O'Reilly & Associates, Inc., December 1994.

[10] Y. Rekhter and T. Li, "A Border Gateway Protocol 4 (BGP-4)," RFC1771, March 1995.

[11] R. Callon, "Use of OSI IS-IS for Routing in TCP/IP and Dual Enviornments," RFC1195, December 1990.