# Avoiding Routing Instability during Graceful Shutdown of OSPF

Aman Shaikh[†]
Rohit Dube[‡]
Anujan Varma[†]

UCSC-CRL-00-20
December 20, 2000

[†] Computer Engineering Department
University of California, Santa Cruz
Santa Cruz, CA 95064

[‡] Xebeo Communications, Inc.
One Cragwood Rd. Suite 100
South Plainfield, NJ 07080

## ABSTRACT

Many recent router architectures decouple the routing engine from the forwarding engine, so that packet forwarding can continue even when the routing software is not active. This implies that one can avoid route flaps that occur when the routing process goes down provided the forwarding engine remains active during that time period. Unfortunately, the current definitions of routing protocols like BGP, OSPF and IS-IS do not support this behavior. In this paper, we propose means of extending OSPF by adding a new capability called *IBB (I'll Be Back) Capability* to it, so that packet forwarding can continue for a certain time period when the OSPF process is down. IBB Capability can be used for avoiding route flaps that occur when the OSPF process is brought down to facilitate protocol software upgrade, operating system upgrade, router ID change, and AS and interface renumbering. We identify the challenges, one of which is the inability of the inactive router to adapt to routing changes, in making OSPF IBB Capable and propose solutions. We then describe a prototype implementation of IBB Capability we have developed using GateD. Using the prototype in an experimental setup, we demonstrate that the overhead of IBB Capability is modest compared to the benefit it offers and has good scaling behavior in terms of network that the overhead of IBB Capability is modest compared to the benefit it offers and has good scaling behavior in terms of network size.

**Keywords:** Routing protocols, Routing stability, OSPF, IBB Capability

# 1 Introduction

While data networks have been increasing in size and complexity over the last few years, little attention has been paid to mechanisms that could ease software maintenance on the switches and routers that make up these networks. Several cases of serious network outages have been reported in recent years, some of which lasted several days. Two of these outages — AT&T (April 1998) and MCI (August 1999) — are known to have been triggered due to software upgrades on the routers in the network [1] and in at least one of them a version of OSPF was being used as the routing protocol to distribute the network topology information. With the society's increasing reliance on data networks, the need to make software maintenance of the routers less disruptive is increasingly being felt.

In this paper we study approaches for preventing routing instabilities when the routing protocol software in a router is made temporarily inactive, for example, to perform a software upgrade. We focus on routers in which the routing engine (software) is decoupled from the forwarding engine (hardware). Most of the current-generation core routers belong to this category. In these systems, there can be situations when the router's forwarding engine is active but its routing engine is down. In these situations, the router can continue forwarding packets even if it cannot participate in the routing protocol(s). The current definitions of routing protocols like BGP [12], OSPF [9, 8] and IS-IS [4] do not support such behavior. These protocols implicitly assume that when the routing process of a router goes down, the router is incapable of forwarding packets and therefore, other routers start routing packets around it. This leads to massive route flapping and instability, especially if the routing process goes down for a short period. This situation arises, for example, when the routing process is brought down for upgrades to the routing software or the operating system, for changes to the router ID or the AS (Autonomous System) number, or for interface renumbering. Route flapping can be avoided if the routing protocol can respond to this special case by letting the affected router continue forwarding packets. This requires extensions to the routing protocols to accommodate such behavior.

In this paper, we focus on the OSPF routing protocol. We propose a way of extending OSPF by adding a new capability called *IBB (I'll Be Back)* to the protocol. This capability enables a router undergoing software maintenance to forward packets for a certain period of time even if its OSPF process is down. We distinguish between two cases in which the OSPF process becomes inactive — graceful shutdown and process crash. IBB applies only to the first case where the OSPF process is brought down in a graceful fashion for a certain period of time. Before going down, the OSPF process informs other routers that it is going down but its forwarding engine is functioning so that the other routers can continue using it for forwarding packets. This avoids route flaps and instability that would have followed had the OSPF process not informed its neighboring routers of its changed state.

Ward and Scudder coined the term IBB while proposing a way of extending BGP to avoid route flaps when the BGP process is brought down [13]. Since BGP relies on underlying Interior Gateway Protocols (IGPs) like OSPF and IS-IS to resolve its next-hops, a similar capability must be built into the IGPs. The IBB extension to OSPF allows us to achieve precisely this and serves as a base for adding a similar capability to BGP as proposed by [13].

Zinin, et al. proposed ways of extending OSPF so that the OSPF process in a router can come up without causing route flaps [14, 15]. Their scheme avoids route flaps only if other routers have not started routing data traffic around the router coming up. This means that the OSPF process in the affected router must come up before other routers detect that it was down. Even if this could be assured, it is difficult to avoid route loops and black holes that can occur due to topological changes while the OSPF process in the affected router was down. In comparison, our IBB scheme provides an elegant way of avoiding routing loops and black holes under similar circumstances. Moreover, we demonstrate the effectiveness of our scheme with a prototype implementation and evaluate its overhead. To our knowledge, this is the first time a comprehensive solution has been proposed, implemented and analyzed for avoiding route flaps and instability when the OSPF process in a router is brought down gracefully.

The paper is organized as follows: In Section 2, we provide a brief overview of the OSPF routing protocol. In Section 3, we describe the IBB extensions that we propose to add to OSPF. In the next section, we discuss details of the prototype implementation that we have developed using the GateD OSPF implementation. In Section 5, we present a detailed account of the experiments we performed to characterize the overhead of IBB extensions. In particular, we discuss the experimentation methodology and results in detail. Finally, in Section 6, we present our conclusions and provide directions for future work.

## 2 Overview of OSPF

This section provides a brief overview of OSPF. For more details, we refer the reader to [9, 8, 10]. OSPF is a widely used intra-AS routing protocol. It is a link-state routing protocol which means that every router running OSPF acquires the entire view of the AS topology. Every router then runs Dijkstra's single-source shortest path algorithm to determine the shortest route to every destination in the AS. Each router describes a part of the topology in one or more *Link State Advertisements (LSAs)*. These LSAs are then flooded reliably throughout the AS. Thus, every router receives LSAs from every other router in the AS. This collection of LSAs forms a link-state database in a router's memory and allows it to create the topological view of the AS.

Every router describes links to all its neighbor routers in a *Router (Type 1) LSA*. Two routers are neighbor routers if they have interfaces to a common network (i.e, they have a link-level connectivity). Neighbor routers form an *adjacency* so that they can exchange routing information with each other. OSPF allows a link between the neighbor routers to be used for forwarding only if these routers have the same view of the topology, i.e., the same link-state database. This ensures that forwarding data packets over the link does not create loops. Thus, two neighbor routers have to make sure that their link-state databases are in sync, and they do so by exchanging parts of link-state database when they establish an adjacency between them. The OSPF specification [9] defines a number of states for an adjacency and ranks them in an increasing order. These states form part of a finite state neighbor machine which is executed (independently) at each router. The highest state of an adjacency is called *full* which indicates that the link-state database of the neighbor is in sync with the router running the state machine. Once the adjacency with a neighbor becomes full, the router can advertise a link to the neighbor in its Router LSA. Every router uses Hello protocol to monitor the state of its adjacencies.

More than two routers (lets say $n$) can have interfaces on a LAN. Instead of forming $n^2$ adjacencies between each pair of $n$ routers on such a LAN, OSPF elects one of the routers as the *Designated Router (DR)* of the network. Every router on the network establishes a full adjacency with the DR. The DR plays a central role in ensuring that the link-state databases of all the $n$ routers are synchronized. The DR also originates a *Network (Type 2) LSA* which contains links to all its fully adjacent neighbors on the network. Each router on the network includes a link to the network (DR) in its Router LSA if it is fully adjacent to the DR. OSPF also elects one of the $n$ routers as a *Backup DR (BDR)* which takes over as DR if the DR fails.

Since LSAs are flooded reliably, all the routers in the AS converge to the same link-state database. Each LSA (Router or Network) in the database represents a vertex and all its outgoing links in the topology graph. Every router runs Dijkstra's algorithm on this topology graph with itself as the root to determine the shortest path to every destination (router and networks) in the AS.

OSPF allows hierarchical routing by splitting an AS into a number of areas. The Router and Network LSAs are area-specific in the sense that they are not flooded beyond area boundaries. Thus, they are said to have an area-level flooding scope. Every interface of a router belongs to exactly one area. Routers that have interfaces in more than one area are called *Border Routers (BRs)*. Every router originates one Router LSA for each area it is attached to. The Router LSA for a particular area describes all the links the originating router has in that area. Border Routers originate *Summary (Type 3 and 4) LSAs* for describing destinations of one area to routers in another area. These LSAs also have an area-level flooding scope like the Router and Network LSAs. OSPF requires one of the areas (area 0) to be designated as the *Backbone area* which plays a central role

in keeping the areas connected and making sure that loops do not occur when the source and the destination of a path are in different areas.

OSPF allows routing information to be imported from other routing protocols like BGP and RIP. The router which imports external routing information into OSPF is called an *AS Border Router (ASBR)*. An ASBR originates *External (Type 5) LSAs* to describe the external routing information. The External LSAs are flooded in the entire AS irrespective of area boundaries, and hence have an AS-level flooding scope.

Every router periodically *refreshes*, i.e., re-originates each of the LSAs it had originated earlier even if the content of the LSA has not changed. This makes OSPF robust against loss or corruption of LSAs. Moreover, each LSA is aged while it resides in the link-state database of a router, and is flushed out of the link-state database when its age reaches *MaxAge* (= 1 hour) after the last refresh [9].

### 2.1 Opaque LSAs

As we observed in the previous section, OSPF has a reliable flooding mechanism which ensures that LSAs are received by all the routers within the flooding scope (area or AS) of the LSA. Opaque LSAs enable the use of this reliable flooding to disseminate application-specific information to all the routers in the area or the AS. The Opaque LSA specification [5] introduces three new types of LSAs in addition to the five types of OSPF LSAs described above: (i) Type 9 LSA which has a link-level flooding scope and is flooded only on the attached links, (ii) Type 10 LSA which has an area-level flooding scope like a Router LSA, and (iii) Type 11 LSA which has an AS-level flooding scope like an External LSA. The application-specific information is stored in the *Opaque Information* field of an Opaque LSA. We make use of the Opaque LSA mechanism to disseminate information related to the IBB extension, as explained in Section 3.

## 3 I"ll Be Back (IBB) Extension to OSPF

In this section, we provide a detailed description of our scheme to introduce IBB capability in OSPF. We discuss the routing database inconsistency problems arising from the addition of IBB capability and propose our solution.

With IBB capability, when the OSPF process on a router $R$ is about to be shut down gracefully, it informs other routers by sending them a message. The message contains the time period during which the OSPF process in $R$ plans to remain inactive. This time period is called the *IBB Timeout* interval. Other routers, upon receiving the message, infer that although $R$'s OSPF process is down, its forwarding engine is still active. Therefore, they continue using $R$ for forwarding data traffic, avoiding route flaps. If $R$'s OSPF process returns to operation within the IBB Timeout interval, IBB capability avoids route flaps completely. On the other hand, if the process remains inactive after the IBB Timeout period, other routers conclude that something is seriously wrong with $R$ and start routing data traffic around $R$. IBB Timeout is a configurable parameter which allows the system administrator to set it to any value based on his/her estimate of how long the OSPF process of $R$ is likely to remain inactive. The set of routers that need to be aware of the state of $R$'s OSPF process forms the *IBB Scope* of $R$. If $R$ is an AS Border Router, its IBB scope is the scope of the External LSAs it originates; otherwise its IBB scope contains all the routers in its attached area(s).

We assume that all the routers within the IBB scope of a router $R$ are IBB capable and agree on the same IBB Timeout value. Otherwise, routing loops and/or black holes can occur. The capability negotiation needed to reach agreement among the routers to support IBB can be done in an out-of-band manner (for example, through configuration).

### 3.1 Link-State Database Inconsistency Problem

As observed in Section 2, OSPF requires all the routers to have the same image of the link-state database. If two routers do not calculate their routing tables from the same image, routing loops or

black holes can result. When router $R$ goes down, its routing table remains frozen until it is back up. If a topological change occurs while the routing process in $R$ is down, it will not be able to update its forwarding database to respond to the change in topology. This implies that within the IBB Timeout period, the forwarding table of $R$ is based on the link-state database image it had at the time it went down. This can lead to routing loops or black holes. Figure 3.1 illustrates how a topological change occurring while the routing process in $R$ is inactive can lead to a routing loop.


(a) Topology when $R$ goes down. (b) Topology at some time after $R$ has gone down.

Figure 3.1: An example of routing loop being formed while $R$ is down.

In the figure, $A$, $B$ and $R$ are routers. As can be seen from the link costs in Figure 3.1(a), $R$'s shortest path to $A$ goes via $B$. Assume that $R$ goes down at this point, thereby freezing its database image and forwarding table. After a while, the link between $A$ and $B$ goes down (Figure 3.1(b)). Under normal circumstances, $B$ could have started using $R$ as the next hop for reaching $A$ because $R$ would have started using the direct link between itself and $A$ to reach $A$. Since $R$'s routing table is frozen, however, a loop is formed if $B$ tries to use $R$ as the next hop for reaching $A$. In summary, when the router $R$ is in IBB shutdown state, other routers such as $B$ cannot calculate their Shortest Path Trees (SPT) as they do normally; router B needs to take into account $R$'s shortest path tree at the time it was shut down, and make sure no loops or black holes are formed. In this paper, we concentrate on loop formation only. We provide a detailed account of when and how loops can form when $R$'s database is not in sync with other routers, and how these loops can be avoided.

**Loop Formation**

To simplify the problem, let us assume that $R$ is the only router in IBB shutdown state, and that all the other routers are using the same image of the database for calculating their Shortest Path Trees (SPTs) and routing tables. $R$'s SPT and routing table are based on the database image it had at the time it was shut down. Since Dijkstra's algorithm makes sure that no loop gets formed when all the routers compute their SPTs from the same image of the database, router $R$ must be part of any loop that gets formed. In fact, the key to the problem lies in the routers that are used by $R$ as next hops for reaching one or more destinations. Since $R$ is down, it is unable to modify its routing table and continues using the same next hops for the IBB Timeout period. This is the root cause of the loop formation problem. To see when packets destined to a particular destination loop, let us pick some arbitrary destination $D$ in the network. We assume that $D$ exists in $R$'s SPT and routing table. If not, $R$ will drop packets destined to $D$ and there cannot be any loops. Let $Y$ be one of the next hops used by $R$ at the time it went down for reaching $D$. During the IBB Timeout period of $R$, $Y$ calculates its SPT as and when required according to the OSPF specification [9]. $D$'s position with respect to that of $R$ in $Y$'s SPT gives rise to the following two cases:

**Case 1:** *$D$ is not in the subtree rooted at $R$ in $Y$'s SPT.*

Figure 3.2 (a) and (b) illustrate the case. In this case, we do not have to worry about loops since packets destined to $D$ cannot loop. Why? Clearly, once the packets reach $Y$, they cannot loop because of the way Dijkstra's algorithm operates. Consider some other router $S$ which has $R$ in its path to $D$ (see Figure 3.2 (c)). The next node after $R$ on the path from $S$ to $D$ in the SPT is shown as $X$ which may or may not be equal to $Y$. But in any case, the packets

From now on, when we state that a router is going down, we mean only its OSPF process is being shut down, unless specified otherwise.

The situation where $R$ or $D$ does not exist in $Y$'s SPT is covered by this case.

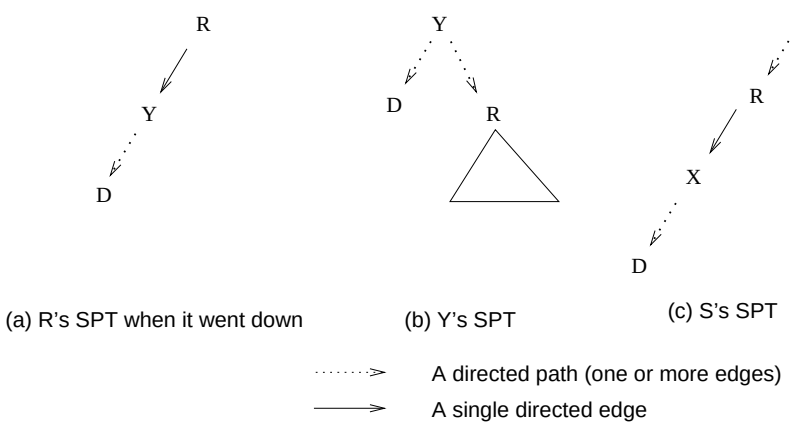



Figure 3.2: $D$ is not in the subtree rooted at $R$ in $Y$'s SPT.

destined to $D$ will reach $R$ without looping. $R$ will forward them to $Y$ instead of forwarding them to $X$ as $S$ would have expected, and we know that once the packets have reached $Y$, they will reach $D$ without looping. Thus, packets destined to $D$ cannot loop in this case.

**Case 2:** *$D$ is in the subtree rooted at $R$ in $Y$'s SPT.*

This case arises because $R$'s SPT has been calculated from the database image which is not in sync with that of other routers (including $Y$). This case definitely leads to the formation of a loop as can be seen from Figure 3.3. Therefore, each router that has $R$ on its path to $D$ needs to calculate a new path to $D$ that does not have $R$ on the path. Those routers which do not have $R$ on their paths can continue using them.

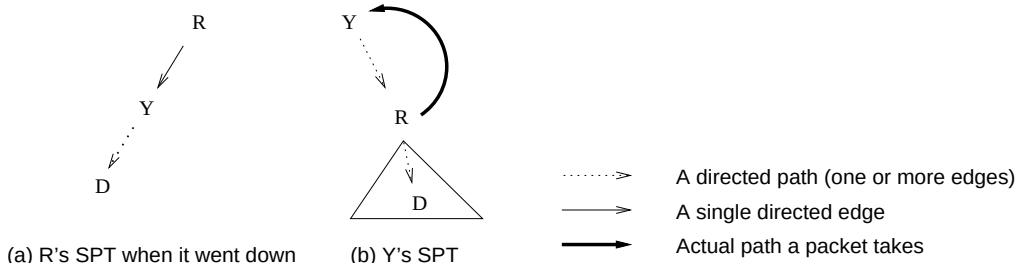



Figure 3.3: $D$ is in the subtree rooted at $R$ in $Y$'s SPT.

To summarize, packets destined to $D$ cannot loop in case 1 whereas they definitely loop in case 2. This loop can be avoided if all the routers calculate a path to $D$ from a topology graph which does not have node $R$ in it. If all the routers (except $R$) perform this calculation, Dijkstra's algorithm makes sure that the packets destined to $D$ cannot loop. Note that only those routers that have $R$ on their paths to $D$ in their original SPT have to actually carry out this calculation. The rest of the routers can continue using their original paths to $D$ since the calculation will not change these paths anyway.

We now discuss how we can avoid the loop in case 2. To start with, assume $Y$ knows that $R$ is using it as the next hop for reaching $D$ at the time it went down (Section 3.1 describes how $Y$ gets this information). Then $Y$ can check $D$'s position relative to that of $R$ every time it calculates its SPT during the IBB Timeout period of $R$. If $Y$ encounters case 1, it does not have to do anything. If it encounters case 2, on the other hand, it needs to recalculate its path to $D$ such that $R$ no longer remains on the path. In addition to that, it also needs to send a message to all the routers asking them not to use $R$ for reaching $D$ (see Section 3.1 for the format of the message and other related details). If some other router has $R$ on its path to $D$, it needs to recalculate its path upon receiving the message just like $Y$ did. The router does not have to do anything if it does not have $R$ on its

path to $D$. Essentially, we make $Y$ "in charge" of ensuring that packets destined to $D$ do not loop because of $R$'s frozen routing table. Generally speaking, for every destination in $R$'s SPT, the next hop used by $R$ is made "in charge" of ensuring that packets destined to that destination do not loop, by informing others not to use $R$ for reaching that destination if required. Looking at this in a different way, we can say that each next hop of $R$ needs to make sure that packets destined to any of its descendents in $R$'s SPT do not loop.

So far we have seen how loops can get formed and how we can avoid them. It remains to be proved that all occurrences of loops resulting from $R$'s out-of-sync database are covered by the case 2 above. We now sketch a proof. Again, we concentrate on some arbitrary destination $D$ in the topology graph. Let $Y$ be the next hop used by $R$ for reaching $D$ as we had in the discussion above. Consider any arbitrary router $S$ in the graph. We want to show that if packets sent by $S$ towards $D$ loop, $Y$ will detect this loop as case 2. Remember that all the routers (this includes both $S$ and $Y$) other than $R$ use the same image of the database for calculating their SPTs. Accordingly, let us say, $S$ calculates its SPT at some point of time. If the path to $D$ does not contain $R$ on it, Dijkstra's algorithm makes sure that packets forwarded to $D$ by $S$ do not loop. Therefore, let us assume that $S$'s path to $D$ does contain $R$ on it and the next node after $R$ on this path is $X$ (see Figure 3.4). If $S$ uses this path to reach $D$, packets will not loop till they reach $R$. Once the packets reach $R$,

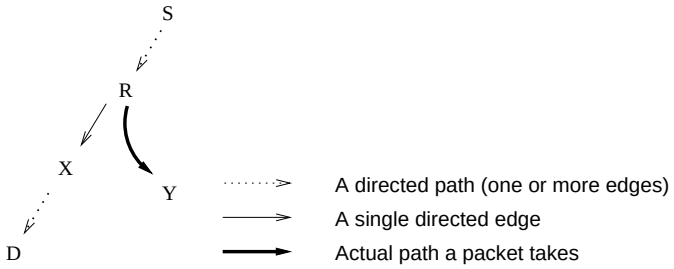



Figure 3.4: Router $S$'s SPT. According to $S$, the next node after $R$ on its path to $D$ should be $X$. But $R$ uses $Y$ as the next hop for the packets destined to $D$.

$S$ would expect $R$ to forward them to $X$, but since $R$ has $Y$ as the next hop for $D$ in its routing table, it would forward the packets to $Y$. Note that $Y$ may or may not be equal to $X$. If it is equal to $X$, packets reach $D$ without looping. Therefore, let us assume that $Y$ is not equal to $X$. Even in this case, any loop that gets formed will definitely involve $R$ and $Y$ because $R$'s routing table is not in sync with other routers, $Y$ is the next hop in its routing table for reaching $D$, and all the other routers use the same database image to calculate their SPTs. The loop may or may not involve $S$ and based on that we get the following two situations:

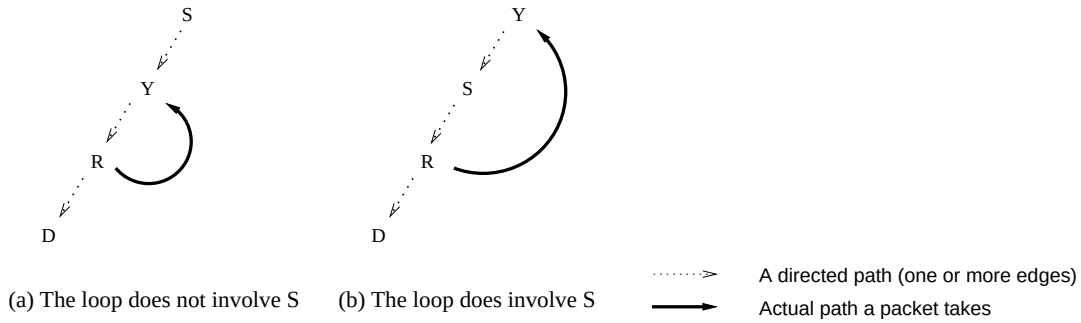



Figure 3.5: Situation where packets sent by $S$ to $D$ loop.

1. The loop does not involve $S$. This happens only if $Y$ lies between $S$ and $R$ on the path from $S$ to $D$. Figure 3.5 (a) shows this case.

2. The loop involves $S$. This is possible only if $S$ lies between $Y$ and $R$ on $Y$'s path to $D$. Figure 3.5 (b) shows this case.

In either of the cases, $Y$ will get $D$ in the subtree rooted at $R$ in its SPT since both $Y$ and $S$ calculate their SPTs from the same image of the database. In other words, $Y$ will encounter case 2 in both the cases and hence will detect the loop. Thus, we have shown that for any general router $S$, packets sent to some destination $D$ will loop only if the next hop $Y$ used by $R$ encounters case 2 as far as $D$ is concerned.

Till now, we have assumed that $R$ is the only router that is down and all the other routers have the same image of the database. Let us relax this and allow more than one router, say $n$ of them, down at the same time. We denote these $n$ routers by $R_1, R_2, \ldots, R_n$. As in the single router case, we can have the next hop used by a particular router $R_i$ for reaching a particular destination $D$ "in charge" of making sure that using $R_i$ to reach $D$ does not create loops. If the router "in charge" detects a loop (case 2 as far as $D$'s position relative to $R_i$ is concerned), it asks the other routers not to use $R_i$ for reaching $D$. Other routers can calculate a new path to $D$ by removing the said $R_i$ from the path if it is there. Note that with $n$ routers down, other routers can get messages asking them not to use $j$ $(1 \leq j \leq n)$ of these $n$ routers for reaching $D$. In that case, these routers can calculate a new path to $D$ without the $j$ forbidden $R_i$'s on it. We take a simpler approach than this. When a router receives a message asking it not to use one or more $R_i$'s on its path to $D$, the router tries to calculate a path to $D$ that does not have any of the $n$ routers, not just the forbidden ones. In order to calculate this path, the router simply calculates a new SPT which does not have any of the $n$ down routers in it. The simplicity comes from the fact that the router can use the same SPT (that is, the one without any of the $n$ down routers in it) for finding paths to all those destinations for which it needs to find a path without some $R_i$'s on it. We earlier showed that the algorithm avoids loop in the case when one router is down. It can be shown that the algorithm just mentioned avoids loops even when $n$ routers are down. Figure 3.6 shows the procedure each router has to follow after calculating its SPT in order to avoid loops when one or more routers are down. We follow a similar approach in tackling the black hole problem.

We now describe the details of communicating IBB-related information between the routers so as to implement the ideas described above. This communication is achieved by the use of Opaque LSAs in OSPF. We now define various Opaque LSAs used for this purpose. For brevity, we omit the detailed message formats of these LSAs.

### Path Info LSA

As we observed in the previous section, the next hop(s) used by router $R$ is "in charge" of making sure that using $R$ to reach a particular destination $D$ does not lead to loops. Thus, before going down, $R$ needs to inform the next hop router that it is using the router as the next hop for reaching $D$. $R$ does so by sending one or more Opaque LSAs [5] out on its attached links and networks. The type of these LSAs is set to 9 which means that the LSAs have a link-level flooding scope. These LSAs are called *Path Info LSAs*, and describes $R$'s OSPF routing table by listing each destination and one or more next hops used by $R$ for reaching that destination. Before sending Path Info LSAs out, $R$ stops accepting any new LSAs and freezes its routing table. Other routers ($R$'s neighbors) hold onto Path Info LSAs for the IBB Timeout period specified by $R$ and use them in the procedure described in Figure 3.6.

### Avoid Router LSA

When a router wants to inform others not to use $R$ for reaching a particular destination $D$, it does so by flooding an Opaque LSA [5] into the associated area. The type of this LSA is set to 10 which means that the LSA has an area-level flooding scope. This LSA is called an *Avoid Router LSA*. The LSA lists the router ID of $R$ and a list of those destinations for which using $R$ on the path leads to a loop as discovered by the originating router. As we saw earlier, the originating router is always one of the next hop routers used by $R$ for reaching each of the destinations mentioned in the LSA. It should also be noted that a router has to originate one Avoid Router LSA per down router

1. For each node $D$ in the SPT, the router needs to perform the following steps:
   (a) Consider each of the equal cost paths the router has to reach $D$ one by one. For each path, perform the following steps:
      i. If the path does not go through any of the $R_i$s, skip the next step.
      ii. The path does go through some $R_i$s. For each such $R_i$, "cancel" this path if either of the following holds true.
         - Some router (this includes the local router also) has sent a message asking others not to use $R_i$ for getting to $D$ (see Section 3.1 for the format of the message).
         - The router checks for the next hops used by $R_i$ for reaching $D$ and at least one of them turns out to be equal to the local router (this corresponds to case 2 in Section 3.1).

      If the path got "canceled" due to the second condition above, "remember" $R_i$ and $D$. At the end, for all such "remembered" $R_i$ and $D$, the router will have to send a message asking others not to use $R_i$ on their paths to $D$.
   (b) If all the paths to $D$ in the router's SPT got "canceled" above, "mark" $D$. Marking of $D$ means that the router needs to find an alternate path to $D$ that does not have any of the $n$ down routers on it (this is done in step ii below).
2. If at least one such $D$ got "marked" in the previous step, calculate a new SPT with all the $R_i$s ($1 \leq i \leq n$) removed.
3. For each un"marked" destination, determine the next hop(s) from the first SPT (the one with all $R_i$s in it). For each "marked" destinations, determine the next hop(s) from the second SPT (the one with all $R_i$s removed). If a "marked" destination does not exist in the second SPT, declare it as unreachable.
4. For each "remembered" pair consisting of some $R_i$ ($1 \leq i \leq n$) and some $D$, the router needs to announce to others not to use $R_i$ for reaching $D$. If no other router has sent a message to that effect so far, the local router should do so now.

Figure 3.6: Extended SPT calculation each router has to perform for down routers. We assume that $R_1, R_2, \ldots, R_n$ denote the routers that are down. Furthermore, we also assume that the IBB Timeout of each of them is yet to elapse.

if there are more than one routers about which it wants to inform others. Any other router, upon receiving an Avoid Router LSA, takes its recent SPT and executes the procedure given in Figure 3.6 for the destinations mentioned in the LSA.

## 3.2 Shutdown Process

When a router $R$ is about to be shut down in the IBB mode, it needs to inform all the routers in its IBB Scope just before it goes down. It does so by flooding an Opaque LSA. In particular, a non-ASBR originates a type 10 Opaque LSA for all its attached areas. An ASBR, whose IBB Scope is equivalent to the External LSAs' flooding scope, originates a type 11 Opaque LSA. This LSA is called an *IBB Cease LSA*, and contains the IBB Timeout value. As mentioned earlier, IBB Timeout value indicates the time period during which $R$'s forwarding engine is considered up even if its OSPF process is down. After $R$ originates an IBB Cease LSA, it originates one or more Path Info LSAs as described in Section 3.1.

When a router receives an IBB Cease LSA from $R$, it holds onto all the LSAs originated by $R$ for the IBB Timeout period of time. Thus, LSAs originated by $R$ are not flushed when their age reaches MaxAge as is done normally in OSPF. Instead, they are deleted only when the age of the IBB Cease LSA reaches IBB Timeout value. This also applies to the Path Info LSAs originated by $R$ as well as Avoid Router and DR Change LSAs (we describe DR Change LSAs next) related to $R$. Moreover, the router includes $R$ as a down router in the procedure given in Figure 3.6. If the

router is fully adjacent to $R$ at the time it receives the IBB Cease LSA from $R$, it needs to continue including links to $R$ in those self-originated Router and Network LSAs that had links to $R$ at the time $R$ went down. This makes sure that there are links leading to $R$ in the link-state database. The router continues doing so till IBB Timeout occurs.

When $R$ is attached to a LAN, the link to the network is represented as a link to the Designated Router (DR)'s IP address in the Router LSA of $R$ [9]. If DR on such a network changes, $R$ normally originates its Router LSA with all the links again. But if $R$ is down, it cannot re-originate its Router LSA when DR on one of its attached network changes. Re-origination of this link needs to be taken care by the newly elected DR. The new DR generates an Opaque LSA called a *DR Change LSA* that describes $R$'s link to the network. A DR Change LSA is a type 10 Opaque LSA, and contains the Router ID of $R$ as well as the IP addresses of the previous DR and the newly elected DR (i.e., itself). When other routers in the associated area receive the DR Change LSA, they replace the link to the previous DR in $R$'s Router LSA with a link to the the new DR. If $R$ itself is a DR or a BDR on some LAN, other routers on the LAN elect a new DR or a BDR when they receive the IBB Cease LSA from $R$. This is because of the central role played by DR and BDR in keeping the link-state database of all the routers on the network in sync. This re-election of DR can lead to some amount of route flapping which unfortunately is hard to avoid.

## 3.3 Returning from IBB Shutdown

If router $R$ that had sent an IBB Cease LSA returns back to operation before the IBB Timeout occurs, the goal is to achieve minimal route flapping and disruption. If it comes back after the IBB Timeout has occurred, it is treated as if it is coming back from a normal shutdown. $R$ needs to write the fact that it sent an IBB Cease LSA along with the IBB Timeout value to some form of permanent storage before it goes down. This allows it to determine whether it is coming back up within the IBB Timeout period. If $R$ finds that it has come back after the period (and hence all the routers consider it as dead at present), it undergoes the normal OSPF procedure as specified in [9]. If it comes back up within IBB Timeout period, however, it originates an Opaque LSA of type 10 or 11 to announce that it is operational again. This LSA is called an *ICB Open LSA*. The router ID used by $R$ in the IBB Cease LSA is listed in the Opaque Information field of the ICB Open LSA. This allows other routers to relate the new router ID of $R$ with its old router ID in the case when $R$ was brought down to facilitate a change in its router ID. $R$ originates any other type of LSAs (including its Router LSA) only after it has originated the ICB Open LSA.

$R$ originates an ICB Open LSA after all its "potentially fully adjacent neighbors" have become fully adjacent. In other words, $R$ does not originate an ICB Open LSA as soon as its adjacency with one neighbor becomes full if possible. Instead, whenever the adjacency with a neighbor becomes full, $R$ checks if its adjacency with any other neighbor is in a state greater than 2-way or not. If it finds no such neighbor, it originates an ICB Open LSA. On the other hand, if $R$ finds that one or more of its neighbors are in a state greater than 2-way, it waits for these neighbors to become fully adjacent. Since $R$ originates all its other LSAs including the Router LSA only after it has originated the ICB Open LSA, waiting for all potential fully adjacent neighbors to actually become fully adjacent allows $R$ to describe most, if not all, links it had before going down in its new Route LSA. This plays a key role in avoiding route flaps that can occur due to premature origination of Router LSA containing only a handful of the links that $R$ had before going down.

When a router receives the ICB Open LSA from $R$, it assumes that $R$'s link-state database is in sync and does not consider $R$ as a down router in the procedure given in Figure 3.6. For this reason, we do not allow $R$ to originate the ICB Open LSA till it has become fully adjacent to at least one neighbor. Even after receiving the ICB Open LSA from $R$, the router holds onto all the LSAs of $R$ for the IBB Timeout period. This allows $R$ to gradually refresh its LSAs after it has come back and avoids route flapping. An important implication is that the system administrator can set the IBB Timeout period to a value that gives $R$ enough time to re-originate all its LSAs after it has come

Like full, 2-way is one of the states of an adjacency [9].

back up. An ICB Open LSA is flushed out when the IBB Timeout occurs. Note that $R$ will not refresh the ICB Open LSA unlike the other LSAs it originates.

# 4 Implementation of IBB Extension

This section describes a prototype implementation of our IBB extension to OSPF. We used GateD (Gate Daemon) [2] which is a popular, public-domain routing software platform for implementing the prototype. This section describes the prototype and throws light on some key implementation issues. We also used the prototype to measure the run-time overhead of the IBB extension, presented in Section 5.

## 4.1 Overview of GateD

GateD is a routing software platform that runs on the UNIX operating system and its variants. We used version 4.0.7 of GateD for the prototype implementation. GateD has implementations of various routing protocols. The version we used has implementations of BGP, EGP, IS-IS, RIP and OSPF.

GateD runs as a single UNIX process. The main abstraction implemented in GateD is that of a *task*. A task is a thread of execution and different tasks can be scheduled independently of one another. Different protocols make use of this abstraction in different ways. For example, the entire OSPF protocol is implemented as a single task whereas BGP uses one task per peer. Each task normally has a socket associated with it. Tasks perform computation when some activity like packet arrival occurs on the associated socket. Tasks can also perform computations by scheduling *jobs*. For example, OSPF schedules its SPT calculation as a job. GateD also provides an abstraction of *timers* which are normally used by protocols for variety of purposes. For instance, OSPF has a HelloTimer associated with each of its interfaces and sends a Hello packet out on the interface when the timer expires. Timers are also used for scheduling jobs. For example, OSPF uses a timer to schedules a job that carries out the SPT calculation.

GateD maintains routes to various destinations in its *RIB (Routing Information Base)*. These routes are installed in the RIB by various routing protocols. GateD also assigns a preference to each route and in case of a tie, the route with the least preference is made the "active" route in the GateD RIB. Changes in the RIB are communicated to the routing protocols using the flash handler routine registered by each protocol. This allows protocols to take appropriate actions when the GateD RIB changes. For example, if the router is an ASBR, OSPF may have to originate an External LSA if it learns of a new route from the RIB. It is important to note here that the *FIB (Forwarding Information Base)* which is used by the operating system kernel for forwarding packets is different from GateD's RIB. GateD is allowed to install routes into the FIB since it has root privileges. GateD can also learn routes from the FIB and install them in its RIB.

## 4.2 Implementation Details

This section describes the implementation details of the IBB extension to GateD. We assume that $R$ is the router that has gone down by sending an IBB Cease LSA for the purpose of the discussion in this section. The prototype increases the binary size of GateD by about 4%.

In the prototype, a router stores information about each down router in an *ibb-info* block. So, if $R$ is the down router, this block stores the router ID of $R$, the IBB Timeout value and other book-keeping information. The block remains active until IBB Timeout occurs even if an ICB Open LSA is received from $R$ before the IBB Timeout period. The life-time of an ibb-info block is divided into two parts: *post-ibb-cease-mode* and *post-icb-open-mode*. The block is in post-ibb-cease-mode when it is created upon receiving an IBB Cease LSA from $R$ and stays there till the router receives an ICB Open LSA from $R$ at which point the block transitions to post-icb-open-mode. It is obvious

GateD consortium has now become a company called NextHop Technologies and is the commercial provider of GateD.

that the ibb-info block of $R$ can spend its entire life-time in post-ibb-cease-mode if no ICB Open LSA is received from $R$ within the IBB Timeout period.

While the ibb-info block of $R$ is in post-ibb-cease-mode, the router ignores all non-IBB LSAs (e.g., Router LSA) originated by $R$. This allows the router to clearly distinguish between the state of $R$ before it went down, the state of $R$ while it is down, and the state after $R$ comes back. Unfortunately, this imposes some ordering requirements across different types of LSAs related to $R$ as described in Section 4.2. Once the ibb-info block transitions to post-icb-open-mode, the router starts accepting non-IBB LSAs originated by $R$ which allows $R$ to gradually refresh its LSAs. The router does not accept any IBB related LSAs other than the ICB Open LSA once the block transitions to post-icb-open-mode. During the entire life-time of the ibb-info block of $R$, LSAs originated by $R$ are not flushed even if their age have reached MaxAge. This is essential to avoid route flaps as we have seen in Sections 3.2 and 3.3. The ibb-info block is deleted when the IBB Timeout occurs. Figure 4.1 summarizes the life-time of the ibb-info block related to $R$.

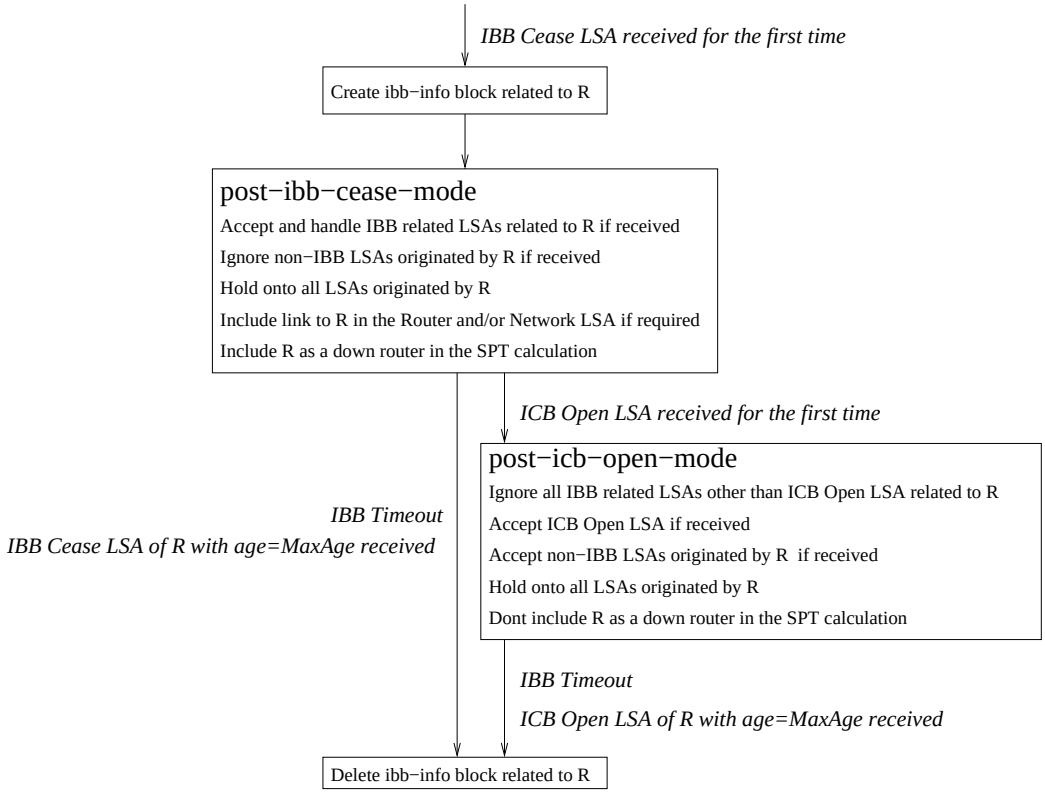



Figure 4.1: The life-time of an ibb-info block related to $R$.

### LSA Ordering

Under normal circumstances, OSPF is not sensitive to the order in which different LSAs are received [9]. This is the reason why GateD does not care about the order in which LSAs are received or sent. Unfortunately, this is not true with the IBB extension. Assume router $R$ goes down after originating an IBB Cease LSA, then comes back and originates an ICB Open LSA. If these LSAs are delivered out of order to some router, the receiving router first ignores the ICB Open LSA because it does not know that $R$ earlier sent an IBB Cease LSA. It subsequently receives the IBB Cease LSA , considers $R$ as down, and may never receive another ICB Open LSA. Thus, it is imperative that the relative order of IBB Cease and ICB Open LSAs is preserved. Apart from this requirement of

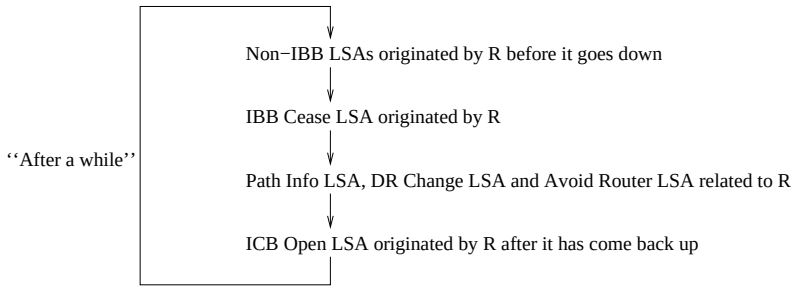



Figure 4.2: The ordering that needs to be preserved amongst LSAs related to $R$ in our prototype.

preserving the order of an IBB Cease LSA and the corresponding ICB Open LSA, the prototype also imposes some more ordering requirements. These requirements are summarized in Figure 4.2.

The figure clearly shows that the ordering requirement is cyclic. It is important to mention here that this ordering applies to LSAs related to $R$ only; no ordering needs to be preserved between LSAs related to different routers. The prototype breaks the cycle shown in the figure by assigning highest preference to the ICB Open LSA. In other words, the prototype sends out the LSAs in the following order: ICB Open LSAs, non-IBB LSAs, IBB Cease LSAs, and other IBB related LSAs. This automatically preserves the order in which LSAs related to $R$ are originated while $R$ is active. A problem arises when some router has both the IBB Cease LSA of $R$ generated before it went down and the ICB Open LSA of $R$ after it came up. The prototype will send out these two LSAs out-of-order which can lead to problems. We believe that chances of this event are rare since the IBB Cease LSA and other IBB related LSAs related to $R$ are deleted when an ICB Open LSA is received.

The prototype imposes the above mentioned ordering by introducing priority levels amongst LSAs that needs to be sent out. For example, ICB Open LSAs have the highest priority, the non-IBB LSAs are at the next priority level, and so on. The LSAs are then served in the strict priority order. It is worth mentioning here that even with this priority-based serving of LSAs, the LSAs can arrive out-of-order at a router due to losses at the link level. We believe that this can be avoided by using sequence numbers in various LSAs related to $R$.

### SPT Calculation

The prototype implements the procedure given in Figure 3.6 for avoiding loops when $R$ is down. As we mentioned earlier, OSPF allows hierarchical routing by splitting an AS into areas. OSPF performs SPT calculation in three stages: In the first stage it calculates all the intra-area routes using Router and Network LSAs, in the second stage it calculates all the inter-area routes using Summary LSAs, and in the final stage it calculates all the external routes using External LSAs [9]. It turns out that we have to apply the procedure given in Figure 3.6 to each stage of SPT calculation.

In order to implement the procedure given in Figure 3.6, the prototype employs two passes of Dijkstra's algorithm. The first pass is done normally, i.e., it includes all the down routers in the SPT calculation. The second pass, on the other hand, excludes all the down routers that are in post-ibb-cease-mode while calculating the SPT. For every destination $D$, the prototype keeps track of the down routers on each of the paths to $D$ during the first pass. At the end of the first pass, the prototype "cancels" one or more paths according to step 1(a) of Figure 3.6 before proceeding with the second pass. These two passes are employed in succession for each stage of the SPT calculation. In other words, the prototype first executes two passes for stage 1 (intra-area routes). At the end of stage 1, it installs routes calculated according to the first or the second pass for each destination depending on whether the destination got "marked" according to step 2 of Figure 3.6 or not. The prototype also keeps track of routes calculated in both the passes for every destination. It also keeps track of the list of down routers along every path calculated in the first pass for each destination. This is required for later stages (inter-area and external route calculation) since these stages build

upon the tree built in stage 1. At present, the prototype implements the two passes only for intra-area stage. We believe that there should not be any fundamental difficulties in implementing the same two pass procedure for inter-area and external stages of the SPT calculation.

It is worth mentioning here that the two passes are invoked only if there is at least one down router in post-ibb-cease-mode. When no such router exists, our prototype executes only one pass with minimal overhead of the procedure. This ensures that the overhead of the procedure given in Figure 3.6 is incurred only when it is actually required.

**Future Enhancements**

Our prototype does not currently support the entire IBB extension described in Section 3. The following are the features it currently does not support, which we plan to add in the future.

1. Change of router ID. Although one of the intentions of IBB extension is to facilitate a seamless change of router ID, the prototype does not support it. We are currently working on the prototype so that it can handle a change of router ID.
2. GateD withdraws the routes that it has installed in the kernel FIB when it goes down. Ideally, this should be prevented. The prototype does not prevent GateD from withdrawing its routes from the kernel FIB for two reasons: First, this requires interaction between OSPF and FIB which is very much router architecture dependent, and hence is not a part of the IBB extension. Second, the objective of the prototype is to see that OSPF running on other routers continue using $R$ in their routing table calculations as if it never went down, and not whether $R$'s FIB is actually "usable" or not.
3. The procedure mentioned in Figure 3.6 is implemented only for the intra-area stage of SPT calculation. We believe that the procedure can be implemented in a similar fashion for the remaining stages of the SPT calculation.
4. The prototype allows only one down router at any given time. Extending the prototype to allow more than one down router at a given time is part of ongoing work.

# 5 Performance

This section describes the results of the experiments we performed to get an idea as to what the overhead of the IBB extension is. First we will describe our experimental setup and then we will present the results.

## 5.1 Experimental Setup

The experiments were carried out in a 5 router test-bed. The test-bed is shown in Figure 5.1. All the routers are PCs running Linux as their operating system. Router pc5 has a 500 MHz Intel Pentium-II CPU with 128 MB of RAM whereas the other routers have 550 MHz AMD-K6-2 CPU with 128 MB of RAM each. Routers pc2, pc3, pc4 and pc5 run Gated as the routing software (IBB extended or original) whereas pc1 runs a modified version of the OSPF implementation (*ospfd*) from John Moy [10]. We have extended ospfd so that it can act as a topology simulator as well as a router. This allows us to make routers other than pc1 believe as if there is a cloud of routers and/or networks "behind" pc1. Note that all the routers in the test-bed and the simulated topology belong to the same area, namely area 0. When ospfd on pc1 comes up, it reads a config file which specifies the topology to be simulated and originates the appropriate LSAs. We do not change the simulated topology once ospfd has started, but ospfd refreshes the simulated LSAs when their age reach *LSRefreshTime* (30 minutes) as it does with its self-originated LSAs [9]. The age of the simulated LSAs when they are originated for the first time is selected randomly so that they are refreshed by ospfd at different times.

We characterize the overhead of IBB extensions in terms of two things: First, overhead per SPT calculation because of the procedure given in Figure 3.6. In order to do this, we measure the mean time it takes for GateD to compute its SPT with and without IBB extension under different

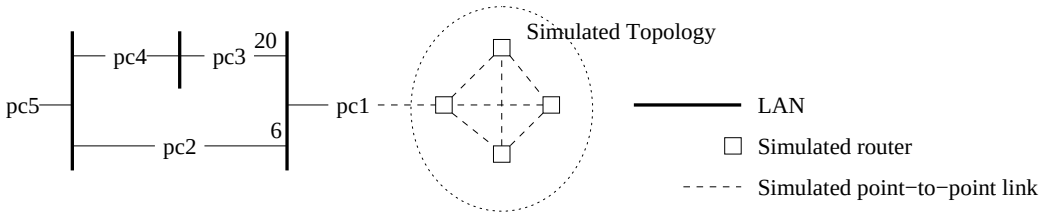



Figure 5.1: Experimental setup used for performance evaluation.

circumstances. Second, overall overhead of IBB extension. For this, we measure the total CPU time used during the experiment by GateD with and without IBB extension. From here on, we refer to the GateD with IBB capability as IBB-Gated and the original unmodified GateD as O-GateD. We use the term GateD generically to refer to both. Both IBB-GateD and O-GateD are instrumented to measure the time it takes for each SPT computation. As mentioned above, pc1 refreshes the simulated LSAs at random times which makes sure that Gated running on the other PCs receive a number of refreshes during the course of an experiment. The number of such refreshes, though random, depend on the number of simulated LSAs originated by pc1. It is worth mentioning here that GateD schedules an SPT computation for every LSA it receives, hence the number of SPT computations depends on the number of LSAs GateD receive during the course of the experiment.

To conduct our experiments, we perform the following sequence of events with IBB-GateD as well as O-GateD using the same simulated topology: At time $t = 0$ minutes, we start ospfd on pc1 and GateD on pc2 through pc5. At $t = 3$ minutes, (GateD on) pc4 goes down for a while. With O-Gated, pc4 simply goes down, so its neighbors (pc2, pc3 and pc5) drop their adjacencies with pc4. With IBB-GateD, on the other hand, pc4 sends an IBB Cease LSA with IBB Timeout set to 10 minutes before going down (it also sends Path Info LSAs to its neighbors). At $t = 8$ minutes, a topology change happens in the testbed. Before the change, cost from pc2 to the LAN(pc1, pc2, pc3) is 6 and the cost from pc3 to the same LAN is 20. Cost of all the other links is 1. This forces pc4 to use pc2 as the next hop for reaching pc1 and all the destinations in the simulated topology. At $t = 8$ minutes, we change the cost of pc3's link to the LAN from 20 to 1. In the case of IBB-GateD, this forces pc2 to generate Avoid Router LSAs for pc1 and destinations in the simulated topology which in turn forces both pc2 and pc5 to use a second pass for selecting the next hops for these destinations. Thus, this change allows us to measure the overhead when a topology change happens while a router is down. At $t = 10.5$ minutes, pc4 comes back which is 2.5 minutes before the IBB Timeout. Therefore, pc4 originates an ICB Open LSA as mentioned in Section 3.3. At $t = 13$ minutes, the IBB Timeout happens and other PCs delete the ibb-info block related to pc4. The sequence terminates at $t = 16$ minutes at which point GateD on pc2 through pc5 and ospfd on pc1 are terminated. Note that we give enough time for the routers to settle down with their adjacency formation and routing tables before pc4 goes down and after the IBB Timeout occurs. We use an Expect [7] script which runs on pc2 to carry out events in the sequence.

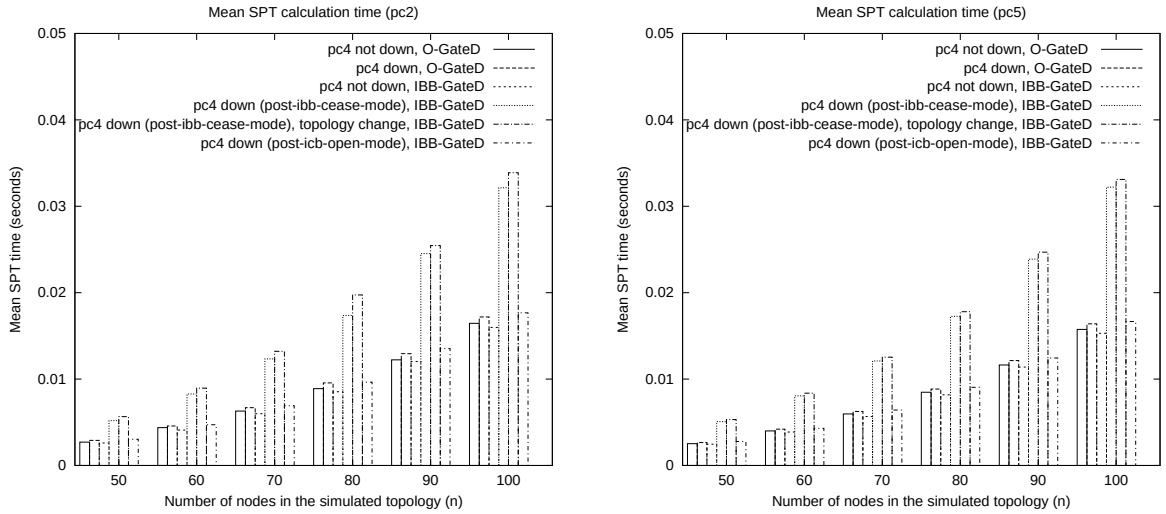
Our instrumentation of GateD code allows us to keep track of the time it takes GateD for each SPT calculation. During the course of the sequence GateD performs a number of SPT calculations as and when required. The instrumented GateD code logs time taken for each of these SPT calculations. Apart from logging the time, GateD also logs enough information for each SPT calculation, so that we can identify principal events that happened during the course of the sequence. For example, for each SPT calculation, GateD logs the number of destinations for which it has to carry out the second pass during that SPT calculation. This allows us to unambiguously determine which SPT calculations happened after the topology change in the sequence. At the end of the sequence, GateD also logs the total amount of CPU time it used during the course of the sequence which we use to quantify the overall overhead of IBB extension as mentioned earlier.

For the simulated topology, we use a fully connected graph of $n$ routers. In particular, each of the $n$ routers is connected by a point-to-point link to every other router in the simulated topology. The cost of each link is a randomly chosen integer between 1 and 10 (inclusive). Though this is not a very realistic intra-area topology, it stresses the SPT calculation the most and hence helps us

measure the overhead in a worst case scenario. We present results for $n$ in the range 50, 60, 70 upto 100 in the next section. For each simulated topology, we carry out the above mentioned sequence for both IBB-GateD and O-GateD. In order to achieve statistical accuracy, we repeat the sequence 12 times for each instance of the simulated topology (i.e., each value of $n$).

## 5.2 Results

Our first set of results show the overhead that IBB extensions introduce to every SPT calculation. The instrumented GateD code uses the Time Stamp Counter (TSC) [3] available on x86-based PCs to measure the time it spends in each SPT calculation. GateD reads the TSC before and after each SPT calculation and uses the difference to calculate the SPT time. The TSC allows very precise measurements of time. The only problem with TSC is that it is not process specific, so the measurement can get skewed if the Linux scheduler intervenes and schedules another process while SPT calculation is underway. We have seen skewed SPT calculation times in our measurements, but the percentage of such skewed measurements is very small. Moreover, the sample variance of mean SPT calculation time is 3 orders of magnitude smaller than the sample mean which validates the accuracy of our measurements. Moreover, we have checked the sanity of our measurements by comparing them with the SPT time values obtained with a function that reports CPU time used by a process, albeit with much less precision than the TSC.




(a) pc2 (b) pc5

Figure 5.2: Comparison of mean SPT calculation time under 6 different cases.

Figure 5.2 shows the mean SPT time as measured on pc2 and pc5 for 6 different cases: (i) when pc4 is up with O-GateD, (ii) when pc4 is down with O-GateD, (iii) when pc4 is up with IBB-GateD, (iv) when pc4 is down (post-ibb-cease-mode) but no topology change has happened with IBB-GateD, (v) when pc4 is down and the topology change has forced the other routers to use second pass for almost all the destinations with IBB-GateD, and (vi) when pc4 has announced it has come back but IBB Timeout is yet to happen (pc4 would be in post-icb-open-mode at other routers) with IBB-GateD. Since cases (i) and (ii) represent the mean SPT time with O-GateD, they serve as the base-points for comparison with mean SPT times obtained for IBB-GateD. The mean SPT time is computed from the appropriate SPT measurements in 12 samples for O-GateD and IBB-GateD. As can be seen from Figure 5.2, SPT computation incurs significant overhead with IBB extension when pc4 is down (cases (iv) and (v)). Moreover, the overhead increases only slightly after the topology change takes effect. This is because our prototype employs two passes even if none of the destinations actually require the second pass. The slight increase after the topology

change stems from the fact that with the topology change, IBB-GateD has to deal with the Avoid Router LSA when it "cancel"s path calculated at the end of the first pass for each destination in the simulated topology. It is also worth noting here that in IBB-GateD, SPT computation incurs negligible overhead when no router is in post-ibb-cease-mode compared to O-GateD (compare mean SPT time in cases (iii) and (vi) with those of cases (i) and (ii)). Another surprising observation is that mean SPT time lowers slightly when no router is down with IBB-GateD (case (iii)) as compared to the case when no router is down with O-GateD (case (i)). We believe that this is a positive side-effect of the changes we had to make to the SPT computation part in GateD while developing the prototype.

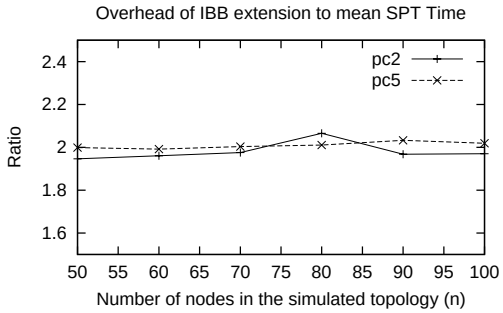
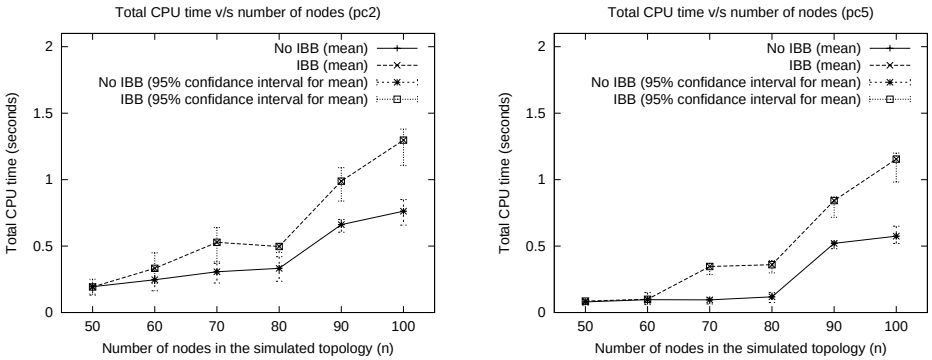



Figure 5.3: Overhead of IBB extension on mean SPT time versus the number of nodes ($n$) in the simulated topology.

Figure 5.2 shows that IBB extension introduces the maximum overhead to SPT computation after the topology change has happened while pc4 is down. An interesting question to ask is how this overhead varies as the size of the network increases. To answer this question, we compute the ratio of mean SPT time in case (v) to that in case (ii). Figure 5.3 shows how this ratio varies with the number of nodes ($n$) in the simulated topology for both pc2 and pc5. As we can see from the figure, the overhead ratio stays flat as the size of the network increases which clearly demonstrates the scalability of the IBB extensions. The figure also shows that the ratio is roughly 2 which is due to the two passes of the Dijkstra algorithm employed in the IBB extension when pc4 is down. We believe that this overhead incurred due to IBB extension is modest compared to the benefits it offers, and it can be reduced further by employing an incremental SPT algorithm [6, 11] for the second pass.




(a) pc2 (b) pc5

Figure 5.4: Total CPU time used by IBB-GateD and O-GateD versus the total number of nodes ($n$) in the simulated topology.

Having seen how much overhead IBB extension introduces to the SPT calculation, let us now turn our attention to how much effect IBB extension has on the total CPU time used by GateD. As

we mentioned earlier, we log the total CPU time used by GateD at the end of every sequence when GateD terminates. Since every sequence runs for a fixed amount of time irrespective of the GateD version and the simulated topology used, we believe that comparing the actual CPU time used by GateD provides a good basis for evaluating IBB extension's effect on the total CPU time. Figure 5.4 shows the mean value of 12 samples of total CPU time for both IBB-GateD and O-GateD as the value of $n$ increases. It is important to mention here that though the total run-time of each sequence is same, the CPU time used by GateD depends on a lot of external events, principal among them is the number of SPT computations which in turn depends on how the simulated LSAs are refreshed by pc1. Since this process is random, the total CPU time has more variation across samples than the mean SPT time. Therefore, Figure 5.4 also shows 95% confidence interval for each data point. As can be seen from the figure, for lower values of $n$, namely, 50 and 60, the total CPU time used by both versions of GateD is almost same. But as $n$ increases, the total CPU time used by IBB-GateD starts increasing more rapidly than that for O-GateD. The reason is that as $n$ increases, GateD spends more percentage of its CPU time in carrying out the SPT calculation. We have observed that when $n$ is equal to 50, GateD spends about 50% of its time in SPT calculation, whereas it spends close to 100% of its time in SPT calculation once $n$ reaches 80. This means that most of the overhead introduced by IBB extensions to GateD OSPF is to the SPT calculation.

## 6 Conclusion

In this paper, we have described how OSPF can be made IBB capable. The IBB capability helps avoid routing instabilities that arise when an OSPF process is brought down, provided the forwarding engine is still functioning. We make use of the Opaque LSA mechanism [5] for communication related to IBB between various routers. In particular, an IBB Cease LSA is used to inform others when the routing process in a router wants to go down temporarily, and to specify the time interval (IBB Timeout) during which the process is expected to be down. Other routers treat the originating router as capable of forwarding packets for that time period. If the routing process in the originating router returns to operation within the specified time period, the network behaves as if the process never went down with respect to packet forwarding. An ICB Open LSA is used for informing others when the originating routing process is operational again, which acts as a trigger to restore normal OSPF operation.

In any link-state routing protocol, it is imperative that all the routers calculate their routing tables from the same image of the link-state database. To ensure that no loops or black holes occur, a router must recalculate its routing table whenever a change in the topology occurs. With the IBB extension, the router that has been gracefully shut down cannot update its link-state database and routing table during the IBB timeout period. This can lead to routing loops and black holes. We presented a detailed analysis of the problem and developed a solution for avoiding such routing instabilities. The solution relies on two new types of LSAs — Path Info LSA and Avoid Router LSA — which enable the routers to determine loop-free paths during the IBB interval.

Although our solution in this paper is specific to OSPF, the fundamental problems addressed by the solution are common to all link-state protocols. Thus, our analysis and solution can be used as a basis for introducing IBB capability in other link-state protocols, such as IS-IS.

We described a prototype implementation of our solution using the GateD implementation of OSPF and used it to evaluate the overhead of introducing the IBB capability in OSPF. Our results clearly demonstrate that most of the overhead is incurred during an SPT calculation. The overhead associated with other IBB-related processing is negligible. Our experimental results also show that the IBB extension places negligible overhead when all the routers are fully operational. All these results demonstrate the viability of using IBB to avoid routing instability.

In the future we plan to investigate ways of lowering the overhead incurred in the SPT calculations. One possible approach is to use an incremental SPT algorithm [6, 11]. We also plan to investigate the situation where more than one router have been gracefully shut down at the same time.

## References

[1] North American Network Operators Group (NANOG), mailing list archives, http://www.nanog.org.

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

[3] Daniel P. Bovet and Marco Cesati. *Understanding the Linux Kernel.* O'Reilly & Associates, Inc., Sebastopol, California, January 2001.

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

[5] Rob Coltun. The OSPF Opaque LSA Option. RFC2370, July 1998.

[6] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Incremental Algorithms for Single–Source Shortest Path Trees. In *Proceedings of Foundations of Software Technology and Theoretical Computer Science*, pages 113–224, December 1994.

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

[8] John T. Moy. *OSPF : anatomy of an Internet routing protocol.* Addison–Wesley Publishing Company, Reading, Massachusetts, January 1998.

[9] John T. Moy. OSPF Version 2. RFC2328, April 1998.

[10] John T. Moy. *OSPF : Complete Implementation.* Addison–Wesley Publishing Company, September 2000.

[11] Paolo Narvaez, Kai-Yeung Siu, and Hong-Yi Tzeng. New Dynamic SPT Algorithm based on a Ball–and–string Model. In *Proceedings of the IEEE Infocom*, 1999.

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

[13] David Ward and John Scudder. BGP Notification Cease: I'll Be Back. Work in progress, draft–ward–bgp4–ibb-00.txt, June 1999.

[14] Alex Zinin, Abhay Roy, and Liem Nguyen. OSPF Out–of–band LSDB Resynchronization. Work in progress, draft–ietf–ospf–oob–resync–00.txt, November 2000.

[15] Alex Zinin, Abhay Roy, and Liem Nguyen. OSPF Restart Signaling. Work in progress, draft–ietf–ospf–restart–00.txt, November 2000.