

Avoiding Instability during Graceful Shutdown of Multiple OSPF Routers

Aman Shaikh, Rohit Dube, Anujan Varma

Abstract

Many recent router architectures decouple the routing engine from the forwarding engine, allowing packet forwarding to continue even when the routing process is not active. This opens up the possibility of using the forwarding capability of a router even when its routing process is brought down for software upgrade or maintenance, thus avoiding the route flaps that normally occur when the routing process goes down. Unfortunately, current routing protocols, such as BGP, OSPF and IS-IS do not support such operation. In an earlier paper [1], we described an enhancement to OSPF, called the *IBB (I'll Be Back) capability*, that enables a router to continue forwarding packets while its routing process is inactive.

When the OSPF process in an IBB-capable router is inactive, it cannot adapt its forwarding table to reflect changes in network topology. This can lead to routing loops and/or black holes. In this paper we focus on the loop problem, and provide a detailed analysis of how and when loops are formed and propose solutions to prevent them. We develop two necessary conditions for the formation of routing loops in the general case when multiple routers are inactive. These conditions can easily be checked by the neighbors of the inactive routers. Simulations on several network topologies showed that checking the two conditions together signaled a loop in most cases only when a loop actually existed.

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. The current practice for maintenance tasks such as software upgrades is to withdraw the router undergoing maintenance from forwarding service. To minimize the impact of the withdrawal, network operators perform upgrades at a time when traffic is low, and stagger upgrades of different routers. Nevertheless, upgrades cause at least some disruption in the forwarding service provided by the network. With society's increasing reliance on data networks, even minor disruptions in the forwarding service may be unacceptable in the future. Moreover, upgrades can also lead to network outages in extreme cases. There have been several cases of serious network outages in recent years, lasting for several days in some

cases. Two of these outages are known to have been triggered while performing software upgrades on the routers in the network [2]. Since routing software is frequently upgraded due to bug-fixes and enhancements, it is of great interest to providers and vendors to make these upgrades as non-intrusive and non-disruptive as possible.

In most of the current-generation core routers, the routing engine (software) is decoupled from the forwarding engine (hardware). Thus, one can potentially continue using these routers for forwarding packets even when its routing software is made temporarily inactive for maintenance. However, current routing protocols such as BGP [3], OSPF [4, 5] and IS-IS [6] do not allow an inactive router to be used for forwarding. The current definitions of these protocols assume that when the routing process of a router goes down, the router is incapable of forwarding packets, forcing other routers to avoid the inactive router for forwarding. To avoid this behavior, the protocol must refrain from re-routing around the router undergoing upgrade at least for a period of time. Several extensions have recently been proposed in the IETF to add such a capability to routing protocols [7, 8]. These proposals are known by various names such as *IBB (I'll Be Back)*, *graceful restart* and *hitless restart*.

In our previous work [1], we proposed an IBB proposal for the OSPF protocol. The aim of the IBB proposal is to enable a router undergoing software maintenance to forward packets for a certain period of time even if its OSPF process is down. Before going down, the OSPF process in the router informs other routers that it is temporarily being shut down, but its forwarding engine remains active. The process also specifies a time interval within which it expects to be back in operation. We called this interval the *IBB Timeout period* in our previous work [1]. Other routers can continue using the router for forwarding packets during the IBB Timeout period. Hence, if the routing process returns to operation within the IBB Timeout period, the scheme avoids the route flaps and instability normally associated with such a disruption. Other routers stop using the affected router for forwarding only if the routing process in the router remains inactive after the IBB Timeout period.

A key problem that an IBB proposal must solve is how to handle topological changes while a router's OSPF process is inactive. Since the process is inactive, it cannot update its forwarding table to reflect these changes. Under these circumstances, continued use of the router for forwarding can lead to problems such as loops or black holes. Recently, Moy has proposed an IBB-like proposal called hitless restart for OSPF [8]. This proposal takes a conservative approach for handling topological changes, by withdrawing the inactive router from forwarding

Aman Shaikh is with AT&T Labs (Research), Florham Park, NJ 07932. Anujan Varma is at the University of California, Santa Cruz, CA 95064. Rohit Dube is with UTStarcom Inc., Alameda, CA 94502.

service as soon as the first topological change is detected by its neighbors. Thus, in the case when the topology does not change within the upgrade interval, the hitless restart proposal avoids the withdrawal completely. However, in a large network running OSPF, changes in network topology occur often. Since the root cause of many changes is malfunctioning hardware or software [9], these changes can occur even when a router is inactive which limits the usefulness of the hitless restart proposal.

The IBB scheme is based on the observation that not every topological change can lead to forwarding loops or black holes even when one of the routers is unable to modify its forwarding table in response to the change. Therefore, it is possible to continue forwarding packets through a router whose routing process is inactive even after a topological change if we ensure that loops or black holes do not form. Thus, our IBB scheme avoids packet forwarding to a destination through such an inactive router only when using the router as the next hop to the specific destination leads to a routing loop or black hole. In this paper, we systematically analyze how loops and black holes are formed when a router does not respond to topological changes, and based on that propose algorithms to avoid them as a part of our IBB scheme. Our previous work [1] mainly focused on a single inactive router and derived a necessary and sufficient condition for avoiding loops when the routing process in one of the routers is made inactive. In this paper, we consider the general case of multiple inactive routers and develop conditions for loop avoidance. We only focus on loop avoidance in this paper because the black hole avoidance procedure developed in [1] applies to multiple inactive router case as well.

This paper is organized as follows: In Section 2, we provide a brief overview of OSPF. In Section 3, we present a summary of the procedure described in [1] for avoiding loops and black holes when a single router is inactive. Sections 4 through 6 describe loop avoidance algorithms to deal with the case of more than one inactive router. In particular, Section 4 describes loop detection conditions that can easily be checked by neighbors of the inactive routers when more than one router is inactive. Section 5 evaluates the effectiveness of these conditions using simulations. Section 6 presents the complete algorithm for loop avoidance in the presence of one or more inactive routers. Finally, in Section 7, we present our conclusions and provide directions for future work.

2 OSPF Overview

OSPF is a widely used intra-domain routing protocol [5]. It is a link-state routing protocol, meaning that each router within the AS discovers and builds an entire view of the network topology. This topology view is conceptually a directed graph. Each router represents a vertex in this topology graph, and each link between neighbor routers represents a unidirectional edge. Each link also has an associated weight that is administratively assigned in the configuration file of the router. Using the weighted topology graph, each router computes a shortest path tree with itself as the root, and applies the results to build its forwarding table. This assures that packets are forwarded along the shortest paths in terms of link weights to their destinations [5]. We will refer

to the computation of the shortest path tree as an *SPF (Shortest Path First) computation*, and the resultant tree as a *Shortest-Path Tree (SPT)*.

The OSPF topology can be divided into areas determining a two level hierarchy. Area 0, known as the *backbone area*, resides at the top level of the hierarchy and provides connectivity to the non-backbone areas (numbered 1, 2, ...). OSPF assigns each link to exactly one area. The routers that have links to multiple areas are called *border routers*. A router maintains the complete topology graph of each area in which it has links. The router does not get to know about the entire topology of remote areas, but knows about the total weight of the path from one or more border routers of its areas to each node in a remote area. OSPF allows routing information to be imported from other routing protocols like BGP. The router that imports routing information from other protocols into OSPF is called an *AS Border Router (ASBR)*.

OSPF performs SPF calculation in three stages. In the first stage, it calculates routes to intra-area destinations by computing SPTs for the topology graph of each area. This stage is termed as the *intra-area stage*. In the second stage, it calculates routes to each remote node by selecting an appropriate border router as an intermediate node based on the path weight information. This stage is termed as the *inter-area stage*. In the final stage, it calculates routes to each external node by selecting an appropriate ASBR as an intermediate node. This stage is termed as the *external stage*.

Every router running OSPF is responsible for describing its local connectivity in a *Link-State Advertisement (LSA)*. These LSAs are *flooded* reliably to other routers in the network, which allows them to build the view of the topology. The set of LSAs in a router’s memory is called a *link-state database* and conceptually forms the topology graph for the router.

3 Handling Topological Changes

When router R goes down¹, its forwarding table gets frozen. Thus, if a topological change occurs while R is inactive, it cannot update its forwarding table to respond to the change in the topology. This can lead to problems such as loops and black holes. Figure 1 illustrates how a topological change occurring while R is inactive can lead to a loop.

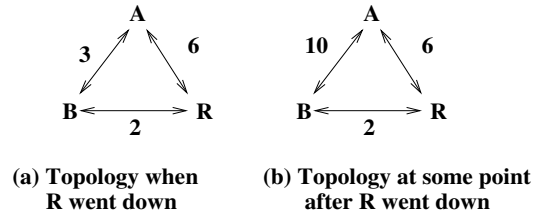


Figure 1: An example of a loop being formed while R is inactive.

In the figure, A , B and R are routers. As can be seen from the link weights in Figure 1(a), R ’s shortest path to A goes via

¹From now on, when we state that a router is going down or is inactive, we mean only its OSPF process is being shut down, unless specified otherwise.

B. Assume that *R* goes down at this point, thereby freezing its database image and forwarding table. After a while, the weight of the $A \leftrightarrow B$ link increases (Figure 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 forwarding table is frozen, however, a loop is formed if *B* tries to use *R* as the next hop for reaching *A*. To avoid the loop, when router *R* is inactive, other routers such as *B* cannot calculate their forwarding tables as they do normally; they need to keep in mind that *R*'s forwarding table is frozen, and make sure no loops or black holes are formed.

Before we describe algorithms for overcoming loops and black holes, let us define some notations. Let *S* be a source and *D* be a destination. For simplicity, we assume that *S* and *D* are routers. Assume that *S* sends a packet to *D*. We will denote the actual path traversed by a packet as $AP_{S \rightarrow D}$. Path $AP_{S \rightarrow D}$ has a *loop* if it has a cycle. Note that if path $AP_{S \rightarrow D}$ has a loop, it never terminates. On the other hand, if $AP_{S \rightarrow D}$ terminates at a node other than *D*, we say it has a *black hole*. Note that node *S* also has a path to *D* in its SPT². Packets may or may not actually travel along this hypothetical path, but the path can never have a loop or a black hole. We will denote this hypothetical path by $SP_{S \rightarrow D}$. Finally, we use $NH(R, D)$ to denote the next hop used by *R* for *D*. Table 1 summarizes the notations we defined here.

Notation	Definition
<i>S</i>	Source
<i>D</i>	Destination
<i>R</i> , <i>R_i</i>	Inactive routers
<i>R_I</i>	Set of inactive routers
$NH(R, D)$	Next hop used by router <i>R</i> for destination <i>D</i>
$AP_{S \rightarrow D}$	Actual path taken by a packet from <i>S</i> to <i>D</i>
$SP_{S \rightarrow D}$	Path in <i>S</i> 's SPT to <i>D</i>

Table 1: Notations used in the paper.

In our previous work [1], we introduced algorithms for overcoming loop and black hole problems. These algorithms consist of two steps: detection and prevention. This two step procedure is applied on a per destination (*D*) and per inactive router (*R*) basis. If a router such as *S* has to detect loops for every destination *D* on its own, it needs to know the next hop used by *R* for *D* when it went down [1]. When this condition is iterated over all possible *D*s, the implication is that *S* has to know *R*'s entire forwarding table at the time it went down. One way *S* can learn about *R*'s forwarding table is to construct *R*'s table based on its local image of the graph at the time it receives *R*'s message. This approach may not produce a correct version of *R*'s forwarding table at *S*. The other approach is to have *R* send its forwarding table to *S* before it goes down. This approach does not scale well. Instead of these two approaches, we propose that *R* send its forwarding table only to its neighbors. This approach

² *S* may have more than one shortest path to *D*. For ease of exposition, we will assume that only one such path exists.

is scalable and at the same time produces a correct version of *R*'s forwarding table. The implication of this approach is that neighbors of each inactive router are assigned the responsibility of detecting loop and/or black hole; neighbors perform detection for every destination independently.

If a neighbor identifies that using the inactive router for reaching *D* leads to a loop or a black hole, it sends out a message to other routers informing them of the problem. All the routers then prevent the loop and/or black hole by calculating a path to *D* such that the inactive router does not lie on it. The router achieves this by removing the inactive router and its incident edges from the graph, and by calculating the shortest path to *D* using the modified graph.

3.1 Single Inactive Router Case

In this section, we summarize the loop avoidance algorithm described in [1] for the single inactive router case. The details for black hole avoidance algorithm can be found in [1].

We denote the inactive router by *R*. Since *R* is the only inactive router, all the routers other than *R* are assumed to be using the same image of the topology graph for calculating their SPTs and forwarding tables. *R*'s SPT and forwarding table are based on the graph it had at the time it went down. Assume that *R* is using *Y* as the next hop for reaching *D*, i.e., $NH(R, D)$ is *Y*. Proposition 1 gives the necessary and sufficient condition for loop detection when a single router is inactive.

Proposition 1 Path $AP_{S \rightarrow D}$ has a loop iff *R* lies on $SP_{Y \rightarrow D}$.

The essence of Proposition 1 is that any loop for destination *D* is detected if *R* lies on *Y*'s path to *D* in the SPT of *Y*. Thus, if *R* sends its forwarding table to *Y* and its other neighbors as mentioned earlier, *Y* detects the loop with respect to *R* and *D* if the following condition based on Proposition 1 holds true.

$$\begin{aligned}
 srtr_cond(Y, R, D) = & \\
 & (Y \text{ is a neighbor of an inactive router } R), \\
 & (NH(R, D) = Y), \text{ and} \\
 & (R \text{ is part of } SP_{Y \rightarrow D}). \tag{1}
 \end{aligned}$$

Once *Y* detects a loop involving *R* and *D*, it recalculates its path to *D* such that *R* no longer remains on the path. Moreover, it also sends a message to other routers asking them not to use *R* for reaching *D*. This message is represented as an *avoid*(*R*, *D*, *L*) message [1] which means router *R* is to be avoided on the path to *D* because of a possible loop³. Other routers recalculate their paths to *D* upon receiving the avoid message.

Figure 2 summarizes the procedure each router has to follow after calculating its SPT in order to avoid loops when a single router is inactive. Step 1(a)i in the figure performs loop detection according to *srtr_cond*.

³ The first parameter in the *avoid* message is the router to be avoided, the second parameter is the destination, and the third parameter indicates the detection condition - 'L' here means a loop was detected with a single inactive router. We include the detection condition in the message since it affects the exact steps carried out by routers for avoidance as we will see later.

1. For each node D in the SPT, perform the following steps:
 - (a) For each of the minimum weight paths the router has to reach D , perform the following steps:
 - i. If the path goes through R , “cancel” this path if either of the following holds true.
 - (a) Some router (this includes the local router also) has sent an $avoid(R, D, L)$ message.
 - (b) One of the next hops used by R for reaching D is the local router.
 - If the path was “canceled” due to condition (b) above, originate an $avoid(R, D, L)$, provided no other router has done it so far.
 - (b) If all the paths to D in the router’s SPT were “canceled” above, mark D . Marking of D means that the router needs to find an alternate path to D that does not have the inactive routers on it (this is done in step 2 below).
2. If at least one such D was marked in the previous step, calculate a new SPT with R removed.
3. For each unmarked destination, determine the next hop(s) from the first SPT (the one with R in it). For each marked destination, determine the next hop(s) from the second SPT (the one without R).

Figure 2: Procedure for loop avoidance with a single inactive router R .

4 Loop Detection with Multiple Inactive Routers

In this section we deal with the case where more than one router, say n of them, are inactive at the same time. Let us denote these n routers by R_1, R_2, \dots, R_n . Furthermore, let us denote the set of these n inactive routers by R_I (i.e., $R_I = (R_1, R_2, \dots, R_n)$). For simplicity, we assume that all the routers (including the inactive ones) belong to a single area. If the routers belong to multiple areas, the condition $srtr_cond$ can be used to detect loops as long as only a single router is inactive within each area.

As in the single router case, we can make the next hop used by a router R_i to the destination D in charge of making sure that using R_i to reach D does not create a loop or a black hole. Note that this requires that adjacent routers are not brought down at the same time. We in fact make an assumption that this is true, i.e., adjacent routers are not brought down simultaneously. We believe that this is not an unreasonable assumption given that it is highly unlikely that operators would perform maintenance on two adjacent routers simultaneously. Unfortunately, with multiple inactive routers, $srtr_cond$ cannot detect all the loops as we show in Section 4.1. We provide two alternative loop detection conditions for multiple inactive routers in Sections 4.2 and 4.3. We should note here that the black hole detection conditions for the case of a single inactive router [1] apply without any modifications even when multiple routers are inactive.

4.1 Example Where Loop Detection Condition for Single Inactive Router Fails

Figure 3 gives an example where $srtr_cond$ fails to detect a loop with two inactive routers R_1 and R_2 .

Figure 3(a) shows the topology when routers R_1 and R_2 become inactive. It also shows the next hop used by various routers for D . Figure 3(b) shows the topology after two links go down while the routers are inactive. It also shows the next hop used by various routers under the changed circumstances. It can be seen that

$$AP_{Z_2 \rightarrow D} = (Z_2, R_2, Y_2, Z_1, R_1, Y_1, Z_2, \dots)$$

This is a loop. It can be verified that proposition 1 does not detect this condition with respect to Y_1, R_1 and D or Y_2, R_2

and D :

$$\begin{aligned} SP_{Y_1 \rightarrow D} &= (Y_1, Z_2, R_2, X_2, D) \\ SP_{Y_2 \rightarrow D} &= (Y_2, Z_1, R_1, X_1, D) \end{aligned}$$

Thus, it is clear that $srtr_cond$ is not sufficient for detecting every loop with multiple inactive routers. In the next two sections, we present two conditions that can detect every loop in the presence of multiple inactive routers.

We should also note here that $srtr_cond$ can lead to false detection of loops when multiple routers are inactive. Figure 4 gives an example of this. It can be seen from Figure 4(b) that

$$SP_{Y_1 \rightarrow D} = (Y_1, R_2, X_2, R_1, X_1, D)$$

This means that $srtr_cond$ holds true with respect to Y_1, R_1 and D . However, packets sent from R_1 reach D without looping.

$$AP_{R_1 \rightarrow D} = (R_1, Y_1, R_2, Y_2, X_1, D)$$

4.2 Condition 1 for Loop Detection

Consider router Z_1 in Figure 3(b). The path to D in its SPT is:

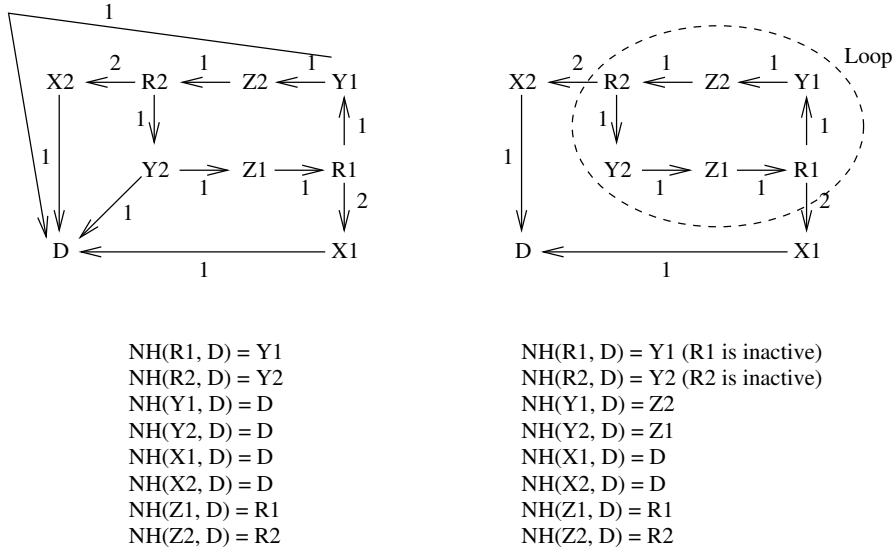
$$SP_{Z_1 \rightarrow D} = (Z_1, R_1, X_1, \dots),$$

where $X_1 \neq NH(R_1, D)$. That is, the node after R_1 in $SP_{Z_1 \rightarrow D}$ is not the next hop used by R_1 . This means that packets will not follow $SP_{Z_1 \rightarrow D}$ if Z_1 attempts to use it. In fact, packets can reach another inactive router R_2 , and return to Z_1 which is exactly what occurs in Figure 3(b).

It turns out that when a loop gets formed, a neighbor of at least one inactive router encounters a situation similar to that of Z_1 . Proposition 2 formalizes this.

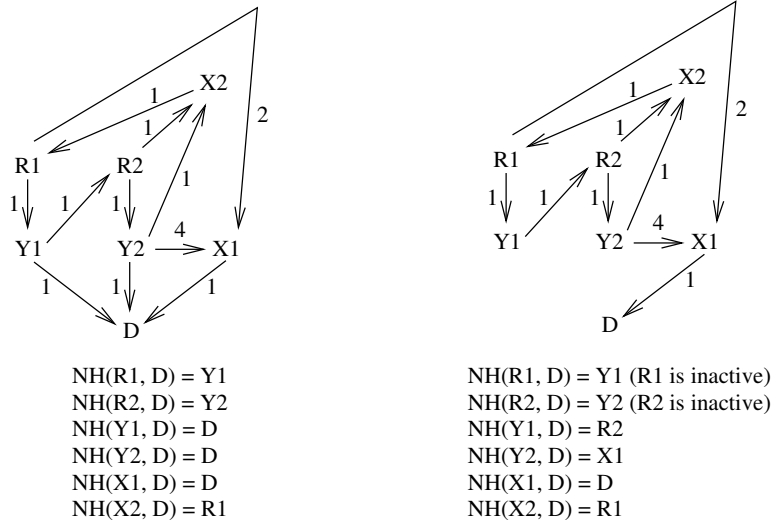
Proposition 2 *If path $AP_{S \rightarrow D}$ has a loop, then there is $R_i \in R_I$ such that the following holds true for at least one neighbor Z_i of R_i : The next node after R_i in $SP_{Z_i \rightarrow D} = X_i \neq NH(R_i, D)$.*

Proof: First consider the case where S is an active router. Let us assume that despite the loop on $AP_{S \rightarrow D}$, for all (R_i, Z_i) pairs (where R_i is an inactive router and Z_i is its neighbor), $NH(R_i, D) \in SP_{Z_i \rightarrow D}$. Since $AP_{S \rightarrow D}$ has a loop, at least one link in this path does not belong to $SP_{S \rightarrow D}$. Let $R_j \rightarrow Y_j$ be the first link, from router R_j to Y_j , that is not in $SP_{S \rightarrow D}$. Because the active routers compute their next hops based on the



(a) Topology when R1 and R2 become inactive (b) Topology changes while R1 and R2 are inactive

Figure 3: Example where *srtr_cond* fails to detect a loop with multiple inactive routers. R_1 and R_2 are inactive routers. (a) Topology when the routers become inactive. (b) Topology after the failure of links $Y_1 \rightarrow D$ and $Y_2 \rightarrow D$.



(a) Topology when R1 and R2 become inactive (b) Topology changes while R1 and R2 are inactive

Figure 4: Example where *srtr_cond* falsely detects a loop with multiple inactive routers. R_1 and R_2 are inactive routers. (a) Topology when the routers become inactive. (b) Topology after the failure of links $Y_1 \rightarrow D$ and $Y_2 \rightarrow D$.

shortest path, R_j cannot be an active router. Furthermore, since S is an active router, there must be a router Z_j preceding R_j on $AP_{S \rightarrow D}$. Since router R_j is inactive, router Z_j (which might be S) must be active since according to our assumption that two adjacent routers cannot be inactive simultaneously. Now consider $SP_{Z_j \rightarrow D}$. Since Z_j is active, $SP_{S \rightarrow D} = SP_{Z_j \rightarrow D}$. As a result, since by hypothesis $R_j \rightarrow Y_j$ does not belong to $SP_{S \rightarrow D}$, it does not belong to $SP_{Z_j \rightarrow D}$ either. In other words, the next node after R_j on $SP_{Z_j \rightarrow D}$ is not same as $NH(R_j, D)$ which is a contradiction.

Next consider the case where S is the inactive router. Let $Y =$

$NH(R_i, D)$. Since $AP_{S \rightarrow D}$ has a loop, so does $AP_{Y \rightarrow D}$. Note that Y must be an active router, and now the argument given above applies. ■

Proposition 2 leads to the following condition that a neighbor of an inactive router can use when multiple routers are inactive:

$$\begin{aligned}
 mrtrs_cond1(Z_i, R_i, D) = & \\
 & (Z_i \text{ is a neighbor of an inactive router } R_i), \text{ and} \\
 & (\text{the next node after } R_i \text{ in } SP_{Z_i \rightarrow D} \neq NH(R_i, D))
 \end{aligned} \tag{2}$$

Below we list a few salient points about Proposition 2 and $mrtrs_cond1$:

- First, Proposition 2 is only a necessary condition for the existence of a forwarding loop. In other words, loop-free operation is guaranteed if $mrtrs_cond1$ is false for all source-destination pairs. However, even if $mrtrs_cond1$ holds true for Z_i , R_i and D , it does not mean that packets sent from R_i to D will loop. Figure 5 shows an example where no loop is formed despite $mrtrs_cond1$ being true. Specifically $mrtrs_cond1$ holds true for (Z_1, R_1, D) and (Z_2, R_2, D) in Figure 5(b). However, packets destined to D do not loop when sent from any node.
- Even if $mrtrs_cond1$ does not hold with respect to a specific router R_i and D , packets sent from R_i to D can still go into a loop. However, in this case there is another inactive router R_j along $AP_{R_i \rightarrow D}$ for which the condition holds true.
- The condition $mrtrs_cond1$ is necessary for the existence of a loop in the single-router case as well, but is weaker than $srtr_cond$. For the single router case, $srtr_cond$ provides a stronger loop detection condition that is both necessary and sufficient.

Just as in the single-router case, the condition $mrtrs_cond1$ can be checked by every neighbor of an inactive router R_i if each of them receives a copy of R_i 's forwarding table. If the condition is found true, the neighbor can then send an $avoid(R_i, D, C1)$ message to other routers, instructing them not to use R_i to reach D because of $mrtrs_cond1$.

4.3 Condition 2 for Loop Detection

Consider router Y_1 in Figure 3(b). Y_1 is the next hop used by inactive router R_1 for D . Note that $SP_{Y_1 \rightarrow D}$ contains another inactive router R_2 in Figure 3(b). Thus, if Y_1 tries to use the path for forwarding traffic to D , packets may not follow $SP_{Y_1 \rightarrow D}$ exactly and may come back to Y_1 . This is exactly what happens in Figure 3(b).

It turns out that when a loop gets formed, neighbor of at least one inactive router encounters a situation similar to that of Y_1 . Proposition 3 formalizes this.

Proposition 3 *If path $AP_{S \rightarrow D}$ has a loop, then there is $R_i \in R_I$ such that $NH(R_i, D) = Y_i$ and $SP_{Y_i \rightarrow D}$ contains a router from R_I .*

Proof: First, if $AP_{S \rightarrow D}$ has a loop, then at least one inactive router must lie on this path. Let R_i be the first inactive router in this path. We can show that the the forwarding path from R_i to D passes through an inactive router (which may be the same as R_i or different). The proof is by contradiction.

Let $NH(R_i, D) = Y_i$ and assume, that the path $SP_{Y_i \rightarrow D}$ does not pass through any inactive routers. Consider a packet transmitted from S to D . Because there are no inactive routers on the path from S to R_i , the path $P_{S \rightarrow R_i}$ is loop-free. R_i then forwards the packet to Y_i . Because there are no inactive routers on the path of $NH(R_i, D)$ to D , the packet will be forwarded along this path and will reach D without looping. This is a contradiction to the hypothesis that $P_{S \rightarrow D}$ has a loop. ■

Proposition 3 leads to the following condition that the neigh-

bors of an inactive router can use to detect loops.

$$\begin{aligned}
 mrtrs_cond2(Y_i, R_i, D) = & \\
 & (Y_i \text{ is a neighbor of an inactive router } R_i), \\
 & (NH(R_i, D) = Y_i), \text{ and} \\
 & (SP_{Y_i \rightarrow D} \text{ contains an inactive router}). \quad (3)
 \end{aligned}$$

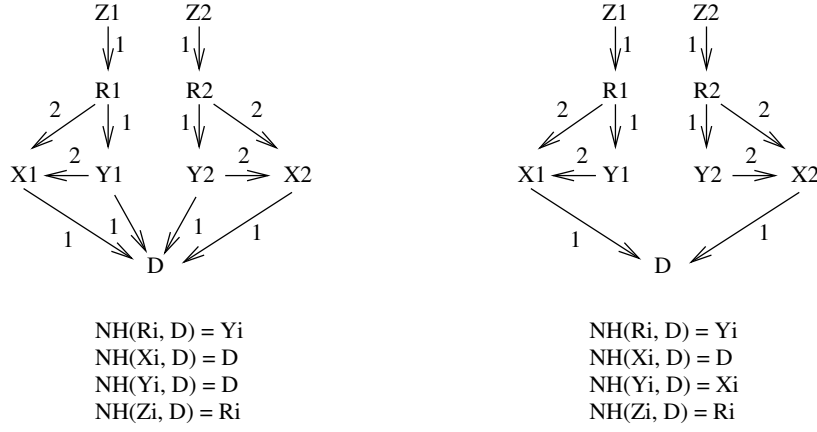
Below we list a few salient points about proposition 3 and $mrtrs_cond2$:

- Like Proposition 2, Proposition 3 is only a necessary condition for the existence of a forwarding loop. Figure 6 shows an example where no loop is formed despite $mrtrs_cond2$ holding true. In Figure 6(b), $mrtrs_cond2$ holds true for (Y_1, R_1, D) . However, packets destined to D do not loop when sent from any node in the network.
- It can be easily proved that if packets sent from R_i to D loop, $mrtrs_cond2$ holds true for R_i itself.
- Condition $mrtrs_cond2$ is a generalization of $srtr_cond$. The former becomes identical to the latter for the special case when the second inactive router is the same as the first inactive router.
- Condition $mrtrs_cond2$ can be used for loop detection even when a single router is inactive. In fact, $srtr_cond$ is a special case of $mrtrs_cond2$ since with a single inactive router, $mrtrs_cond2$ holds true for $NH(R, D)$. However, $srtr_cond$ provides a more precise loop detection condition. Furthermore, a disadvantage of $mrtrs_cond2$ is that a mere addition of a router in the set R_I can invalidate the use of some inactive routers for reaching D . This is because if next hop Y_i used by an already inactive router R_i has the new inactive router on its path to D , Y_i declares a loop.

The condition $mrtrs_cond2$ can be checked by every neighbor of an inactive router R_i , in the same way as $mrtrs_cond1$. If the condition is found true, the neighbor can then generate an $avoid(R_i, D, C2)$ message to other routers, instructing them not to use R_i to reach D because of $mrtrs_cond2$.

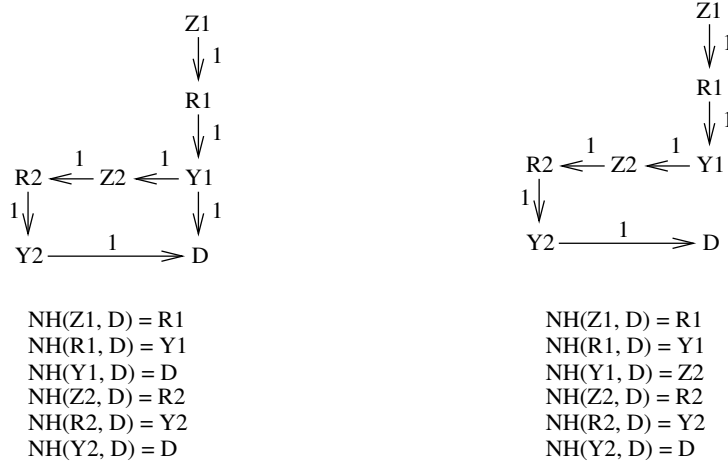
5 Performance Evaluation of Loop Detection Conditions 1 and 2

In this section we evaluate the effectiveness of loop detection conditions in the case of multiple inactive routers using realistic topology graphs. Given realistic topology graphs, the aim is to determine how often $srtr_cond$ generates false positives and false negatives, and how often $mrtrs_cond1$ and $mrtrs_cond2$ generate false positives. A false positive is a case when the condition indicates the possibility of a routing loop, but no such loop exists. A false negative refers to the case where the condition ascertains loop-free operation, while a loop actually exists. False negatives can occur only when the single-router condition $srtr_cond$ is used even though multiple inactive routers are present. We use two sources for realistic topology graphs: weighted topologies generated by the Rocketfuel project [10, 11], and areas of a tier 1 ISP's OSPF domain. First we describe the methodology, and then present the results.



(a) Topology when R1 and R2 became inactive (b) Topology changes while R1 and R2 are inactive

Figure 5: Example showing that converse of proposition 2 is not true. R_1 and R_2 are inactive routers. (a) Topology when the routers become inactive. (b) Topology after the failure of the link $Y_1 \rightarrow D$.



(a) Topology when R1 and R2 became inactive (b) Topology changes while R1 and R2 are inactive

Figure 6: Example showing that converse of proposition 3 is not true. R_1 and R_2 are inactive routers. (a) Topology when the routers become inactive. (b) Topology changes while the routers are inactive.

5.1 Methodology

For each topology graph, we select two routers at a time and bring them down. Subsequently, we introduce a topology change by bringing down one or two links. We then observe how this change affects various destinations of the graph in terms of loop formation and detection. Figure 7 describes the main steps of the methodology used.

As the figure shows, R_1 and R_2 are chosen such that they are not neighbors of each other. We assume these two routers are brought down for maintenance. For a given (R_1, R_2) pair, the method considers each node of the graph as a destination D . Step 1a of *main* calculates the SPT rooted at D . Step 1c of *main* determines the set of links from R_1 and R_2 to D in SPT. These sets are denoted by L_1 and L_2 , respectively. Step 1d selects one link from L_1 and, if necessary, another link from

L_2 and brings them down⁴. Bringing down these links forces a change in the paths from each R_i to D . Links not belonging to L_1 and L_2 are not considered because bringing them down does not affect the path from R_i to D . Moreover, choosing one link from each R_i 's path to D increases the likelihood of a loop formation. Step 2 of *bring_down_links* calculates an SPT rooted at D without l_1 and l_2 ; the incremental SPF algorithm proposed by Ramalingam and Reps [12] is used for calculating the SPT. Step 3 of *bring_down_links* applies loop detection conditions to see if a potential loop exists for packets destined to D if R_i is used on the path. Finally, step 4 of *bring_down_links* determines whether a packet from R_1 and R_2 to D actually loop or not. Since R_1 and R_2 are inactive, the procedure determines their

⁴We also conducted experiments involving single link failures. However, very few single link failures — 0.1% or less — led to loop formation (and false loop detection). Hence we omit the results for these cases, and instead focus on the more interesting case of two-link failures.

- **Procedure *main*()**
 1. For each pair (R_1, R_2) and for each node D such that $R_1 \neq R_2 \neq D$, and R_1 and R_2 are not neighbors
 - (a) Calculate SPT rooted at D .
 - (b) Bring down R_1 and R_2 .
 - (c) Identify two sets of links: $L_1 = \{\text{Set of links from } R_1 \text{ to } D\}$, and $L_2 = \{\text{Set of links from } R_2 \text{ to } D\}$.
 - (d) For each link l_1 in set L_1
 - i. If l_1 belongs to L_2 , call *bring_down_links*(l_1, l_1)
 - ii. Else for each link l_2 in set L_2 call *bring_down_links*(l_1, l_2)
- **Procedure *bring_down_links*(Link l_1 , Link l_2)**
 1. Bring down l_1 and l_2 .
 2. Calculate a new SPT rooted at D .
 3. Determine whether *srtr_cond* (Section 3.1), *mrtrs_cond1* (Section 4.2) and *mrtrs_cond2* (Section 4.3) hold true for R_1 and R_2 , their neighbors and D .
 4. Determine whether packet sent from R_1 and R_2 to D actually loop or not.

Figure 7: Methodology used for evaluating the loop detection conditions when two routers are inactive.

next hops for D from the first SPT calculated in step 1a of *main*; for the remaining routers the next hops are determined from the second SPT.

As mentioned earlier, we make use of the topology graphs generated by the Rocketfuel project [10], and areas of a tier 1 ISP. Specifically, we consider five out of six AS topology graphs for which Rocketfuel inferred link weights [10]. Moreover, we consider areas 0, 1, 2, 3 and 4 of the ISP. Each Rocketfuel AS topology is considered a single OSPF area. Thus, these five AS topologies and five areas of the ISP provide us with ten topology graphs over which we applied the method of Figure 7. Table 2 provides information about these topology graphs. All the information given in the table pertains to the methodology described in Figure 7. For example, the second, third and fifth columns of the table represent the number of (R_1, R_2) pairs, D s and (R_1, R_2, D) triplets, respectively, that were considered in *main* for the associated topology graph. The sixth column represents the total number of samples produced. The total number of samples is equal to twice the number of times procedure *bring_down_links* is called from *main* since each call to the procedure produces two samples: one for R_1 and the other for R_2 . The last column of the table represents the number of samples for which the topology change results in a loop from R_i to D ; the column also provides the percentage with respect to the total number of samples.

5.2 Results

We consider four metrics to gauge the performance of the loop detection conditions. Before we describe the four metrics, let us define some variables. Let us denote the number of cases where loop exists for a given R_i and D by $loops(R_i, D)$. Furthermore, we denote the number of cases where *srtr_cond* holds true with respect to R_i and D as $scond(R_i, D)$. We define $cond1(R_i, D)$ and $cond2(R_i, D)$ to be the number of cases where *mrtrs_cond1* and *mrtrs_cond2*, respectively, hold true. Finally, $cond1\&2(R_i, D)$ denotes the number of cases where both *mrtrs_cond1* and *mrtrs_cond2* hold true with respect to R_i and D . Having defined these variables, let us first define the four metrics:

1. *srtr_fneg*: Fraction of cases where a loop exists but the loop detection condition for a single inactive router fails to detect the loop. Formally,

$$srtr_fneg = 1 - \frac{\sum_{(R_1, R_2, D)} (scond(R_1, D) + scond(R_2, D))}{\sum_{(R_1, R_2, D)} (loops(R_1, D) + loops(R_2, D))}$$

2. *cond1_fpos*: Fraction of cases where loop detection condition 1 for multiple inactive router case detects a loop, but no loop is actually formed. Formally,

$$cond1_fpos = 1 - \frac{\sum_{(R_1, R_2, D)} (loops(R_1, D) + loops(R_2, D))}{\sum_{(R_1, R_2, D)} (cond1(R_1, D) + cond1(R_2, D))}$$

3. *cond2_fpos*: Fraction of cases where loop detection condition 2 for multiple inactive router case detects a loop, but no loop is actually formed. Formally,

$$cond2_fpos = 1 - \frac{\sum_{(R_1, R_2, D)} (loops(R_1, D) + loops(R_2, D))}{\sum_{(R_1, R_2, D)} (cond2(R_1, D) + cond2(R_2, D))}$$

4. *cond1&2_fpos*: Fraction of cases where loop detection condition 1 and 2 both detect a loop, but no loop is actually formed. Formally,

$$cond1\&2_fpos = 1 - \frac{\sum_{(R_1, R_2, D)} (loops(R_1, D) + loops(R_2, D))}{\sum_{(R_1, R_2, D)} (cond1\&2(R_1, D) + cond1\&2(R_2, D))}$$

We also considered false positive cases for *srtr_cond*. We observed only one instance of such a case, which was in Area 0 of the ISP. Therefore, we do not consider false positives for *srtr_cond* further in this section. Tables 3–6 present these four metrics for various topologies.

Here are the salient points of the results:

- Condition *srtr_cond* misses about 5–10% of loops for Rocketfuel topologies. On the other hand, the false negative rate for ISP areas exhibit a lot of variation.

Topology graph	Number of (R_1, R_2) pairs	Number of D s (= Number of vertices)	Number of edges	Number of (R_1, R_2, D) triplets	Total number of samples	Number of samples with a loop (% of total samples)
Rocketfuel 1221	5,625	108	306	596,250	6,391,938	118,268 (1.85%)
Rocketfuel 1755	3,580	87	322	304,300	4,394,948	359,209 (8.17%)
Rocketfuel 3257	12,552	161	656	1,995,768	41,299,386	1,292,845 (3.13%)
Rocketfuel 3967	2,934	79	294	225,918	3,729,308	294,372 (7.89%)
Rocketfuel 6461	9,496	141	748	1,319,944	17,505,258	234,202 (1.34%)
ISP area 0	$\approx 7,000$	≈ 100	≈ 700	961,000	18,711,286	657,681 (3.51%)
ISP area 1	$\approx 2,000$	≈ 50	≈ 320	162,978	407,058	4,727 (1.16%)
ISP area 2	$\approx 1,000$	≈ 50	≈ 200	73,458	21,560	432 (2.00%)
ISP area 3	$\approx 2,000$	≈ 50	≈ 300	149,410	43,002	742 (1.73%)
ISP area 4	≈ 1000	≈ 50	≈ 200	39,689	8,606	142 (1.65%)

Table 2: Information about topology graphs. We provide approximate values for some parameters of ISP areas due to proprietary nature of the data.

Topology graph	Number of loops detected	Number of false negatives	<i>srr_fneg</i>
Rocketfuel 1221	103,957	14,311	0.1377
Rocketfuel 1755	339,406	19,803	0.0583
Rocketfuel 3257	1,255,156	37,689	0.0300
Rocketfuel 3967	284,299	10,073	0.0354
Rocketfuel 6461	223,887	10,315	0.0461
ISP area 0	640,586	17,096	0.0267
ISP area 1	3,187	1,540	0.4832
ISP area 2	216	216	1.0000
ISP area 3	395	347	0.8785
ISP area 4	71	71	1.0000

Table 3: Results for *srr_cond*.

Topology graph	Number of loops detected	Number of false positives	<i>cond2_fpos</i>
Rocketfuel 1221	164,966	46,698	0.2831
Rocketfuel 1755	410,514	51,305	0.1250
Rocketfuel 3257	1,580,130	287,285	0.1818
Rocketfuel 3967	349,950	55,578	0.1588
Rocketfuel 6461	334,745	100,543	0.3004
ISP area 0	831,535	173,854	0.2091
ISP area 1	6461	1974	0.3055
ISP area 2	432	0	0.0000
ISP area 3	744	2	0.0027
ISP area 4	142	0	0.0000

Table 5: Results for *mrtrs_cond2*.

Topology graph	Number of loops detected	Number of false positives	<i>cond1_fpos</i>
Rocketfuel 1221	1,047,633	940,532	0.8978
Rocketfuel 1755	1,447,496	1,101,607	0.7610
Rocketfuel 3257	9,034,742	7,771,005	0.8601
Rocketfuel 3967	1,411,714	1,122,429	0.7951
Rocketfuel 6461	5,700,073	5,472,299	0.9600
ISP area 0	5,654,932	5,006,611	0.8854
ISP area 1	119,554	115,172	0.9633
ISP area 2	10,485	10,068	0.9602
ISP area 3	22,564	21,857	0.9687
ISP area 4	3,672	3,546	0.9657

Table 4: Results for *mrtrs_cond1*.

Topology graph	Number of loops detected	Number of false positives	<i>cond1&2_fpos</i>
Rocketfuel 1221	112,984	5,883	0.0521
Rocketfuel 1755	361,363	15,474	0.0428
Rocketfuel 3257	1,309,579	45,842	0.0350
Rocketfuel 3967	313,854	24,569	0.0783
Rocketfuel 6461	253,974	26,200	0.10316
ISP area 0	700,125	51,804	0.0740
ISP area 1	4,742	360	0.0759
ISP area 2	417	0	0.0000
ISP area 3	707	0	0.0000
ISP area 4	126	0	0.0000

Table 6: Results for *mrtrs_cond1* and *mrtrs_cond2*.

- Most of the loops detected by *mrtrs_cond1* are false positives. Compared to that *mrtrs_cond2* performs much better. The reason is that *mrtrs_cond1* holds true when next hop used by R_i for D changes which occurs quite often when a link is removed from R_i 's path to D . Unfortunately, very few cases actually result in a loop. On the other hand, *mrtrs_cond2* requires another inactive router to appear on

the path from R_i to D . The heights of the trees for these topologies are low which makes it highly unlikely for another inactive router to move onto R_i 's path to D when a link is removed.

- Using *mrtrs_cond1* and *mrtrs_cond2* together decreases the number of false positives significantly (to approximately 10 percent or lower). However, it does not eliminate false

positives altogether.

- Areas 2, 3 and 4 of the ISP have very similar graphs which is reflected in almost identical results obtained for them. Area 1 is slightly different from these areas, and Area 0 has a very different topology compared to the remaining areas of the ISP.

6 Loop Avoidance Procedure with Multiple Inactive Routers

When multiple routers are inactive, we use *mrtrs_cond1* in conjunction with *mrtrs_cond2* to detect loops as this results in the least number of false positives. It is possible that, for a given (R_i, D) pair, the router that detects *mrtrs_cond1* to be true is different from the router that detects *mrtrs_cond2* to be true. A router detecting the condition *mrtrs_cond1* sends out an *avoid*($R_i, D, C1$) message to other routers. Similarly, a router detecting *mrtrs_cond2* sends out an *avoid*($R_i, D, C2$) message. Other routers remove R_i from the SPT for destination D only if they have received both the messages *avoid*($R_i, D, C1$) and *avoid*($R_i, D, C2$).

The prevention step is similar to that used for the single inactive router case. Routers calculate a new path to D by removing R_i from the path if it is part of the path. With n inactive routers, it may be necessary to remove j of these routers ($1 \leq j \leq n$) in the SPT for reaching a given destination D . A new path to D can then be computed without the j forbidden R_i s on it. Since the set of forbidden routers can be different for different routers, a router might have to perform $O(|N|)$ SPF calculations, the overhead of which may not be acceptable in some circumstances. Fortunately, we can reduce the number of SPF calculations to two with the following scheme.

When a router has to avoid one or more of the inactive routers on its path to D , the router attempts to calculate a path to D that does not have any of the n inactive 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 inactive routers on it. The advantage of this method is that the router can use the same SPT (that is, the one without any of the n inactive routers in it) for finding paths to all the destinations for which it needs to find a path without one or more R_i s on it. In other words, the router first calculates its SPT with all the inactive routers in it. Subsequently, for each destination D , the router marks the destination if the path to the destination has at least one forbidden router on it. For all the unmarked D s, the router selects next hops from the original SPT. For all the marked D s, the router computes a second SPT after removing all the inactive routers, and selects next hops from this second SPT. Figure 8 summarizes this procedure.

One problem with this approach is the possibility of partitioning the network when n inactive routers are removed from the graph. We believe that this is highly unlikely since the scheme is primarily applicable to the case when routers are taken out of service for maintenance/upgrade within a single AS (Autonomous System), and network administrators very rarely take down enough routers at once to partition the network. Another problem with the simplified approach is that some routers may

use the first SPT (one with all the inactive routers in it) for a destination D , while other routers may use the second SPT (that does not include any of the inactive routers). This is because some of the active routers may not have any forbidden routers on their paths to D , while others may have at least one forbidden router on their paths. This raises the question whether the disparity in how routers select next hops for D can lead to loops or black holes. Proposition 4 proves that this disparity does not lead to a loop.

Proposition 4 *Let S denote an arbitrary router in the topology graph and D an arbitrary destination. Suppose the set of forbidden routers for D is selected according to Proposition 2. S determines its $SP_{S \rightarrow D}$ from first SPT if path to D does not have any forbidden routers on it, otherwise from the second SPT. Under this circumstances, $AP_{S \rightarrow D}$ does not have a loop.*

Proof: We assume that none of the inactive routers in R_I are pair-wise neighbors. In other words, for every destination D and R_i , $NH(R_i, D)$ does not belong to R_I . Let R_D be the set of forbidden routers for D . Obviously, $R_D \subseteq R_I$. We assume that R_D is determined according to Proposition 2. After S computes its first SPT, it may have one or more forbidden routers on its path to D . Let R_{SD} denote the set of forbidden routers S has on its path to D . Naturally, $R_{SD} \subset R_D$.

If R_{SD} is empty, S does not have any forbidden routers on its path to D . In other words, S picks $SP_{S \rightarrow D}$ from its first SPT. Let $SP_{S \rightarrow D} = (S, \dots, Z_{i1}, R_{i1}, X_{i1}, \dots, Z_{ik}, R_{ik}, X_{ik}, \dots, D)$. Since R_{SD} is empty, Proposition 2 does not hold true for any R_{ij} 's on $SP_{S \rightarrow D}$. In other words, $SP_{S \rightarrow D} = (S, \dots, Z_{i1}, R_{i1}, NH(R_{i1}, D), \dots, Z_{ik}, R_{ik}, NH(R_{ik}, D), \dots, D)$. Therefore, a packet from S to D actually follows $SP_{S \rightarrow D}$, i.e., $AP_{S \rightarrow D} = SP_{S \rightarrow D}$. Hence, $AP_{S \rightarrow D}$ cannot have a loop.

If R_{SD} is non-empty, S does have some forbidden routers on its path to D in the first SPT. Therefore, S picks $SP_{S \rightarrow D}$ from the second SPT and hence does not have any routers from R_I . If every router on the path $SP_{S \rightarrow D}$ also uses its second SPT to select a path to D , then $AP_{S \rightarrow D}$ cannot have a loop. Otherwise, $AP_{S \rightarrow D}$ will be same as $SP_{S \rightarrow D}$ up to a router that has used its first SPT to pick its path for D . Let us call this router X . Thus, $AP_{S \rightarrow D}$ is identical to $SP_{S \rightarrow D}$ until a packet reaches X . From X onwards, $AP_{S \rightarrow D}$ may diverge from $SP_{S \rightarrow D}$. In fact, $AP_{S \rightarrow D} = (S, \dots, AP_{X \rightarrow D})$. Consider the situation from X 's perspective. Since X is using its first SPT to select $SP_{X \rightarrow D}$, X does not have any forbidden routers on its path to D , i.e., R_{XD} is empty. And we have already proved above that $AP_{X \rightarrow D}$ cannot have a loop. This combined with the fact that $AP_{S \rightarrow D}$ is equal to $SP_{S \rightarrow D}$ up to router X proves that $AP_{S \rightarrow D}$ cannot have a loop. ■

7 Conclusions

In this paper, we evaluated an IBB extension to OSPF, focusing on the general case when the OSPF process in one or more routers is inactive, while their forwarding capability is still enabled. The IBB capability helps avoid routing instabilities that arise when the OSPF process is brought down in a router for maintenance tasks such as hardware or software upgrade, con-

1. For each node D in the SPT, perform the following steps:
 - (a) For each of the minimum weight paths the router has to reach D , perform the following steps:
 - i. If the path does not go through any of the R_i s, skip the next step (ii).
 - ii. The path goes through some R_i s. For each such R_i , do the following:
 - If $srtr_cond(S, R_i, D)$ holds true, send an $avoid(R_i, D, L)$ message to other routers.
 - If $mrtrs_cond1(S, R_i, D)$ holds true, send an $avoid(R_i, D, C1)$ message to other routers.
 - If $mrtrs_cond2(S, R_i, D)$ holds true, send an $avoid(R_i, D, C2)$ message to other routers.
 - “Cancel” this path if either of the following holds true.
 - If number of inactive routers is 1 and some router (this includes the local router also) has sent an $avoid(R_i, D, L)$ message.
 - If number of inactive routers is more than 1, and some router(s) (including the local router) have sent $avoid(R_i, D, C1)$ and $avoid(R_i, D, C2)$ message.
 - (b) If all the paths to D in the router’s SPT were “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 inactive routers on it (this is done in step 2 below).
2. If at least one such D was marked in the previous step, calculate a new SPT with all the R_i s ($1 \leq i \leq n$) removed.
3. For each unmarked destination, determine the next hop(s) from the first SPT (the one with all R_i s in it). For each marked destination, determine the next hop(s) from the second SPT (the one with all R_i s removed).

Figure 8: Procedure for loop avoidance. We assume that S denotes the router performing the procedure, and R_1, R_2, \dots, R_n denote the inactive routers.

figuration update, change of router ID, etc., provided the forwarding engine is still functioning. When a router is shut down in IBB mode, it informs other routers that its routing process is going to be inactive for a while and specifies a time interval (IBB Timeout period) within which it expects to be back. Other routers treat the originating router as capable of forwarding packets during that time period. If the routing process on 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.

Upon a topological change, the inactive router cannot update its link-state database and forwarding table. There are three alternatives to deal with the resulting consistency problem: do nothing, stop using the inactive router for forwarding, or keep using the inactive router but make sure no loops or black holes are formed. We have chosen the third approach, where we attempt to use the forwarding capability of the inactive router as much as possible, except in cases where it can lead to routing loops or black holes. Based on a systematic analysis of how loops and black holes are formed, we developed procedures for avoiding them. The key idea behind these procedures is to make the neighbors of the inactive router in charge of detecting loops and black holes. The neighbors, in turn, ask other routers not to use the inactive router for such destinations.

For the general case of more than one router in the inactive state, loop detection is more difficult than the single-router case. We formulated two conditions, each of which can be checked by the neighbors of an inactive router, that are necessary for the existence of a forwarding loop when multiple routers are inactive. Our simulations with real network topologies show that checking the conditions together eliminates most of the cases where a loop is signaled by the algorithm when none exists. Although we have developed the procedures for OSPF, we believe that they are general and can be applied to other link-state protocols, such as IS-IS.

Acknowledgment

We would like to thank Kenneth Calvert and the anonymous reviewers for their valuable comments and feedback. We would also like to thank Jennifer Rexford and Albert Greenberg for several helpful discussions, and feedback on the earlier draft of the paper. Finally, we would like to thank Ratul Mahajan for Rocketfuel OSPF topologies, Joel Gottlieb for tier-1 ISP topologies, and Luciana Buriol for her help with incremental SPF algorithms and their implementation.

References

- [1] Aman Shaikh, Rohit Dube, and Anujan Varma, “Avoiding Instability during Graceful Shutdown of OSPF,” in *Proc. IEEE INFOCOM*, 2002, vol. 2, pp. 883–892.
- [2] “North American Network Operators Group (NANOG), mailing list archives, <http://www.nanog.org>,”.
- [3] Y. Rekhter, T. Li, and S. Hares, “A Border Gateway Protocol 4 (BGP-4),” RFC4271, January 2006.
- [4] John T. Moy, “OSPF Version 2,” RFC2328, April 1998.
- [5] John T. Moy, *OSPF : Anatomy of an Internet Routing Protocol*, Addison-Wesley, January 1998.
- [6] R. Callon, “Use of OSI IS-IS for Routing in TCP/IP and Dual Environments,” RFC1195, December 1990.
- [7] Srihari Sangli, Yakov Rekhter, Rex Fernando, John Scudder, and Enke Chen, “Graceful Restart Mechanism for BGP,” Work in progress, Intenet Draft, July 2003.
- [8] John T. Moy, Padma Pillay-Esnault, and Acee Lindem, “Graceful OSPF Restart,” RFC3623, November 2003.
- [9] Aman Shaikh, Chris Isett, Albert Greenberg, Matthew Roughan, and Joel Gottlieb, “A Case Study of OSPF Behavior in a Large Enterprise Network,” in *Proc. ACM SIGCOMM Internet Measurement Workshop (IMW)*, 2002, pp. 217–230.
- [10] Ratul Mahajan, Neil Spring, David Wetherall, and Tom Anderson, “Inferring Link Weights using End-to-End

Measurements,” in *Proc. ACM SIGCOMM Internet Measurement Workshop (IMW)*, 2002, pp. 231–236.

- [11] “Rocketfuel: An ISP Topology Mapping Engine,” <http://www.cs.washington.edu/research/networking/rocketfuel>.
- [12] G. Ramalingam and T. Reps, “An Incremental Algorithm for a Generalization of the Shortest-path Problem,” *Journal of Algorithms*, vol. 21, no. 2, pp. 267–305, 1996.

Aman Shaikh (M’03) is a Technical Specialist at AT&T Labs (Research). He obtained his Ph.D. and M.S. in Computer Engineering from the University of California, Santa Cruz in 2003 and 2000 respectively. He also holds a B.E. (HONS) in Computer Science and an M.Sc. (HONS) in Mathematics from the Birla Institute of Technology and Science, Pilani, India. His current research interests include IP routing, and network management and operations. He has published several research and technical papers in these areas.

Rohit Dube is a veteran of the Data Networking Industry. Most recently he was at UTStarcom as an Internal Engineering Consultant and Product Manager. Rohit came to UTStarcom via the acquisition of Xebec Communications where he was a Co-founder and the Director of Software Engineering. In previous lives, Rohit was a Member of Technical Staff and a Technical Lead at Bell Labs (Lucent Technologies) and a Founding Engineer at Torrent Networking Technologies (acquired by Ericsson). Rohit holds an M.S. in Computer Science from the University of Maryland at College Park and a B.Tech in Computer Science and Engineering from the Indian Institute of Technology at Bombay. He has published several research and technical papers in the fields of Data Networking and Computer Communications and co-chairs the Internet Engineering Task Force OSPF Working Group.

Anujan Varma (M’86) is a Professor of Computer Engineering at University of California, Santa Cruz. He received his Ph.D. in computer engineering from University of Southern California in 1986, and was with the IBM Thomas J. Watson Research Center at Yorktown Heights, New York, until 1991. His current research interests are in high-speed switching and routing hardware, traffic management, congestion control and scheduling, and transport of video and audio over packet networks. He has published more than 100 papers in refereed journals and conference proceedings. Dr. Varma has received several awards including the National Science Foundation Young Investigator Award, IEEE Darlington Award, and a teaching innovation award from the University of California. He has consulted extensively for the networking and semiconductor industries.