Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computer Networks 46 (2004) 273–293

COMPUTER NETWORKS

www.elsevier.com/locate/comnet

OSPF-based hybrid approach for scalable dissemination of QoS parameters ☆

Turgay Korkmaz [a,*], Marwan Krunz [b], Jyothi Guntaka [a]

[a] *Department of Computer Science, The University of Texas at San Antonio, San Antonio, TX 78249, USA*
[b] *Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ 85721, USA*

Received 23 June 2003; received in revised form 29 February 2004; accepted 3 March 2004
Available online 26 May 2004

Responsible Editor: S. Fdida

Abstract

Current link-state routing protocols (e.g., OSPF) use flooding to disseminate link-state information throughout the network. Despite its simplicity and reliability, flooding incurs unnecessary communication and processing overheads in control plane since nodes may receive multiple copies of the same advertisement. These overheads become significant in protocols that support quality-of-service (QoS) routing, where links are associated with dynamic metrics (e.g., available bandwidth) that need to be advertised frequently. The overheads can be significantly reduced using tree-based broadcasting approaches. Although a number of such approaches have been proposed in the literature, they have not been used in real networks because of their complexity and/or unreliability. In this paper, we propose a hybrid link-state dissemination approach that combines the best features of flooding and tree-based broadcasting. Our approach is particularly suited for the dissemination of "dynamic" link metrics (e.g., available bandwidth), which are often used in QoS-based path selection and traffic engineering. In our approach, topological changes and *first-time* LSAs (link-state advertisements) are flooded, whereas *refresh* LSAs (the ones that provide updated information on the *dynamic* metrics) are sent using tree-based broadcasting. The broadcast trees in our approach are constructed dynamically during the flooding of the first-time LSA, without the need for the complex algorithms of previously proposed tree-based approaches. Two versions of the proposed scheme are provided; one being more suitable for quasi-static topologies (i.e., link failure rate is low) while the other is aimed at highly dynamic networks. We show how both versions can be integrated into the OSPF protocol. We further provide a working implementation of both versions, obtained after modifying Moy's OSPF source code [OSPF: Complete Implementation (with CD-ROM), Addison Wesley, Reading, MA, 2000]. We contrast the communications and processing overheads of our scheme with those of flooding and pure tree-based broadcasting, using both analysis and simulations. Our results indicate that the hybrid approach has a significantly lower overhead than flooding; yet it enjoys the simplicity, reliability, and fast convergence of flooding.
© 2004 Elsevier B.V. All rights reserved.

☆ The work of M. Krunz was supported in part by the National Science Foundation through grants ANI-0095626, ANI-0313234, and ANI-0325979; and in part by the Center for Low Power Electronics (CLPE) at the University of Arizona. CLPE is supported by NSF (grant # EEC-9523338), the State of Arizona, and a consortium of industrial partners. An abridged version of this paper was presented at the *IEEE GLOBECOM 2002—High Speed Networks Symposium*, Taipei, Taiwan, November 17–21, 2002.

* Corresponding author. Tel.: +1-210-458-7346.

E-mail addresses: korkmaz@cs.utsa.edu (T. Korkmaz), krunz@ece.arizona.edu (M. Krunz), jguntaka@cs.utsa.edu (J. Guntaka).

1389-1286/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2004.03.034

Keywords: Flooding; Tree-based broadcasting; OSPF; Link-state routing; Scalable QoS routing

1. Introduction

In link-state routing, nodes in a routing domain try to maintain an accurate "map" of the underlying network. To achieve this goal, each node encodes the state information related to its outgoing links into a link-state advertisement (LSA) packet and disseminates this LSA throughout the network. To disseminate LSAs in a simple and reliable manner, existing link-state routing protocols (e.g., OSPF [18,19]) use flooding, in which an incoming LSA is forwarded to all neighbors except the one from which the LSA was received. As a result, a given node may receive the same LSA several times, causing unnecessary communications and processing overhead.

One can argue that the overhead of flooding which is in control plane is small when compared to data traffic. However, this is primarily the case for static link metrics that are advertised rather infrequently. Even in that case, routing protocols (e.g., OSPF) try to minimize the overhead of flooding by advertising small-size LSAs every 30 min. This infrequent dissemination is sufficient for a best-effort service, but not for QoS-oriented network services, in which multiple link metrics (e.g., available bandwidth, link delay, jitter, etc.) need to be disseminated. With more parameters being advertised, the size of the LSA inevitably gets larger. Moreover, some of these parameters (e.g., available bandwidth) are quite dynamic, and thus must be frequently disseminated (e.g., using triggered updates [1]) to provide an accurate representation of the underlying network. As we start to incorporate dynamic metrics (such as the available link bandwidth) into the routing architecture, the amount of control overhead will increase by orders of magnitude, depending on the dynamic nature of the link-state parameters. For example an egress link that carries few connections, a change in the bandwidth of any of these connections (including the case of connection termination) can, in principle, trigger a new bandwidth advertisement. These changes can occur at the time scale of seconds, not minutes, creating a large number of update messages per time unit. While QoS routing architectures are yet to be deployed and more insight into and understanding of their impact on the operation of the network are still needed, it is our belief that such architectures will generate large control-traffic volume. Note that the large volume of control traffic not only impacts the bandwidth usage, but also the efficiency of packet switching at the router and delay variation for data packets. Because the router gives higher switching priority to control packets, the arrival of a new control packet triggers an interrupt and forces the processor at the router to switch from one task to another, which degrades the performance of the router and increase the delay for data packets. Therefore, schemes that reduce the anticipated control-traffic volume will have a great impact on the success of link-state routing protocols.

To reduce the overhead of flooding in link-state protocols, researchers have investigated tree-based broadcasting approaches [4,14], in which LSAs are forwarded over broadcast trees such that every node receives exactly one copy of each LSA. While tree-based broadcasting reduces the overhead, it introduces a challenging problem, namely how to determine and maintain *consistent* broadcast trees throughout the network. Previously proposed solutions for this problem rely on complex algorithms and protocols (see Section 2 for details), making them impractical to deploy in real networks.

In this paper, we propose a hybrid mechanism that combines the best features of flooding and tree-based broadcasting. The proposed mechanism alternates between flooding and tree-based broadcasting modes. In the flooding mode, "first-time" LSAs are flooded to establish the broadcast trees, over which subsequent (refresh) LSAs are advertised in the tree-based broadcasting mode. The proposed hybrid mechanism can, in principle, be used with any link-state routing protocol. For concreteness, we focus on the current OSPF protocol and its QoS extensions as described in [2]. [1] In general, the proposed approach requires three modifications to OSPF: (1) redefining some of the currently unused bits in the LSA header, (2) adding a table in each router for maintaining information relevant to the broadcast trees, and (3) adding and/or modifying some of the steps in the "flooding procedure" of OSPF. We implemented and tested these modifications using the source code and simulation environment provided by Moy [20]. The hybrid mechanism is shown to achieve a significant reduction in the communications overhead compared to flooding; yet, it maintains the simplicity and reliability of flooding. Compared to pure tree-based broadcasting, the proposed mechanism incurs a slight extra overhead, but this overhead is overshadowed by the simplicity of this mechanism and its amenability to practical application in real networks. Furthermore, in contrast to previous tree-based broadcasting mechanisms, in which the broadcast trees are determined based on the hop count, ours uses the minimum delay experienced during the flooding of a first-time LSA to compute the broadcast trees. As a result, it enjoys the same fast convergence of flooding.

The rest of this paper is organized as follows. In Section 2, we give background information on flooding in OSPF and review the literature on tree-based broadcasting. In Section 3 we present the basic version of the proposed hybrid mechanism, describe how to integrate it into the OSPF protocol, analyze its overhead, and contrast this overhead with those of flooding and pure tree-based broadcasting using both analysis and simulations. In Section 4 we present a variant of the hybrid mechanism that is suited for dynamic topologies with a high rate of link failures. Extending the hybrid mechanism to hierarchical OSPF networks is described in Section 5. The paper is concluded in Section 6. In the Appendix A, we give highlights of our software implementation of the hybrid scheme, performed by modifying Moy's OSPF source code. A complete description of this implementation is available online at www.cs.utsa.edu/~korkmaz/research/hftb.

[1] The Opaque LSA Option [8] is another proposal to provide QoS extensions in OSPF. Since it also relies on flooding, it can be easily integrated into our hybrid approach.

2. Background and related work

2.1. Flooding

Flooding is used in OSPF to disseminate the link-state information to all routers in the same domain. Each router periodically generates LSAs representing the parameters of its outgoing links and sends these LSAs to all of its neighbors. Routers forward received LSAs to their neighbors except the ones that sent the LSAs. Typically, received LSAs are explicitly acknowledged, although in some cases the acknowledgement (ACK) is implicit (for example, if two neighbors send the same LSA to each other at around the

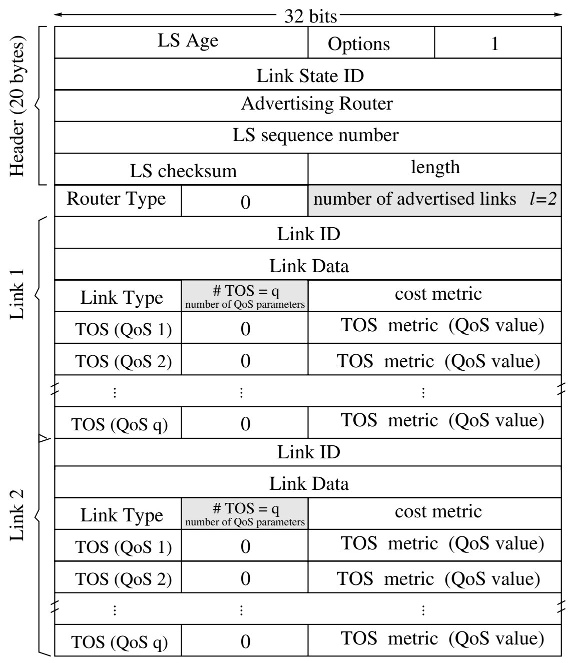


Fig. 1. Structure of an OSPF router-LSA with two advertised links.

same time, then each of them can be sure that the other has the LSA, so there is no need to generate explicit ACKs). This flooding process allows every node to acquire the same map of the network.

OSPF uses five types of LSAs. For a network with point-to-point links, only one of these types, known as *router-LSA*, is used. The basic structure of a router-LSA is shown in Fig. 1. Every router-LSA starts with a 20-byte header that contains information that uniquely identifies the LSA. The rest of the LSA contains the values of the "cost" metric and any additional QoS parameters that are associated with the outgoing links of the LSA's originator. Note that OSPF currently uses a single metric, but the type-of-service (TOS) field in OSPF, which has not been much used in the past, can be redefined to advertise multiple link parameters (see [2] for details). Consider the shaded fields in Fig. 1. In this example, the number of advertised links is indicated by l. The link-state information is then repeated l times with different values. Each link has some identification information followed by the link parameters. The number of link parameters is indicated by q. Each parameter is encoded using a four-byte field that includes the parameter name and its value. Hence, the size (in bytes) of a router-LSA originating from node u is given by $S(u) \overset{\text{def}}{=} 20 + 4 + l(u)(12 + 4q)$, where $l(u)$ is the number of advertised outgoing links from node u (i.e., $l(u) = \deg(u)$). Note that an LSA ACK packet consists of only a 20-byte header.

Flooding has also been used for path determination in mobile ad hoc networks (MANETs), and reducing its overhead has been the focus of much research. In MANETs, the routing protocol may be *proactive* (table driven) or *reactive* (on-demand). [2] Examples of proactive protocols are DSDV [25], WRP [21], and CGSR [7]. In proactive protocols, nodes periodically broadcast their link (or path) costs and use Dijkstra (or Bellman–Ford) like algorithms to compute the "shortest" paths to various destinations. These protocols are somewhat similar to link-state and distance-vector protocols used over the wired Internet (e.g., OSPF and BGP). Reactive protocols, on the other hand, operate *on demand*; when the source node does not have in its cache a path to a given destination, it broadcasts a route request (RREQ) packet, querying its neighbors about the availability of such a path. Dynamic source routing (DSR) [15], ad-hoc on-demand distance vector (AODV) [24], and the temporally ordered routing algorithm (TORA) [23] are all examples of reactive routing protocols. In their basic forms, reactive protocols rely on flooding for path discovery, whereby the source node floods the network with its RREQ packet. When an intermediate node receives the RREQ packet, it responds back with a route reply (RREP) if it has a path to the destination. Otherwise, the intermediate node continues to flood the packet. Eventually, at least one node (possibly the destination itself) will respond affirmatively with a RREP. The overhead of flooding via wireless broadcast is even more significant than flooding over point-to-point links, and it can lead to a "broadcast storm problem" [22] (neighboring nodes that receive an RREQ packet rebroadcast it simultaneously, leading to collisions, backoffs, retransmissions, etc.). Several attempts were made to reduce the overhead of flooding in reactive routing protocols (e.g., [6,16]). For example, Ko and Vaidya [16] proposed the *location aided routing* (LAR) scheme, in which the Global Positioning System (GPS) is used to localize path queries and limit their propagation. A similar objective is achieved in [6] but without the need for GPS-based location information. Note that such techniques are aimed at on-demand routing approaches, and are not applicable to table-driven proactive protocols like OSPF.

The overhead of flooding has also been addressed in the context of establishing multicast trees (e.g., [5]). In [5] the authors use consecutive flooding of a multicast request (a technique known as *expanding rings*) to establish a multicast tree for a given set of receivers. This is somewhat similar to flooding a packet with a limited time-to-live (TTL) value, where the TTL value in incrementally increased until all receivers are included in the multicast tree.

[2] A combination of reactive and proactive approaches can also be used, as in the zone routing protocol (ZRP) [13].

2.2. Tree-based broadcasting

State dissemination based on tree-based broadcasting appears in the literature in two forms: single broadcast tree (SBT) and multiple broadcast trees (MBT). In the SBT approach, all nodes compute a common broadcast tree (e.g., a spanning-tree), and every node marks its own links on that tree. Every node then receives LSAs via one of its marked links and forwards the LSAs through its other marked links. The SBT approach has two main disadvantages. First, it results in an unbalanced load distribution since LSAs are sent over a fixed subset of the network links (i.e., the links that belong to the broadcast tree). Second, it is quite possible for nodes that are neighbors according to the network graph to lie far away from each other on the broadcast tree, a situation that delays the convergence of the routing protocol. In [3] the authors explored the viability of the SBT approach for state dissemination in the PNNI protocol. They provided a distributed spanning-tree algorithm for determining the broadcast tree. However, finding this tree and maintaining it in a consistent manner involves complex operations such as exchanging extra control packets besides LSAs and executing the spanning-tree algorithm in a distributed manner.

In the MBT approach, every node has its own broadcast tree (e.g., a shortest path tree). For illustration, consider the trees originating from nodes 1 and 3 in Fig. 2. The LSAs originating from a given node are disseminated over that node's broadcast tree. For example, node 1 generates an LSA and sends it to nodes 2 and 3. However, only node 2 forwards this LSA to node 4 (according to the broadcast tree of node 1). To disseminate LSAs over their originators' broadcast trees, every node needs to know the broadcast trees of all other nodes. This can be done as follows. Every node determines its parent and children on every broadcast tree and stores these parent–children relationships in a table [14]. Fig. 3 illustrates an example. Let T_i be the broadcast tree originating from node i, $i = 1, 2, \ldots, n$, where n is the number of nodes in the network. Consider the representation of T_1 at node 2. Since node 2 has only one child on T_1 (namely, node 4), it marks node 4 as its child in the first row of its table. So whenever node 2 receives an LSA originating from node 1, it forwards this LSA to node 4 only. Node 2 also stores its parent on T_1 (namely, node 1) in the first row of its table. Since LSAs are disseminated over different shortest-path trees, the MBT approach provides some form of load balancing. The convergence time of this approach is also faster than that of the SBT approach.

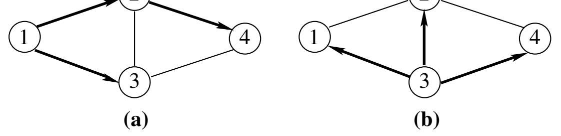


Fig. 2. Multiple broadcast trees originating from nodes 1 and 3.

A key issue in the MBT approach is how to determine the broadcast trees in a distributed manner. Previously proposed approaches achieve that through additional control packets (besides the LSAs) and by relying on protocols that execute some variant of the shortest path algorithm [4,14]. This complicates the establishment and maintenance of consistent broadcast trees. The main objective of these complicated mechanisms is to always disseminate LSAs over the broadcast trees. In [14] the authors addressed the issue of determining broadcast trees while the topology information is still being disseminated over these trees. In [4] the authors considered the idea behind reverse-path forwarding (RPF) in [11] and proposed a new topology dissemination protocol called TBRPF, in which broadcast trees are computed based on full topological information received over the broadcast trees themselves. In TBRPF, every node executes Dijkstra's algorithm to determine a reverse minimum-hop tree, and then exchanges some information with neighbors to determine its parent and children from the standpoints of other nodes. Although TBRPF provides more reliability than other existing methods, it suffers from the overhead associated with computing the trees and communicating with neighbors whenever a topological change occurs. The ideas behind SBT and MBT have also been used in various multicast protocols (e.g., CBT, DVMPR, PIM [12,17,26]). Specifically, SBT is similar to the shared multicast trees while MBT is similar to source-based multicast trees. In general, multicast protocols are being designed to optimize the performance for a subset of nodes (receiving nodes). In our case, however, we try to provide better performance in broadcasting sender's packets to all nodes in the network.

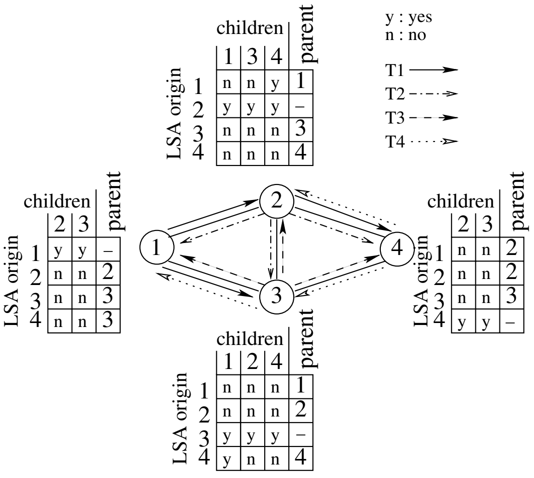
3. Hybrid dissemination mechanism

In this section, we first introduce our Hybrid Flooding and Tree-based Broadcasting (HFTB) approach and prove its correctness. We then discuss how it can be integrated into OSPF. Finally, we compare the overhead of HFTB with that of flooding and pure tree-based broadcasting using analytical and simulation results. For now, we assume that the underlying network consists of a single area (i.e., domain) and that no link failures take place within the standard OSPF update interval (30 min). In later sections, we modify HFTB to deal with topologies that exhibit frequent link failures and we also address its extension to hierarchical OSPF-based networks.

3.1. HFTB and its correctness

As indicated before, the 30 min update interval of OSPF [18] is sufficient for the relatively static cost metric, but it is not enough for other link parameters (e.g., available bandwidth) that may change several times within the 30 min period. Such dynamic parameters need to be frequently disseminated, e.g., using triggered updates. The objective of HFTB is to disseminate triggered LSAs using tree-based broadcasting while continue to use flooding for disseminating the relatively static cost metric and connectivity information every 30 min.

Basically, HFTB is similar to previous MBT approaches in the sense that every node maintains the same parent–children relationships, as shown in Fig. 3. However, in contrast to previous MBT approaches, HFTB uses flooding of the *first* LSA in every 30 min update interval to establish the broadcast trees, which are then used to disseminate subsequent "refresh" LSAs generated *within*



LSA origin	children 1	children 3	children 4	parent
1	n	n	y	1
2	y	y	y	–
3	n	n	n	3
4	n	n	n	4

LSA origin	children 2	children 3	parent
1	y	y	–
2	n	n	2
3	n	n	3
4	n	n	3

LSA origin	children 2	children 3	parent
1	n	n	2
2	n	n	2
3	n	n	3
4	y	y	–

LSA origin	children 1	children 2	children 4	parent
1	n	n	n	1
2	n	n	n	2
3	y	y	y	–
4	y	n	n	4

Fig. 3. Representing the parent–children relationships of the broadcast trees in the MBT approach.

the update interval. No extra control packets or complex protocols are needed to establish the broadcast trees.

The broadcast trees are established as follows. Let LSA_u denote a flooded LSA that was generated by some node u. Suppose that LSA_u arrives at some node i for the first time through node j, as shown in Fig. 4. Node i selects node j as its parent from the standpoint of node u and acknowledges node j. When node j receives the acknowledgment, it records node i as its child from the standpoint of node u. If LSA_u arrives at node i again via another node v, then node i acknowledges the LSA as in OSPF without establishing a new parent–child relationship. After the flooding of the first LSA_u, every node can determine its parent and children on the broadcast tree of node $u(T_u)$. In contrast to previous approaches that establish the broadcast trees with respect to (w.r.t.) the minimum hop count, HFTB dynamically determines the broadcast trees w.r.t. the actual minimum delay, and

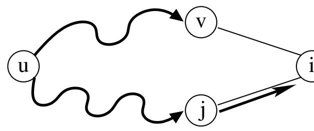


Fig. 4. Establishing broadcast trees in the HFTB approach.

broadcasts LSAs over these trees, suggesting that HFTB converges as fast as flooding.

One concern here could be whether the initially established optimal broadcast trees will stay optimal for the next 30 min. It is true that the initially optimal trees might not stay optimal for the next 30 min due to delay variations in the network. However, the delay variation on trees can be reduced by giving priority to OSPF packets. In fact, existing routers (e.g., Cisco IOS) assign an IP precedence value of 6 (i.e., Internetwork Control) to OSPF and other control packets on the control plane, and process them before queued data packets (see http://www.cisco.com/warp/public/105/rtgupdates.html for details). Giving priority to OSPF and other control packets means that these packets will not see the high delay variations seen on the data paths. Therefore, from the point of view of the *refresh* LSAs, the "optimal" path obtained based on first-time (flooded) LSAs will likely be stable (i.e., its delay does not change significantly) over a 30 min period.

Theorem 1 (Correctness of HFTB). *Consider a network $G = (V, E)$ with bidirectional links. Suppose that an arbitrary node $s \in V$ generates an LSA and floods it throughout the network. Furthermore, suppose that the LSA experiences some delay $d(u, v)$ while being processed and forwarded from node u to node v. Under HFTB, T_s is established throughout the network in finite time (i.e., every node can determine its parent and children on T_s). Moreover, T_s converges to a shortest-paths tree.*

Proof. Initially, T_s consists of only node s. So, the theorem is trivially true. Consider T_s after flooding LSA$_s$ through some nodes in the network. In flooding, an arbitrary node u may receive the same LSA$_s$ several times. However, node u forwards the incoming LSA$_s$ to its neighbors only once, upon the first arrival of this LSA. In addition, node u selects its parent on T_s by acknowledging the node from which LSA$_s$ was received for the first time. Subsequent arrivals of LSA$_s$ at node u are acknowledged without establishing a parent–child relationship. Let $t[u]$ be the time at which node u receives LSA$_s$ for the first time along the path $p = \langle v_0 = s, v_1, \ldots, v_k = u \rangle$. Without loss of generality, we assume that $t[s] = 0$. So, $t[u] = \sum_{i=1}^{k} d(v_{i-1}, v_i)$.

To prove that T_s is a shortest-paths tree, we need to show that the path p does not contain any cycle and that p is the shortest path from s to u. The proof follows similar arguments to those used in [10, pp. 523–525] for shortest-paths trees. However, instead of the process of relaxing a link (u, v) in [10], we consider the process of forwarding first-time LSA$_s$ from node u to node v. So it is sufficient to show that forwarding LSA$_s$ from node u to node v upon its first arrival is the same as relaxing the link (u, v) in the computation of a shortest-paths tree.

In computing the shortest-paths tree, a node u with minimum $t[u]$ is selected and every link (u, v) is considered for relaxation in a sequential manner. If $t[u] + d(u, v) < t[v]$, then link (u, v) is relaxed, i.e., the parent of node v is set to node u and $t[v]$ is set to $t[u] + d(u, v)$. In HFTB, paths are explored in parallel. So node u starts forwarding LSA$_s$ through (u, v) as soon as it receives LSA$_s$ for the first time. In other words, a node u with minimum $t[u]$ is automatically selected in parallel and every link (u, v) is considered. If node v receives LSA$_s$ for the first time via node u, then the parent of node v is set to node u and $t[v]$ is set to $t[u] + d(u, v)$; this is the same as relaxing (u, v) in [10]. Otherwise, no parent–children relationship is established again. The rest of the proof follows the same proofs in [10, pp. 523–525].

Finally, forming the broadcast tree T_s takes finite time since in flooding every node receives LSA$_s$ in a finite amount of time. □

3.2. Integrating HFTB into OSPF

OSPF can be easily modified to support the proposed approach, as outlined in Fig. 5. In general, we make three modifications to OSPF. The first one is to designate one of the unused bits of the Options field in the LSA header as *Flooding or Tree-based broadcasting* (FT)-bit. If the FT-bit is set to 0, the LSA will be flooded throughout the network as in the standard OSPF and will be used to establish the broadcast tree of that LSA. If this bit is set to 1, then the LSA will be disseminated

```
Basic HFTB executed at node $i$
 Upon becoming operational
 1. node $i$ initializes its parent-children table
 2. node $i$ synchronizes its link-state database as in OSPF
 3. node $i$ goes into flooding mode
 Repeat every 30 minutes
 1. node $i$ goes into flooding mode
 2. node $i$ generates a new LSA$_i$ that describes the state of its outgoing links
 Upon failure of link $(i, j)$
 1. node $i$ goes into flooding mode and generates a new LSA$_i$
 Upon generating a new LSA$_i$
 1. if node $i$ is in flooding mode then
 1.1    set FT-bit in Options to 0
 1.2    send the LSA$_i$ to all neighbors
 1.3    node $i$ goes into tree-based broadcasting mode
 2. else if node $i$ is in tree-based broadcasting mode then
 2.1    set FT-bit in Options to 1
 2.2    send the LSA$_i$ to all children of node $i$
 3. end if
 Upon receiving an LSA$_u$ originating from node $u$ via node $j$
 1. if the LSA$_u$ is the most recent then
 1.1    if FT-bit in Options is 0 then
 1.1.1     send the LSA$_u$ to all neighbors except node $j$
 1.1.2     table[LSA's originator, i.e., node $u$].parent = node $j$
 1.1.2     table[LSA's originator, i.e., node $u$].childlost = no
 1.1.3     send LSA$_u$ ACK to node $j$ by setting FT-bit in Options to 1
 1.2    else if FT-bit in Options is 1 then
 1.2.1     send the LSA$_u$ to all children according to LSA's originator, i.e., node $u$
 1.2.2     send LSA$_u$ ACK to node $j$ (FT-bit is set to 1)
 1.3    end if
 2. end if
 Upon receiving an ACK fir LSA$_u$ from node $j$
 1. if FT-bit in Options is 1 then
 1.1    table[LSA's originator, i.e., node $u$].children[node $j$] = yes
 2. else if FT-bit in Options is 0 then
 2.1    table[LSA's originator].children[node $j$] = no
 3. end if
end HFTB
```

Fig. 5. Integrating HFTB into the OSPF protocol.

over its originator's broadcast tree. The FT-bit is also used in LSA ACKs. In this case, if the FT-bit is 1, then this is an indication that the receiver has selected the sender as its parent; otherwise, the receiver has a different parent. When a sender node receives an ACK with FT-bit = 1, this sender records the receiver as its child. To maintain these parent–children relationships, we need to create a table at each node, which is the second required modification. The third modification is to add/change some steps in the flooding procedure of OSPF (as listed in Fig. 5), so that the broadcast trees can be established during the flooding of the first LSAs and used during the tree-based dissemination of the subsequent LSAs generated within the 30 min interval.

We implemented HFTB, starting with the OSPF's source code provided in [20] and modifying this code to account for the above required changes. The implementation was tested on a "real" network of four Linux-based PCs acting as OSPF nodes. It was also tested using the simulation tool provided in [20]. The details of our modifications and implementations are summarized in Appendix A and described in detail in a technical report that is available online at www.cs.utsa.edu/~korkmaz/research/hftb.

3.3. Performance comparisons

To compare the overheads of various dissemination mechanisms, we consider two performance measures: (1) the total number of LSAs and ACKs (TNLA), which gives an indication of the processing overhead, and (2) the total size (in bytes) of the exchanged LSAs and ACKs (TSLA), which gives an indication of the communications overhead. These measures can be determined analytically under some ideal link conditions, namely, no LSA losses, no retransmissions, and no implicit ACKs. We first present such results for flooding, HFTB, and pure tree-based broadcasting. We then compare these results with the ones obtained via simulations, using the tool provided in [20]. Note that our analytical results are intended to provide lower bounds on the TNLA and the TSLA. We simply use the analytical results (the lower bounds on the TNLA and the TSLA) to illustrate the impacts of different system parameters on the performance of the existing and proposed solutions, and compare the general performance trends in these solutions. In other words, our analytical results are not intended to replace the simulation results, which take into account the dynamic behavior of the system at hand. In our emulations/simulations, we relax the simplifying assumptions made for analytical results and measure the TNLA and TSLA of the exiting and proposed schemes under the actual operation of OPSF.

3.3.1. Analytical results

Consider a network with n nodes and m links. Each node u periodically (every 30 min) generates a router-LSA and floods it as in the standard OSPF. Within the 30 min period, each node may be triggered to generate additional router-LSAs that advertise the most recent values of the link-state parameters. Let λ indicate the average number of triggered advertisements within a 30 min interval. Recall that the size of an ACK packet is 20 bytes while the size of an LSA packet originating from node u is $S(u) = 20 + 4 + l(u)(12 + 4q)$ bytes, where $l(u)$ is the number of advertised links (i.e., the degree of node u) and q is the number of parameters associated with every link. If no link failure occurs in a given 30 min interval, the TNLA and TSLA of flooding are

$$\text{TNLA}_{\text{flooding}} = 2(\lambda + 1)nm, \tag{1}$$

$$\text{TSLA}_{\text{flooding}} = (\lambda + 1)\sum_{u=1}^{n} m[S(u) + 20]. \tag{2}$$

In the above equations, we assume that every LSA is explicitly acknowledged. So when two neighboring nodes, say u and v, receive the same LSA from a third node, one of them, say u, forwards it first to the other, which in turn acknowledges the LSA. As a result, the total number of forwarded LSAs in the network will be the same as the total number of ACKs (i.e., half of the analytically computed TNLA are LSAs). In a real network, however, u and v may forward the same LSA to each other around the same time. In this case, each of them can be sure that the other has the LSA, and there is no need to generate an explicit ACK. Compared to the scenario assumed in the analysis, there is one additional generated LSA but one less ACK, which means that the actual TNLA value is roughly equal to the analytically computed one (the latter is slightly smaller because the analysis ignores retransmissions of LSAs). As for the TSLA measure, its actual value will be much larger than the analytically predicted one, because there are more LSAs than ACKs in the actual TNLA value.

In pure tree-based broadcasting, LSAs are disseminated over trees, each consisting of $n - 1$ links. Thus, the TNLA and TSLA of tree-based broadcasting are

$$\text{TNLA}_{\text{tree}} = 2(\lambda + 1)n(n - 1), \tag{3}$$

$$\text{TSLA}_{\text{tree}} = (\lambda + 1)\sum_{u=1}^{n} (n - 1)[S(u) + 20]. \tag{4}$$

Note that establishing the broadcast trees also involves some protocol overhead, which we are ignoring since our focus is on the TNLA and

TSLA during the state dissemination phase. Also note that there are no implicit ACKs in pure tree-based broadcasting.

For HFTB, no extra protocol overhead is incurred due to the establishment of broadcast trees during the flooding of the first LSA. The TNLA and TSLA of HFTB are

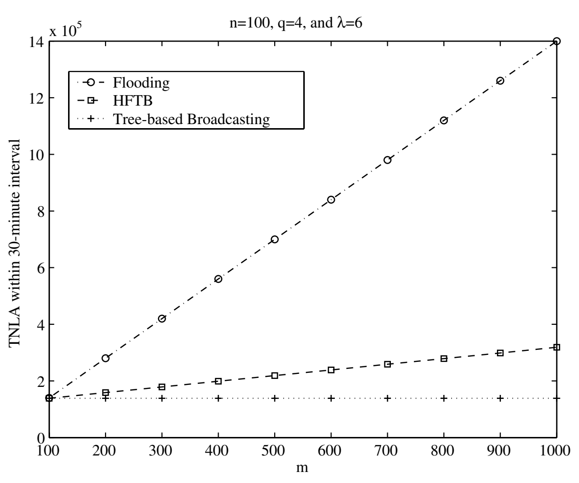
$$\mathrm{TNLA}_{\mathrm{HFTB}} = 2(nm + \lambda n(n-1)), \tag{5}$$

$$\mathrm{TSLA}_{\mathrm{HFTB}} = \sum_{u=1}^{n} m[S(u)+20] + \lambda \sum_{u=1}^{n} (n-1).[S(u)+20]. \tag{6}$$

In practice, some first-time LSAs will be implicitly acknowledged during the flooding phase (but not during the tree-based broadcasting phase). So the relative improvement of HFTB over flooding will be more pronounced in practice than what is being predicted by the above equations (as shown later in simulations).

The TNLA depends on three parameters (n, m, λ) while the TSLA depends on four parameters (n, m, q, λ). In the numerical examples below, we consider topologies with $n = 100$, and we vary m, q, and λ. The same trends have been observed for other values of n. Fig. 6 depicts the TNLA of various dissemination mechanisms versus m and λ. As m and λ increase, the TNLA (and thus the processing overhead) of compared schemes increases linearly. However, the TNLA of flooding increases at a higher rate than the other two schemes. Fig. 7 depicts the TSLA of various dissemination mechanisms versus m, q, and λ. As m increases, the TSLA (and thus the communications overhead) of flooding increases quadratically, while this increase is linear in other mechanisms. As shown in the figure, the TSLA of all mechanisms increases linearly with q. However, the TSLA of flooding increases at a higher rate than those of tree-based mechanisms. The TSLA of all mechanisms is linearly proportional to λ. Once again, the TSLA of flooding increases at a much higher rate than that of other mechanisms.

In general, pure tree-based broadcasting is expected to provide the best possible efficiency in state dissemination, given that the broadcast trees are already established. However, establishing such trees in the pure tree-based approach requires complex algorithms and protocols to maintain consistent trees throughout the network. Because of the complexities and/or unreliabilities of previous tree-based mechanisms, current Internet protocols do not use tree-based broadcasting, and instead rely on flooding despite its high communications overhead. The proposed HFTB mechanism takes advantage of both flooding and tree-based broadcasting. HFTB is easy to incorporate into the current link-state protocols and provides



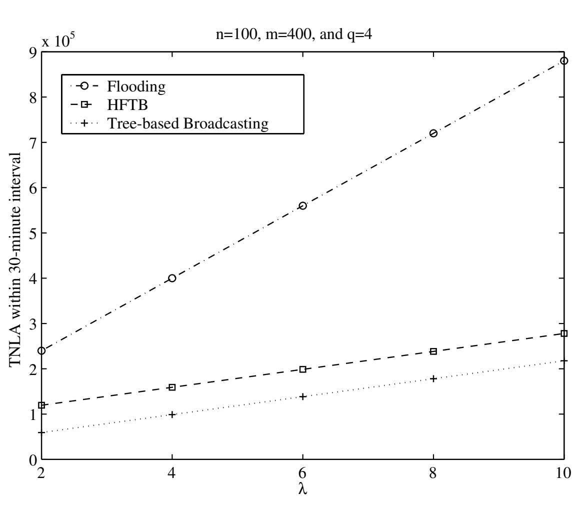


Fig. 6. TNLA (processing overhead) versus m and λ (based on analysis).

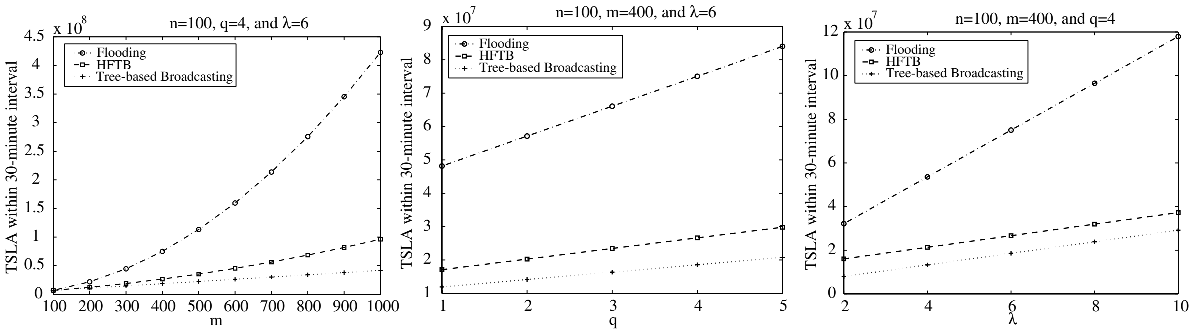


Fig. 7. TSLA (communications overhead) versus m, q, and λ (based on analysis).

significantly better performance than flooding. It is particularly advantageous in networks that support QoS routing, where the values of q and λ are typically larger than those of best-effort networks.

3.3.2. Simulation results

We also compare the performance of our implementation of HFTB against flooding using the simulation tool `ospfd_sim` provided in [20]. This tool takes an entire OSPF network as a configuration file and executes a copy of the actual `ospfd` software for each router, allowing us to test/evaluate actual implementations (and modifications) of the OSPF protocol as if we had real OSPF routers. In our simulations, we use the topologies shown in Fig. 8, which are modified versions of the ANSNET topology [9]. The configuration files for these topologies are available online at www.cs.utsa.edu/~korkmaz/research/hftb. Since the general trends in TNLA and TSLA as functions of λ and q are clear, we just select one reasonable value for λ and q. For example, we assume that each link is associated with three QoS parameters (i.e., $q = 3$) whose values are refreshed five times within each 30 min period (i.e., $\lambda = 5$). A smaller value would not show clearly the difference between HFTB and flooding. On the other hand, a larger value would require more simulation time and resources, without necessarily depicting a different trend in the behavior. For each topology, the simulation program was ran for three hours. The obtained TNLA and TSLA values, normalized and averaged over a 30 min interval, are shown in Fig. 9. The figure also shows the TNLA and TSLA values computed using the previous formulas. For the TNLA, the analysis and the simulations produce comparable results, with the difference attributed to the random initialization of nodes in the simulations and the enabling of LSA retransmission. In the case of the TSLA, the difference between the analysis and the simulations is significant, particularly for flooding. The cause of this big gap is mainly attributed to the assumption used in the analysis that all LSAs are sent in one

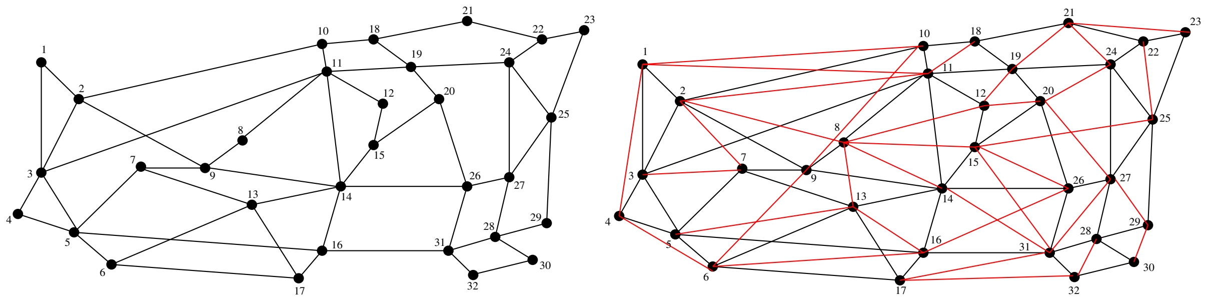


Fig. 8. Modified 32-node ANSNET topologies with 54 and 91 links.

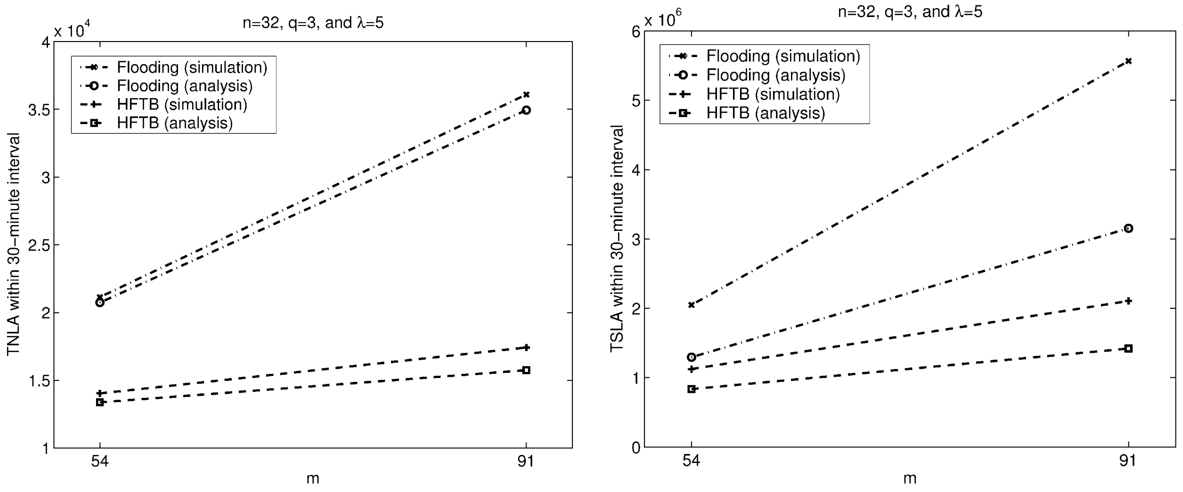


Fig. 9. TNLA and TSLA versus *m* for flooding and HFTB (based on analysis and simulation).

direction over a link and *explicitly* acknowledged in the other direction. In the simulations, however, several LSAs are sent in both directions over a link and implicitly acknowledged, resulting in more LSAs than ACKs. For example, in one simulation run on the network with 54 links, we counted 14,994 LSAs and 6148 ACKs in the case of flooding. Their total (21,142) is the TNLA value, which is close to its analytically obtained counterpart (20,736). In summary, the number of implicit ACKs (and thus the number of LSAs) increases during actual flooding, resulting in significantly higher TSLA than what is analytically expected. In case of HFTB, some implicit ACKs are used during the flooding of the first-time LSAs. However, since subsequent LSAs are disseminated over the broadcast trees, implicit ACKs are no longer used, resulting in only a slightly higher TSLA than what is analytically expected. Accordingly, we should expect that in practice, HFTB will provide a much better performance gain over flooding than what is analytically predicted.

4. Enhanced hybrid mechanism

The previously discussed HFTB mechanism computes the broadcast trees once every 30 min. However, due to topological changes, particularly link failures, some broadcast trees may become disconnected shortly after they have been updated. If no action is taken to repair these trees, some nodes may not receive the up-to-date values of link parameters for at most 30 min. The basic HFTB is still a viable solution if the probability of a link failure is low or if the underlying path selection algorithm is capable of dealing with inaccurate state information. However, if highly accurate state information is needed at every node, then the disconnected trees should be repaired *during* the tree-based broadcasting phase. This can be done by using a slightly modified version of HFTB, which we refer to as HFTB with *repair tree option* (HFTB-RT).

In the absence of link failures, HFTB-RT is similar to the basic HFTB. When a link fails, HFTB-RT dynamically repairs the disconnected broadcast tree(s) by switching the LSA of that tree to the flooding mode at the disconnection point. This is similar to the node at disconnection point instantiating the LSA. As an example, consider the situation in Fig. 4. Suppose that link (j, i) has gone down, disconnecting T_u. Upon detecting the link failure, node j realizes that it has lost its child i from the standpoint of node u while node i has lost its parent. Now assume that node j receives an

LSA from node u in tree-based broadcasting mode. In this case, since node j knows that node i and other successors will not receive LSA_u from any other nodes, it will flood LSA_u throughout the network to make sure every node gets the same LSA. Although the LSA is flooded with the same sequence number, we mark some bits in the header to indicate that this LSA is more recent than the previous ones so that the nodes that already have it will not discard it. For example, as shown by the dashed arrows in Fig. 10, node u receives LSA_u flooded back from j. However, node u *does not* discard it, since the RT-bit that we describe later in this LSA is set to 1. So node u updates its database and further floods this LSA to other neighbors. At some point during this flooding, node v receives the LSA and forwards it to node i. Note that in general it is possible that LSA_u might reach node i through a node other than node u or v. In any case, the node from which node i receives the LSA_u first time is selected as the new parent of node i on T_u. Accordingly, node i sends an ACK message to that node (e.g., v), indicating the establishment of a new parent–child relationship. When node v receives this ACK, it records node i as a child. Note that the objective here is to repair the tree rather than establishing it from scratch. So if a node receives the same LSA from its child, then this node should not select its child as its parent. For example, in Fig. 10, node u gets the LSA back from node j, but does not select it as a parent. As a result, the broadcast tree originating from node u is repaired, as shown by the solid arrows in Fig. 10. Also note that the same LSA is sent over some links twice, increasing the TNLA and TSLA. However, our simulations indicate that HFTB-RT still gives much better performance than flooding.

Integrating HFTB-RT into OSPF can be done as described in the pseudo-code in Fig. 11. In

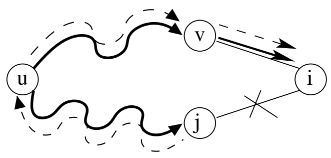


Fig. 10. Repairing broadcast trees in HFTB-RT.

addition to the OSPF modifications described in Section 3.2, we need to make two other changes. First, we need to designate a second unused bit in the Options field of the LSA header as the Repair Tree (RT) bit. Following a link failure, nodes at both ends of the failed link generate new LSAs with RT-bit = 1 and *flood* these LSAs throughout the network. These LSAs will be used to repair the broadcast trees. Suppose that some node i receives LSA_u with FT-bit = 0 and RT-bit = 1 via some node v. Node i first checks whether it has a parent on T_u. If not, it selects node v as its parent and sends an ACK message with FT-bit = 1 and RT-bit = 0 to node v. Upon receiving this ACK, node v records node i as its child.

The second change is to add a new step in the flooding procedure of OSPF, as follows. If the link connecting node i to one of its children on the tree of node u goes down, then node i should switch an incoming LSA of node u from tree-based broadcasting mode to flooding mode (i.e., change the FT-bit and the RT-bit from 1 and 0 to 0 and 1, respectively). Node i then floods that LSA to all neighbors including its own parent. During this flooding, those nodes that have lost their parents due to the link failure can select a new parent, repairing the disconnected trees. Again by modifying the OSPF's source code provided in [20], we have implemented HFTB-RT and tested our implementation using both a real network with four OSPF nodes and also using the simulation tool in [20].

When no link failures take place, HFTB-RT and HFTB have the same TNLA and TSLA performance. Under link failures, HFTB-RT uses the flooding mode more often than HFTB, and its TNLA and TSLA values increase with the frequency of failures. Analytical estimation of these values is not possible at present, since it requires knowledge of the number of trees that become disconnected following a link failure. Instead, we use simulation results to compare flooding and HFTB-RT. Again we use the ANSNET topologies and we let $q = 3$ and $\lambda = 5$. The simulations are ran for three hours with various numbers of link failures happening at different times within the three hours. The TNLA and TSLA values averaged over a 30 min interval are shown in Fig. 12

```
HFTB-RT executed at node $i$
 Upon becoming operational
 1. perform the same tasks as in Basic HFTB
 Repeat every 30 minutes
 1. perform the same tasks as in Basic HFTB
 Upon failure of link $(i, j)$
 1. for each originator node $u$ do
 1.1      if table[originator $u$].children[node $j$] = yes then
 1.1.1        table[originator $u$].childlost = yes
 1.2      end if
 2. end for
 3. node $i$ goes into flooding mode and generates a new LSA$_i$
 Upon generating a new LSA$_i$
 1. perform the same tasks as in Basic HFTB
 Upon receiving an LSA$_u$ via node $j$
 1. if the LSA$_u$ is the most recent then
 1.1      if FT-bit in Options is 0 then
 1.1.1        if RT-bit in Options is 0 then
 1.1.1.1          send the LSA$_u$ to all neighbors except node $j$
 1.1.1.2          table[LSA's originator, i.e., node $u$].parent = node $j$
 1.1.1.3          table[LSA's originator, i.e., node $u$].childlost = no
 1.1.1.4          send LSA$_u$ ACK to node $j$ by setting FT-bit=1 and RT-bit=0
 1.1.2        else (i.e., RT-bit is 1)
 1.1.2.1          if node $i$ has the LSA$_u$ (with FT-bit=1 and RT-bit=0) then
 1.1.2.1.1            if table[LSA's originator, i.e., node $u$].parent is node $j$ then
 1.1.2.1.1.1              send LSA$_u$ ACK to node $j$ by setting FT-bit=1 and RT-bit=1
 1.1.2.1.2            else
 1.1.2.1.2.1              send LSA$_u$ ACK to node $j$ by setting FT-bit=0 and RT-bit=1
 1.1.2.1.3            end if
 1.1.2.2          else
 1.1.2.2.1            table[LSA's originator, i.e., node $u$].parent = node $j$
 1.1.2.2.2            send LSA$_u$ ACK to node $j$ with FT-bit=1 and RT-bit=1
 1.1.2.3          end if
 1.1.2.4          send LSA$_u$ to all neighbors except node $j$
 1.1.3        end if
 1.2      else (i.e., FT-bit in Options is 1)
 1.2.1        if table[LSA's originator, i.e., node $u$].childlost = yes then
 1.2.1.1          set FT-bit to 0 and RT-bit to 1 (flood LSA$_u$ to repair the broadcast tree of node $u$)
 1.2.1.2          send LSA$_u$ to all neighbors including node $j$
 1.2.1.3          table[originator $u$].parent = node j
 1.2.1.4          table[originator $u$].childlost = no
 1.2.1.5          send LSA$_u$ ACK to node $j$ with FT-bit=1 and RT-bit=1
 1.2.2        else
 1.2.2.1          send the LSA$_u$ to all children according to LSA's originator, namely node $u$
 1.2.2.2          send LSA$_u$ ACK to node $j$ (FT-bit=1, RT-bit=0)
 1.2.3        end if
 1.3      end if
 2. end if
 Upon receiving an ACK for LSA$_u$ from node $j$
 1. perform the same tasks as in Basic HFTB
end HFTB-RT
```

Fig. 11. Integrating HFTB-RT into the OSPF protocol.

for the 54-link topology (similar trends were also observed for the 91-link topology).

As the number of link failures increases, the TNLA and TSLA of flooding decrease while those of HFTB-RT increase gradually. The reason for this is that, since the number of active links in the network decrease due to link failures, the flooding sends LSAs through less number of links and thus

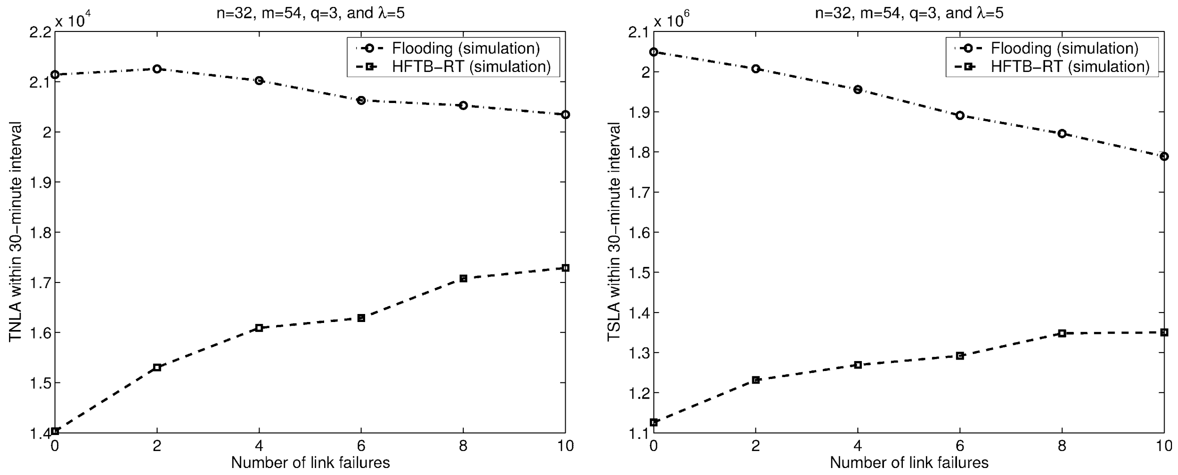


Fig. 12. Overhead of HFTB and flooding versus the number of link failures (simulations).

reduce the TNLA and TSLA. In HFTB-RT, however, since some LSAs in tree-based broadcasting mode are switched to flooding mode in case of link failures and flooded throughout the network, the TNLA and TSLA of HFTB-RT increase gradually with the increase of link failures. However, HFTB-RT is still giving better performance than the flooding under moderate number of link failures. Note that as the number of link failures increases, the network becomes more sparse (i.e., looks like a tree). In this case, flooding may perform better than HFTB-RT. In practice, however, core networks do not experience frequent link failures that lead to sparse topologies. Therefore, HFTB-RT is a viable solution for disseminating QoS-related state information accurately and efficiently in practice.

5. Extending HFTB(-RT) to OSPF-based hierarchical networks

For scalability reasons, OSPF supports two-level hierarchical routing, in which the network is divided into areas. In each area, the *area border routers* capture the routing information within that area, summarize this information into *summary-LSAs*, and then *flood* these LSAs throughout the other areas and their border routers. So far, we have explained the operation of HFTB and HFTB-RT *within* an area. We now explain how to use/extend the proposed hybrid approach to hierarchical OSPF networks. First, note that it is possible to use HFTB(-RT) within areas while still using flooding between areas. This may be particularly appropriate when the number of areas is small or when the inter-area connectivity is sparse.

Basically, since OSPF also uses flooding between areas (but involving different types of LSAs), a similar approach to the one used within an area can be used here. We need to compute and maintain broadcast trees for the upper layer of the hierarchy, and to make the HFTB-related modifications (FT-bit in the router LSAs) to the other types of LSAs (e.g., summary LSAs) that are exchanged between areas. The rest is similar to what was described before. In other words, an area border router will flood the first summary LSA within the 30 min interval so that each node can establish the broadcast trees for upper layer areas. After the flooding of the first summary LSA, all the nodes within the network would use the broadcast trees to disseminate the subsequent summary LSAs throughout the network. Note that the broadcast trees established in for upper layer would be consistent with the ones created within the area because intra-area routing has precedence over inter-area routing. We are currently in the process of making these modifications to the source code of the OSPF.

6. Conclusions and future work

We provided a hybrid state dissemination mechanism that combines flooding and tree-based broadcasting (HFTB) to achieve simple yet reliable and efficient link-state dissemination. Such a mechanism is particularly needed in the context of QoS routing, which involves frequent dissemination of several dynamic parameters. In contrast to previous tree-based broadcasting approaches, which require complex algorithms and protocols to determine and maintain the broadcast trees, the proposed HFTB simply determines the broadcast trees by flooding first-time LSAs. Subsequent LSAs that update the QoS-related state of an existing link(s) are then disseminated over the broadcast trees. To deal with link-failures, we provided a modified version of HFTB, called HFTB-RT that can repair disconnected trees. We described how to integrate the proposed hybrid mechanisms into OSPF. For this purpose, two of the currently unused bits of the Options field in the LSA header are defined as FT-bit and RT-bit. Using these bits, we slightly modified the flooding procedure of OSPF to determine the broadcast trees and to disseminate LSAs over these trees. Accordingly, we implemented HFTB and HFTB-RT by modifying the existing source code of OSPF. We compared the proposed mechanisms with the flooding using numerical and simulation results. If there are no link failures, HFTB-RT has the same performance of HFTB. Otherwise, it incurs some extra overhead over HFTB. However, by using HFTB-RT, nodes dynamically repair disconnected broadcast trees in the case of link failures and acquire the most recent LSAs in a simple and efficient manner. In spite of this overhead, the HFTB-RT provides significantly better performance than currently used flooding while maintaining the simplicity and reliability of flooding under reasonable number of link failures. Moreover, in simulation we observed that, the flooding procedure of OSPF sends the same LSAs over some links on both directions and counts them as implicit ACKs. Since this results in having more LSAs than ACKs in the network, the TSLA of the flooding excessively increases in practice. However, the proposed tree-based hybrid approach avoids implicit ACKs and thus provides much better TSLA performance than the flooding in practice.

As a future work, we plan to implement the proposed hybrid mechanism under the hierarchical structure of OSPF while investigating how to further simplify it and deploy in an incremental manner. We also study the stability and performance of QoS-based path selection algorithms under QoS-enhanced OSPF using the proposed mechanism. Finally, we plan to investigate better analytical models that can give good accuracy when compared with simulations.

Appendix A. OSPF-based implementation of HFTB(-RT)

Both HFTB and HFTB-RT have been implemented by modifying the OSPF source code provided in [20]. We now explain the main features of our implementation. A detailed description of this implementation along with the source code can be found at www.cs.utsa.edu/~korkmaz/research/hftb. Implementation wise, HFTB can be regarded as a special case of HFTB-RT, so we only describe the implementation of HFTB-RF.

As indicated before, integrating HFTB-RT into OSPF requires making three main modifications to OSPF: (1) redefining two of the unused bits of the Options field of the LSA header, (2) creating and maintaining a parent–children table at every node, and (3) adding and changing some steps in the OSPF algorithm at a node.

A.1. Redefining unused bits

The header of an OSPF LSA has several unused bits in the Options field. We designated the two most significant bits of this field as FT-bit and RT-bit, respectively. To access/check/set these bits later, we defined the following masks in hexadecimal format:

```
SPO_FT = 0x80
SPO_RT = 0x40
```

These masks are included in `spfpkt.h`, where all other bits are defined.

A.2. Creating and maintaining a parent–children table

In Fig. 3, we described the parent–children table as a two-dimensional array. In practice, the number of nodes and the number of neighbors are not known in advance, so we need a dynamic data structure to maintain the parent–children relationships as they become realized. To provide a proof-of-concept, we simply used a linked list implementation. However, one can use binary trees or hash tables to improve the efficiency in accessing the entries of the parent–children table.

As shown in Fig. 3, each entry of the parent–children table consists of three components: origin, parent, and a list of children. To maintain such entries using a linked list, we first defined the following classes in a new file called `pctable.h`.

```
class PCTable{
  public:
    rtid_t origin;
    SpfNbr *parent;
    Children *childlist;
    PCTable *next;
    /* some flags that are used in
    HFTB-RT */
    int need_flood;
    seq_t repair_seqno;
    int repaired;
    PCTable(); /* Constructor to
    initialize the parent–children
    table */
};

class Children{
  public:
    SpfNbr *child;
    Children *next;
    Children(){/* Constructor to
    initialize the children linked
    list */
      child = 0;
      next = 0;
    }
  friend class PCTable;
};
```

As shown above, the `PCTable` class also contains few other flags for each entry, which are used to repair trees in HFTB-RT. To incorporate the parent–children table into OSPF, we included the following into the original `OSPF class` whose declaration is given in `ospf.h`: (a) a pointer to the parent–children table, and (b) new functions to update or access the entries of the parent–children table:

```
class OSPF{
  ...
  PCTable *pctable_head;
  ...
  void add_parent_pctable(rtid_t
  orig, SpfNbr *par);
    /* Origin and parent nodes are
      given. If origin does not ex-
      ist, it is created and parent
      is added. Otherwise, the
      given parent is added to the
      existing origin. */

  void add_child_pctable(rtid_t
  orig, SpfNbr *ch);
    /* Origin and the child node are
      given. If the child is not
      already present, it is added to
      the children list of the
      origin. Otherwise, it is not
      added. */

  void delete_child_pctable(rtid_t
  orig, SpfNbr *ch);
    /* Origin and the child node are
      given. If the child is not
      present, it is not deleted.
      Otherwise, it is deleted
      from the children list of the
      origin. */

  bool is_child_pctable(rtid_t
  orig, SpfNbr *ch);
    /* Origin and child node are
      given. Returns true if the
      given child node is present in
      the origin's children list.
      Otherwise, returns false. */
```

```
SpfNbr *get_parent_pctable(rtid_t
orig);
  /* Given the origin, the parent
    node corresponding to it
    is returned. */

/* The following four functions
   are used in HFTB-RT to set or
   check the values of the flags
   need_flood and repaired that are
   needed to deal with link failures
   and disconnected trees */
void set_flood_pctable(SpfNbr *ch);
bool need_flood_pctable(rtid_t
orig);
void set_tree_repair_pctable
(rtid_t orig, seq_t seqno);
bool need_tree_repair_pctable
(rtid_t orig, seq_t seqno);
```

In the standard OSPF implementation, the above functions are given in `ospf.C`. In our implementation, we put them in the newly created file `pctable.C`.

A.3. Modifying OSPF procedures

The required modifications to OSPF procedures were outlined in Fig. 11. We now describe how such modifications were implemented. For ease of exposition, we follow the same presentation order of Fig. 11.

A.3.1. Upon becoming operational

The constructor of the `OSPF class` which is in `ospf.C` is called. We included the followings into that constructor: (a) the parent–children table is initialized as an empty list (i.e., `pctable_head` = `0`); (b) a flag named `hftb` defined in `ospf.h` is set to 0, indicating that the node is initially in flooding mode.

A.3.2. Repeat every 30 min

The `refresh_lsas()` function of `OSPF class` which is in `ospf.C` is called. This function periodically generates a new LSA describing the state of its outgoing links. In this function, before generating a new LSA, we reset the `hftb` flag (i.e., set it to 0), making the node go into the flooding mode every 30 min.

In addition to the periodic generation of LSAs, our implementation allows for LSAs to be generated upon a trigger. This feature was implemented in the new function `hftb_lsas()` in `dbage.C` by modifying `refresh_lsas()` to generate trigger-based LSAs.

A.3.3. Upon the failure of a link

The destructor of `SpfIfc class` which is in `spfifc.C` is called. We added new statements into that destructor to set the `need_flood` flag in the parent–children table for every originator whose tree is disconnected as a result of that link failure. This flag is later used in the `flood()` function to repair trees by setting the RT-bit of a received LSA to 1 and flooding that LSA.

A.3.4. Upon generating a new LSA

The `lsa_reorig()` function in `spforig.C` is called. If the node is in the flooding mode (i.e., `hftb` = `0`), then the FT-bit is set to 0 for that LSA while it is being built in the function `rl_orig()` (which is in the file `rtrlsa.C`). After flooding the LSA to all the neighbors in the `flood()` function, we set the `hftb` flag is to 1, indicating that the node is switching into the tree-based broadcasting mode for the subsequent LSAs. If the node is already in the tree-based broadcasting mode (i.e., `hftb` = `1`), then the FT-bit of the generated LSA is set to 1, again while it is being built in `rl_orig()`. The LSA is then sent to all the nodes in the children list using the `flood()` function. To avoid sending the LSA to non-children nodes, we simply added few if-statements in the `flood()` function in `spflood.C`.

A.3.5. Upon receiving via node *j* an LSA that originated from node *u*

The `recv_update()` function in `spflood.C` is called. If the received LSA is the most recent one, this function updates the link-state database and calls the `flood()` function, in which the following changes are made. Check the options field of the LSA. Then

1. if FT = 0 and RT = 0,
   Node *j* is added as a parent to the originator *u* in the parent–children table using `add_parent_pctable( )`. The LSA is flooded to all neighbors except node *j*. An acknowledgement is sent to node *j*, setting the FT-bit to 1.
2. if FT = 0 and RT = 1,
   The received LSA is compared against the database copy using the function `cmp_opts( )`. This function is added to the file `spflood.C`.

```
/* In flooding mode, if the receiv-
ing node is chosen to be the parent
  send an ACK to it with FT-bit set
  to 1*/
int LSA::cmp_opts(LShdr *hdr){
  LShdr *dbcopy;
  /* Create network-ready version
  of database copy */
  dbcopy = ospf->BuildLSA(this);
  if ((dbcopy->ls_seqno = = hdr->
  ls_seqno) && (dbcopy->ls_opts
  & 128) = = 128){
    return 0;
  }
  else{ /* LSA most recent; add to
  DB */
    return 1;
  }
}
```

   If the node has a database copy with FT = 1 and RT = 0, then an acknowledgement is sent to node *j* with the FT-bit set to 0. The LSA is sent to all the neighbors except node *j*.
   Otherwise, node *j* is added as a parent to the originator *u* in the parent–children table using `add_parent_pctable( )`. An acknowledgement is sent to node *j* with the FT-bit set to 1. The LSA is sent to all the neighbors except node *j*.
3. if FT = 1,
   If the flag `need_flood` corresponding to originator *u* indicates that the broadcast tree is disconnected (i.e., `need_flood` is set to 1), then the LSA's FT-bit is set to 0 and the RT-bit is set to 1. The flag `need_flood` in the parent–children table is set to 0 to indicate that the disconnected tree has been repaired. The LSA is sent to all neighbors including node *j*. An acknowledgement is sent to node *j* with FT-bit equals to 1.
   Otherwise, the LSA is sent to all the nodes in the children list. An acknowledgement is sent to node *j* with the FT-bit equals to 1.

A.3.6. Upon receiving an acknowledgement from node j

The `recv_ack( )` function in `spfack.C is called`. In this function, we made the following changes:

(a) if the FT-bit in the received LSA's options field is 1, then add node *j* to the children list of the received LSA's originator using the function `add_child_pctable( );`
(b) if the FT-bit in the received LSA's Options field is 0, then delete node *j* from the children list of the received LSA's originator. Deleting the child is done by using the function `delete_child_pctable( )`.

References

[1] G. Apostolopoulos, R. Guerin, S. Kamat, S.K. Tripathi, Quality of service based routing: A performance perspective, in: Proceedings of the ACM SIGCOMM '98 Conference, Vancouver, British Columbia, Canada, August–September 1998, pp. 17–28.

[2] G. Apostolopoulos, D. Williams, S. Kamat, A. Guerin, R. Orda, T. Przygienda, QoS routing mechanisms and OSPF extensions. RFC 2676, IETF, August 1999.

[3] E. Basturk, P. Stirpe, A hybrid spanning tree algorithm for efficient topology distribution in PNNI, in: Proceedings of the 1st IEEE International Conference on ATM (ICATM '98), 1998, pp. 385–394.

[4] B. Bellur, R.G. Ogier, A reliable, efficient topology broadcast protocol for dynamic networks, in: Proceedings of the INFOCOM '99 Conference, vol. 1, IEEE, New York, 1999, pp. 178–186.

[5] K. Carlberg, J. Crowcroft, Building shared trees using a one-to-many joining mechanism, ACM Computer Communication Review 27 (1) (1997) 5–11.

[6] R. Castaneda, S.R. Das, Query localization techniques for on-demand routing protocols in ad hoc networks, in: Proceedings of the Fifth International Conference on Mobile Computing and Networking (MobiCom'99), August 1999, pp. 186–194.

[7] C.-C. Chiang, H.-K. Wu, W. Liu, M. Gerla, Routing in clustered multihop mobile wireless networks with fading channel, in: The IEEE Singapore International Conference on Networks (SICON), April 1997, pp. 197–211.
[8] R. Coltun, The OSPF opaque LSA option, Technical Report RFC 2370, IETF, July 1998.
[9] D.E. Comer, Internetworking with TCP/IP, vol. 1, third ed., Prentice-Hall, Englewood Cliffs, NJ, 1995.
[10] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, sixteenth ed., The MIT Press and McGraw-Hill, Cambridge, MA, 1996.
[11] Y.K. Dalal, R.M. Metcalfe, Reverse path forwarding of broadcast packets, Communications of the ACM 21 (1978) 1040–1048.
[12] C. Diot, W. Dabbous, J. Crowcroft, Multipoint communication: a survey of protocols, functions, and mechanisms, IEEE Journal on Selected Areas in Communications 15 (3) (1997) 277–290.
[13] Z.J. Haas, M.R. Pearlman, The zone routing protocol (ZRP) for ad-hoc networks, Technical report, Internet-draft, IETF MANET Working Group, July 2002.
[14] P.A. Humblet, S.R. Soloway, Topology broadcast algorithms, Computer Networks and ISDN Systems 16 (1989) 179–186.
[15] D.B. Johnson, D.A. Maltz, Y.-C. Hu, The dynamic source routing protocol for mobile ad hoc networks, Technical report, Internet-draft, IETF MANET Working Group, February 2003.
[16] Y.-B. Ko, N.H. Vaidya, Location-aided routing (LAR) in mobile ad hoc networks, in: Proceedings of the ACM/IEEE MobiCom '99 Conference, November 1998, pp. 66–75.
[17] V.O.K. Li, Z. Zhang, Internet multicast routing and transport control protocols, Proceedings of the IEEE 90 (3) (2002) 360–391.
[18] J. Moy, OSPF version 2. Standards Track RFC 2328, IETF, April 1998.
[19] J.T. Moy, OSPF: Anotomy of an Internet Routing Protocol, Addison Wesley, Reading, MA, 1998.
[20] J.T. Moy, OSPF: Complete Implementation (with CD-ROM), Addison Wesley, Reading, MA, 2000.
[21] S. Murthy, J.J. Garcia-Luna-Aceves, An efficient routing protocol for wireless networks, Mobile Networks and Applications 1 (2) (1996) 183–197.
[22] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, J.-P. Sheu, The broadcast storm problem in a mobile ad hoc network, in: Proceedings of the Fifth International Conference on Mobile Computing and Networking (MobiCom'99), August 1999, pp. 151–162.
[23] V. Park, S. Corson, Temporally-ordered routing algorithm (tora) version 1 functional specification, Technical Report, Internet-draft, IETF MANET Working Group, February 2003.
[24] C.E. Perkins, E. Belding-Royer, S. Das, Ad hoc on-demand distance vector routing, Technical Report, Network Working Group, RFC 3561, July 2003.
[25] C.E. Perkins, P. Bhagwat, Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers, ACM SIGCOMM Computer Communication Review 24 (4) (1994) 234–244.
[26] L.H. Sahasrabuddhe, B. Mukherjee, Multicast routing algorithms and protocols: a tutorial, IEEE Network 14 (1) (2000) 90–102.

Turgay Korkmaz received the B.Sc. degree with the first ranking from Computer Science and Engineering at Hacettepe University, Ankara, Turkey, in 1994, and two M.Sc. degrees from Computer Engineering at Bilkent University, Ankara, and Computer and Information Science at Syracuse University, Syracuse, NY, in 1996 and 1997, respectively. He received his Ph.D. degree from Electrical and Computer Engineering at University of Arizona, Tucson, AZ, in December 2001. In January 2002, he joined the University of Texas at San Antonio, where he is currently an Assistant Professor of Computer Science department. His research interests include QoS-based routing, multiple constrained path selection, efficient dissemination of network-state information, topology aggregation in hierarchical networks, and performance evaluation of QoS-based routing protocols. He is a Co-PI on the NSF High Performance Network Connections (HPNC) Award to provide Internet 2 Connectivity for UTHSCSA and UTSA. He was the co-chair for the ACM Symposium on Applied Computing (SAC 2003), Special Track on Parallel and Distributed Systems and Networking and the SAC 2004 Special Track on Computer networks. He also served on the technical program committee of IEEE INFOCOM 2004.

Marwan Krunz is an associate professor of Electrical and Computer Engineering at the University of Arizona. His research interests lie in the field of computer networks, especially in its performance and traffic control aspects. His recent work has focused on the provisioning of quality of service (QoS) over wireless links, QoS routing, traffic modeling, bandwidth allocation, video-on-demand systems, and power-aware protocols for ad hoc networks. He has published more than 70 journal articles and refereed conference papers in these areas. He is a recipient of the National Science Foundation CAREER Award (1998–2002). He currently serves on the editorial board for the IEEE/ACM Transactions on Networking and the Computer Communications Journal. He was a Guest Co-editor for a Feature Topic on QoS Routing (IEEE Communications, June 2001) and a Special Issue on Hot Interconnects (IEEE Micro, January 2002). He is the Technical Program Co-chair for the IEEE INFOCOM 2004 Conference (Hong Kong, March 7–11, 2004), and previously served as the Technical Program Co-chair for the 9th Hot Interconnects Symposium (Stanford University, August 2001). He has served and continues to serve on the executive and technical program committees of many international conferences. He serves as a reviewer and a panelist for NSF proposals, and is a consultant for several corporations in the telecommunications industry.

Jyothi Guntaka received Bachelor of Technology degree with a distinction in Computer Science and Engineering from Andhra University, Visakhapatnam, India. She received her M.S. in Computer Science from The University of Texas at San Antonio, San Antonio, Texas. Her research interests include Quality of Service, Congestion Control and Security in computer networks.