Scalable, Distributed Cycle-Breaking Algorithms for Gigabit Ethernet Backbones

Francesco De Pellegrini

Create-Net Research Center, Trento, Italy
and
Department of Information Engineering, University of Padova
depe@dei.unipd.it

David Starobinski, Mark G. Karpovsky, and Lev B. Levitin

Department of Electrical and Computer Engineering at Boston University
{*staro,markkar,levitin*}*@bu.edu*

Ethernet networks rely on the so-called spanning tree protocol (IEEE 802.1d) in order to break cycles, thereby avoiding the possibility of infinitely circulating packets and deadlocks. This protocol imposes a severe penalty on the performance and scalability of large Gigabit Ethernet backbones, since it makes inefficient use of fibers and may lead to bottlenecks. In this paper, we propose a significantly more scalable cycle-breaking approach, based on the novel theory of turn-prohibition. Specifically, we introduce, analyze and evaluate a new algorithm, called Tree-Based Turn-Prohibition (TBTP). We show that this polynomial-time algorithm maintains backward-compatibility with the IEEE 802.1d standard and never prohibits more than 1/2 of the turns in the network, for any given graph and any given spanning tree. We further introduce a distributed version of the algorithm that nodes in the network can run asynchronously. Through extensive simulations on a variety of graph topologies, we show that the TBTP algorithm can lead to an order of magnitude improvement over the spanning tree protocol with respect to throughput and end-of-end delay metrics. In addition, we propose and evaluate heuristics to determine the replacement order of legacy switches that results in the fastest performance improvement. © 2005 Optical Society of America

OCIS codes: 060.4250, 060.4510.

1. Introduction

For many years, Ethernet has been the prevalent local area network (LAN) technology, offering a wide-range of services in a simple and cost-effective manner. With the standardization of Gigabit Ethernet protocols, the scope of Ethernet has widened further [1]. A large number of corporations and service providers are now adopting Gigabit Ethernet as their backbone technology. Gigabit Ethernet backbones cope with the increasing traffic demand resulting from the deployment of high-speed LANs, home networks, Voice over IP, and high-bandwidth applications. The key advantage of Gigabit Ethernet over alternate technologies, such as ATM, is to maintain backward-compatibility with the over one hundred millions Ethernet nodes deployed world-wide and the large number of applications running on these nodes [2, 3].

Gigabit Ethernet has the same plug-and-play functionalities as its Ethernet (10 Mb/s) and Fast Ethernet (100 Mb/s) precursors, requiring minimal manual intervention for connecting hosts to the network. In addition, Gigabit Ethernet relies on full-duplex technologies and on a flow control (backpressure) mechanism that significantly reduce the amount of congestion and packet loss in the network [1, 4]. More specifically, the flow control mechanism (IEEE 802.3x) prevents switches from loosing packets due to buffer overflow. This protocol makes use of *Pause* messages, whereby a congested receiver can ask the transmitter to suspend (pause) its transmissions. Each Pause message includes a timer value that specifies how long the transmitter needs to remain quiet.

Currently, the network topology for Gigabit Ethernet follows the traditional rules of Ethernet. The spanning tree protocol (IEEE 802.1d) is used to avoid the occurrence of any cycle in the networks, thus pruning the network into a tree topology [5].

The reasons for breaking cycles are two-fold. The first is to avoid broadcast packets (or packets with unknown destination) from circulating forever in the network. Unlike IP, Ethernet packets do not have a Time-to-Live (TTL) field. Moreover, Ethernet switches must be transparent, which means that they are not allowed to modify headers of Ethernet packets.

The second reason is to prevent the occurrence of deadlocks as a result of the IEEE 802.3x flow control mechanism [6]. Such deadlocks may occur when Pause messages are sent from one switch to another along a circular path, leading to a situation where no switch is allowed to transmit. The use of a spanning tree precludes this problem, since deadlocks cannot arise in an acyclic network [7].

The spanning tree protocol works well in LAN networks, which are often organized hierarchically and under-utilized [8]. However, it imposes a severe penalty on the performance and scalability of large Gigabit Ethernet backbones, since a spanning tree allows the use of only one cycle-free path in the entire network. As pointed out by the Metro Ethernet Forum, an industry-wide initiative promoting the use of optical Ethernet in metropolitan area networks, this leads to inefficient utilization of expensive fiber links and may result in uneven load distribution and bottlenecks, especially close to the root [9, 10].

One of the current approaches to address this issue is to overlay the physical network with logical networks, referred to as virtual LANs [5]. A spanning tree instance is then run separately for each virtual LAN (or group of virtual LANs). This approach of maintaining multiple spanning trees can add significant complexity to network management and be very CPU-intensive [9]. Other approaches based on multiple spanning trees are described in [11, 12].

In this paper, we propose a significantly more scalable approach, based on the novel theory of *turn-prohibition* [13, 14], in order to solve the cycle-breaking problem in Gigabit Ethernet backbones. Turn-prohibition is much less restrictive than link-prohibition, the approach employed to construct a spanning tree. The main idea is to consider pairs of links around nodes, referred to as turns [15], and show that all the cycles in a network can be broken through the prohibition of carefully selected turns in the network (a turn (a, b, c) around node b is prohibited if no packet can be forwarded from link (a, b) to link (b, c)).

One of the main challenges in making use of the turn-prohibition approach is to maintain backward-compatibility with the IEEE 802.1d standard. Our main contribution in this paper is to propose and analyze a novel algorithm, called Tree-Based Turn-Prohibition (TBTP), that addresses this issue. This algorithm receives a graph along with a spanning tree, as its input, and generates a set of prohibited turns, as its output.

The TBTP algorithm possesses several key theoretical properties. First, it breaks all the cycles in the networks and preserves connectivity. Second, it *never* prohibits turns along the given spanning tree. In particular, if the tree is generated by the IEEE 802.1d protocol, then legacy switches can gradually be replaced by switches capable of running turn-prohibition. Third, the algorithm prohibits at most 1/2 of the turns in the network. Thus, the total number of permitted turns in the network always exceeds the total number of prohibited turns. This result is valid for *any* given graph and spanning tree on it. Furthermore, it is generalizable to weighted graph topologies. In this latter case, the algorithm guarantees that the total weight of permitted turns always exceeds the total weight of prohibited turns in the network. We note that the constraint of permitting all the turns along a given spanning tree is critical for backward-compatibility. Without this constraint, a tighter bound on the fraction of prohibited turns can been achieved, namely 1/3 [13, 14]. Finally, we show that the TBTP algorithm can be implemented in a distributed fashion and the number of messages exchanged on each link is independent of the network size (assuming that the maximum degree of the graph is fixed).

The rest of this paper is organized as follows. In Section 2, we give a brief overview of the IEEE 802.1d protocol and discuss related work. In Section 3, we introduce the TBTP algorithm, prove its main properties, and analyze its worst-case time complexity. In Section 4, we present a distributed version of the TBTP algorithm. In Section 5, we provide a general framework for maintaining backward-compatibility in an heterogeneous network composed of both “intelligent” switches, capable of running turn-prohibition, and legacy switches. We also propose heuristics to determine the order in which legacy switches should be replaced in order to achieve the fastest performance improvement. In Section 6, we present numerical results, comparing the TBTP algorithm with the standard spanning tree algorithm and an earlier turn-prohibition approach called Up/Down [16]. Through extensive simulations on a variety of graph topologies, we show that the TBTP algorithm significantly outperforms the two other schemes with respect to throughput and end-to-end delay metrics. The last section is devoted to concluding remarks.

2. Background and Related Work

2.A. The Spanning Tree Algorithm (IEEE 802.1d)

The spanning tree algorithm, first proposed in the seminal paper of [17], is the standard for interconnecting LANs, according to the IEEE 802.1d protocol.

This algorithm requires a unique identifier (ID) for every switch and every port within a switch. Using a distributed procedure, it elects the switch with the smallest ID as the root. A spanning tree is then constructed, based on the shortest path from each switch to the root (switch and port IDs are used to break ties).

Every switch brings the ports connected to its parent and children into a *forwarding* state. All remaining ports are placed in a *blocking* state. Ports in the forwarding state are used to forward data frames, while ports in the blocking state can only be used for forwarding signaling messages between switches.

Packet forwarding is based on a *backward–learning* process. When a switch receives a packet from a certain host S, via one of its active ports P, it assumes that the same port can be used in the reverse direction to forward packets to host S. This way, each switch progressively constructs a table, maintained in a local cache, that maps each destination with its corresponding port in the switch. Note that cache entries are valid only for a limited amount of time, determined by an associated timer. If the destination of a packet is unknown, then the packet is forwarded over all active ports except the incoming one. This action is commonly referred to as *flooding* or *broadcast*.

Newer versions of the spanning tree protocol include the rapid spanning tree protocol (RSTP) (IEEE 802.1w) that increases the speed of convergence of the original algorithm. RSTP takes advantage of the presence of point-to-point links and is tailored for optical backbones. Doing so, the convergence speed scales down from the order of tens of seconds to the order of hundreds of milliseconds. Recent work, however, indicate that RSTP may exhibit undesirable count-to-infinity behavior [18].

2.B. Improvements

Several enhancements have been proposed in the literature in order to mitigate congestion near the root of the tree.

The Distributed Load Sharing (DLS) technique enables use of some of the links not belonging to the tree [19, 20]. Under specific topological constraints, this technique can provide alternate paths to the spanning tree, thus helping to relieve local congestion. In general, however, no guarantee is provided on the overall performance improvement of the network.

A recent improvement over the DLS idea is proposed in [21]. This work devises a backward-compatible scheme, called STAR (Spanning Tree Alternate Routing), that finds alternate paths shorter than paths on the spanning tree, for any additive metric. It also introduces optimization techniques to determine which legacy switches should be candidates for replacement by new switches. As with previous DLS solutions, the STAR scheme does not provide bounds on the amount of prohibited resources in the network and does not address the problem of deadlocks caused by the IEEE 802.3x protocol. Readers are referred to [21] for other previous work related to the STAR protocol.

In [22], a solution, based on the technique of diffusing computation, is proposed in order to avoid infinite packet loops. Unfortunately, this solution is not backward-compatible with the IEEE 802.1d protocol and does not address the issue of deadlocks.

A scheduling approach, suitable for a lossless Gigabit Ethernet LAN, is proposed in [6] in order to avoid deadlocks. Although this solution avoids changes in the headers of Ethernet frames, it requires the replacement of all the switches in the network and is inherently incompatible with the spanning tree approach.

Our contribution substantially differs from previous work by providing a provable bound on the amount of prohibited resources (turns) in the network. Moreover, the TBTP algorithm that we propose is a general graph-theoretic approach for breaking cycles in networks. This algorithm is, thus, application-independent and can be used to avoid both infinite packet loops and deadlocks. Finally, our proposed algorithm is purposefully designed to be backward-compatible, as it relies on the spanning tree generated by the IEEE 802.1d protocol.

A preliminary version of this paper appeared in [23]. The main addition of the current paper is in the introduction, analysis, and performance evaluation of the distributed version of the TBTP algorithm. This algorithm differs from previous distributed turn-prohibition algorithms [16, 24] by providing bounds on the fraction of prohibited turns and guaranteeing backward-compatibility.

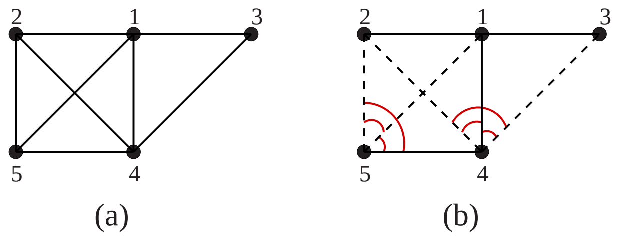


Fig. 1. (a) Example of a graph G. (b) Solution of the Up/Down algorithm for graph G. Tree-links are represented by solid lines and cross-links by dashed lines. Arcs represent prohibited turns (six turns are prohibited in total).

3. Scalable Cycle-Breaking Algorithms

In this section, we present the Tree-Based Turn-Prohibition (TBTP) algorithm for breaking cycles in a scalable manner in Gigabit Ethernet Networks. We also briefly review an earlier turn-prohibition algorithm called *Up/Down* [16] and discuss the theoretical advantages of TBTP over that algorithm. Beforehand, we introduce our network model and notations, and provide a formal statement of the problem.

3.A. Model

We model a Gigabit Ethernet network by a directed graph $G(V, E)$ where V is a set of nodes (vertices) representing switches and E is a set of links (edges). We do not consider end-hosts, since they would just be leaves on the graph. We restrict our attention to *bi-directional* network topologies, that is, networks where nodes are connected by full-duplex links. It is worth noting that essentially all modern Gigabit Ethernet networks make use of full-duplex links (in contrast to the original Ethernet where nodes were communicating over a shared medium link).

We define a *cycle* to be a path whose first and last links are the same, for instance, $(n_1, n_2, n_3, \ldots, n_{\ell-1}, n_\ell, n_1, n_2)$. Our goal is to break all such cycles in the underlying graph in order to avoid deadlocks and infinitely circulating packets.

Note that the literature in graph-theory typically defines a cycle as a path such that the initial and final nodes in the path are the same [25]. We refer to this latter definition as a *cycle of nodes*. A spanning tree breaks all cycles of nodes in a graph.

Breaking all cycles of nodes is, however, unnecessary in general. For instance, referring to Figure 1(a), the path $(5, 1, 4, 3, 1, 4)$ contains a cycle, while the path $(5, 1, 4, 3, 1, 2)$ does not (although it contains a cycle of nodes). A cycle is, thus, created only when the same buffer or port is traversed twice, in the same direction. In particular, a path may traverse several times the same node without creating a cycle.

A pair of input-output links around a node is called a *turn*. The three-tuple (i, j, k) will be used to represent a turn from link (i, j) to link (j, k), with $i \neq k$. For instance, in Fig. 1(a), the three-tuple $(2, 1, 3)$ represent the turn from link $(2, 1)$ to link $(1, 3)$ around node 1.

Due to the symmetrical nature of bi-directional graphs, we will consider the turns (i, j, k) and (k, j, i) to be the same turn. As shown in the sequel, an efficient approach for breaking cycles in a network is based on the *prohibition* of turns. For example, in Fig. 1(a), prohibiting the turn $(2, 1, 3)$ means that no packet can be forwarded from link $(2, 1)$ to link $(1, 3)$ (and from link $(3, 1)$ to link $(1, 2)$). Note that turns involving leaves in the graph can always be permitted, since they do not belong to any cycle. We will denote by $R(G)$ the set of all turns in the network. If the degree of each node i is d_i, then $|R(G)| = \sum_{i=1}^{|V|} d_i(d_i - 1)/2$.

3.B. Statement of the Problem

Suppose a spanning tree $T(G) = (V_T, E_T)$ providing connectivity for a graph G is given *a-priori*. We refer to links belonging to this tree as *tree-links*. Other links are referred to as *cross-links*.

Denote a set of prohibited turns by $S_T(G)$. Our goal is to determine a set $S_T(G)$, such that

1. Every cycle in the network contains at least one turn from $S_T(G)$, i.e., all the cycles in the network are broken.
2. $S_T(G)$ contains a minimum number of elements.
3. All the turns between tree-links in $T(G)$ are permitted.

Our problem, therefore, is to determine a minimal cycle-breaking set $S_T(G)$ that does not include turns from $T(G)$. If the tree is generated by the IEEE 802.1d protocol, then the solution to this problem would allow us to gradually replace legacy switches by switches capable of running turn-prohibition. We present, now, two algorithms to address this problem. For simplicity of exposition, we first assume that all nodes in the network are *intelligent* nodes, that is, nodes capable of implementing turn-prohibition. We address backward-compatibility issues with legacy nodes in Section 5.

3.C. Up/Down

A possible approach for the construction of a cycle-breaking set $S_T(G)$, is the so-called *up/down routing* algorithm [16]. In this approach, a spanning tree $T(G)$ is first constructed. Then, nodes are ordered according to the level at which they are located on the tree. The level of a node is defined at its distance (in number of hops) from the root. Nodes at the same level are ordered arbitrarily.

Once the nodes are ordered, a link (i, j) is considered to go "up" if $i > j$. Otherwise, it is said to go "down". A turn (i, j, k) is referred to as an up/down turn if node if $i > j$ and $j < k$. Respectively, a down/up turn is a turn such that $i < j$ and $j > k$. Clearly, any cycle must involve at least one up/down turn and one down/up turn. Therefore, it is possible to break all the cycles in a graph by prohibiting all the down/up turns. This means that packets can be transmitted over cross-links as long as they are not forwarded over down/up turns. An illustration of the Up/Down algorithm is provided in Fig. 1(b).

The Up/Down routing algorithm achieves much higher performance than the simple spanning tree approach implemented by the IEEE 802.1d protocol, as shown by our simulations in Section 6. However, the performance of this algorithm depends critically on the selection of the spanning tree and its root node. In particular, it has been shown that the fraction of turns prohibited by this scheme, that is, the ratio $|S_T(G)|/|R(G)|$, can be arbitrarily close to 1 in some networks [13].

3.D. The Tree-Based Turn-Prohibition (TBTP) Algorithm

We now introduce a novel cycle-breaking algorithm, called Tree-Based Turn-Prohibition (TBTP). This algorithm prohibits at most 1/2 of the turns for *any* given graph G and spanning tree $T(G)$. This is in contrast with the Up/Down algorithm that does not provide any guarantees on the fraction of prohibited turns in the network.

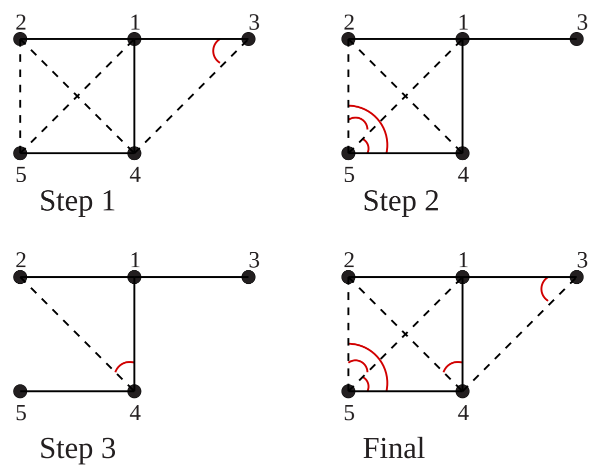


Fig. 2. Successive steps of the TBTP algorithm and final result (five turns are prohibited in total).

3.D.1. The Algorithm

The TBTP algorithm is a greedy procedure that receives a graph G and an associated spanning tree $T(G)$ as its arguments. At each iteration, the algorithm breaks all the cycles involving some selected node i^*. Towards this end, the algorithm prohibits all the turns around node i^* (except for turns between tree-links) and permits all turns of the type (i^*, j, k), where (i^*, j) are cross-links originating from node i^*. All the cross-links connected to node i^* are then deleted and the procedure is repeated until all the nodes have been selected and no more cross-links remain. Following is a pseudo-code for the algorithm:

Algorithm TBTP$(G, T(G))$:

1. For each node i, that has adjacent cross-link(s), construct two sets of turns

$$\begin{aligned} A(i) &= \{(i,j,k)|(i,j) \in E \backslash E_T, (j,k) \in E\} \\ P(i) &= \{(j,i,k)|(j,i) \in E \backslash E_T, (i,k) \in E\} \end{aligned}$$

2. Select node i^* maximizing $|A(i)| - |P(i)|$ (if there are multiple such nodes, choose one at random).

3. Add all the turns from $P(i^*)$ into $S_T(G)$.

4. Delete all cross-links $(i^*, j) \in E \backslash E_T$, and repeat the procedure until all the cross-links have been deleted.

The intuition behind the node selection is as follows. Each node i, is associated with a potential set of permitted turns $A(i)$ and a potential set of prohibited turns $P(i)$. At each iteration, the selected node is the one that maximizes the difference between the cardinality of the sets, namely $|A(i)| - |P(i)|$. We will show in the sequel that there always exists at least one node for which this difference is greater or equal to zero. Thus, this selection strategy guarantees that the algorithm prohibits at most 1/2 of the turns in the graph.

Figure 2 illustrates the algorithm. At the first step of the algorithm, the sets $A(i)$ and $P(i)$ are computed for each node i. For instance, for node 1, these sets are $A(1) = \{(1,5,2),(1,5,4)\}$ and $P(1) = \{(2,1,5),(3,1,5),(4,1,5)\}$. The selected node is node 3 for which $|A(3)| - |P(3)| = 2$ (either node 2 or 5 could have been selected as well). As a result, turn $(1, 3, 4)$ is prohibited. The procedure is then repeated, but without cross-link $(3, 4)$. As shown in the figure, the procedure continues until no more cross-links remain. The final set of prohibited turns is $S_T(G) = \{(1, 3, 4), (1, 4, 2), (1, 5, 2), (1, 5, 4), (2, 5, 4)\}$ (other solutions are also possible). Thus, the algorithm ends up in prohibiting five turns which is one fewer than Up/Down.

3.D.2. Analysis

We now prove the main properties of the TBTP algorithm.

Theorem 1 *The TBTP algorithm preserves network connectivity.*

Proof

The algorithm never prohibits turns between tree-links. Thus, network connectivity is provided by the spanning tree $T(G)$.

Theorem 2 *The TBTP algorithm breaks all the cycles.*

Proof

If any cycle exists, then it must involve at least one cross-link. Thus, in order to prove the theorem, we need to show that a turn containing a cross-link cannot belong to any cycle.

The proof is by induction. The induction hypothesis is that each iteration of the algorithm, none of the turns around nodes previously selected, and containing a cross-link, belong to any cycle. This hypothesis is clearly true for the first node selected, say node i_1. This is because the TBTP algorithm prohibits all the turns of the type (j, i_1, k) around node i_1, where (j, i_1) are cross-links.

Now suppose that $n-1$ nodes have already been selected and none of the turns, containing a cross-link, around these nodes belong to any cycle. The next chosen node is, say, node i_n. We distinguish between two types of turns around node i_n. First, we consider turns which contain a cross-link adjacent to one of the previously selected nodes. Some of these turns may have been permitted in one of the former steps of the algorithm, but can not lead to a cycle, by the induction hypothesis. Second, we consider the turns around node i_n, containing a cross-link, and that do not involve a previously selected node. The TBTP algorithm prohibits all these turns. As a consequence, the algorithm breaks all the remaining cycles that could have involved node i_n with one of its adjacent cross-links. The induction hypothesis is thus justified, and the proof of the theorem is complete.

We now show that the TBTP algorithm prohibits at most 1/2 of the turns in the network. We first prove the following lemma:

Lemma 1 *At each step of the algorithm, the following inequality holds*

$$\sum_{i \in G} (|A(i)| - |P(i)|) \geq 0$$

Proof

Each turn (i, j, k), $\{(i, j) \in E\backslash E_T, (j, k) \in E\}$, can appear only once as a prohibited turn, namely in the set $P(j)$. On the other hand, the same turn will appear as a permitted turn in the set $A(i)$, and possibly also in the set $A(k)$ if $(j, k) \in E\backslash E_T$. Thus, each turn is counted once in the set of prohibited turns and, at least, once in the set of permitted turns, thereby proving the lemma.

Theorem 3 *The TBTP algorithm prohibits at most 1/2 of the turns in the network.*

Proof

From Lemma 1, there must exist at least one node i for which the difference $|A(i)| - |P(i)|$ is greater or equal to zero. Since, at each step, the algorithm selects the node i^* that maximizes this difference, it is guaranteed that the number of permitted turns is greater or equal to the number of prohibited turns.

It is worth noting that the number of turns permitted by the TBTP algorithm is actually strictly greater than the number of prohibited turns. This is because turns between tree-links are all permitted as well.

3.D.3. Algorithm complexity

We next show that the TBTP algorithm has practical computational complexity.

Theorem 4 *The computational complexity of the TBTP algorithm is $\mathcal{O}(|V|^2 d^2)$, where d represents the degree of graph G (i.e., the maximum degree of any node in the graph).*

Proof

The analysis of the computational complexity can be broken down into the following components:

1. The construction of the spanning tree $T(G)$. This component has complexity $\mathcal{O}(|V|d)$.
2. Computations of the sets $A(i)$ and $P(i)$. Computing the number of permitted/prohibited turns around any node i is of the order $\mathcal{O}(d^2)$. At each iteration of the algorithm, this computation is performed for all $|V|$ nodes in the graph. Since the number of iterations is at most $|V|$, we obtain that the overall complexity of this component is $\mathcal{O}(|V|^2 d^2)$.
3. The selection of node i^*. At each iteration, the node that maximizes the difference $|A(i)| - |P(i)|$ is selected, which requires $\mathcal{O}(|V|)$ computations. Since there are at most $|V|$ iterations, the complexity of this component is $\mathcal{O}(|V|^2)$.

We therefore obtain that the algorithm complexity is $\mathcal{O}(|V|^2 d^2)$.

Note that the above analysis assumes a straight-forward implementation of the algorithm. The computational complexity could be reduced by using advanced data structures, such as priority queues. Likewise, at each step of the algorithm, we do not have to repeat the computations of the set $A(i)$ and $P(i)$ for each node i, but only for a subset of the nodes.

3.D.4. Extension for Weighted Graphs

So far, we have only considered the case of unweighted graphs. However, in switched Ethernet networks, different links may have different capacity, e.g., 100 Mb/s versus 1 Gb/s. Consequently, the relative importance of different turns vary as well.

In order to address this issue, we can extend our results to weighted graphs. Each turn (i, j, k) in the graph is associated with a weight $w(i, j, k)$. This weight can be set according to any metric of interest [14].

The TBTP algorithm for weighted graphs remains the same as for unweighted graphs. We just have to replace $|A(i)|$ and $|P(i)|$ by the sum of the weight of turns in the corresponding sets. Moreover, since the proof of Theorem 3 holds unchanged, we obtain the following result for weighted graph topologies:

Corollary 1 *The sum of the weights of prohibited turns by the TBTP algorithm is at most* $1/2$ *of the sum of the weights of all turns in* G.

As an illustration, consider the example of Fig. 2, but with the following distribution of weights for the turns: $w(1, 3, 4) = 10$ and the weights of all other turns set to 1. In such a case, the algorithm would prohibit the following set of turns $S_T(G) = \{(2, 4, 1), (2, 4, 3), (2, 4, 5), (3, 4, 1), (3, 4, 5), (2, 5, 1),$
$(2, 5, 4)\}$. The overall weight of the prohibited turns is 7 which represents 25% of the total weight of the turns in the network.

4. Distributed TBTP

4.A. The Algorithm

A direct implementation of the TBTP algorithm, as described in the previous section, requires each switch in the network to have global knowledge of the network topology. Such global knowledge can be obtained in a distributed fashion, if every switch broadcasts its list of neighbors (neighborhood list), to the entire network. In essence, this is the approach implemented by link-state protocols, such as OSPF [5]. The problem with this approach is that it often does not scale well as the network grows.

In the following, we describe a more scalable, distributed implementation of the TBTP algorithm, referred to as dTBTP. This distributed version of the TBTP algorithm requires only local topological knowledge at each node. The dTBTP algorithm preserves all the important theoretical properties of the original TBTP algorithm. In particular, it guarantees that at most 1/2 of the turns on the network are prohibited. However, as shown in Section 6, its performance is slightly inferior to that of TBTP because it follows a local optimization procedure instead of a global one.

The dTBTP algorithm works as follows. Each node i maintains two sets, $A(i)$ and $P(i)$, that are defined as in the previous section. To build these sets, node i just needs to receive the neighborhood list of each of its neighbors. Node i then computes the cardinality of the sets $A(i)$ and $P(i)$. Now, we observe that one can break all the cycles containing node i either by prohibiting all the turns in set $P(i)$, i.e., turns around node i, or by prohibiting all the turns in set $A(i)$, i.e., turns starting from node i. In order to guarantee that at most 1/2 of the turns are prohibited, we prohibit turns belonging to the set of smaller cardinality. Thus, if $|A(i)| > |P(i)|$, then all the turns in $A(i)$ are permitted and those in $P(i)$ are prohibited. Otherwise, the turns in $P(i)$ are permitted and those in $A(i)$ are prohibited. This procedure guarantees that all the cycles containing node i are broken and that node i prohibits at most 1/2 of the turns under its consideration.

When implementing the above approach, one needs to avoid conflicting decisions by different nodes and make sure that only one node is allowed to permit or prohibit any given turn. In order to enforce this constraint, we allow a node to permit or prohibit turns only if it has the highest ID within its 2-hop neighborhood (note that two nodes belonging to the same turn must be within a distance of at most two hops). Once a node computes its sets of permitted turns and prohibited turns, it deletes itself and its cross-links and sends a deletion message to all the nodes within its 2-hop neighborhood. Upon receiving this message, another node with highest ID within its 2-hop neighborhood will be able to perform the same procedure and so on until all the nodes delete themselves.

A pseudo-code for the dTBTP algorithm is as follows:
Distributed Algorithm dTBTP$(G, T(G))$:

1. Each node i computes two sets of turns $A(i)$ and $P(i)$, defined the same way as in the pseudo-code of the TBTP algorithm, and checks if it has the highest ID in its 2-hop neighborhood.

2. If node i has no cross-links, then it removes itself and sends a deletion signal to its 2-hop neighborhood.

3. If node i has adjacent cross-links but does not have the highest ID, it waits until it receives a *deletion* signal from a node in its 2–hop neighborhood. Once this happens, it repeats step 1.

4. If node i has adjacent cross-links and has the highest ID, it does the following operations:

   (a) If $|A(i)| > |P(i)|$, then all the turns in $A(i)$ are permitted and those in $P(i)$ are prohibited. Otherwise, all the turns in $A(i)$ are prohibited and those in $P(i)$ are permitted.

   (b) Node i deletes itself and all of its cross-links and sends a deletion signal to its 2-hops neighborhood. Node i informs its neighbors about its set of prohibited/permitted turns.

A few comments are in order here. First, we note that the algorithm can be run completely asynchronously by different nodes in the network. Second, a node without cross-links can immediately remove itself, even if it does not have the highest ID. This optimization is implemented by the algorithm in step 2). Third, if some (non-deleted) node j receives a deletion signal from node i, then it must recompute its sets $A(j)$ and $P(j)$. Specifically, node j must remove all the cross-links adjacent to node i from graph G. However, node i and the tree-links adjacent to node i must stay in the graph, except for a special case described in the next paragraph. Finally, we note that at each point of time, there always exists at least one node in the graph that can remove itself. This is the node with the highest ID in the graph. However, in general, many nodes are able to remove themselves concurrently, as shown by our simulations in Section 6.

The performance of dTBTP can be further optimized using a number of heuristics. For instance, nodes having degree 1 after their cross-links are deleted (i.e., they are connected to a single tree-link) can be safely deleted from the graph since they cannot belong to any cycle. This approach can prevent the unnecessary prohibition of some turns. Furthermore, one can improve performance if nodes assign IDs to themselves according to their distance to the root (the higher the distance, the higher the ID). Note that when the tree $T(G)$ is generated by the IEEE 802.1d protocol, each node knows its distance to the root. To implement such a label assignment, we can break the ID of each node into two parts. The most significant bits of the ID correspond to the distance to the root (which is known once a tree is built), while the least significant bits correspond to the physical ID of the switch, thereby breaking any possible tie. Note that the node IDs used by dTBTP are logical quantities that do not have to be the same as the physical IDs of the switches. This approach improves performance because it speed-ups the deletion of leaves, which are nodes connected to a single tree link.

4.B. Message Complexity Analysis

A key performance metric for any distributed algorithm is the amount of messages required for the algorithm to converge. For the sake of simplicity, we assume that all the messages consist of a unit of information, but the analysis below can easily be extended to the case where messages have non-uniform sizes. During the initialization phase, independent of the topology, the number of messages needed is $2|E|$ packets, since over every link (i,j) of graph G, node i and node j will exchange a message containing the list of their neighbors, which means two messages per link. During the running phase, node i, when deleting itself, sends a deletion message to all the nodes within a ball of radius 2, i.e. $|B(i,2)| = \sum_{j\in N(i)}(d_j - 1)$ messages, where $N(i)$ is the set of neighbors of node i. Thus, the overall amount of messages M required both during the initialization and the running phase can be upper bounded as follows:

$$\begin{aligned} M &= 2|E| + \sum_{i\in G}\sum_{j\in N(i)}(d_j - 1) \\ &= 2|E| + \sum_{i\in G}\sum_{j\in N(i)}(d_j - 1) \\ &\leq 2|E| + \sum_{i\in G} d(d-1) \\ &= 2|E|d, \end{aligned} \tag{1}$$

where d is the degree of G. Finally, if we calculate $M/|E|$, we obtain the following bound on the average number of messages exchanged over each link during a run of dTBTP:

Theorem 5 *The average number of messages exchanged over each link by dTBTP never exceeds $2d$, where d represents the degree of graph G.*

The above theorem states that the message complexity per link of dTBTP is independent of the size of the graph (assuming d is constant).

5. Backward–Compatibility

In the description of the Up/Down and TBTP algorithms in Section 3, we have assumed that all switches in the network are capable to perform turn-prohibition. In large networks, however, a massive replacement of all switches is a major issue and very unlikely to happen. In this section, we suggest a strategy for a smooth transition towards a complete replacement.

5.A. Approach

We consider an heterogeneous network consisting of both intelligent and legacy nodes. In order to ensure backward-compatibility, we require intelligent switches to be able of running both the spanning tree algorithm (IEEE 802.1d) and turn prohibition.

At the very beginning, all the switches run the distributed spanning tree algorithm. As a result, every node belongs to a spanning tree $T(G)$ that is generated by the IEEE 802.1d protocol.

Next, intelligent nodes send "neighbor discovery" messages to their directly connected neighbors. As a result, an intelligent node knows if it is connected to another intelligent node or to a legacy node.

Now suppose that two intelligent nodes are connected by a cross-link. Then use of this link may be possible, depending on the turn-prohibition algorithm being employed. However, if a cross-link connects between an intelligent node and a legacy node, then use of this link is not possible. This is because the legacy node would not accept to forward or receive packets over that link.

Therefore, a backward-compatible solution has to examine each turn (i, j, k), where (i, j) is a cross-link and both nodes i and j are intelligent, and decide if this turn can be permitted or not. We now distinguish between the cases of Up/Down, TBTP, and dTBTP.

5.A.1. Up/Down

We remind that the Up/Down algorithm requires nodes to be ordered. This information can readily be obtained from the IEEE 802.1d protocol, since each node n knows its distance $d(n)$ from the root. Thus, each turn (i, j, k) for which $d(i) \neq d(j)$ and $d(j) \neq d(k)$ can be resolved. It will be prohibited only if node j is farther away from the root than the two other nodes. Also, turns such that $d(i) = d(j)$ can be resolved using the IDs of the switches, that is, if $\mathrm{ID}(i) > \mathrm{ID}(j)$ then $i > j$ and vice-versa.

Note that this approach works as well when node k in the turn (i, j, k) is a legacy node, because node j knows whether node k is its parent or one of its children (remind that if node k is a legacy node, then link (j, k) must be a tree-link).

5.A.2. TBTP

As mentioned in Section 3, the TBTP algorithm requires each node to have global knowledge of the network topology. This is of course not possible in an heterogeneous network, since legacy nodes would not participate in the process of collecting the topology (e.g., by generating or forwarding link-state packets).

Our solution is to implement the TBTP algorithm independently for each connected group of intelligent nodes. We refer to such groups as *components of connectivity.* Figure 3(a) gives an example of an heterogeneous network with two components of connectivity, namely nodes 1,3,4 and nodes 6,7,8.

In each component connectivity, the intelligent nodes collect the internal topology of that component and run the TBTP algorithm to decide which turns should be permitted or prohibited.

In order to show why this approach breaks all the cycles, consider the following modified topology G'. This topology consists of the original topology G, but without all the cross-links connected to legacy nodes (on one or both ends). We now observe that our backward-compatible solution is equivalent to having run the centralized version of the TBTP algorithm, described in Section 3, on the topology G'. Therefore, it follows from Theorem 2 that all the cycles in the network are broken. Figure 3 depicts an example of an heterogeneous graph G and its corresponding modified topology G'.

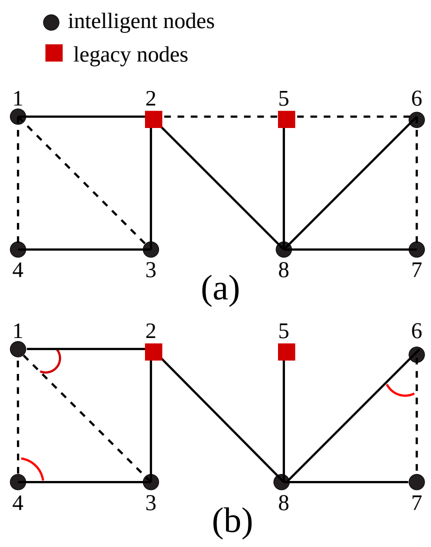


Fig. 3. (a) An heterogeneous graph, G, composed of legacy and intelligent nodes. (b) Corresponding modified topology G' and prohibited turns.

5.A.3. dTBTP

The distributed TBTP algorithm of Section 4 requires nodes to exchange neighborhood lists. Such neighborhood lists can only be generated by intelligent nodes. The neighborhood list generated by some node i should contain the list of all the neighbors of node i connected by a tree-link as well as all the intelligent neighbors of node i connected by a cross-link. Legacy neighbors connected by a cross-link should not be included in the list since they cannot forward packets. Except for this provision, the algorithm can be implemented exactly the same way as described in Section 4.

5.B. Packet Forwarding (Routing)

In a turn prohibition-based algorithm, such as Up/Down or TBTP, several paths may exist between a source and a destination, as opposed to a spanning tree where only a single path exists. Thus, there is a need to deploy a routing algorithm that can determine the “best” path, according to some appropriate metric, from each source to each destination.

In [13], we have described one such routing algorithm that is a generalization of the Bellman-Ford routing algorithm (other routing algorithms can also be generalized using the Turnnet concept of [26]). In the case of TBTP, the implementation of the generalized Bellman-Ford algorithm adds little overhead, since every node knows the full topology. Similarly to the original Bellman-Ford algorithm, the generalized version can be implemented in a fully distributed fashion. Therefore, when used with Up/Down or dTBTP, nodes do not need to have global knowledge of the topology. Furthermore, since the network contains no cycles, the count-to-infinity problem will not arise.

In the case of heterogeneous networks, the generalized Bellman-Ford algorithm is implemented separately in each component connectivity. In order to maintain backward-compatibility, intelligent nodes that are directly connected to legacy nodes must also implement the backward-learning process of the IEEE 802.1d protocol. In such a case, when an intelligent node learns about a certain destination outside its component of connectivity, it must inform all the other nodes within its component.

Finally, each routing entry has an associated time-out, e.g. 15 minutes. If the entry is not refreshed within this period of time, then it is removed from the routing table. As with the IEEE 802.1d protocol, a switch that receives a packet for an unknown destination will broadcast the packet along links of the spanning tree $T(G)$.

5.C. Heuristics for Node Replacement

By implementing turn-prohibition instead of link-prohibition, the TBTP and Up/Down algorithms clearly outperform the simple spanning tree algorithm. Therefore, the gradual replacement of legacy nodes by intelligent node will improve network performance.

An interesting question within this context is as to which legacy nodes should be replaced first in order to achieve maximum improvement. One simple strategy is the *random* one in which a legacy node is picked at random and replaced by an intelligent node.

We propose here another heuristic. This heuristic relies on the fact that a random replacement could lead to sets of intelligent nodes completely isolated, i.e., no cross-link can be enabled for forwarding. Instead, we propose a *Top-Down* approach whereas the replacement starts from the root, i.e., replacing the level 1 node first, then proceed with level 2 nodes and so on. This replacement strategy will guarantee the existence of at most one component of connectivity in the network. Moreover, it will relieve congestion from the root. Therefore, we expect that this heuristic will lead to a faster performance improvement than the random one. Our numerical results in the next section confirm this rationale.

5.D. Reconfiguration

In the case of node/link failures, the spanning tree $T(G)$ is reconfigured according to the specifications of the IEEE 802.1d standard or its recently improved version, IEEE 802.1w, which supports faster spanning tree reconfiguration. Once the spanning tree is reconfigured, a new set of permitted/prohibited turns is determined using the procedure described above.

6. Numerical Results

In this section, we compare the performance of the various cycle-breaking approaches described in the previous sections. In order to obtain comprehensive sets of results, we make use of both *flow level* and *packet level* simulations.

Flow level simulations are based on fluid models. They provide fast, qualitative results and can be run over large number of graphs with different topology properties.

Packet level simulations, on the other hand, model network dynamics at the granularity of single packets. They can provide more accurate estimates on quality of service metrics, such as end-to-end delay and throughput. Unfortunately, they are computationally intensive, especially for large graphs.

6.A. Flow Level Simulations

The goal of our flow level simulations is to compare the performance of the algorithms as related to different network topology properties, such as the size of the network or the degree of the nodes. Another important objective is to evaluate the heuristics for node replacement discussed in Section 5.

Our simulations are based on random, connected graphs. Each node has fixed degree d, and the network size (number of nodes in the network) is $|V|$. We assume that all links in the network have the same capacity C and we set $C = 1$ Gb/s. All the results presented correspond to average over 100 graphs with identical parameters.

Once a random graph is generated, each of the cycle-breaking algorithms (spanning tree, Up/Down and TBTP) is run on top of it in order to determine a set of prohibited links/turns. Routing matrices over a turn-prohibited graph are computed using the generalized Bellman-Ford algorithm of [13].

As a reference, we also simulate an ideal scheme when no turn in the graph is prohibited. We refer to this scheme as *shortest path*, since each flow is established along the shortest path from the source to the destination.

We consider two metrics. The first is the fraction of prohibited turns in the network, that is, the ratio $|S_T(G)|/|R(G)|$. A lower value for this quantity is considered to be better since it implies less unutilized network resources.

The second metric is throughput. We compute this metric as follows. We assume a fluid model where flows are transmitted at a fixed rate. Each node establishes a session (flow) with k other nodes in the network, picked at random. In all of our simulations, we set $k = 4$. Each flow is routed along the shortest-path over the turn-prohibited graph (if multiple routes exist, then one is selected at random). Next, we determine the bottleneck link, which is the link shared by the maximum number of flows. The throughput is then defined as the capacity of the bottleneck link divided by the number of flows sharing it. In other words, the throughput is the maximum rate at which each flow can be transmitted without saturating the network.

6.A.1. Complete Replacement of the Switches

In our first set of simulations, we compare the performance of the algorithms assuming that all switches in the network are intelligent, that is, capable of performing turn-prohibition.

We first evaluate the performance and scalability properties of each algorithm as a function of the number of nodes in the network $|V|$. We assume the degree of each node to be $d = 8$.

Table 1 shows that the TBTP algorithm prohibits about 10% fewer turns than Up/Down. As expected, the simple spanning tree approach performs far worse, prohibiting about 90% of the turns in the network (although the spanning tree approach prohibits links, it is still possible to count the fraction of prohibited turns in the network). These results are almost insensitive to the network size.

Figure 4 depicts the throughput performance with 99% confidence intervals. The results presented are in agreement with those obtained for the fraction of prohibited turns. We observe that, for 32 nodes, the TBTP achieves a throughput

$\|V\|$	TBTP	Up/Down	Spanning Tree
32	0.29	0.31	0.91
56	0.28	0.30	0.91
88	0.28	0.30	0.91
120	0.28	0.30	0.91
152	0.27	0.30	0.91

Table 1. Fraction of prohibited turns for networks of varying size and fixed degree $d = 8$.

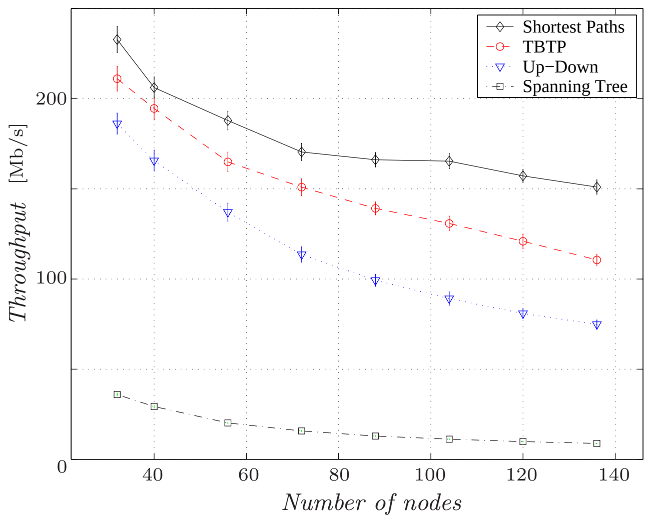


Fig. 4. Throughput versus increasing number of nodes $|V|$. The degree of each node is $d = 8$.

approximately 10% higher than that of Up/Down. Moreover, the relative difference in the performance between these two algorithms increases with the network size. Another important observation is that the throughput of TBTP is within a factor of at most 1.5 from that of the "shortest-path" scheme. This means that the cost of breaking all the cycles in the network may not be too significant in terms of network performance. This is in clear contrast with the spanning tree approach which achieves an order of magnitude lower throughput.

Next, we evaluate the effect of the graph density on the performance of the algorithms. We vary the degree of the nodes d, but keep the number of nodes fixed to $|V| = 120$. Table 2 provides evidence on the scalability of the turn-prohibition techniques, as compared to the spanning tree. While the fraction of prohibited turns prohibited by the TBTP and Up/Down algorithms increase slightly with the degree d, the spanning tree approach experiences a much sharper increase.

Figure 5 shows the throughput performance of the various algorithms as a function of the node degree. We note that increasing the degree of nodes implies an addition of links in the network. The spanning tree algorithm is the only algorithm that does not benefit from this increase. The reason is that a spanning tree permits the use of only $|V| - 1$ links in the network, independently of the topology.

Figure 5 also depicts the performance of the Turn-Prohibition (TP) algorithm developed in [13]. This algorithm prohibits at most 1/3 of the turns in any network,

d	TBTP	Up–Down	Spanning Tree
4	0.23	0.25	0.74
6	0.27	0.29	0.86
8	0.28	0.30	0.91
10	0.28	0.31	0.93
12	0.29	0.31	0.95

Table 2. Fraction of prohibited turns for networks of fixed size, $|V| = 120$, and nodes of varying degree d.

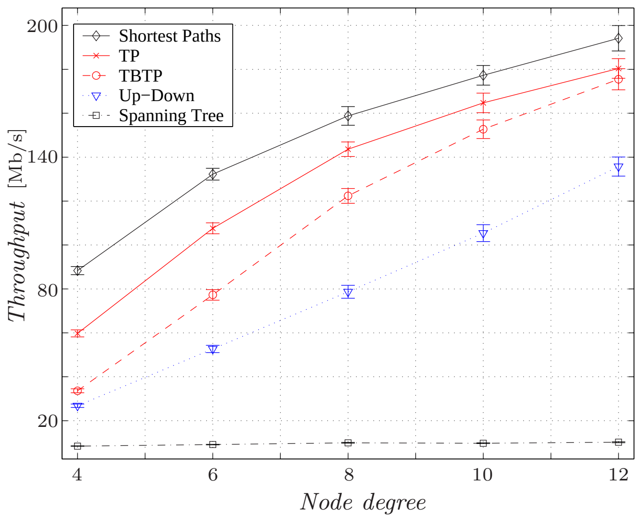


Fig. 5. Throughput versus increasing degree d, for $|V| = 120$ nodes.

but is not backward-compatible with the IEEE 802.1d protocol. We observe that the performance of the new TBTP algorithm lies in-between those of the TP and Up/Down algorithms. Moreover, the performance of TBTP gets closer to TP as graphs become denser. Thus, the TBTP algorithm shows a good trade–off between the performance of the TP approach and the constraint imposed by permitting all the turns along a given spanning tree.

6.A.2. Partial Replacement of the Switches

We now evaluate and compare the heuristics for node replacement described in Section 5. Simulations are run for random graphs of $|V| = 120$ nodes and degree $d = 8$. For each graph G, a spanning tree $T(G)$ is first constructed. Then legacy nodes are replaced according to one of the heuristics, that is, either the random or the Top-Down approach.

Figure 6(a) plots the throughput as a function of the number of intelligent nodes in the network, for each of the two heuristics. The results depicted in that figure are for the case where intelligent nodes implement the TBTP algorithm. As expected, the figures shows that the Top-Down heuristic leads to a faster improvement in performance than the random heuristic. For instance, when 60 out of the 120 network nodes are intelligent, Top-Down achieves a throughput roughly 50% higher than that obtained by the random approach. For both heuristics, the marginal gain in the throughput increases with the number of intelligent nodes.

Figure 6(b) shows similar results when intelligent nodes implement the Up/Down algorithms, though the difference between the two heuristics is less significant in this case.

6.A.3. Performance of the Distributed Version

In this section, we evaluate the performance of the distributed TBTP algorithm. Figure 7 depicts the throughput achieved by the dTBTP algorithm for the same graph parameters used for Fig. 5. The performance of dTBTP with random ID assignment (labelled as "dTBTP random") is markedly worse than TBTP and only slightly better than Up/Down.

In order to overcome such a behavior, the introduction of optimizations for node IDs assignment is beneficial. As we mentioned in Section 4, one can label nodes according to their distance from the root. In this case, the performance of dTBTP (simply labelled as "dTBTP") is much closer to the behavior of TBTP.

Next, we simulate the convergence time or number of rounds needed for dTBTP to complete. We count a new round, each time the dTBTP algorithm performs steps 2) and/or 4) (i.e., a node generates a deletion signal). We note that, within a given ball, several nodes may perform step 2) within the same round but at most one node may perform step 4). Nodes belonging to different balls may perform step 4) within the same round, if they each have the highest ID within their respective ball.

Fig. 8 shows the number of rounds taken by dTBTP for graphs of degree $d = 4$ and size increasing from $|V| = 32$ nodes to $|V| = 1000$ nodes. We observe from the figure that, after an initial phase, the converges time of dTBTP grows slowly with the network size. This result shows that dTBTP scales well as the network size gets larger.

6.B. Packet Level Simulations

In this section, we present packet level simulation results obtained with the NS2 simulator [27]. These simulations allow us to estimate the average end-to-end delay of packets as a function of the traffic load, for each of the cycle-breaking methods.

We consider two sample topologies. The first one, referred to as graph G_1, is similar to that adopted for the flow level simulation and consists of a randomly generated graph with 32 nodes of degree 8. The second graph, G_2, has also 32 nodes. It is generated using the BRITE topology generator [28], based on the so-called Barabási–Albert model [29]. This model captures the power-law node degree distribution that has been observed in a variety of networks.

Our traffic model is as follows. Each node establishes an UDP session (flow) with four other nodes, picked at random in the network. Each UDP session generates traffic according to an exponential ON–OFF source model. The average ON and OFF periods are 460 ms and 540 ms, respectively. The average traffic rate (in bit/s) is a simulation parameter and denoted by λ. The size of each packet is 600 bytes long. Each link has a propagation delay of 0.5 ms and capacity of 1 Mb/s. Note that due to memory and computation constraints, we could not simulate gigabit links. We conjecture, however, that the simulation results would have been qualitatively similar to those presented here.

The results for the end-to-end delay are presented in Figure 9. The results obtained for the two sample graphs are similar. We observe that the average end-to-end delay incurred with the spanning tree algorithm is always higher than with the two turn-prohibition approaches. Moreover, the maximum sustainable throughput, i.e., the traffic rate value λ at which the end-to-end delay starts to diverge, is increased by a factor of about five when the turn-prohibition techniques are employed. The TBTP algorithm achieves a higher throughput than Up/Down, as the latter prohibits a larger number of turns.

Finally, it interesting to compare Figures 4 and 9, for the case of $|V| = 32$ nodes. We observe that flow level simulations predict well how each scheme performs one with respect to the other (we remind that the flow-level simulations use 1 Gb/s links). This result validates the use of flow level simulations as a fast and reliable method to predict network performance.

7. Conclusion

In this paper, we have addressed the problem of breaking cycles in a scalable manner in Gigabit Ethernet networks. This problem has gained particular importance recently with the emergence of large metropolitan area networks (MANs), based on the Gigabit Ethernet technology. Toward this goal, we have proposed, analyzed and evaluated a novel cycle-breaking algorithm, called Tree-Based Turn-Prohibition (TBTP). This polynomial-time algorithm guarantees the prohibition of at most 1/2 of the turns in the network, while permitting all the turns belonging to a predetermined spanning tree. We have shown that the algorithm can be generalized to the case where turns are assigned non-uniform weights (these weights can be set according to any metric of choice). In that case, the TBTP algorithm prohibits at most 1/2 of the total weight of turns in the network. This generalization is especially useful for networks with links of non-uniform capacity.

In addition, we have introduced a distributed implementation of the TBTP algorithm, called dTBTP. With dTBTP, nodes need only to have local knowledge of the network topology, i.e., they need only to know the nodes and links that are within a radius two hops. We have shown that dTBTP satisfies the same theoretical properties as the original TBTP algorithm. Moreover, the number of messages sent on each link is bounded by $2d$, where d is the maximum degree of the graph.

We have also presented a general framework for incrementally deploying TBTP-capable switches in a way that is backward-compatible with the existing IEEE 802.1d standard. Since several paths may exist between any source-destination pair, we have described a method to perform packet forwarding (routing) in an heterogeneous networks consisting of intelligent and legacy switches.

The performance of the proposed algorithm was thoroughly evaluated using both flow-level and packet-level simulations. The simulation showed that the TBTP algorithm achieves a throughput that is about an order magnitude higher than that obtained with the spanning tree standard. Furthermore, for a wide range of topology parameters, the performance of the TBTP-algorithm differs only by a small margin from that of shortest path routing (which achieves the highest possible throughput in theory, but does not break cycles). The performance of dTBTP is inferior to that of TBTP (although its performance can be improved using various heuristics) but still noticeably better than Up/Down.

Finally, we have proposed a heuristic, called Top/Down, to determine the order in which legacy switches should be replaced. This scheme proceeds by first replacing the root node, then nodes at level 1, and so forth. We have shown that Top/Down replacement outperforms a scheme where legacy nodes are replaced at random, achieving a throughput 50% higher in some cases. An interesting open research area is to investigate whether this strategy might be further improved and devise other possible replacement schemes.

Acknowledgment

This work was supported in part by DOE ECPI grant DE-FG02-04ER25605 and NSF CAREER grant CNS-0132802.

References and Links

[1] H. Frazier and H. Johnson, "Gigabit Ethernet: from 100 to 1,000 Mbps," IEEE Internet Computing **3**(1), 24–31 (1999).

[2] H. Frazier, "The 802.3z Gigabit Ethernet Standard," IEEE Network **12**(3), 6–7 (1998).

[3] D. Clark, "Are ATM, Gigabit Ethernet Ready for Prime Time?" IEEE Computer **31**(5), 11–13 (1998).

[4] W. Noureddine and F. Tobagi, "Selective Backpressure in Switched Ethernet LANs," in Proceedings of Globecom 1999, vol. 2, pp. 1256–1263 (Vancouver, British Columbia, Canada, 6-10 June, 1999).

[5] R. Perlman, *Interconnections* (Addison-Wesley, 2000).

[6] M. Karol, S. Golestani, and D. Lee, "Prevention of Deadlocks and Livelocks in Lossless Backpressured Packet Networks," in Proceedings of INFOCOM 2000, vol. 3, pp. 1333–1342 (Tel Aviv Israel, 26-30 March, 2000).

[7] J. Duato, "A Necessary and Sufficient Condition for Deadlock-Free Routing in Cut-Through and Store-and-Forward Networks," IEEE Transactions on Parallel and Distributed Systems **7**(8), 841–854 (1996).

[8] M. Soha and R. Perlman, "Comparison of Two LAN Bridge Approaches," IEEE Network **2**(1), 37–48 (1988).

[9] "Metro Ethernet Networks-A Technical Overview," Metro Ethernet Forum White Paper, 2002-2004. Available online at `http://www.metroethernetforum.org/`.

[10] L. Bosack and C. Hedrick, "Problems in Large LANs," IEEE Network **2**(1), 52–56 (1988).

[11] G. Ibanez, A. Garcia, and A. Azcorra, "Alternative Multiple Spanning Tree Protocols (AMSTP) for Optical Ethernet Backbones," in Proceedings of IEEE LCN 2004, vol. 3 (Tampa, Florida, 16-18 Nov., 2004).

[12] S. Sharma, K. Gopalan, S. Nanda, and T. C. Chiueh, "Viking: A Multi-Spanning-Tree Ethernet Architecture for Metropolitan Area and Cluster Networks," in Proceedings of IEEE INFOCOM 2004 (Hong Kong, 7-11 March, 2004).

[13] D. Starobinski, M. Karpovsky, and L. Zakrevski, "Application of Network Calculus to General Topologies using Turn–Prohibition," IEEE/ACM Transactions on Networking **11**(3), 411–421 (2003).

[14] L. Levitin and M. Karpovsky, "Deadlock Prevention in Networks Modeled as Weighted Graphs," in Proceedings of 8th Int. Conf. on Information Networks, Systems and Technologies, ICINSAT 2002 (St.-Petersburg, 16-19 Sept., 2002).

[15] C. Glass and L. Ni, "The Turn Model for Adaptive Routing,," Journal of ACM **41**(5), 874–902 (1994).

[16] M. D. Schroeder, A. D. Birrell, M. Burrows, H. Murray, R. M. Needham, and T. L. Rodeheffer, "Autonet: A High-Speed, Self-Configuring Local Area Network Using Point-to-Point Links," IEEE Journal on Selected Areas in Communications **9**(8), 1318–1335 (1991).

[17] R. Perlman, “An Algorithm for Distributed Computation of a Spanning Tree in an Extended LAN,” in Proceedings of Ninth ACM Data Communications Symposium, pp. 44–53 (Whistler Moutain, British Columbia, Canada, 1985).

[18] A. Myers, T. Ng, and H. Zhang, “Rethinking the Service Model: Scaling Ethernet to a Million Nodes,” in Proceedings of HotNets-III (San Diego, CA, 15-16 Nov., 2004).

[19] J. Hart, “Extending the IEEE 802.1 MAC Brige Standard to Remote Bridges,” IEEE Network **2**(1), 10–15 (1988).

[20] R. Perlman, W. Hawe, and A. Lauck, “Utilization of redundant links in bridged networks,” (22 September, 1992). United States Patent Number 5,150,360.

[21] K. Lui, W. C. Lee, and K. Nahrstedt, “STAR: a transparent spanning tree bridge protocol with alternate routing,” ACM SIGCOMM Computer Communication Review **32**(3), 33–46 (2002).

[22] T. L. Rodeheffer, C. Thekkath, and D. Anderson, “SmartBridge: A Scalable Bridge Architecture,” in Proceedings of ACM SIGCOMM 2000, pp. 205–216 (Stockholm, Aug. 28 - Sept. 1, 2000).

[23] F. de Pellegrini, D. Starobinski, M. Karpovsky, and L. Levitin, “Scalable Cycle-Breaking Algorithms for Gigabit Ethernet Backbones,” in Proceedings of INFOCOM 2004 (Hong Kong, 7-11 March, 2004).

[24] J. Echague, M. Prieto, J. Villadangos, and V. Cholvi, “A Distributed Algorithm to Provide QoS by Avoiding Cycles in Routes,” in Proceedings of QofIS 2004 and Lecture Notes in Computer Science, vol. 3266, pp. 224–236 (Barcelona, Spain, 29 September - 1 October, 2004).

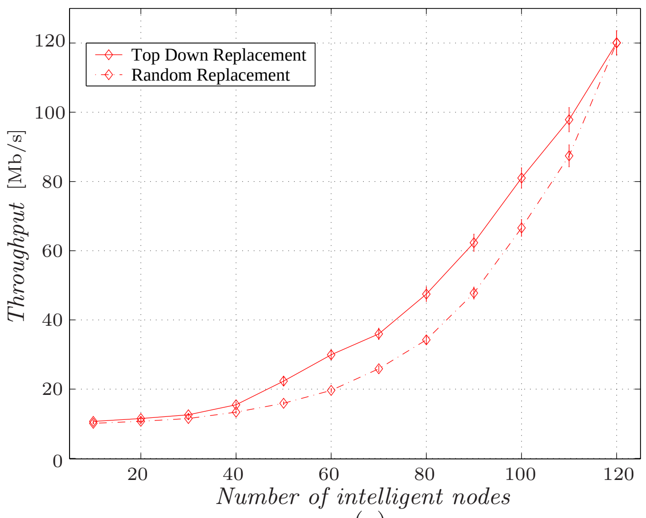
[25] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to algorithms* (McGraw-Hill, 1990).

[26] M. Fidler and G. Einhoff, “Routing in Turn-Prohibition Based Feed-Forwad Networks,” in Proceedings of IFIP Networking 2004, Springer LNCS 3042, pp. 1168–1179 (Athens, Greece, 9-14 May, 2004).

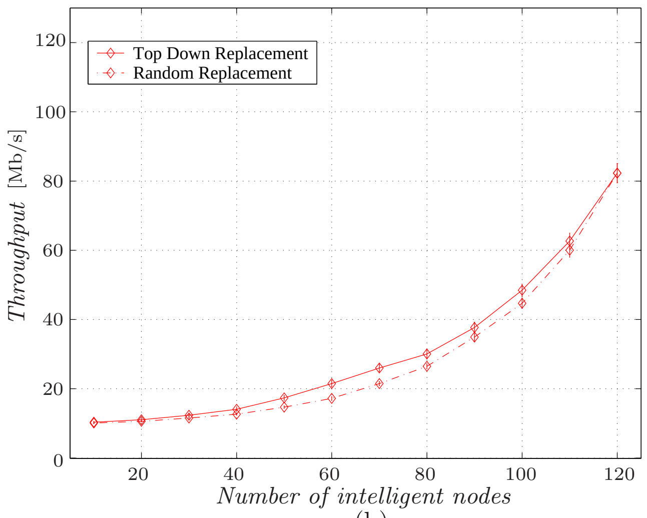
[27] The Network Simulator - ns-2. Available online at `http://www.isi.edu/nsnam/ns/`.

[28] Boston University Representative Topology Generator - BRITE. Available online at `http://www.cs.bu.edu/brite/`.

[29] A.-L. Barabási and R. Albert, “Emergence of Scaling in Random Networks,” Science **286**, 509–512 (1999).



(a)



(b)

Fig. 6. Throughput versus number of intelligent nodes (total number of nodes is fixed at $|V| = 120$): (a) TBTP (b) Up–Down.

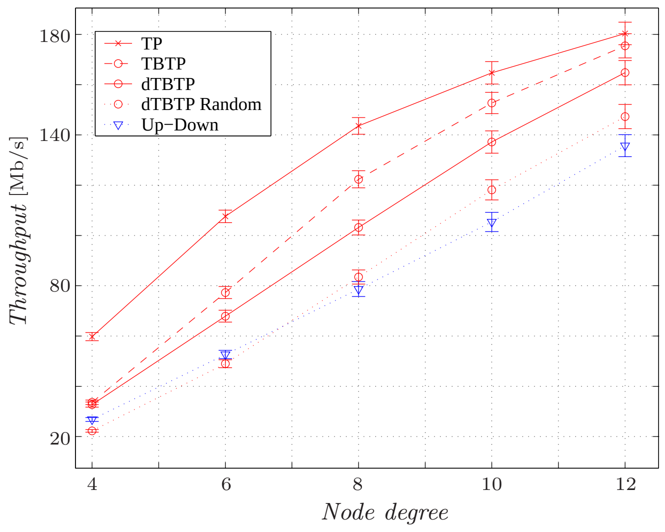


Fig. 7. Performance of dTBTP with random ID assignment and distance-based ID assignment.

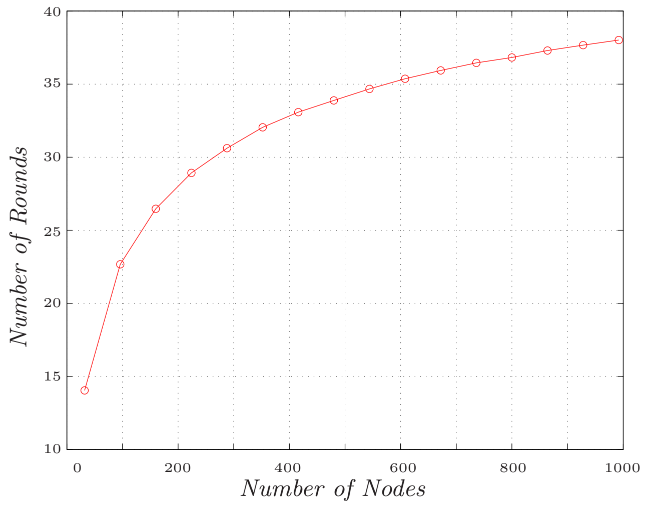
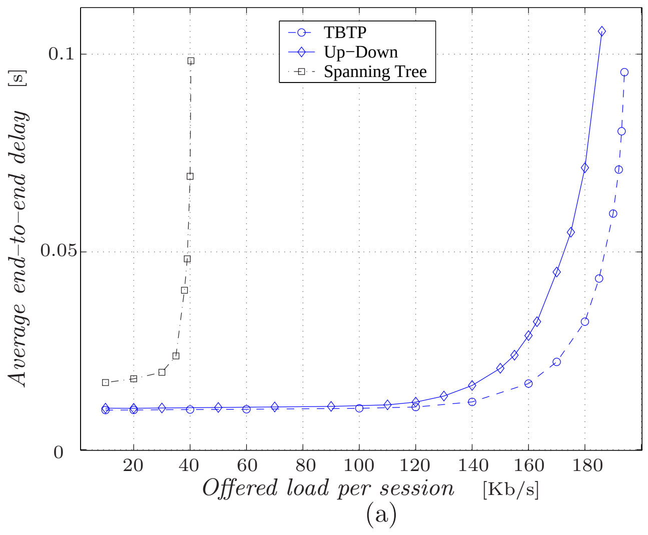


Fig. 8. Convergence time of dTBTP.



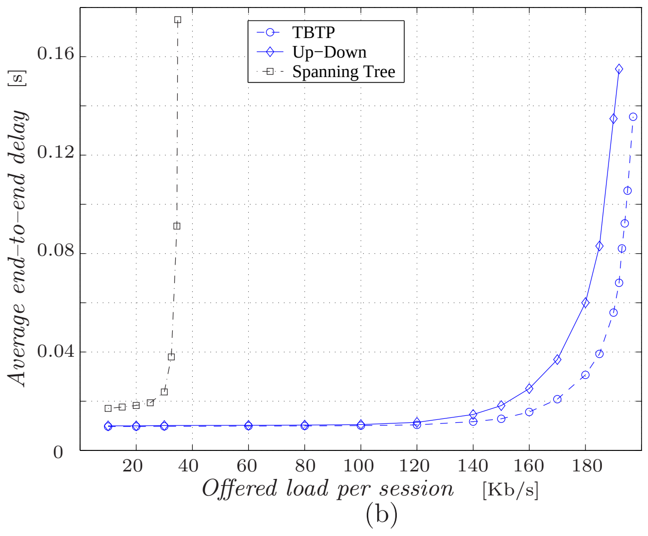


Fig. 9. Average end–to–end delay versus offered load λ, (a) Sample graph G_1 (b) Sample graph G_2.