Available online at www.sciencedirect.com
ScienceDirect
Computer Communications 31 (2008) 1824–1831
computer communications
www.elsevier.com/locate/comcom

Transforming general networks into feed-forward by using turn-prohibition ☆

Juan Echagüe [a], Jesús Villadangos [b], Vicent Cholvi [a,*], Manuel Prieto [b]

[a] Universitat Jaume I, Campus del Riu sec, 12071 Castellón, Spain
[b] Universidad Pública de Navarra, Campus de Arrosadía, 31006 Pamplona, Spain

Received 3 March 2007; received in revised form 27 November 2007; accepted 28 November 2007
Available online 8 December 2007

Abstract

The issue of breaking cycles in communication networks is an important topic for several reasons. For instance, it is required when transparent bridges are filling the forwarding tables. It is also needed to prevent the occurrence of deadlocks caused by certain routing protocols. Furthermore, most of the techniques used to work with communication networks can only be applied if the network topology is free of cycles.

We present a distributed protocol which, applied to any network topology, provides a cycle-free topology. Our approach is based on the prohibition of only certain turns in the network. In contrast to previous proposals, our protocol is fully distributed, and thus does not require nodes to have knowledge of the global network topology. Furthermore, it allows multiple nodes to initiate the protocol in an independent manner. This feature can be used to cope with new nodes entering the system, as well as with nodes leaving it (voluntarily or due to a failure). We provide a detailed description of our proposal, formal proof of correctness, and an analysis of its performance.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Cycle-free networks; Feed-forward networks; QoS routing

1. Introduction

The problem of breaking cycles in communication networks is an issue that has been broadly addressed, since it offers several important benefits:

- It prevents broadcast packets from circulating forever in the network (note that Ethernet packets do not have a TTL field and switches are not allowed to modify the headers).
- It also guarantees that, provided the total load injected at each link does not exceed the link rate, the network will be stable [1,2]. That is, the number of undelivered packets in the network can be bounded.
- Furthermore, it precludes the occurrence of deadlocks in certain networks. For example a deadlock can occur as a result of using flow control mechanisms in IEEE 802.3x [3]. Also, deadlocks may arise in cyclic networks if the wormhole routing protocol is used [4].
- Finally, the application of most of the methods and techniques used to reason about performance guarantees in current telecommunication networks can only be applied if the links in the network are unable to form cycles [5,6]. Clearly, since real networks are generally not free of cycles, this severely restricts direct application of the above mentioned theories.

Taking the above mentioned reasons into account and in order to avoid the problems that are associated with the existence of cycles, a natural solution is to restrict the use of the network in such a way that it becomes impossible to create any cyclic dependency.

A simple approach to transform any network topology into a cycle-free network topology is to construct a spanning tree and prohibit the use of links not belonging to the span-

☆ A preliminary version of this paper appeared in the Proceedings of the International Workshop on QoS Routing, 2004. This work has been partially supported by the CICYT under Grant TSI2004-02940 and by Bancaixa under Grant P1-1B2003-37.

* Corresponding author. Tel.: +34 964728332.
E-mail address: vcholvi@uji.es (V. Cholvi).

ning tree. However, although a spanning tree maintains graph connectivity, it is quite inefficient since it prohibits the use of a high percentage of links and increases the traffic in the links close to the spanning tree's root node [7,8].

A more scalable approach is not to prohibit the use of complete links, but only of certain *turns*, where a turn is defined as a triplet of nodes connected by two links. A prohibited turn (a, b, c) would forbid the forwarding of a packet from link (a, b) to link (b, c) (and from link (c, b) to link (b, a)). The main idea is to break all the cycles in the network through the prohibition of carefully selected turns. Although routing protocols are usually not equipped to handle forbidden turns, protocols such as *Turnnet* [9] make it possible to transform an arbitrary network with no prohibited turns into a new one without prohibited turns, without raising the routing complexity in an unacceptable manner. This allows us to use arbitrary routing schemes.

In this paper, by following the turn-prohibition approach, we propose a protocol that guarantees that the resulting network topology will be free of cycles (i.e., feed-forward). On the contrary to previous proposals, our protocol is fully distributed, and does not require nodes to have global knowledge of the network topology. Furthermore, it allows multiple nodes to initiate the protocol in an independent manner, even at the same time. This feature can be used to cope with new nodes entering the system, as well as with nodes leaving the system (voluntarily, or due to a failure).

The rest of the paper is arranged as follows. In Section 2, we introduce the proposed protocol, and describe how it works. In Section 3, we prove the correctness of the above mentioned protocol, and bound the maximum number of prohibited turns. An experimental evaluation is carried out in Section 4. Finally, in Section 5 we present our conclusions.

2. The *DMITP* protocol

In this section, we present a distributed protocol that uses the turn-prohibition mechanism to transform any network topology into an equivalent feed-forward topology. We call it the *Distributed Multiple Initiator Turn-Prohibition* protocol (*DMITP*). In contrast to centralized solutions, where a single node performs all the actions, and broadcasts the resulting topology after completion [10,11], *DMITP* works in a distributed fashion, forbidding turns while carrying out the network exploration.

2.1. Definitions and notation

We model the network topology as a graph $G = (V, E)$ consisting of a set of vertices V and a set of edges E. Vertices represent network nodes and edges represent bidirectional links. The terms node and vertex, and link and edge are used interchangeably. We denote a link between node i and j as (i, j). A *path* is defined as a list of nodes $(i, j, k \dots, p)$, such that adjacent nodes are connected by a link. We say that a path forms a *direct cycle* if it contains the same edge at least twice. Note that a path may traverse the same node several times without creating a cycle.

Algorithm 1. Code of *initiate_DMITP*$_i(t)$

```
1. $ID_i \leftarrow (i, t)$ {$ID_i$ consists of a pair of values $(i, t)$, where $i$ denotes the node that started the instance, and $t$ is the time at which the instance was started. Identifiers are ordered according to the instance time, breaking ties by using the node's name. Initially $(i, 0)$ for all $i$}
2. $father_i \leftarrow i$ {$father_i$ stores the father of node $i$. Here, we mark node $i$ as an initiator. Initially nil}
3. $explored_i \leftarrow true$ {$explored_i$ is a Boolean variable that takes the value $true$ when node $i$ has been explored (i.e., it has received a GO message). Initially takes the value false}
4. $out_i \leftarrow select(V_i)$ {$V_i$ is the set of neighbors of node $i$. $select(S)$ returns a node randomly selected from $S$}
5. $eNodes_i \leftarrow out_i$ {$eNodes_i$ contains the set of nodes that have sent/received a GO message to/from node $i$. Initially $\emptyset$}
6. send $[\mathsf{GO}, ID_i, \emptyset]$ to $out_i$ {Start the exploration}
```

A pair of input–output links around a node is called a *turn*. We represent a turn around node j by a triplet (i, j, k), where $i, j, k \in V$ and $(i, j), (j, k) \in E$. The prohibition of turn (i, j, k) means that no path can contain the sublists (i, j, k) or (k, j, i). That is, no path can traverse link (j, k) after traversing (i, j) and vice versa. We are not considering the turns comprising the same link (i.e., (i, j, i)).

Algorithm 2. Code executed at node i on the reception of message $[\mathsf{GO}, d, Q]$ from node j

```
1. if $(ID_i < id)$ then
2.    $eNodes_i \leftarrow \emptyset$;
3.    $explored_i \leftarrow false$;
4.    append $(j, i)$ to $L(Q)$;
5.    $eNodes_i \leftarrow eNodes_i \cup j$;
6. end if
7. if $(explored_i = \text{false})$ then
8.    $ID_i \leftarrow id$;
9.    $explored_i \leftarrow$ true;
10.   $father_i \leftarrow j$;
11.   $out_i \leftarrow select(V_i - eNodes_i)$;
12.   if $(out_i \neq \emptyset)$ then
13.      $eNodes_i \leftarrow eNodes_i \cup out_i$;
14.      send $[\mathsf{GO}, ID_i, Q]$ to $out_i$;
15.   else
16.      send $[\mathsf{DONE}, Q]$ to $father_i$;
17.   end if
18. else
19.   $eNodes_i \leftarrow eNodes_i \cup j$;
20.   send $[\mathsf{BACK}, Q]$ to $j$
21. end if
```

2.2. Description of the protocol

In our proposal, any node can initiate the protocol, at any time, by executing the function *initiate_DMITP$_i$(t)*, where i denotes the initiating node and t is the time[1] at which the function is invoked (see Algorithm 1).

The main body of the protocol is made up of the actions associated with the reception of three types of messages: GO, BACK and DONE. The GO message is in charge of performing network exploration (see Algorithm 2). The BACK message is sent as a reply to a GO message when a cycle has been detected (see Algorithm 3). Finally, the DONE message is used to turn back when a part of the network has finished being explored (see Algorithm 4).

The core structure used throughout the execution of the protocol is formed by a set of pairs of the type $Q \equiv (L(Q), T(Q))$ (where $L(Q)$ is the set of already explored links and $T(Q)$ is a set of forbidden turns).

In Algorithm 4, the structure Q is used by the function $findTurn_i(Q)$ to detect some turns intended to be forbidden in order to transform the original topology into a feed-forward. To implement $findTurn_i(Q)$, we first have used a modified version of the well-known Floyd's cycle-finding algorithm [13] to detect cycles in Q. Then, we extracted the turns in the form $(\cdot, i, j)$, with $father_j \neq i$, that are part of some cycle in $L(Q)$ and are not already included in $T(Q)$. These turns are returned by $findTurn_i(Q)$ to be subsequently forbidden.

As it has been stated previously, *DMITP* can be initiated, independently, by several nodes, even at the same time. This can be done without any problem since *DMITP* is designed in such a way that, after completion, all nodes belong to the same instance (see Theorem 1). Furthermore, this feature can be used to cope with new nodes entering and leaving the system. In the case of an entering node, such a node only has to initiate a new instance of the protocol to create a new feed-forward topology containing it. Similarly, when a node leaves the system, either voluntarily or due to a crash, it is only necessary for one of its neighbors to initiate a new instance of the protocol.

Algorithm 3. Code executed at node i on the reception of message [BACK, Q] from node j

```
1. forbid turn $(father_i, i, j)$;
2. append $(father_i, i, j)$ to $T(Q)$;
3. if $(V_i - eNodes_i = \emptyset)$ then
4.    if $(father_i \neq i)$ then
5.       send [DONE, $Q$] to $father_i$;
6.    end if
7. else
8.    $out_i \leftarrow select(V_i - eNodes_i)$;
9.    $eNodes_i \leftarrow eNodes_i \cup out_i$;
10.   send [GO, $ID_i$, $Q$] to $out_i$;
11. end if
```

[1] We assume that, in order to be able to compare the different instances according to their stating time, a distributed clock synchronization algorithm is used (see for instance [12]).

2.3. Example of application of DMITP

To illustrate how *DMITP* works, here we provide an example of application to a simple graph (see Fig. 1), assuming that there is only one initiator.

Algorithm 4. Code executed at node i on the reception of message [DONE, Q] from node j

```
1. if $(V_i - eNodes_i = \emptyset)$ then
2.    $C \leftarrow findTurn_i(Q)$
3.    for all $c \in C$
4.       forbid turn $c$;
5.       append $c$ to $T(Q)$;
6.    end for
7.    if $(father_i \neq i)$ then
8.       send [DONE, $Q$] to $father_i$;
9.    end if
10. else
11.   $out_i \leftarrow select(V_i - eNodes_i)$;
12.   $eNodes_i \leftarrow eNodes_i \cup out_i$;
13.   send [GO, $ID_i$, $Q$] to $out_i$
14. end if
```

(a) Fig. 1(a) shows the initial scenario, which contains the following cycles: $(0,1,2,0,1)$, $(0,2,3,0,2)$ and $(0,1,2,3,0,1)$. The protocol is started by node 1, at time t. The instance identifier is $ID_1 = (1,t)$ and $Q = \emptyset$.

(b) Node 1 sends [GO, $(1,t)$, Q] to node 2 through link $(1,2)$. Upon receiving such a message, node 2 sets $ID_2 = (1,t)$, and appends (1,2) to $L(Q)$. Fig. 1(b) shows the above mentioned action.

(c) Node 2 sends [GO, $(1,t)$, Q] to node 3 through link (2,3). Then, node 3 sets $ID_3 = (1,t)$, appends (2,3) to $L(Q)$ and sends [GO, $(1,t)$, Q] to node 0.

(d) Node 0 sets $ID_0 = (1,t)$, appends (3,0) to $L(Q)$ and sends [GO, $(1,t)$, Q] to node 1. Node 1 appends $(0,1)$ to $L(Q)$ and, as it has already been explored, it replies with [BACK, Q] to node 0, which prohibits the turn (3, 0, 1) and appends (3,0,1) to $T(Q)$. Fig. 1(c) shows the resulting scenario.

(e) As node 0 still has an unexplored neighbor, it sends [GO, $(1,t)$, Q] to node 2. Since node 2 has already been explored, it also replies with [BACK, Q], which makes node 0 append (2,0) to $L(Q)$, forbid the turn (3, 0, 2) and append (3,0,2) to $T(Q)$. Fig. 1(d) shows the resulting scenario.

(f) Node 0 has explored all its adjacent nodes, so it sends [DONE, Q] to its father (i.e., node 3).

As node 3 has already explored all its adjacent nodes, it checks for undetected cycles by means of the function $findTurn_3(Q)$, finding that there are no cyclic dependencies. After that, node 3 sends [DONE, Q] to its father (i.e., node 2).

As node 2 has also explored all its neighbors, it invokes $findTurn_2(Q)$, detecting that there is a cyclic dependency that can be broken by prohibiting the turn $(1,2,0)$. There-

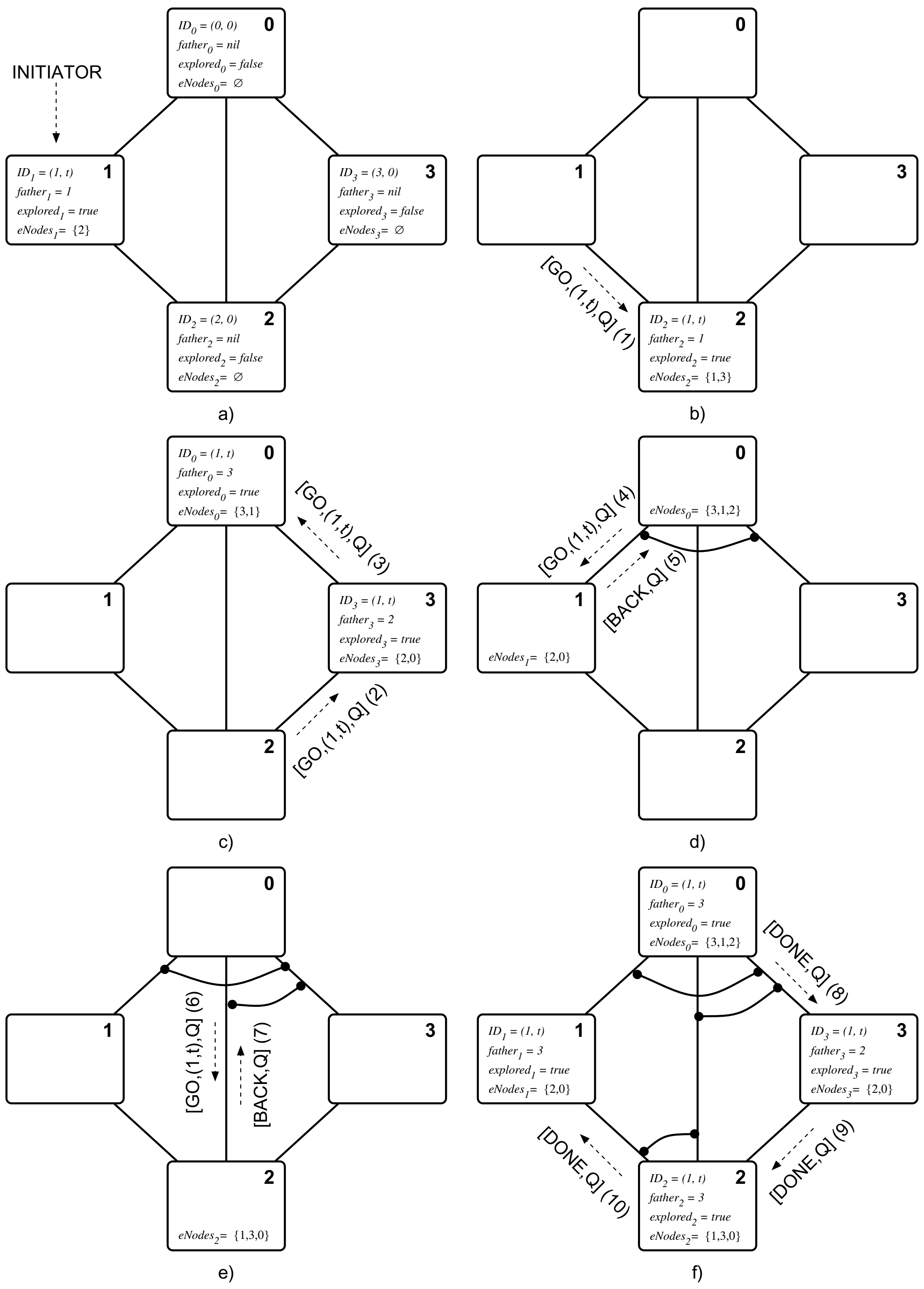
INITIATOR
$ID_0 = (0, 0)$ $father_0 = nil$ $explored_0 = false$ $eNodes_0 = \varnothing$
$ID_1 = (1, t)$ $father_1 = 1$ $explored_1 = true$ $eNodes_1 = \{2\}$
$ID_3 = (3, 0)$ $father_3 = nil$ $explored_3 = false$ $eNodes_3 = \varnothing$
$ID_2 = (2, 0)$ $father_2 = nil$ $explored_2 = false$ $eNodes_2 = \varnothing$
a)
[GO,(1,t),Q] (1)
$ID_2 = (1, t)$ $father_2 = 1$ $explored_2 = true$ $eNodes_2 = \{1,3\}$
b)
$ID_0 = (1, t)$ $father_0 = 3$ $explored_0 = true$ $eNodes_0 = \{3,1\}$
[GO,(1,t),Q] (3)
$ID_3 = (1, t)$ $father_3 = 2$ $explored_3 = true$ $eNodes_3 = \{2,0\}$
[GO,(1,t),Q] (2)
c)
$eNodes_0 = \{3,1,2\}$
[GO,(1,t),Q] (4)
[BACK,Q] (5)
$eNodes_1 = \{2,0\}$
d)
[GO,(1,t),Q] (6)
[BACK,Q] (7)
$eNodes_2 = \{1,3,0\}$
e)
$ID_0 = (1, t)$ $father_0 = 3$ $explored_0 = true$ $eNodes_0 = \{3,1,2\}$
[DONE,Q] (8)
$ID_1 = (1, t)$ $father_1 = 3$ $explored_1 = true$ $eNodes_1 = \{2,0\}$
$ID_3 = (1, t)$ $father_3 = 2$ $explored_3 = true$ $eNodes_3 = \{2,0\}$
[DONE,Q] (10)
[DONE,Q] (9)
$ID_2 = (1, t)$ $father_2 = 3$ $explored_2 = true$ $eNodes_2 = \{1,3,0\}$
f)

Fig. 1. Evolution of the execution of *DMITP* algorithm assuming that there is a single initiator (node 1). Each message has been labeled in order of occurrence. Within the boxes that represent the nodes, we show the values of the data structures when the algorithm starts and ends, and when they change as a result of receiving a message. The arcs between links show the forbidden turns.

fore, node 2 prohibits that turn, appends $(1,2,0)$ to $T(Q)$ and sends [DONE, Q] to node 1.

Since node 1 is the initiator, it ends the algorithm. Fig. 1(e) shows the situation described. The resulting topology is a cycle-free topology.

We note that, depending on the initiator node and on the order in which the network is explored, the specific prohibited turns and the number of them may differ. For instance, if the exploration of nodes had followed the path $(1,2,0,3)$, instead of $(1,2,3,0)$, our protocol would have forbidden a total of two turns (which is the minimal number of forbidden turns for transforming this topology into feed-forward).

3. Analysis of the *DMITP* protocol

In this section, we present several properties that are satisfied by the *DMITP* protocol. For the sake of clarity, here we summarize the results without including formal proofs. These proofs are deferred to the Appendix.

Perhaps the most important feature that must be shown when considering a protocol is that it behaves as expected. In our case, this means that the *DMITP* protocol must transform any network topology into a feed-forward topology, in such a way that the resulting network has to remain connected. Theorem 2 provides a formal proof for the first of these two results (i.e., it shows that the *DMITP* protocol transforms any network topology into a feed-forward topology), while Theorem 3 proves that after applying the *DMITP* protocol, the resulting network remains connected.

At this point, we would like to remark that in the two theorems mentioned above it was assumed that it was a single active instance of the protocol. However, in Theorem 1 we have shown that even when multiple executions of the *DMITP* protocol are active at the same time, after completion, all nodes belong to the same instance, which will be the greatest. According to the protocol's behavior, their different instances work in an independent fashion. In particular, when an instance arrives at a node, it first checks whether such a node "belongs" to another active instance of higher priority (if any). If so, the arriving instance stops working. Otherwise, it acts as if the node did not belong to any instance (i.e., it acts as if the new instance were the only active instance at that node). Clearly, at the very end, the instance with the highest priority will prevail over the rest, and the situation will be the same as if only such an instance were executed. Therefore, simply assuming that there was a single active instance of the protocol was enough to prove the correctness of Theorems 2 and 3 in the general case.

Regarding the performance of the *DMITP* protocol in terms of prohibited turns, in Lemma 1 it is shown that it prohibits only one turn per cycle. Furthermore, the *DMITP* protocol also prohibits at most $\frac{1}{2}$ of the total number of turns in the network (see Theorem 4). This is greater than the optimal bound provided by the protocol proposed by Starobinski et al. [11], which is a third of the total number of turns. However, it must be taken into account that, in contrast to the *DMITP* protocol, the solution proposed in [11] is a centralized solution that requires a global knowledge of the network. The maximum number of messages and steps taken by any instance of the protocol clearly depends on the number of edges E of the network topology where the protocol is executed. Specifically, it uses at most $2|E|$ messages and takes $2|E|$ time steps (see Theorem 5).

4. Experimental evaluation

In this section, we present an experimental evaluation of the performance of *DMITP*. For any particular experiment, we generate 99 different topologies using the *GT-ITM* graph generator [14,15] (33 flat random, 33 hierarchical and 33 transit-stub), averaging the results with a confidence interval of 95%. Furthermore, we also compare *DMITP* with three well-known cycle-avoidance protocols, namely, the Spanning Tree [16], the Up/Down [10] and the Turn-Prohibition [11] protocols.

4.1. First experiment

Our first experiment analyzes how the increase in the number of nodes affects the percentage of prohibited turns. We have generated topologies with an average network degree of 4, varying the number of nodes from 16 to 255.

Fig. 2 shows that *DMITP* performs much better than the Spanning Tree, slightly better than the Up/Down and slightly worse than the Turn-Prohibition (note that, whereas in relative terms *DMITP* performs 50% worse than the Turn-Prohibition, in absolute terms it is less than 5% worse). However, it must be taken into account that unlike *DMITP* which is fully distributed, both the Up/Down and the Turn-Prohibition are centralized protocols (i.e., they need to know the whole network topology).

Furthermore, Fig. 2 shows that, although Theorem 4 bounds the maximum number of prohibited turns provided by *DMITP* by 50% of the total, in practice it does not exceed 25%.

4.2. Second experiment

In our second experiment, we analyze how an increase in the network degree affects the percentage of prohibited turns for the different turn-prohibition protocols. We have generated topologies with a fixed number of nodes equal to 120, varying the network degree from 4 to 10.

Fig. 3 confirms that *DMITP* performs much better than the Spanning Tree, slightly better than the Up/Down and slightly worse than the Turn-Prohibition. Furthermore,

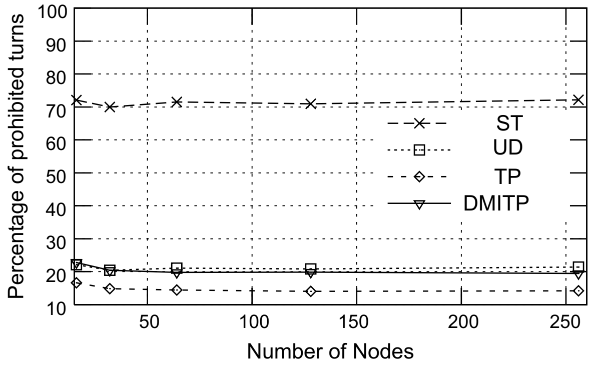

Fig. 2. Percentage of prohibited turns when varying the number of nodes (for a fixed network degree of 4).

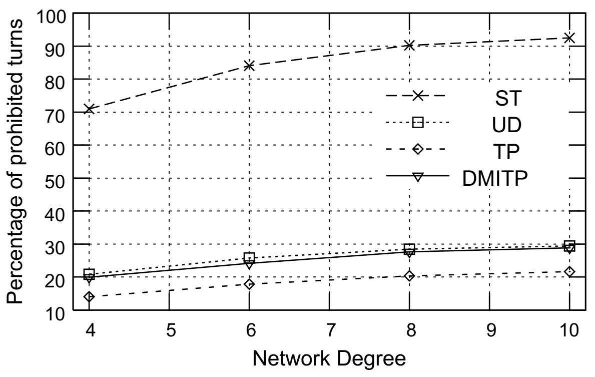

Fig. 3. Percentage of prohibited turns when varying the network degree (for a fixed number of nodes of 120).

Fig. 3 also confirms the observation made in the previous experiment (i.e., the worst case bound provided by Theorem 4 is never reached in practice).

Overall and as a result of our experimental results, we have shown that *DMITP* presents both the advantages of the Spanning Tree protocol (i.e., it is fully distributed) and of the Up/Down and Turn-Prohibition protocols (i.e., it is very efficient).

5. Conclusions

In this paper, a distributed protocol has been presented, which, when applied to any network topology, transforms it into a feed-forward, while maintaining network connectivity. The protocol is based on the turn-prohibition mechanism, which, as has been shown here, is more efficient than forbidding the use of complete links. In contrast to centralized solutions, where a single node performs all the actions and broadcasts the resulting topology after completion, *DMITP* works in a distributed fashion, forbidding the turns while carrying out the network exploration.

An interesting property of *DMITP* is that several nodes can initiate it independently, and even at the same time. This feature can be used to cope with new nodes entering and leaving the system.

Finally, we evaluated the performance of our algorithm, showing that it presents both the advantages of the Spanning Tree protocol (i.e., it is fully distributed) and of the Up/Down and Turn-Prohibition protocols (i.e., it is very efficient).

Appendix. Correctness proofs

Theorem 1. *The DMITP protocol guarantees that, after completion, all nodes will belong to the same instance (which will be the greatest one), regardless of the number of initiated instances.*

Proof. Assume that multiple initiators start a *DMITP* execution and assume, by the way of contradiction, that when all the executions have finished, not all nodes belong to the highest instance.

Let i be a node whose current instance is not the highest one. By taking into account the protocol's behavior, node i will eventually receive a GO message from another node with the highest instance. However, messages with the highest instance are never discarded (line 1 of Algorithm 2). So, node i will join the highest instance (line 8 of Algorithm 2) and will discard any message belonging to a lower instance. This contradicts our assumption and proves the Theorem. □

Theorem 2. *The DMITP protocol transforms any network topology into a feed-forward topology.*

Proof. We perform the demonstration by contradiction. Assume that after applying the *DMITP* protocol to a network topology, it is still possible to create, at least, one cycle. Let us call it c, and denote as i the first node in c that receives a GO message (note that in a connected graph, all the nodes, except the initiator, receive a GO message at least once).

We base our proof on the following two properties:

- Node i will execute $findTurn_i(Q)$, for some Q: Since node i is the first one in c that receives a GO message, it will have at least one "unexplored" adjacent node. Therefore, when it receives a GO message for the first time, it marks itself as *explored* (line 10 of Algorithm 2) and sends a GO message to one of its unexplored adjacent nodes (line 16 of Algorithm 2), which will mark node i as its father (line 10 of Algorithm 2). Therefore, node i will have, at least, one son. However, since each node eventually receives a DONE message from any of its sons (i.e., the condition in line 1 of Algorithm 4 is fulfilled), then node i will execute $findTurn_i(Q)$ (line 2 of Algorithm 4). This proves the property.
- When node i executes $findTurn_i(Q)$, $L(Q)$ already contains the set of links that form c: By contradiction. Assume that when node i executes $findTurn_i(Q)$, $L(Q)$ does not contain a given link l belonging to c. Let us denote the nodes that connect link l as m and n.
  Since node i is the first one in c that receives a GO message, then it has necessarily started the exploration process (line 14 of Algorithm 2). Furthermore, node i can execute $findTurn_i(Q)$ only if nodes m and n have sent it a DONE message through link l (line 1 of Algorithm 4). But, in this case, l will be included in $L(Q)$, thus reaching a contradiction.

Therefore, from the two above mentioned properties, we know that node i will execute $findTurn_i(Q)$ and will return a turn that, once prohibited, will break c (line 4 of Algorithm 4), thus contradicting the initial hypothesis. □

Theorem 3. *After applying the DMITP protocol the resulting network remains connected.*

Proof. During any execution of the *DMITP* protocol, a spanning tree is configured, which is rooted in the initiator node, and which contains all the nodes of the system.

Therefore, by using the links of the spanning tree, connectivity is guaranteed. However, it is necessary to prove that the *DMITP* protocol does not forbid any turn belonging to that spanning tree.

The *DMITP* protocol performs the prohibition of a turn (line 1 of Algorithm 3 and line 4 of Algorithm 4) in either of the following two scenarios:

- Node i receives a BACK message from node j. In this case the turn $(father_i, i, j)$ is forbidden. Thus, it has to be proved that such a turn does not belong to the spanning tree. But the spanning tree is formed by the links where GO and DONE messages have been sent, and this is not the case for the link (i, j).
- The turns are returned by $findTurn_i(Q)$ (line 2 of Algorithm 4). In this case, the forbidden turns will be of the form $(\cdot, i, j)$, where, $father_j \neq i$. Therefore, neither the link between node i and node j, nor the turn $(\cdot, i, j)$ belong to the spanning tree. □

Lemma 1. *The DMITP protocol prohibits exactly one turn per cycle in any network topology.*

Proof. We know that turns are prohibited either when detecting a cycle during the network exploration (line 1 of Algorithm 3), or as a result of invoking $findTurn_i(Q)$ (line 4 of Algorithm 4). So, we make a case analysis.

(1) Turns forbidden only during the network exploration: By contradiction. Assume that the same cycle causes the prohibition of more than one turn during the network exploration. This implies that during the exploration at least two nodes of the same cycle have received a BACK message through the links involved in the cycle. However, this situation is not possible, thus reaching a contradiction.
(2) Turns forbidden only by $findTurn_i(Q)$: Any invocation of $findTurn_i(Q)$ will obtain the turns forbidden by previous invocations and, therefore, it only forbids one turn per cycle.
(3) Turns forbidden both during the network exploration and by $findTurn_i(Q)$: In this case, we show that the same cycle can not cause the prohibition of one turn during the network exploration, and another turn during the invocation of $findTurn_i(Q)$. Assume, by way of contradiction, that a given cycle causes the prohibition of two turns, one during the network exploration and the other during the invocation of $findTurn_i(Q)$. Due to the protocol's behavior, the first turn must be forbidden during the exploration. Therefore, when $findTurn_i(Q)$ is invoked, $L(Q)$ will already contain such a cycle and its associated turn. This will prevent $findTurn_i(Q)$ from forbidding another turn to break that cycle. □

Theorem 4. *The DMITP protocol prohibits at most $\frac{1}{2}$ of the total number of turns in any network topology.*

Proof. By Lemma 1, we have that the *DMITP* protocol breaks all cycles of the topology by prohibiting only one turn per cycle. Here, we will prove that, if we prohibit only one turn per cycle then, at most half of the total number of possible turns are prohibited.

The proof is performed by induction on the number of nodes (n) in the network.

Base case: When $n = 1$ it is trivially true.
Inductive Hypothesis: Assume that it is true for $n = i - 1$.
Inductive Step: Now, we add a new node, denoted i, to the previous network topology.

Regarding the number of new turns, we have that it is equal to $(d_i(d_i - 1))/2 + \sum_{j\in V_i}(d_j - 1)$, where V_i denotes the set of neighbors of node i and d_j denotes the degree of node j. The first component is due to the turns created by the new node and its neighbors, and the second one due to the turns where one of the nodes is not in V_i.

Taking into account that cycles that contain node i must include two of the new turns, then the maximum number of turns that must be prohibited to break each new cycle is $\sum_{j\in V_i}(d_j - 1)/2$, which is less than half of the total number of new turns. This concludes the proof. □

Theorem 5. *Any instance of the DMITP protocol uses, at most, $2|E|$ messages and takes $2|E|$ time steps, E being the number of edges of the network topology where the protocol is executed.*

Proof. Since the "token" (i.e., the messages of the protocol's execution taken together and in order) is sent once in each direction through each link, then $2|E|$ messages are passed before the protocol terminates, and it takes $2|E|$ time steps (since each message takes one time unit to be transmitted). □

References

[1] R.L. Cruz, A calculus for network delay. Part I: Network elements in isolation, IEEE Transactions on Information Theory 37 (1) (1991) 114–131.
[2] M. Andrews, B. Awerbuch, A. Fernandez, T. Leighton, Z. Liu, J. Kleinberg, Universal-stability results and performance bounds for greedy contention-resolution protocols, Journal of the ACM (2001) 39–69.
[3] M. Karol, S. Golestani, D. Lee, Prevention of deadlocks and livelocks in lossless backpressured packet networks, IEEE/ACM Transactions on Networking 11 (6) (2003) 923–934.
[4] 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) (1996) 841–851.
[5] B. Haverkort, Performance of Computer Communication Systems: A Model-Based Approach, John Wiley & Sons, 1999.

[6] C. Chang, Performance Guarantees in Communication Networks, Springer-Verlag, New York, 2000.

[7] L. Bosack, C. Hedrick, Problems in large LANs, IEEE Network 2 (1) (1988) 49–56.

[8] Metro ethernet networks – a technical overview, Metro Ethernet Forum White Paper. Available from: <http://www.metroethernetforum.org> (2002).

[9] M. Fidler, G. Einhoff, Routing in turn-prohibition based feed-forward networks, in: Networking, LNCS, vol. 3042, 2004, pp. 1168–1179.

[10] M. Schroeder, A. Birrell, M. Burrows, H. Murray, R. Needham, T. Rodeheffer, Autonet: A high-speed, self-configuring local area network using point-to-point links, IEEE JSAC 9 (8) (1991) 1318–1335.

[11] D. Starobinski, M. Karpovsky, L. Zakrevski, Application of network calculus to general topologies using turn-prohibition, IEEE/ACM Transactions on Networking 11 (3) (2003) 411–421.

[12] H. Attiya, J. Welch, Distributed Computing: Fundamentals, Simulations and Advanced Topics, Wiley and Sons, 2004.

[13] R.W. Floyd, Nondeterministic algorithms, Journal of the ACM 14 (4) (1967) 636–644.

[14] K. Calvert, M. Doar, E.W. Zegura, Modeling internet topology, IEEE Communications Magazine 35 (6) (1997) 160–163.

[15] E.W. Zegura, K. Calvert, J. Donahoo, A quantitative comparison of graph-based models for internet topology, IEEE/ACM Transactions on Networking 5 (6) (1997) 770–783.

[16] R. Perlman, Interconnections, Addison-Wesley, 2000.