# Application of Network Calculus to General Topologies using Turn-Prohibition

David Starobinski, Mark Karpovsky, and Lev Zakrevski

*Abstract*— **Network calculus is known to apply in general only to feed-forward routing networks, i.e., networks where routes do not create cycles of interdependent packet flows. In this paper, we address the problem of using network calculus in networks of arbitrary topology. For this purpose, we introduce a novel algorithm, called turn-prohibition (TP), that breaks all the cycles in a network and, thus, prevents any interdependence between flows. We prove that the TP-algorithm prohibits the use of at most 1/3 of the total number turns in a network, for any network topology. Using analysis and simulation, we show that the TP-algorithm significantly outperforms other approaches for breaking cycles, such as the spanning tree and up/down routing algorithms, in terms of network utilization and delay bounds. Our simulation results also show that the network utilization achieved with the TP-algorithm is within a factor of two of the maximum theoretical network utilization, for networks of up to 50 nodes of degree four. Thus, in many practical cases, the restriction of network calculus to feed-forward routing networks may not represent a significant limitation.**

## I. INTRODUCTION

NETWORK calculus is a general paradigm for the provision of Quality of Service (QoS) in communication networks [6], [9], [17]. The main principle of network calculus is to show that if all the input flows to a network satisfy a certain set of constraints, then so do all the flows within the network. The formulation of the constraints is simple enough to allow the computation of bounds on various performance measures, such as delay and queue length, at each element of the network.

A well-known network calculus is the $(\sigma, \rho)$ calculus, first introduced in [9] and further developed in [18] which provides deterministic bounds on delay and buffering requirements in a communication network. This model is useful for applications requiring deterministic QoS guarantees [21].

The network calculus framework applies also to statistical services [4], [14]. In particular, the minimum envelop rate (MER) [5] and exponentially bounded burstiness (EBB) [27] network calculi provide exponential bounds on various metrics of interest. More recently, a network calculus, termed stochastically bounded burstiness (SBB) calculus [23], [25], was developed in order to capture the multiple time-scale and self-similar behavior of the traffic, as observed over the Internet [16], [19], [26]. The SBB calculus provides general stochastic bounds at each node of a network.

A central problem shared by all network calculi is of determining the conditions under which a network is stable, meaning that the queue length at each element of the network is bounded according to some appropriate metric [5], [10]. It turns out that network stability is easy to establish only for *feed-forward* routing networks, i.e., networks where routes do not create cycles of interdependent packet flows. Such network are stable if and only if the traffic load (utilization) at each element is smaller than one [5], [10]. This condition is known as the *throughput condition* [27]. The case of non-feed-forward networks is generally much more complicated, with only a few notable exceptions (e.g. [18]). While the throughput condition remains necessary for the stability of such networks, it is no longer sufficient. A number of examples given in [15] illustrate this fact. Recent results show that even networks of FIFO queues with $(\sigma, \rho)$ sessions may be unstable [1].

An upper bound on the delay in arbitrary non-feed-forward networks has recently been derived in [7]. Unfortunately, this bound is useful only for very small link utilization. Specifically, the maximum achievable link utilization is inversely proportional to the maximum route length of any flow in the network. For instance, for a network diameter of 6 hops, the maximum utilization on any link does not exceed 20%.

In summary, network calculus is mostly useful for feed-forward routing networks. This fact leads to the natural question of how network calculus can be applied to networks of arbitrary topology. The main contribution of this paper is to take on this problem and provide an efficient solution to it.

Our approach is to pro-actively break all the possible cycles in a network, by prohibiting (disabling) the use of some of the network resources. This way, any interdependence between different flows can be prevented. The main challenge with this approach resides in minimizing the amount of resources that need to be prohibited.

The simplest approach for breaking cycles in a network is to construct a spanning tree and prohibit the use of links not belonging to the tree. However, a spanning tree approach is highly inefficient since a large number links are unused, and links close to the root become congested.

Instead, we propose to resort to a more sophisticated approach based on the prohibition of *turns*. Here, a turn is defined as a specific pair of input-output links around a node [11]. The main claim is that in order to break all the cycles in a network, it is sufficient to prohibit a set of turns instead of a set of links, as is the case with spanning trees (a turn $(a, b, c)$ around some node $b$ is prohibited if not packets can be forwarded from link $(a, b)$ to link $(b, c)$). For instance, while a spanning tree may fully prohibit the transmission of any packet through an output link of some node, a turn-prohibition approach may allow the use of this link, as long as packets arrive from a pre-determined set of input links.

In this paper, we introduce a novel algorithm using this approach, called the turn-prohibition algorithm (TP-algorithm) [28]. This algorithm ensures that all the cycles in a network are broken, while maintaining global connectivity. Moreover, for *any* network topology, it never prohibits more

D. Starobinski and M. Karpovsky are with the ECE department at Boston University. E-mail: {staro,markkar}@bu.edu.

L. Zakrevski is with the ECE department at the New Jersey Institute of Technology. E-mail: zakr@adm.njit.edu.

The work of the second and third authors was supported in part by the National Science Foundation under Grant MIP 9630096

than 1/3 of the total number of turns in the network. This property provides a meaningful upper bound on the maximum amount of resources that need to be sacrificed in order to guarantee a cycle-free network. The TP-algorithm exhibits a reasonable computational complexity that is a polynomial in the number of nodes. The proposed approach applies to network nodes with general, non-blocking switching architectures.

It is worth noting that the TP-algorithm is not the first algorithm based on the concept of turn-prohibition. In particular, the up/down routing scheme, developed in the context of a local area network called Autonet, uses a similar concept [22]. However, this scheme does not systematically attempt at minimizing the amount of prohibited turns in the network and its performance is much less predictable. In particular, we show in the sequel that the fraction of turns prohibited by this scheme may tend to 1 in some networks. Furthermore, our numerical results in Section V show that the TP-algorithm typically achieves a throughput 10% to 20% higher than the up/down algorithm.

This paper is organized as follows. In Section II, we introduce our graph-theoretic model of the network and related notations. Next, in Section III, we summarize the spanning tree and up/down routing algorithms and indicate their limitations. Then, in Section IV, we introduce the TP-algorithm, prove its main properties, and illustrate it with concrete examples. Our simulation results are presented in Section V, where we compare the throughput and delay bounds achieved by the three algorithms. The last section of this paper is devoted to concluding remarks.

## II. Model

We model a network by a directed graph. We define the graph $G$ to be a collection of $N$ nodes and $M$ links. A pair $(n_1, n_2)$ denotes a link directed from node $n_1$ to node $n_2$.

We restrict our attention to the typical case of *bi-directional* network topologies, that is networks where nodes are connected by bi-directional links. We will define the *degree* of a node as the number of output links of the node. For a bi-directional graph, this number is equivalent to the number of input links to the node.

A *path* from nodes $n_1$ to $n_\ell$ in a graph $G$, is a sequence of nodes $(n_1, n_2, n_3, \ldots, n_{\ell-1}, n_\ell)$, such that each two subsequent nodes are connected by a link. We say that a graph $G$ is *connected*, if for each node $i$ there exists a path to every other node $j$.

A *cycle* in $G$ is defined to be a path where the first *link* and the last *link* of the path are the same, for instance, $(n_1, n_2, n_3, \ldots, n_{\ell-1}, n_\ell, n_1, n_2)$. Note that the literature in graph-theory typically defines a cycle as a path such that the first node and the last node in the path are the same [8]. We will refer to this latter definition as a *cycle of nodes*.

Breaking all cycles of nodes is a too strong requirement for network calculus. For instance, referring to Figure 1, the path $(4, 1, 2, 4, 1)$ creates a cycle in the network, while the path $(3, 4, 1, 2, 4, 6)$ does not (although it contains a cycle of nodes). A cycle is thus created only when the same port (or link) is visited twice. 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 $(a, b, c)$ will be used to represent a turn around node $b$ from link $(a, b)$ to link $(b, c)$. For instance, in Fig. 1, the three-tuple $(1, 2, 4)$ represent the turn from link $(1, 2)$ to link $(2, 4)$ around node 2.

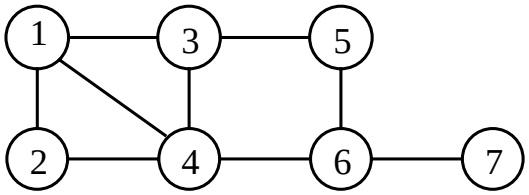


Fig. 1. Simple example of a connected graph

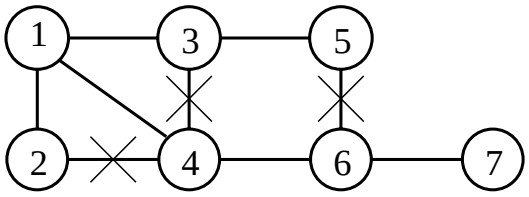


Fig. 2. A spanning tree for the the graph of Fig. 1. A link with an X represents a prohibited link.

Due to the symmetrical nature of bi-directional graphs, we will consider the turns $(a, b, c)$ and $(c, b, a)$ to be the same turn. The number of turns around a node of degree $d_i$ is equal to $d_i(d_i - 1)/2$. For instance, in Fig. 1, there are six turns around node 4.

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, prohibiting the turn $(1, 3, 4)$ means that no packet can be forwarded from link $(1, 3)$ to link $(3, 4)$ (and from link $(4, 3)$ to link $(3, 1)$).

## III. Summary of Previous Approaches for Breaking Cycles

In this section, we summarize two earlier approaches that preserve network connectivity and break all the cycles in a network of arbitrary topology.

### A. Spanning Trees

The simplest approach for breaking all the cycles in a graph is to construct a *spanning tree*. A spanning tree is a connected sub-graph of $G$ that includes all the nodes of $G$ and does not contain any cycle of nodes.

Spanning trees can be generated in various ways. One involves picking a root node at random (for instance, the node with the lowest id), and construct a shortest path tree using a Breadth First Search (BFS) procedure [8]. An example of a spanning tree generated that way is depicted in Fig. 2. In this example, node 1 is chosen as the root.

A major drawback of the spanning tree approach is that a large number of links are unused. Moreover, links near the root of the spanning tree get congested, thus limiting the throughput of the whole network.

### B. Up/Down Routing

The spanning tree approach turns out to be too restrictive in preventing all cycles of nodes. As explained earlier, cycles in

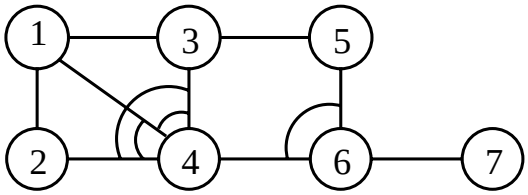


Fig. 3. An up/down spanning tree for the the graph of Fig. 1. An arc represents a prohibited turn.

networks arise only when a path can traverse the same link twice or more. Instead of fully prohibiting use of links, a more efficient approach is to prohibit use of turns.

With the *up/down routing* algorithm [22], a spanning tree is first constructed, and 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 from the root). Nodes at the same level are ordered arbitrarily. Such an ordering of the nodes in a spanning tree is illustrated in Fig. 2 where node 1 is the root of the tree, nodes 2, 3 and 4 are level 1 nodes, nodes 5 and 6 are level 2 nodes, and node 7 is a level 3 node. Note that if another node were chosen as the root, then the labels of the nodes would be changed.

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 $(a, b, c)$ is referred to as an up/down turn if node if $a > b$ and $c > b$. Respectively, a down/up turn is a turn such that $a < b$ and $c < b$. A key observation is that any cycle must involve at least one up/down turn and one down/up turn. Therefore all the cycles in a graph can be broken by prohibiting all the down/up turns. This also means that all the other types of turns can be permitted. Thus, packets are allowed to traverse links not belonging to the spanning tree as long as they are not forwarded along down/up turns. An up/down spanning tree, for the graph of Fig. 1, is depicted in Fig. 3. The arcs represent the prohibited turns. For instance, around node 4, turns $(2, 4, 3)$, $(2, 4, 1)$ and $(1, 4, 3)$ are all prohibited. On the other hand, turn $(2, 4, 6)$ is permitted. Thus, the up/down spanning tree approach still allows to make use of link $(2, 4)$, unlike the simple spanning tree approach.

The up/down routing algorithm leads to markedly better performance than the simple spanning tree approach. Nevertheless, links near the root still remain more likely to get congested than other links in the network.

Another problem is that the performance of the up/down scheme depends critically on the selection of the spanning tree and on which node is chosen as the root of the tree. To illustrate this fact, Figures 4 and 5 show the same graph, but with different node labeling. We note that the total number of turns in this graph is on the order of $N^2$. For the case of Fig. 4, only the turns $(i, i + 1, 1)$, $1 < i < N$, are prohibited. The fraction of prohibited turns is thus on the order of $1/N$, and tends to 0 as $N$ gets very large. On the other hand, for the case depicted in Fig. 5, all the turns around node $N$ are prohibited, and the fraction of prohibited turn in the graph tend to 1 as $N$ gets very large. In general, the the up/down scheme will perform somewhere between these two extremes depending on the location of the root.

*Example 1:* In order to make the previous example concrete in term of delay bounds, assume that a $(\sigma, \rho)$ session is established

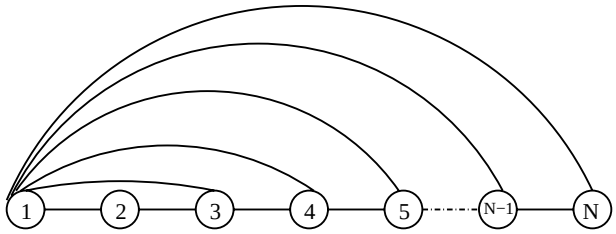


Fig. 4. Up/Down: A node labeling resulting in a small fraction of prohibited turns.

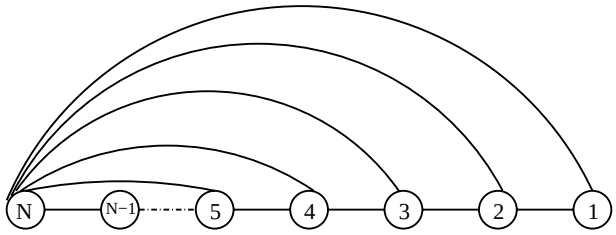


Fig. 5. Up/Down: A node labeling resulting in a large fraction of prohibited turns.

in both directions between each pair of nodes in the network of Figs. 4 and 5. Here, the parameters $\rho$ and $\sigma$ correspond to the bucket rate and bucket depth (burstiness), respectively. Assume further that flow shapers are implemented at each node such that the $(\sigma, \rho)$ characterization of each flow remains the same within the network [17]. Each node implements a FIFO output queueing architecture, and the capacity of each link is $C$. Assuming that the throughput condition is satisfied on each link, the delay bound for $k$ flows sharing a same link is $\bar{D} = k\sigma/C$ [9]. For the case of Fig. 4, each node can communicate with each other node within two hops, via node 1. Therefore, in this case, an end-to-end delay bound for each flow will be $\bar{D} = 2(N-1)\sigma/C$, since $N - 1$ flows share each link starting from node 1 or ending at it. For the case of Fig. 5, all the communication needs to be done over the bottom links, that is over the links $(i, i + 1)$. It can easily be checked that the delay bound on each link $(i, i + 1)$ is $\bar{D} = i(N - i)\sigma/C$, which can be on the order of $\bar{D} = N^2\sigma/C$. Worse, since a flow between node 1 and node $N$ traverses $N - 1$ hops, the resulting end-to-end delay bound for this flow is on the order of $\bar{D} = N^3\sigma/C$. This shows that the selection of the root can have a tremendous effect on the network performance and the delay bounds. ∎

In summary, the choice of a "good" spanning tree is critical for the performance of the up/down routing approach. Unfortunately, there are currently no simple methods to find the tree that minimizes the amount of prohibited turns, except for performing an exhaustive, prohibitive search.

## IV. The Turn-Prohibition Algorithm

In this section, we describe a new algorithm that provides a robust upper bound on the amount of restricted network resources [28]. In particular, the algorithm guarantees that the fraction of prohibited turns, for *any* given graph, never exceeds 1/3. This is in contrast to the up/down routing algorithm, for which the fraction of prohibited turns may tend to 1, as shown

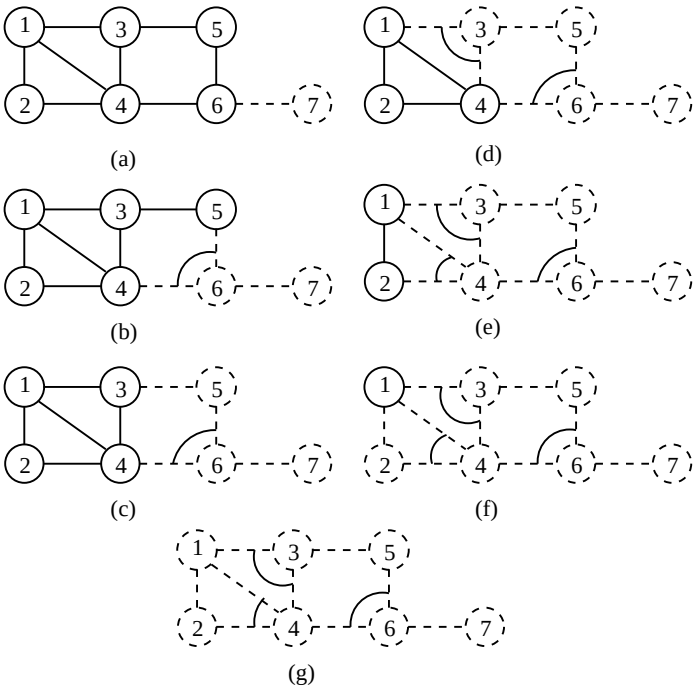


Fig. 6. Successive iterations of the TP-algorithm

in the previous section.

The turn-prohibition algorithm does not make use of a spanning tree. Instead, it considers iteratively each node in the network and uses turn prohibition to break all the possible cycles involving the node under consideration. At each step, the algorithm selects the node of minimal degree. We will show that this selection is the key in guaranteeing that the fraction of prohibited turns never exceeds 1/3. A basic version of the TP-algorithm is described in more details in the next section.

A major complication comes from the connectivity requirement, that is, the requirement that all the nodes must remain connected at the end of the algorithm. The basic version of the TP-algorithm does not generally guarantee connectivity. In Section IV-B, we describe a full version of the TP-algorithm that breaks all the cycles, preserves connectivity, and guarantees that no more than 1/3 of the turn are prohibited. This algorithm is recursive in nature. Note that the up/down routing algorithm automatically satisfies the connectivity requirement by first constructing a spanning tree.

### *A. Basic Version*

We first introduce a simpler version of the TP-algorithm, that forms the basis of the full version. The algorithm performs the following iteration:

*Step 1:* Select a node of minimal degree. Denote this node as node $a$.
*Step 2:* Prohibit all the turns around node $a$, that is, prohibit all the turns of the type $(b, a, c)$.
*Step 3:* Permit all the turns starting from node $a$, that is, permit all the turns of the type $(a, b, c)$.
*Step 4:* Delete node $a$ and the links incident to it, and repeat the procedure until all the nodes in the graph have been deleted.

The iterations of the TP-algorithm are illustrated in Fig. 6. First, node 7 is selected. There is no turn around node 7, and thus no turn is prohibited. Turns $(7, 6, 5)$ and $(7, 6, 4)$ are permitted. Node 7 and link $(6, 7)$ are then deleted and the procedure is repeated. At the next stage, either of nodes 2, 5 or 6 can be selected, as they are all of the same minimal degree 2. Assume that node 6 is selected. Then, the turn $(4, 6, 5)$ is prohibited, and the turns $(6, 4, 1)$, $(6, 4, 2)$, $(6, 4, 3)$ and $(6, 5, 3)$ are all permitted. Node 6 and links $(4, 6)$ and $(5, 6)$ are now deleted from the graph. The procedure continues until all the nodes have been considered. The final set of prohibited turns is $(4, 6, 5)$, $(1, 3, 4)$ and $(2, 4, 1)$ (other solutions are also possible). Note that this one turn less than for the up/down spanning tree example given in the previous section.

The TP-algorithm satisfies the following properties:

**Theorem 1** *The TP-algorithm breaks all the cycles.*

*Proof:* The proof is by induction. The induction hypothesis is that, at each step of the algorithm, all the cycles involving a deleted node have already been broken. This hypothesis is clearly true for the first node selected, since all the turns around it are prohibited. Now suppose that $i$ nodes have already been deleted, and all the cycles involving them have been broken. The next node under consideration is, say, node $n_{i+1}$. We distinguish between two types of turns around node $n_{i+1}$. First, we consider the turns that involve at least one node that has already been deleted. These turns have been permitted in one of the previous steps of the algorithm, but can not lead to a cycle since, by the induction hypothesis, all the cycles involving a deleted node have already been broken. Second, we consider the turns around node $n_{i+1}$ that do not involve a previously deleted node. The TP-algorithm prohibits all these turns, and thus breaks all the remaining cycles that could have involved node $n_{i+1}$. The induction hypothesis is thus justified, and the proof of the claim is complete. ∎

*Remark:* Consider the last node deleted by the algorithm. All the turns around it are permitted. Thus, no cycle of nodes can originate from it, since otherwise a cycle (of links) would automatically be created. In other words, no path can traverse the last deleted node more than once. We will use this property in the sequel.

**Theorem 2** *The TP-algorithm prohibits at most 1/3 of the turns.*

*Proof:* At each step, the algorithm selects a node of minimal degree. Suppose that the selected node $n_i$ has a degree $d_i$. The total number of prohibited around node $n_i$ is $d_i(d_i - 1)/2$.

By definition, node $n_i$ has $d_i$ neighbors. Each neighbor $n_j$ has a degree, $d_j$, that is larger or equal than $d_i$. The number of permitted turns starting from node $n_i$ and involving node a neighbor $n_j$ is $d_j - 1$. The total number of permitted turns starting from $n_i$ is thus $\sum_{j=1}^{d_i} d_j - 1 \geq d_i(d_i - 1)$. This quantity is at least twice as large as the number of prohibited turns. Thus, at most 1/3 of the turns are prohibited. ∎

**Theorem 3** *The basic version of the TP-algorithm preserves connectivity under the assumption that, at each step of the algorithm, the graph consisting of the non-deleted nodes remains connected.*

*Proof:* Suppose that, at step $i$, node $n_i$ is deleted. We select a link from node $n_i$ to one of its non-deleted neighbors, say node $n_j$, and refer to the link as a *selected link*. For instance in Fig. 6, the links $(7, 6)$, $(6, 4)$, $(5, 3)$, etc., are successively picked as selected links. Note that the TP-algorithm guarantees that all the turns between selected links are permitted. From the assumption, once node $n_i$ has been deleted, there remains a connected graph of $N - i$ non-deleted nodes. The same procedure is repeated for each of these nodes. As the algorithm terminates, we end up with a graph of $N$ nodes and $N - 1$ (selected) links. The only graph that satisfies such a property is a spanning tree. Thus, connectivity is guaranteed. ∎

### *B. Full Version*

Unfortunately, the assumption, on which Theorem 3 is based, does not hold in general, as illustrated in Fig 7. Suppose that the first node selected is node 7, and turn $(6, 7, 8)$ is prohibited. Then, the graph is broken into two components of connectivity, namely, nodes $\{1, 2, 3, 4, 5, 6\}$ and $\{8, 9, 10, 11\}$, and the global connectivity is lost.

In general, once all the turns around some node $a$ have been prohibited, the remaining nodes in the graph will be split into $K \geq 1$ different components of connectivity $G_1, G_2, \ldots, G_K$. In order to preserve global connectivity, the full version of the algorithm must permit some of the turns around node $a$. To this end, the full algorithm select $K$ links connecting node $a$ to each component. These links are referred to as *special links*. All the turns between special links are permitted. In Fig. 8, the special links are links $(7, 6)$ and $(7, 8)$.

At a first glance, it seems that all what remains to do is to recursively run the algorithm within each component of connectivity. Unfortunately, such a scheme would indeed break all the cycles *within* components of connectivity, but not necessarily cycles *across* components of connectivity. This fact is illustrated in Fig. 8. Here the turn $(6, 7, 8)$ has been permitted (to guarantee connectivity) and the TP-algorithm has been run within each component connectivity. Cycles within the components have indeed been broken, but cycles across components, such as the cycle $(7, 8, 9, 10, 11, 8, 7, 6, 5, 3, 4, 6, 7, 8)$, still exist.

Note that cycles across components of connectivity arise only due to the presence of cycles of nodes within components. Specifically, for each component, these cycles of nodes must start from the node connected to the special link (in Fig. 8, these are nodes 6 and 8).

The solution to this problem consists of preventing cycles of nodes that originate from a node connected to a special link. We remind now the property that that no cycles of nodes can originate from the last node deleted by the TP-algorithm (see the remark following Theorem 1). We are now going to take advantage of this property and make sure that, for each component of connectivity, the node connected to the special link will be the last one to be deleted. For this purpose, we will mark the node and refer to it as a *special node*. By making sure that the special node is the last one to be deleted, the full TP-algorithm guarantees that no cycles on node will originate from it.

Due to the recursive nature of the TP-algorithm, one of the $K$ components of connectivity, say component 1, may already contain a special node. For this component, the node connected to the selected link is not marked as special. Thus, a cycle of nodes originating from this node may exist. However, cycles across components are still prevented since none of the other components of connectivity contains a cycle of nodes (a cycle across components of connectivity arises only if at least two components contain a cycle of nodes originating from the node connected to the special link). The following theorem summarizes our results:

**Theorem 4** *The full version of the TP-algorithm breaks all the cycles and preserves global connectivity.*

The formal description of the full TP-algorithm follows. It is based on a recursive procedure **TP(**$G'$**)**, where the argument $G'$ represents a component of connectivity:

Procedure **TP(**$G'$**)** :

*Step 1:* Select the node of minimal degree in $G'$, excluding the special node (if there is such one). If several nodes of minimal degree are available, then select first a node that is not a neighbor of the special node. Denote the selected node as node $a$.

*Step 2:* Prohibit all the turns around node $a$, that is, prohibit all the turns of the type $(b, a, c)$.

*Step 3:* Permit all the turns starting from node $a$, that is, permit all the turns of the type $(a, b, c)$.

*Step 4:* If the remaining graph is broken into $K \geq 1$ components of connectivity $G_1, G_2, \ldots, G_K$, then select $K$ special links connecting node $a$ to each component, and permit all the turns between the special links.

*Step 5:* If a special node exists in $G'$, then it should be in $G_1$. For each of the other components of connectivity $G_2, G_3, \ldots, G_K$, the node connected to the special link is marked as a special node.

*Step 6:* Delete node $a$ and the links incident to it. If there is only one remaining node in $G'$ then delete it and return the procedure. Otherwise, invoke recursively the procedure for each component of connectivity, that is, perform **TP**($G_1$),**TP**($G2$),. . .,**TP**($G_K$).

The algorithm is started by invoking **TP**($G$), where $G$ corresponds to the whole graph. Note that initially, no node is marked as special. An illustration of the results of the TP-algorithm is given in Fig. 9. In this case, we suppose that node 7 is the first selected node. After deleting node 7 and the links incident to it, the graph is broken into two components of connectivity. Then, links $(7, 6)$ and $(7, 8)$ are selected as special links and node 8 is selected as a special node. Note that node 6 could also have been selected as a special node, but this is unnecessary since $K$ components of connectivity require only $K - 1$ special nodes, as explained earlier.

We now show that the first step of the algorithm guarantees that at most 1/3 of the turns are prohibited.

**Theorem 5** *The full version of the TP-algorithm prohibits at most 1/3 of the turns.*

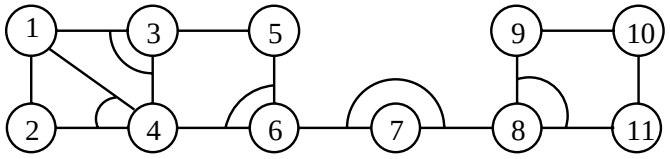


Fig. 7. An implementation where all the cycles are broken, but connectivity is lost

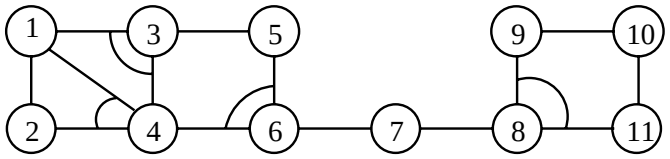


Fig. 8. An implementation where connectivity is preserved, but not all the cycles are broken

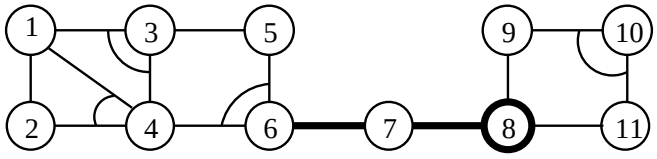


Fig. 9. The full TP-algorithm. The special node and special links are marked in bold. Connectivity is preserved, and all the cycles are broken.

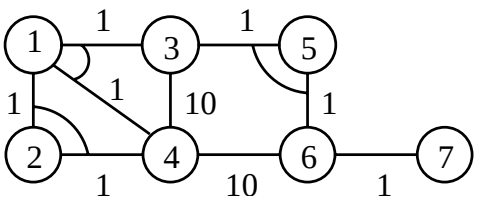


Fig. 10. Applying the generalized TP-algorithm to a weighted graph. The number adjacent to each link represents the weight of the link.

*Proof:* At each step, the algorithm selects the node of minimal degree in the component of connectivity, excluding the special node. If there exist several nodes of minimal degree, then the algorithm selects first a node that is not a neighbor of the special node.

Suppose that the selected node, $n_i$, is of degree $d_i$. The number of prohibited turns around $n_i$ is at most $d_i(d_i-1)/2$ (maybe less if there are special links). We now distinguish between two cases.

First, suppose that $n_i$ is not a neighbor of the special node. Then, all the neighbors of $n_i$ have a degree greater or equal than $n_i$, and the fraction of prohibited turns is smaller or equal than 1/3, exactly as in the proof of theorem 2.

Next, suppose that the selected node $n_i$ is a neighbor of the special node. Let's denote the degree of the special node by $d'$. If $d_i \leq d'$, then, again, it follows immediately that at most 1/3 of the turns are prohibited. Now, suppose that $d_i > d'$. Clearly, node $n_i$ has at least $d_i - d'$ neighbors that are not connected to the special node. The degree of these neighbors must be strictly greater than $d_i$, otherwise one of them would have been selected instead of $n_i$. The other $d' - 1$ neighbor have a degree larger or equal to $d_i$. The last neighbor is the special node, of degree $d'$. Overall, the total number of permitted turns is at least $(d_i - d')d_i + (d' - 1)(d_i - 1) + (d' - 1) = d_i(d_i - 1)$, which is twice as large as the number of prohibited turns. Thus, at most 1/3 of the turns are prohibited. ∎

*Example 2:* Assume the same setting as in Example 1. It is easy to check that the final set of prohibited turns obtained by TP-algorithm is the same as with the "intelligent" node labeling of Fig. 4. Thus, the end-to-end delay bound for each flow will be $\bar{D} = 2(N-1)\sigma/C$. ∎

### C. Generalization for Links of Non-Uniform Weights

So far, we have only considered networks where all the links have the same weight (value). In reality, different links have different characteristics such as data rates or physical lengths. Consequently, different turns have varying importance as well.

In order to take on this issue, we propose the following framework. We suppose that each link $(a, b)$ has a weight $w_{ab}$, and $w_{ba} = w_{ab}$. We assume that the weights of links are additive. We can then define the weight of a turn $(a, b, c)$ as $w_{abc} = w_{ab} + w_{bc}$. Thus, more important turns are represented by higher weights.

We now introduce a "generalized" TP-algorithm that aims at minimizing the sum of the weights of prohibited turns. For this purpose, only the first step of the algorithm needs to be modified. Denote by $W_i$ the sum of the weights of the links incident to node $n_i$, i.e., $W_i = \sum_{j=1}^{d_i} w_{ij}$. Then, instead of selecting a node of minimal degree, the generalized algorithm will select a node $n_i$ for which $W_i$ is minimal (excluding the special node, if there is such one). For instance, in Fig. 10, node 7 with $W_7 = 1$ is first selected. Next, either node 2 or node 5 are selected since $W_2 = W_5 = 2$. The procedure then continues until all the nodes are considered and deleted.

Using an approach similar to the proofs of Theorem 2 and 5, it can be shown that the overall weight of the turns prohibited by the generalized TP-algorithm is at most 1/2 of the total weight of the turns in the network. This bound is valid for any graph topology and any distribution of weights. An illustration of the generalized TP-algorithm for a weighted graph is given in Fig. 10.

### D. Discussion

*Computational Complexity:* The worst-case computational complexity of the TP-algorithm is polynomial in $N$, more precisely $O(N^2 d)$, where $d$ represents the maximal degree of any node in the network. This complexity is certainly reasonable, as long as the network topology does not change too often.

A derivation of this complexity is as follows. At each step of the algorithm, one node is deleted from further consideration. Thus, the algorithm consists of at most $N$ steps. At each of these steps the following computations are performed:

1. A node of minimal degree is selected, an operation of complexity $O(N)$.
2. The components of connectivity are determined. Using a spanning tree construction algorithm, the complexity of this operation is $O(Nd)$.
3. On the order of $O(d^2)$ turns are considered for permission/prohibition.

Therefore, the computational complexity of each step is $O(Nd)$, and the overall complexity of the algorithm is $O(N^2d)$.

*Irreducibility:* The TP-algorithm, as described in the previous section, does not guarantee irreducibility, which means that a final set of prohibited turns may include one or more redundant turns. In other words, it is possible that one of the prohibited turns could in fact have been permitted without creating any cycle.

A version of the TP-algorithm guaranteeing irreducibility is described in [28]. The basic idea is to mark a node as special only if it is absolutely needed. In general, the performance of a TP-algorithm guaranteeing irreducibility is only marginally better than the one that was described therein.

Note that even when irreducibility is guaranteed, the TP-algorithm may not necessarily lead to a minimum solution, that is, a solution with the smallest possible number of prohibited turns. This is because the first step of the algorithm permits to select arbitrarily one among several nodes which satisfy the same constraints. The selection of different nodes at a particular stage may lead to different final sets of prohibited turns with different number of elements.

*Decentralized Implementation:* The current implementation of TP-algorithm requires knowledge of the full network topology, unlike the spanning tree and up/down spanning tree algorithms. Nevertheless, the TP-algorithm can still be implemented in a decentralized fashion as a link-state algorithm like OSPF [20].

*Routing:* A set of prohibited turns constructed by the TP-algorithm does not completely specify the routing strategy since several valid routes may exist between any source/destination pair (the connectivity property guarantees that at least one route exists). A reasonable goal is thus to develop a decentralized algorithm that can determine the shortest path between any source and destination, while forwarding packets only over permitted turns.

It turns out that the traditional Bellman-Ford routing algorithm can be generalized in order to perform this task [12], [28]. The memory required by this algorithm is on the order of $O(Nd)$, at each node, since a different vector of distances to the destinations needs to be maintained for each of the $d$ input links. We note that the problem of finding a good routing algorithm is separate from the problem of constructing a set of prohibited turns, which means that the same routing algorithm can be used with up/down routing restrictions.

## V. Simulation Results

In this section, we present the results of simulations comparing the performance of the spanning tree, up/down routing, and turn-prohibition algorithms.

Our simulator is based on randomly generated, connected graphs. Every node in these graphs has the same degree, i.e. $d = 4$, but the total number of nodes varies. The links have identical weights.

Once a random graph is generated, each of three cycle-breaking algorithms are run on top it in order to determine a set of prohibited links/turns. Routing matrices are then determined using the generalized version of the Bellman-Ford algorithm. All the results presented correspond to averages over 100 graphs with identical parameters.

| *# nodes* | *TP-alg* | *up/down* | *sp. tree* |
|---|---|---|---|
| 16 | 0.23 | 0.27 | 0.72 |
| 32 | 0.23 | 0.27 | 0.73 |
| 64 | 0.22 | 0.26 | 0.73 |
| 128 | 0.21 | 0.26 | 0.73 |
| 255 | 0.21 | 0.25 | 0.73 |

TABLE I

Fraction of turns prohibited by each scheme as a function of the total number of nodes. All the nodes are of degree four.

### A. Fraction of Prohibited Turns

We first compute the fraction of turns prohibited by each scheme, as a function of the total number of nodes. This metric gives a good indication on the amount of unused network resources. Ideally, the fraction of prohibited turns should be as small as possible.

The performances of the three schemes are compared in Table I. We remark that the TP-algorithm prohibits about 10% to 20% fewer turns than the up/down scheme. The simple spanning tree algorithms performs significantly worse than the two other algorithms. Interestingly, the results seem to be rather insensitive to the total number of nodes in the network.

### B. Throughput

We now consider the throughput achieved by the three different schemes. This metric is computed as follows. We assume that a flow is established between each pair of nodes in the network, in both directions. Each flow is routed along the shortest path over the turn-prohibited graph (if multiple paths of same length are available, then one of them is arbitrarily selected). 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 possible rate at which each flow can transmit without saturating the network.

Notice that our definition of throughput is not the only possible one. In particular, flows that do not traverse the bottleneck link could in fact transmit at a higher rate. We may then resort to a max-min criterion [2], or any other similar criteria, to determine the appropriate transmission rate for each flow. However, we expect the relative performance of the schemes to remain about the same.

Table II compares the throughput achieved by the three algorithms. We present results that are normalized by the throughput obtained with the TP-algorithm. We remark that the performance of both the up/down and spanning tree algorithms degrades, as a function of the total number of nodes, compared to the TP-algorithm. The probable reason for this behavior is that spanning trees become deeper with the increase in the number of nodes, and therefore links close to the root get more and more congested.

We next compare the throughput achieved by the TP-algorithm with the maximum theoretical throughput in a net-

| *# nodes* | *TP-alg* | *up/down* | *sp. tree* |
|---|---|---|---|
| 16 | 1 | 0.95 | 0.31 |
| 32 | 1 | 0.92 | 0.28 |
| 64 | 1 | 0.88 | 0.23 |
| 128 | 1 | 0.82 | 0.19 |
| 255 | 1 | 0.74 | 0.16 |

TABLE II

THROUGHPUT OF EACH SCHEME. THE VALUES ARE NORMALIZED BY THE THROUGHPUT OBTAINED WITH THE TP-ALGORITHM.

| *# nodes* | *shortest-path* | *TP-algorithm* |
|---|---|---|
| 16 | 1 | 0.85 |
| 32 | 1 | 0.62 |
| 64 | 1 | 0.45 |
| 128 | 1 | 0.32 |
| 255 | 1 | 0.21 |

TABLE III

COMPARISON BETWEEN THE THROUGHPUT OF THE TP-ALGORITHM AND THE, SO-CALLED, SHORTEST PATH SCHEME THAT ACHIEVES THE MAXIMUM THEORETICAL THROUGHPUT.

work. The maximum throughput is achieved when no network resources are prohibited. In this case, we can employ a "shortest-path" scheme, where every flow takes the shortest possible path from any source to any destination. Note that the throughput achieved with the shortest-path scheme represents an upper bound on the best possible achievable throughput in any feed-forward routing network. Of course, a shortest-path scheme can not be implemented in practice as network stability is not guaranteed.

Table III presents interesting results. We observe that with up to 64 nodes, the throughput achieved by the TP-algorithm is within a factor of about two of the maximum theoretical throughput. This result indicates that the restriction of network calculus to feed-forward routing network may not be too significant for small to mid-size networks, in terms of network utilization.

### C. Delay Bounds

Finally, we compare the performance of the TP-algorithm and the up/down routing algorithm with respect to delay bounds. We assume the same setting as in Example 1 of section III-B, except that we now consider random graphs. The end-to-end delay bound $\bar{D}$ for a flow is computed as follows. Suppose that a flow traverses $K$ links and the number of flows on each link is $m_k$, where $1 \leq k \leq K$. Then, an expression for the end-to-end delay bound is given by $\bar{D} = \sum_{k=1}^{K} m_k \sigma / C$.

In Fig. 11, we depict the fraction of flows with delay bound exceeding some threshold of $x$ time units, for networks of $64$ nodes (a time unit corresponds to the quantity $\sigma/C$). The results are consistent with those of the previous section, where the TP algorithm is shown to outperform the up/down algorithm. For instance, 18% of the flows have a delay bound exceeding 600 time units in the case of up-down, while this fraction is reduced to 10% for the TP-algorithm.

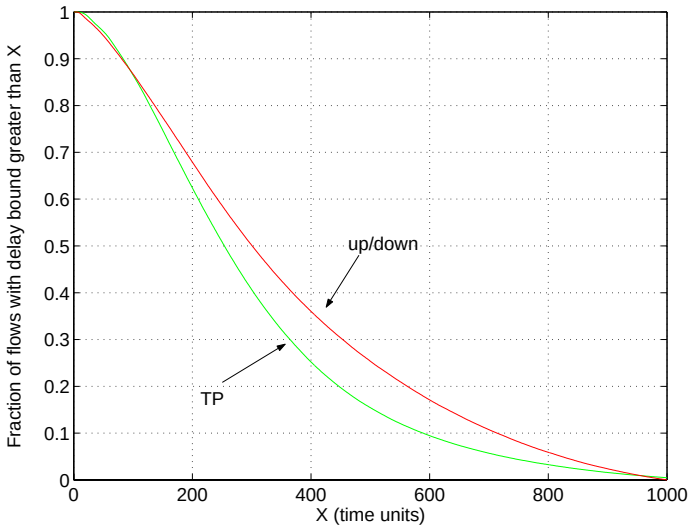


Fig. 11. Delay bound: turn-prohibition versus up/down.

## VI. CONCLUDING REMARKS

In this paper, we have addressed and proposed a concrete solution to the problem of using network calculus in networks of arbitrary topology. We introduced the turn-prohibition (TP) algorithm that breaks all the cycles in any given network, and prohibits the use of at most 1/3 of the turns. We showed that the TP-algorithm can be generalized to networks with weighted links. Moreover, the computational complexity of the algorithm was shown to be only quadratic in the number of nodes. Using analysis and simulations, we showed that the TP-algorithm achieve higher performance, in terms of throughput and delay, than other algorithms used for breaking cycles, such as the spanning tree and the up/down routing algorithms. We note, though, that the difference between the TP and spanning tree algorithms is much more significant than between the TP and up/down algorithms. Our simulations also revealed that, for networks of moderate size, the network utilization achieved by the TP-algorithm is reasonably close to the maximum theoretical network utilization. Specifically, our simulations showed that, for networks of up to 50 nodes of degree four, the network utilization obtained with the TP-algorithm is at least half the highest possible network utilization. We expect the difference to be even smaller for nodes of larger degrees. Thus, in many practical cases, the restriction of network calculus to feed-forward routing networks may not represent a significant limitation, rendering this framework particularly appealing for implementation in practical QoS architectures, such as DiffServ [3], [7].

We conclude this paper by noting that the TP-algorithm represents a universal method for breaking cycles, and, as such, can potentially improve the performance of many other networking applications. In particular, it brings the potential of significantly improving the performance of local area networks, such as Gigabit Ethernet, where packet-forwarding loops and deadlocks need to be prevented [13], [24]. These networks currently implement the simple spanning-tree algorithm [20]. It can also be useful for preventing the appearance of deadlocks in wormhole routing

networks, such as networks of workstations (NOWs) [29], [30]. These examples illustrate the general problematic nature of cycles in networks, and the promise of turn-prohibition to provide a unifying solution methodology.

## ACKNOWLEDGEMENTS

The assistance of Shameek Gupta in running the numerical experiments is gratefully acknowledged.

## REFERENCES

[1] M. Andrews, "Instability of FIFO in Session-Oriented Networks," in the proceedings of *SODA 2000*, pp. 440-447, San Francisco, January 2000.
[2] D. Bertsekas and R. Gallager, *Data Networks*, Prentice-Hall, 1992.
[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," IETF Request for Comments: 2475, December 1998.
[4] R. Boorstyn, A. Burchard, J. Liebeherr, and C. Oottamakorn, "Statistical Service Assurances for Traffic Scheduling Algorithms," *IEEE JSAC*, Vol. 18, No. 12, pp. 2651-2664, December 2000.
[5] C.S. Chang, "Stability, Queue Length and Delay of Deterministic and Stochastic Queueing Networks," *IEEE Trans. on Automatic Control*, Vol. 39, No. 5, pp. 913-931, May 1994.
[6] C.S. Chang, *Performance Guarantees in Communication Networks*, Springer Verlag, 2000.
[7] A. Charny and J.-Y. Le Boudec, "Delay Bounds in a Network with Aggregate Scheduling," in the proceedings of *QOFIS*, pp. 1-13, Berlin, October 2000.
[8] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, McGraw-Hill, 1990.
[9] R. Cruz, "A Calculus for Network Delay, Part I: Network Elements in Isolation," *IEEE Trans. on Information Theory*, Vol. 37, No. 1, pp. 114-131, January 1991.
[10] R. Cruz, "A Calculus for Network Delay, Part II: Network Analysis," *IEEE Trans. on Information Theory*, Vol. 37, No. 1, pp. 132-141, January 1991.
[11] C. Glass and L. Ni, "The Turn Model for Adaptive Routing," *Journal of ACM*, Vol. 5, pp. 874-902, 1994.
[12] S. Jaiswal, M. Mustafa, L. Zakrevski, and M. Karpovsky, "Unicast Wormhole Message Routing in Irregular Computer Networks," in the proceedings of *7th Int. Conf. On Parallel and Distributed Computer Systems (PDCS-2000)*.
[13] M. Karol, S. Golestani, and D. Lee, "Prevention of Deadlocks and Livelocks in Lossless, Backpressured Packet Networks," in the proceedings of *INFOCOM 2000*, pp. 1333-1342, Tel Aviv, Israel.
[14] E. Knightly and N. Shroff, "Admission Control for Statistical QoS: Theory and Practice," *IEEE Network*, Vol. 13, No. 2, pp. 20-29, March-April 1999.
[15] P. Kumar, "A Tutorial on Some New Methods for Performance Evaluation of Queueing Networks," *IEEE JSAC*, vol. 13, no. 6, pp. 970-980, August 1995.
[16] W. Leland, M. Taqqu, W. Willinger and D. Wilson, "On the Self-Similar Nature of Ethernet Traffic (Extended Version)," *IEEE/ACM Trans. on Networking*, Vol. 2, No. 1, pp. 1-15, 1994.
[17] J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queueing Systems for the Internet*, Springer-Verlag Lecture Notes on Computer Science #2050, available on-line at http://icawww.epfl.ch.
[18] A. Parekh and R. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case," *IEEE/ACM ToN*, Vol. 2, No. 2, pp. 137-150, April 1994.
[19] V. Paxson and S. Floyd, "Wide Area Traffic: The Failure of Poisson Modeling," *IEEE/ACM Trans. on Networking*, Vol. 3, No. 3, pp. 226-244, June 1995.
[20] R. Perlman, *Interconnections Second Edition: Bridges, Routers, Switches, and Internetworking Protocols*, Addison Wesley, 2000.
[21] S. Shenker, C. Partridge, and R. Guerin, "Specification of Guaranteed Quality of Service," *IETF Request for Comments: 2212*, September 1997.
[22] M. Shoreder et al. ,"Autonet: A High-Speed, Self-Configuring Local Area Network Using Point-to-Point Links," *IEEE JSAC*, Vol. 9, No. 8, pp. 1318-1335, October 1991.
[23] D. Starobinski, *Quality of Service in High Speed Networks with Multiple Time-Scale Traffic*, Ph.D. Dissertation, Technion - Israel Institute of Technology, Haifa, Israel, 1999.
[24] D. Starobinski and M. Karpovsky, "Turn Prohibition-based Packet Forwarding in Gigabit Ethernets," in the proceeding of the *Gigabit Networking Workshop (GBN 2001)*, Anchorage, Alaska, April 2001. http://www.comsoc.org/tcgn/conference/gbn2001/
[25] D. Starobinski and M. Sidi, "Stochastically Bounded Burstiness for Communication Networks," *IEEE Trans. on Info. Theory*, Vol. 46, No. 1, pp. 206-212, January 2000.
[26] D. Starobinski and M. Sidi, "Modeling and Analysis of Power-Tail Distributions via Classical Teletraffic Methods," *Queueing Systems (QUESTA)*, Vol. 36, Nos. 1-3, pp. 243-267, November 2000.
[27] O. Yaron and M. Sidi, "Performance and Stability of Communication Networks via Robust Exponential Bounds," *IEEE/ACM ToN*, Vol. 1, No. 3, pp. 372-385, June 1993.
[28] L. Zakrevski, *Fault-Tolerant Wormhole Message Routing in Computer/Communication Networks*, Ph.D. Dissertation, Boston University, 2001.
[29] L. Zakrevski, S. Jaiswal, L. Levitin, and M. Karpovsky,"A New Method for Deadlock Elimination in Computer Networks with Irregular Topologies," in the proceedings of *6th Int. Conf. On Parallel and Distributed Computer Systems (PDCS-99)*.
[30] L. Zakrevski, M. Mustafa, and M. Karpovsky, "Turn Prohibition Based Routing in Irregular Computer Networks," in the proceedings of *7th Int. Conf. On Parallel and Distributed Computer Systems (PDCS-2000)*.