Routing in Turn-Prohibition Based Feed-Forward Networks

Markus Fidler and Gerrit Einhoff

Aachen University, Department of Computer Science,
Ahornstr. 55, 52074 Aachen
fidler@i4.informatik.rwth-aachen.de
gerrit@einhoff.com

Abstract. The application of queuing theory to communications systems often requires that the respective networks are of a feed-forward nature, that is they have to be free of cyclic dependencies. An effective way to ensure this property is to identify a certain set of critical turns and to prohibit their use. A turn is a concatenation of two adjacent, consecutive links.
Unfortunately, current routing algorithms are usually not equipped to handle forbidden turns and the required extensions are nontrivial. We discuss the relevant issues for the example of the widely deployed Dijkstra algorithm. Then, we address the general case and introduce the Turnnet concept, which supports arbitrary combinations of routing algorithms with turn-prohibiting feed-forward mechanisms.

1 Introduction

Classical queuing theory has been investigated for a long time to better understand many qualities of communication systems [7]. It has recently been complemented by Network Calculus [10, 2], which extends known queuing theory by means of a worst-case analysis to provide deterministic performance bounds. A field of application of Network Calculus are Quality of Service (QoS) enabling architectures, like the Differentiated Services framework [1], where it allows to efficiently compute delay bounds [6, 14] for a so-called Premium Service [3].

1.1 The Feed-Forward Property

Unfortunately, a variety of methods from the field of classical queuing theory, as well as the direct application of Network Calculus have one important prerequisite, namely the network has to be of a feed-forward nature.

Definition 1 (Feed-Forward Property). *A feed-forward queuing network is a network, in which all queues can be ordered in such a way that whenever a traffic flow traverses from queue i to queue j, this implies that $i < j$ [7], or in a more verbatim way: the links of a feed-forward network cannot form any cycles, i.e. it is impossible for traffic flows to create cyclic dependencies on each other [2].*

Dependencies occur, for example in case of Network Calculus, if two flows use the same queuing and scheduling unit on an outgoing link. In this scenario the service offered to each of the flows individually depends on the service that is consumed by the respective other flow. Now, consider a network consisting of three nodes a, b, and c and three links (a, b), (b, c), and (c, a). Assume two flows use the network, whereby the path of flow 1 is $a \rightarrow b \rightarrow c$ and the path of flow 2 is $b \rightarrow c \rightarrow a$. The service that remains for flow 2 at link (b, c) depends on the service that is consumed by flow 1 at the same link, which in turn depends on the output of flow 1 from link (a, b). Fortunately, the output of flow 1 from link (a, b) does not depend on flow 2. Thus, the dependency is not cyclic and the system can be solved in an inductive manner. However, adding a third flow that traverses the path $c \rightarrow a \rightarrow b$ creates a cyclic dependency. The output of flow 1 from link (a, b) depends on the output of flow 3 from link (c, a). Flow 3's output from link (c, a) depends, however, on flow 2's output from link (b, c), and again on flow 1's output from link (a, b), which completes the cycle.

1.2 Feed-Forward Mechanisms

Obviously, real-world networks are not necessarily of a feed-forward nature, unless they are for example star-shaped. One way to nevertheless realize Network Calculus based QoS offerings [6] is to take provisions to prevent from the creation of cyclic dependencies between different flows.

To ensure the feed-forward property in an arbitrary network, the usual approach is to restrict the usage in a certain way that makes it impossible for flows to create a cyclic dependency. The simplest way to do so is to build a spanning tree covering all nodes and to prohibit the use of all links not belonging to that tree. Since a spanning tree cannot contain any circles by definition, the feed-forward property is ensured. On the other hand this approach can disable large parts of the network, potentially causing a big performance impact [5].

A more intelligent approach is not to prohibit the use of complete links, but only of certain turns. A turn is a triple of three nodes connected by two links. For example a prohibited turn (a, b, c) would disallow a flow to utilize the path $a \rightarrow b \rightarrow c$, but it could still use $a \rightarrow b \rightarrow d$, provided the link $b \rightarrow d$ exists and the turn (a, b, d) is permitted.

Two possible algorithms that determine a set of turns, which have to be prohibited to make a network feed-forward compliant, are Up/Down Routing [13] and Turn Prohibition [15]. Both algorithms return a set of turns that have to be prohibited within a given network topology. As expected the performance impacts of the two turn-prohibiting algorithms on the routing performance are a lot smaller than with the link-prohibiting spanning tree approach [5].

2 Routing in Networks with Prohibited Turns

Using a turn-prohibiting mechanism creates a problem though. Routing algorithms are usually not equipped to handle forbidden turns. In difference to link-prohibiting mechanisms that return a smaller, but still valid network consisting

only of nodes, links, and metrics that routing algorithms can work with[1], turn-prohibiting mechanisms require that the routing algorithm takes the forbidden turns into account and does not use them. Obviously, commonly used routing schemes do usually not fulfill this requirement.

2.1 The Challenge of Routing with Prohibited Turns

One option to apply a routing algorithm to a network with prohibited turns, is to adapt the algorithm to honor the forbidden turns. In [15] an example is provided for the Bellman-Ford algorithm. However, a solution that is applicable to arbitrary routing algorithms is not self-evident and to our knowledge missing in current literature.

An algorithm used by a lot of routing schemes [5] is Dijkstra's shortest path algorithm. Examples include Shortest Path First (SPF) and its enhancements [11, 12, 17, 5]. Yet, Dijkstra's algorithm is not aware of prohibited turns, although it does not seem to hard to extend it by just cancelling the consideration of a new path as soon as it includes a prohibited turn. However, figure 1 gives a motivating example, why this approach does not work.

Part (a) shows a simple network with four nodes. The number at each link specifies the additive link-costs. When searching a least-cost path $1 \rightarrow 4$ using Dijkstra's algorithm, it finds the correct path $1 \rightarrow 3 \rightarrow 4$ with a cost of 2 as shown in (b). Assuming that the turn $1 \rightarrow 3 \rightarrow 4$ is prohibited (see (c)), the correct least-cost path from 1 to 4 is now $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ with a cost of 5 as can be seen in (d). However, using Dijkstra's algorithm extended in the way described above, it would find the path $1 \rightarrow 2 \rightarrow 4$ (see (e)) with a cost of 6, which is *not* the path with the least costs.

The reason for this wrong result is that the algorithm finds the shortest path to each node in an incremental way, i.e. once it has found the shortest path to a node, that path is fixed. In the given example the first node that is examined is node 3. The correct shortest path $1 \rightarrow 3$ is identified and the backpointer of node three is set to point to node 1. With prohibited turns, however, the shortest path to a node depends on the next node of the path. Unfortunately, Dijkstra's algorithm does not consider the following node in its local shortest path decision as it is not designed to do so. With prohibited turns, a node may need more than one backpointer, depending on the destination node. For the request $1 \rightarrow 3$, the backpointer of node 3 should point to node 1, but for the request $1 \rightarrow 4$ it should point to 2 (see (f)).

It is obvious that extending Dijkstra's algorithm to work with prohibited turns is far from trivial and also would not constitute a general solution for all routing schemes. For example the Maximum Disjoint Paths [16] and the Minimum Interference Routing [9] algorithm–although making use of Dijkstra's algorithm–would need individual adaptations to honor prohibited turns. Thus, a general purpose concept that allows for an arbitrary combination of routing algorithms with feed-forward mechanisms is needed.

[1] Note, that routing in a spanning tree is trivial, since there exists only a single path between any two nodes.

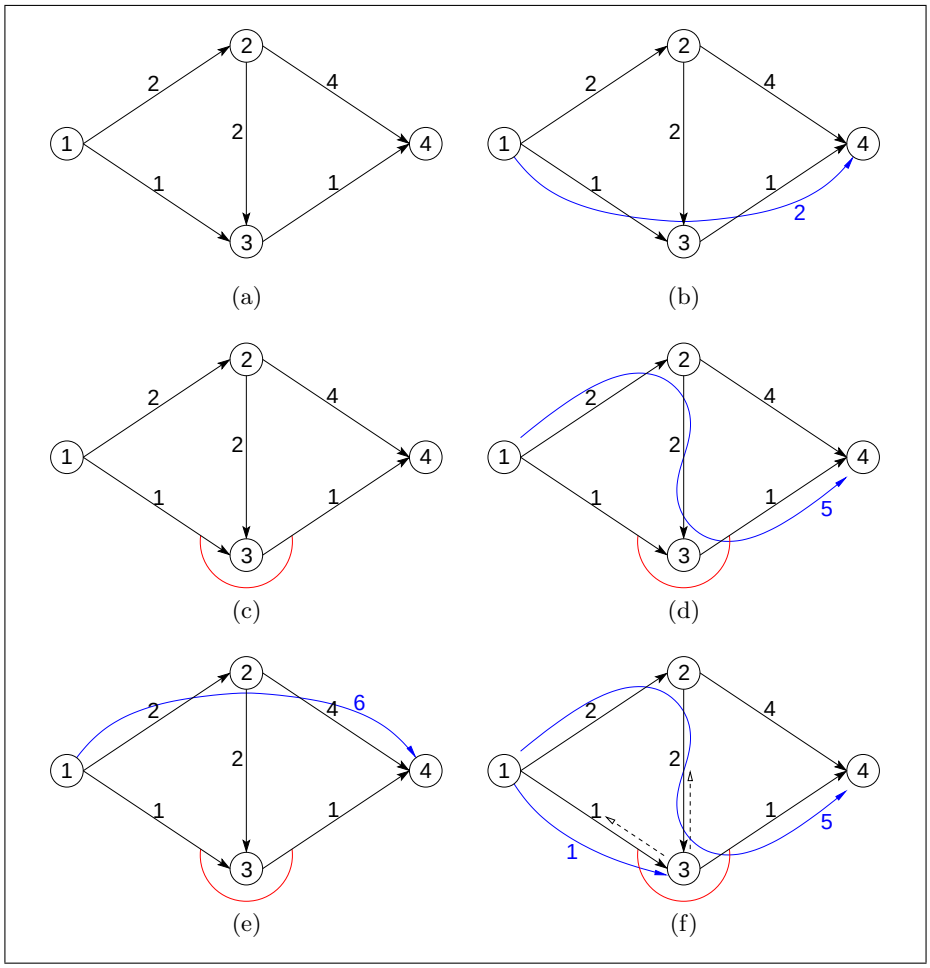

Fig. 1. Dijkstra's algorithm does not work with forbidden turns.

2.2 Formal Requirements Specification

Routing algorithms expect a network to consist of nothing else but nodes, links, and link metrics, which can be used without any restrictions. To work with prohibited turns one solution is to transform a network with a given set of prohibited turns into another network without prohibited turns, with the constraint that routing results can be transferred back to the original network without impacts on their correctness.

The following definition helps in the formal specification of this requirement.

Definition 2 (Path-Conserving). *Given two networks* $G^1 = (N^1, E^1)$ *and* $G^2 = (N^2, E^2)$ *consisting of nodes* N^i *and edges* E^i*. Select two nodes* $s^1, d^1 \in N^1$ *with* $s^1 \neq d^1$ *and define* P_{s^1,d^1} *to be the set of possible paths between* s^1 *and* d^1*.*

G^2 is called path-conserving *to G^1, if there exist $s^2, d^2 \in N^2$ and a bijective function f between P_{s^1,d^1} and P_{s^2,d^2} so that all defined path metrics are the same for p and $f(p)$, i.e. $m(p) = m(f(p))$ with $m()$ being the metrics for a path p.*

The function $m()$ hereby specifies the accumulated result of the metrics of a path. For an additive metric this means the sum of all link metrics on that path. For example the path p in Fig. 1 (d) would result in $m(p) = 5$. If a network has a vector of different metrics for each link, $m()$ results in a vector also.

From this definition, the following corollary can be extracted immediately.

Corollary 1. *If a network G^2 is path-conserving to a network G^1 with function f, any routing algorithm that finds optimal paths by link metrics and produces a path p^2 in G^2 would produce a path p^1 in G^1 with $m(p^1) = m(f^{-1}(p^2)) = m(p^2)$.*

Proof. By contradiction. Assuming the routing algorithm would produce a path p^1 in G^1 and a path p^2 in G^2 and $m(p^1) \neq m(f^{-1}(p^2))$ would hold. If $m(p^1) > m(f^{-1}(p^2))$, i.e. $f^{-1}(p^2)$ is a better path in G^1, then the routing algorithm should have found that path and is therefore no optimizing algorithm contrary to the assumption. If $m(p^1) < m(f^{-1}(p^2))$, i.e. $f^{-1}(p^2)$ is a worse path in G^1, then by definition $m(f(p^1)) < m(p^2)$ holds and therefore the routing algorithm should have found $f(p^1)$ in G^2 and is thus no optimizing routing algorithm contrary to the assumption. □

To use this result for the combination of routing algorithms and turn prohibiting feed-forward mechanisms, an algorithm is needed that, if given a network $G = (N, E)$, a source and a destination node $s, d \in N$, a set of prohibited turns $T \subseteq \{(i, j, k) : i, j, k \in N \wedge (i, j) \in E \wedge (j, k) \in E\}$, and for each link $(i, j) \in E$ a set of m additive, multiplicative[2], or concave[3] metrics $m^t_{(i,j)}, 1 \leq t \leq m$ (e.g. propagation delay, $1-$loss probability, or bandwidth), generates a new network G^2 that is path-conserving to G with respect to the set of prohibited turns T. Additionally, a transforming function f^{-1} must be known, respective an algorithm that produces $f^{-1}(p)$ for the input p.

The Turnnet algorithm provides exactly that and is described in the following section.

3 The Turnnet Concept

The basic idea behind Turnnet is that a routing algorithm should not look at a path node-by-node but link-by-link, i.e. it should shift its focus from the visited nodes to the visited links. Going from one node to the next includes crossing a link, but going from one link to the next includes crossing a turn around a node. So by focusing on the link-steps in a path, rather than the node-steps, the turns are included in the observations.

[2] A multiplicative metric can be converted into an additive metric applying the logarithmic transformation.

[3] A concave metric is a metric that is accumulated by forming the minimum, i.e. $m = \min(m_1, m_2, ..., m_n)$.

3.1 The Algorithm

To achieve this, Turnnet transforms an arbitrary network with prohibited turns into a new one without prohibited turns with the following steps:

1. Add two special nodes to the original network, one connected to the source node, the other connected from the destination node. Set the link metrics of the new links to neutral, that is zero for additive metrics and infinity for concave metrics.
2. For each link in the original network, generate a node in the new network.
3. For each turn in the original network, generate a link in the new network connecting the nodes corresponding to the two links of the original turn.
4. Set the link metrics of the new links to be the same as the metrics of the second link of the corresponding turn in the original network.
5. Delete all links from the new network, whose corresponding turns in the original network are prohibited.

The routing algorithm is then run on the new network using the nodes corresponding to the newly added special links in the original network as source and destination nodes.

Transforming a path from the new to the original network can be done efficiently by cycling through the nodes of the new path and replacing them with corresponding nodes in the original network like this:

1. Cut the last link from the path (which is the link to the node corresponding to the second special node in the original network).
2. Cycle through the nodes of the path and append the destination node of the corresponding link in the original network to the new path.

3.2 The Initial Example Revisited

Figure 2 continues the previous example from figure 1, by applying the Turnnet algorithm to the network and showing that Dijkstra's algorithm produces optimal results this time. In part (a) the original network is shown, with the new special nodes connected to the source and destination nodes. In (b) the network has been transformed as described in steps 2-4. The new nodes are marked with the labels of the source and destination nodes of the corresponding links in the original network in (a). In the following they will be labelled with the "$\rightsquigarrow$" symbol. Step 5 of the algorithm is shown in (c). The original network has only one prohibited turn, namely $1 \rightarrow 3 \rightarrow 4$, i.e. $T = \{(1, 3, 4)\}$. This turn corresponds to the link $(1 \rightsquigarrow 3) \rightarrow (3 \rightsquigarrow 4)$ in the new network. Thus, in compliance with step 5, the use of that link is prohibited. After running Dijkstra's algorithm on the new network, two paths are found as shown in (d), the best path being $(-1 \rightsquigarrow 1) \rightarrow (1 \rightsquigarrow 2) \rightarrow (2 \rightsquigarrow 3) \rightarrow (3 \rightsquigarrow 4) \rightarrow (4 \rightsquigarrow -2)$ with a cost of 5. Transforming this path back into the original network one gets the path $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ which is identical to the path shown in part (d) of figure 1 and indeed the correct shortest path.

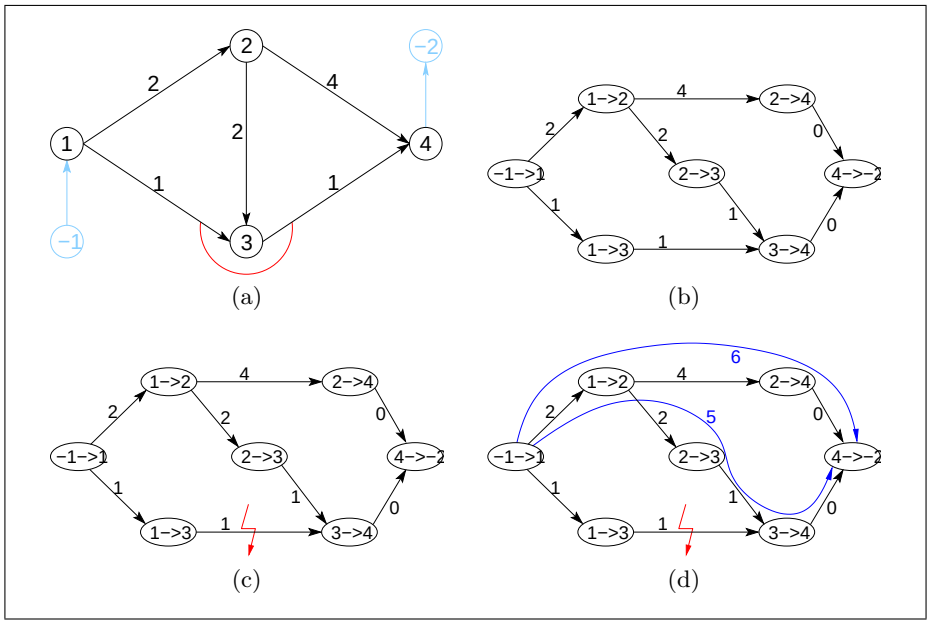

Fig. 2. Example for the transformation of a network with the Turnnet concept.

3.3 Formal Definition and Proof of Correctness

In the following a formal definition of the Turnnet concept is given and its path-conserving property is proven.

Definition 3 (Turnnet algorithm). *Given a network $G = (N, E)$, a source and a destination node $s, d \in N$, a set of prohibited turns $T \subseteq \{(i,j,k) : i,j,k \in N \wedge (i,j) \in E \wedge (j,k) \in E\}$, and for each link $(i,j) \in E$ a set of m additive, multiplicative, or concave metrics $m^t_{(i,j)}, 1 \leq t \leq m$, the Turnnet algorithm produces two special nodes n_s and n_d, a network $G^{TN} = (N^{TN}, E^{TN})$, two nodes $s^{TN}, d^{TN} \in N^{TN}$, and a set of link metrics m^{TN} as follows:*

$$N^{TN} = \{(i \rightsquigarrow j)\} : (i,j) \in E\} \cup \tag{1}$$

$$\{(n_s \rightsquigarrow s), (d \rightsquigarrow n_d)\} \tag{2}$$

$$E^{TN} = \{\{((i \rightsquigarrow j), (j \rightsquigarrow k)) : (i,j),(j,k) \in E\} \cup \tag{3}$$

$$\{((n_s \rightsquigarrow s), (s \rightsquigarrow i)) : i \in N \wedge (s,i) \in E\} \cup \tag{4}$$

$$\{((j \rightsquigarrow d), (d \rightsquigarrow n_d)) : j \in N \wedge (j,d) \in E\}\} \setminus \tag{5}$$

$$\{((i \rightsquigarrow j), (j \rightsquigarrow k)) : (i,j,k) \in T\} \tag{6}$$

$$s^{TN} = (n_s \rightsquigarrow s) \tag{7}$$

$$d^{TN} = (d \rightsquigarrow n_d) \tag{8}$$

$$m^{TN,t}_{((i \rightsquigarrow j),(j \rightsquigarrow k))} = m^t_{(j,k)} \quad \forall\ ((i \rightsquigarrow j), (j \rightsquigarrow k)) \in E^{TN} \tag{9}$$

The two nodes $(n_s \rightsquigarrow s)$ and $(d \rightsquigarrow n_d)$ in (2) emerge from the addition of the two special nodes n_s and n_d to the original graph G in step 1 of the algorithm. Equation (9) only holds for the links in E^{TN} that have a corresponding link in E. For the new links leading to the second special node (specified in (5)), the metrics have to be set to "neutral", i.e. they should not influence the total path metric. This is given in (10).

$$m^{TN,t}_{((j \rightsquigarrow d),(d \rightsquigarrow n_d))} = \begin{cases} 0, & \text{if } m^t \text{ is an additive metric} \\ 1, & \text{if } m^t \text{ is a multiplicative metric} \\ \infty, & \text{if } m^t \text{ is a concave metric} \end{cases} \tag{10}$$

Definition 4 (Transformation Function). *Given a path $p^{TN} = ((i \rightsquigarrow j)_1, ..., (i \rightsquigarrow j)_q)$ with $(i \rightsquigarrow j)_x \in N^{TN}$, the transformation function tr to the corresponding path p in G is defined as given in (11):*

$$tr(p^{TN}) = (j_1, j_2, ..., j_{q-1}) \tag{11}$$

Given these definitions the following theorem shows that the Turnnet algorithm indeed provides a method to combine arbitrary routing algorithms with turn-prohibiting feed-forward mechanisms.

Theorem 1. *Given a network G, source and destination nodes s, d, a set of prohibited turns T, and for each link a set of metrics $m^t_{(i,j)}$, the Turnnet algorithm produces a network G^{TN} that is path-conserving to G with respect to the prohibited turns.*

Proof. The proof is divided into two parts. First, it is shown that tr is a bijective function between G and G^{TN} and then $m(p) = m(tr^{-1}(p))$ is proven.

Given a path p in G from the source node s to the destination node d. The path is defined as $p = (p_1, ..., p_q)$ with $p_1 = s$, $p_q = d$, and $p_x \in N$ for $1 < x < q$. The inverse transformation function tr^{-1} transforms this path p to a path p^{TN} in G^{TN} like this:

$$p^{TN} = (p^{TN}_1, ..., p^{TN}_{q+1}) \tag{12}$$

$$p^{TN}_x = \begin{cases} s^{TN}, & \text{if } x = 1 \\ (p_{x-1} \rightsquigarrow p_x), & \text{if } 1 < x < q+1 \\ d^{TN}, & \text{if } x = q+1 \end{cases} \tag{13}$$

To prove that p^{TN} is a valid path in G^{TN} it suffices to show that all its nodes are valid, i.e. $p^{TN}_x \in N^{TN}$ for $1 \leq x \leq q+1$, and that they are connected to each other, i.e. $(p^{TN}_{x-1} \rightarrow p^{TN}_x) \in E^{TN}$ for $2 \leq x \leq q+1$.

Clearly, all p^{TN}_x are valid nodes in N^{TN}, because all $p_{x-1} \rightarrow p_x$ are links in G (otherwise p would not be a path) and N^{TN} includes all links from G (see (1)).

Since every pair of nodes (p_{x-1}^{TN}, p_x^{TN}) in p^{TN} corresponds to the two links $((p_{x-2} \to p_{x-1}), (p_{x-1} \to p_x))$ in E, according to (3) there is also a link $p_{x-1}^{TN} \to p_x^{TN}$ in E^{TN}. Therefore, every node in p^{TN} is connected to its predecessor node and thus, p^{TN} is a valid path in G^{TN}.

The other way around, i.e. given p^{TN}, the tr function from definition 4 produces p, because $p_x^{TN} = (p_{x-1} \rightsquigarrow p_x)$ for $1 < x < q+1$ and $p_1^{TN} = s^{TN} = (n_s \rightsquigarrow s)$ (see (7)) still holds and therefore $p = (p_1, ..., p_q)$ according to (11).

Thus, tr^{-1} produces an unique, valid path in G^{TN} and $p = tr(tr^{-1}(p))$ holds and consequently tr is a bijective function.

Since tr is a bijective function, it suffices to show that $m(p) = m(tr^{-1}(p)) = m(p^{TN})$ to prove that G^{TN} is path-conserving to G.

Because $p = (p_1, ..., p_q)$, $p^{TN} = (p_1^{TN}, ..., p_{q+1}^{TN})$ and, with (9), $m^{TN,t}_{(p_{x-1}^{TN} \to p_x^{TN})} = m^t_{(p_{x-1} \to p_x)}$ for $2 \leq x \leq q$, for the path metric $m(p) = m((p_1^{TN}, ..., p_q^{TN}))$ holds, i.e. the path metrics for p in G and the first q nodes of $p^{TN} = tr^{-1}(p)$ in G^{TN} is the same.

Now, since the metrics for the last link of p^{TN}, namely $p_q^{TN} \to p_{q+1}^{TN}$ which is $p_q^{TN} \to (d \rightsquigarrow n_d)$, are set to neutral according to (10), they have no influence of the accumulated path metrics. Thus, $m(p) = m((p_1^{TN}, ..., p_q^{TN})) = m((p_1^{TN}, ..., p_{q+1}^{TN})) = m(p^{TN})$ holds and G^{TN} is path-conserving to G. □

3.4 Application to the G-WiN Topology

This section provides a real-world scenario applying the G-WiN topology of the German Research Network (DFN) as of 2000 [8] that is shown in the left of figure 3. It consists of a dense level one mesh, which allows for multiple alternative paths, thus achieving redundancy. The level two sites are each connected to a single level one site only, however, using two links in parallel, thereby providing backup capabilities.

In the right of figure 3 the level one mesh is reproduced, including a set of forbidden turns that are derived by Turn Prohibition [15]. The level one nodes have been visited by the Turn Prohibition algorithm in the order of their numbering. The star-shaped level two components are excluded here, because all routing decisions are already determined. Further on, the star structure assures that the routes are feed-forward compliant anyway.

Figure 4 illustrates the Turnnet that corresponds to the G-WiN level one topology excluding the prohibited turns from figure 3. It can be immediately seen, that the Turnnet graph provides a valid order for an inductive application of Network Calculus. However, adding any of the prohibited turns that are shown in figure 3, for example $(7, 1, 2)$, will render an inductive approach impossible. The Turnnet graph represents the dependencies that exist between the links in the original network. A related structure is also known as channel dependency graph from [4], where it is used to analyze deadlock conditions.

Adding special nodes as described in section 3.1 and applying Dijkstra's algorithm to the Turnnet in figure 4 allows to derive shortest paths without introducing cycles that can be transformed backwards to the original network.

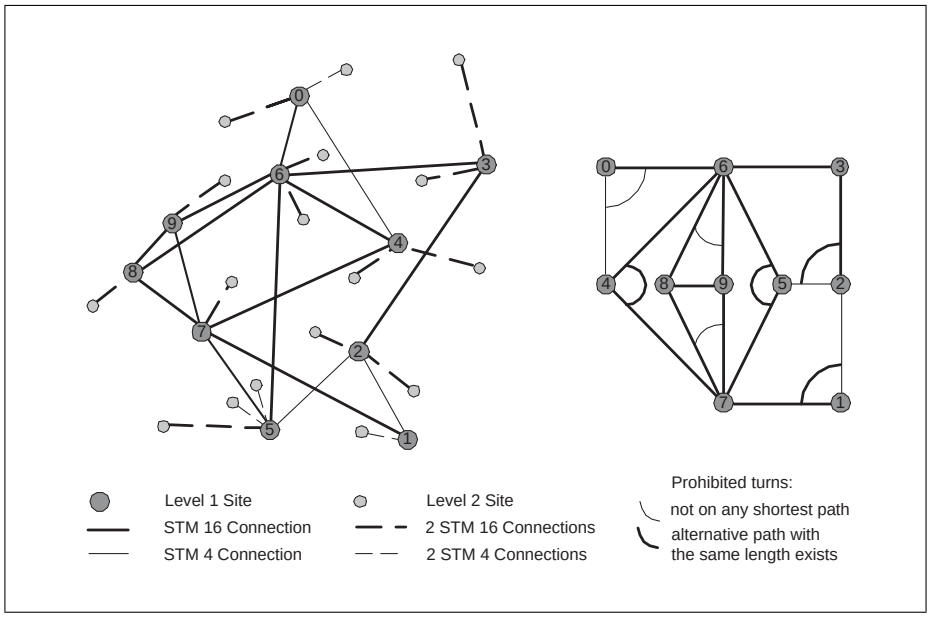

Fig. 3. G-WiN topology and Turn Prohibition example.

For example to derive the shortest path from node 5 to node 3 two special nodes -1 and -2 have to be connected to the original network by links $(-1, 5)$ and $(3, -2)$. These are then transformed to the Turnnet and become nodes $(-1 \rightsquigarrow 5)$ and $(3 \rightsquigarrow -2)$ and links $((-1 \rightsquigarrow 5), (5 \rightsquigarrow 2))$, $((-1 \rightsquigarrow 5), (5 \rightsquigarrow 6))$, $((-1 \rightsquigarrow 5), (5 \rightsquigarrow 7))$, $((2 \rightsquigarrow 3), (3 \rightsquigarrow -2))$, and $((6 \rightsquigarrow 3), (3 \rightsquigarrow -2))$. The shortest path that is found in the Turnnet is $(-1 \rightsquigarrow 5) \rightarrow (5 \rightsquigarrow 6) \rightarrow (6 \rightsquigarrow 3) \rightarrow (3 \rightsquigarrow -2)$, which becomes $5 \rightarrow 6 \rightarrow 3$ after backwards transformation.

An analysis of the paths that can be derived with or without Turn Prohibition allows to classify the turns as shown in the right of figure 3. Three of the seven prohibited turns do not impact any shortest paths and the remaining four forbid potential shortest paths, for which alternatives with the same hop count exist.

3.5 Discussion

From theorem 1 and corollary 1 it immediately follows that any routing results found in the Turnnet network are also valid in the original network, when transformed back with function tr. Thus, it is proven that the Turnnet algorithm solves the problems described in section 2 and allows for the combination of arbitrary routing algorithms and turn-prohibiting feed-forward mechanisms.

This flexibility, however, comes at the price of increased complexity. According to the definition, the Turnnet algorithm has to be executed for each source/destination pair. Fortunately, it is very easy to implement Turnnet in a way that allows for a single execution at initialization time and a very simple

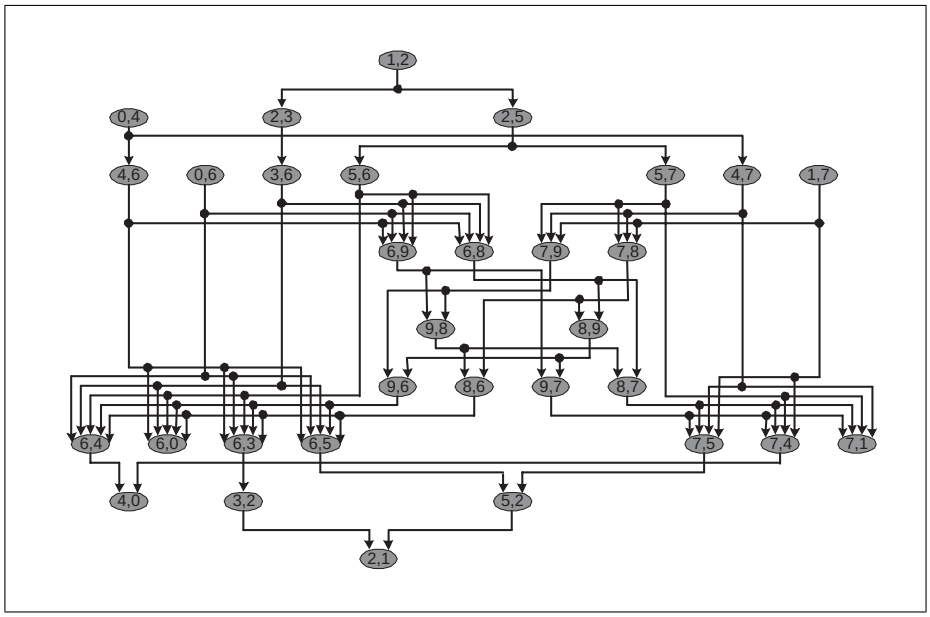

Fig. 4. G-WiN Turnnet excluding prohibited turns.

adjustment of the source/destination nodes for each request. The complexity of the initial Turnnet computation is in $O(|E|^2)$. However, the resulting network G^{TN} is bigger than the original one, i.e. $|N^{TN}| = (|E| + 2)$ and $|E^{TN}|$ depends on the number of turns in the original network. Therefore, the computational complexity of the applied routing algorithm may increase. For routing schemes based on Dijkstra's algorithm the complexity rises from $O(n^2)$ to $O^{TN}(|E|^2)$.

A big advantage of the Turnnet concept is that routing algorithms do not have to be aware of it. For a routing algorithm it makes no difference if the network it operates on is a Turnnet or not, which can be used efficiently for practical implementations.

4 Conclusions

The application of feed-forward mechanisms to data networks is relatively new and the problem of applying conventional routing algorithms to networks with prohibited turns has to our knowledge not been investigated in detail so far. By developing the Turnnet concept, we have evolved a general-purpose solution, which allows to use arbitrary routing schemes with prohibited turns.

The Turnnet algorithm is not very complicated. It does not raise the routing complexity in an unacceptable manner and can be easily implemented. Thus, offering a service with delay guarantees based on Network Calculus and the application of a feed-forward mechanism in conjunction with the use of Turnnet for routing, is a viable and recommendable option for network operators.

Acknowledgments

This work was supported in part by the Path Allocation in Backbone Networks (PAB) project funded by the German Research Network (DFN) and the Federal Ministry of Education and Research (BMBF) and in part by the German Research Community (DFG) under grant GRK (Graduate School) 643.

References

1. S. Blake, D. Blake, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. *RFC 2475*, December 1998.
2. C.-S. Chang. *Performance Guarantees in Communication Networks*. Springer, 2000.
3. B. Davie, A. Charny, J.C.R. Bennett, K. Benson, J.Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis. An Expedited Forwarding PHB (Per-Hop Behavior). *RFC 3246*, March 2002.
4. J. Duato, S. Yalamanchili, and N. Lionel. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann, 2003.
5. Gerrit Einhoff. Quality of Service Routing for an IP Premium Service based on MPLS Traffic Engineering. Master's Thesis, Aachen University, June 2003.
6. M. Fidler, and V. Sander. A Parameter Based Admission Control for Differentiated Services Networks, *Elsevier Computer Networks*, 44(4):463–479, 2004.
7. B. R. Haverkort. *Performance of Computer Communication Systems: A Model-Based Approach*. John Wiley & Sons, January 1999.
8. G. Hoffmann. G-WiN - the Gbit/s Infrastructure for the German Scientific Community. *Proceedings of Terena Networking Conference*, 2000.
9. M. S. Kodialam and T. V. Lakshman. Minimum Interference Routing with Applications to MPLS Traffic Engineering. *Proccedings of IEEE INFOCOM (2)*, pages 884–893, 2000.
10. J.-Y. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queueing Systems for the Internet*. Number 2050 in LNCS. Springer, July 2002.
11. Q. Ma and P. Steenkiste. On Path Selection for Traffic with Bandwidth Guarantees. *Proceedings of IEEE International Conference on Network Protocols*, October 1997.
12. Q. Ma, P. Steenkiste, and H. Zhang. Routing High-Bandwidth Traffic in Max-Min Fair Share Networks. *Proceedings of ACM SIGCOMM*, pages 206–217, 1996.
13. M. D. Schroeder et al. 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, October 1991.
14. V. Sander. *Design and Evaluation of a Bandwidth Broker that Provides Network Quality of Service for Grid Applications*, volume 16 of *NIC*. February 2003. PhD Thesis, Aachen University.
15. D. Starobinski, M. Karpovsky, and L. Zakrevski. Application of Network Calculus to General Topologies using Turn-Prohibition. *IEEE/ACM Transactions on Networking*, June 2003.
16. N. Taft-Plotkin, B. Bellur, and R. Ogier. Quality-of-Service Routing Using Maximally Disjoint Paths. *Proceedings of IEEE/IFIP IWQoS*, pages 119–128, June 1999.
17. Z. Wang and J. Crowcroft. Quality-of-Service Routing for Supporting Multimedia Applications. *IEEE Journal of Selected Areas in Communications*, 14(7):1228–1234, 1996.