Preserving the Independence of Flows in General Topologies using Turn-Prohibition

Markus Fidler[1], Oliver Heckmann[2], and Ralf Steinmetz[2]

[1] Centre for Quantifiable Quality of Service (Q2S), NTNU Trondheim, Norway
`fidler@ieee.org`
[2] Multimedia Communications Lab (KOM), TU Darmstadt, Germany
`{heckmann,steimetz}@kom.tu-darmstadt.de`

Abstract. Various elegant and powerful theories for network performance evaluation have to assume independence to be efficient. While traffic sources are often supposed to be independent, the implications of this assumption regarding flows in arbitrary networks are largely unknown. Recently, turn-prohibition was proposed to solve a related problem concerning feed-forward networks.
In this paper we extend the concept of turn-prohibition to address the issue of independence of flows in general topologies. To this end we evolve an algorithm which derives a set of critical turns that provide full connectivity while conserving the independence of flows up to multiplexing points. In an iterative procedure further turns are added to improve connectivity. The developed algorithm is proven and exemplified.

1 Introduction

Emerging quality of service architectures gave rise to various new approaches to network performance evaluation. Beyond classical queuing theory, for example [2, 8], methods like the theory of effective bandwidths, see [4, 11, 18] and references therein, and deterministic network calculus, for a comprehensive overview see [4, 12], were developed. Recently, network calculus extensions towards a probabilistic equivalent started to evolve, for example [3, 4, 6, 13, 17, 19].

Independence of flows is critical for the applicability, accuracy, and efficiency of a variety of these methods. While statistical multiplexing of independent flows is known to smooth out burstiness, dependent flows are still subject to worst-case analysis where bursts are cumulated. Yet, adequate preconditions that ensure independence have not been devised for general topologies. A related issue concerning feed-forward networks has recently been solved [7, 15, 16]. The feed-forward property facilitates inductive analyses, for example applying network calculus and its probabilistic extensions. While networks usually are not of a feed-forward type, turn-prohibiting algorithms [15, 16] have been developed, which allow modifying the routing [6] to ensure this property.

In this paper we extend the concept of turn-prohibition in a way such that arbitrary routing algorithms can generate paths that conserve the independence of flows in general topologies. We apply a network model where routers are represented by vertices V and links and the belonging queuing and scheduling units by directed edges E. Bidirectional links are presumed and often displayed as such in figures, where each bidirectional link corresponds to two directed and opposing edges in E. For brevity and ease of presentation, we assume that the objective of the routing algorithm is to minimize the path length measured in hops. Traffic sources are expected to be uncorrelated, that is traffic flows are stochastically independent before entering the network. For clarity we distinguish between two conditions under which dependencies are created:

1. **Direct Dependencies:** Consider two flows i and j that pass a common queue. The flows i and j are dependent when leaving the queue.
2. **Indirect Dependencies:** Consider three flows i, j, and k. Let flows i and j traverse a queue and afterwards j and k traverse another queue. When leaving the queues, flows i and j respective j and k are directly dependent. Further on flow k depends via flow j indirectly on the properties of flow i.

Dependencies can further on be classified as plain or cyclic, where in the latter case a number of flows form a cycle of direct dependencies, such that the characteristics of a flow when leaving a queue indirectly depend on themselves. In the following we will introduce conditions under which dependencies cannot occur, namely the feed-forward property and unique dependency paths.

The remainder of this paper is organized as follows: Section 2 briefly recalls known methods to ensure the feed-forward property and introduces the concept of dependency graphs. In Sect. 3, we develop and prove an extended turn-prohibition algorithm that allows ensuring unique dependency paths. An example is discussed in Sect. 4 while Sect. 5 gives concluding remarks and recommendations for application.

2 The Feed-Forward Property

Definition 1 (Feed-Forward Property). *In a feed-forward queuing network the queues can be labeled in a way, such that whenever traffic flows from queue i to queue j this implies that $i < j$ [8]. That is the queues of a feed-forward network cannot form any cycles, respective it is impossible for traffic flows to create cyclic dependencies [4].*

Certain network topologies, for example sink-tree networks [4], are generally of a feed-forward nature. However, considering general topologies, few approaches are known which allow ensuring the feed-forward property. Among these are edge-prohibiting methods, for example based on spanning trees, and turn-prohibiting approaches, like up-down routing [15] and the turn-prohibition algorithm [16] where a directed turn (a, b, c) refers to the concatenation of two successive edges (a, b) and (b, c). Generally, turn-prohibiting methods may have significantly less performance impact then edge-prohibiting ones [16] since a prohibited turn only bans the combination of the belonging edges but not the edges themselves.

Here, we apply the turn-prohibition algorithm [16] and a related data structure referred to as dependency graph [5] respective turn network [7] which we briefly introduce in the following subsections.

2.1 Turn-Prohibition

The turn-prohibition algorithm [16] generally breaks all cycles in networks with bidirectional links. The steps of the basic version are summarized in Alg. 1 where Pre(.) denotes the set of predecessors of a vertex and Suc(.) the set of successors. Given a network graph $G = (V, E)$ with vertices V and edges E and an empty set of turns P, the algorithm inspects all turns around each of the vertices in increasing order of their degree that is the cumulated capacity of the connected edges. While inspecting a vertex, all turns around it are included in the set of prohibited turns P and the vertex and all adjacent edges are removed from the graph before the next vertex for inspection is determined.

Algorithm 1 Calculate prohibited turns P

```
Require: $G = (V, E), P = \emptyset$
  while $V \neq \emptyset$ do
    Select a vertex $b \in V$ with minimal degree
    for all $a \in \text{Pre}(b), c \in \text{Suc}(b), a \neq c$ do
      $P \leftarrow P \cup \{(a, b, c)\}$
    for all $a \in \text{Pre}(b)$ do
      $E \leftarrow E \setminus \{(a, b)\}$
    for all $c \in \text{Suc}(b)$ do
      $E \leftarrow E \setminus \{(b, c)\}$
    $V \leftarrow V \setminus b$
```

Figure 1 gives an example. In (a), four flows are displayed which create a cyclic dependency. In (b) and (c), turn-prohibition breaks this cycle. The vertices are investigated in the order of their labelling. No turns exist around vertex 0, thus vertex 0 and edges $(0, 2)$ and $(2, 0)$ are removed. The same applies for vertex 1. In case of vertex 2 the turns $(3, 2, 5)$ and $(5, 2, 3)$ are prohibited as indicated by the arc around vertex 2 before vertex 2 and the adjacent edges are removed from the graph. The following steps do not prohibit further turns. Finally in (d), shortest paths that do not utilize the prohibited turns are shown for the flows from the initial problem. Clearly turn-prohibition resolved the cyclic dependency.

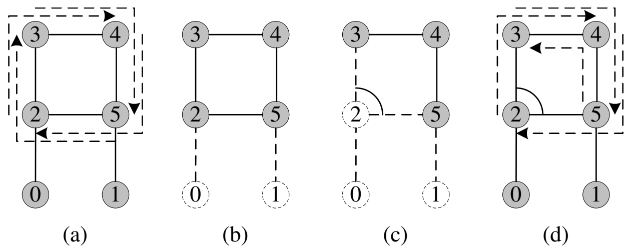

Fig. 1. Example application of turn-prohibition

2.2 Dependency Graphs

A structure which allows for efficient analysis of both cyclic and non-cyclic dependencies is the so-called dependency graph [5], also referred to as turn network [7] since it essentially consists of edges and turns. The graph of the turn network $G^* = (E, T)$ results from transformation of the initial network graph $G = (V, E)$ where edges become vertices and turns become edges. Thus, the notation of the set of predecessors Pre(.) and successors Suc(.) naturally extends to edges where it becomes Pre((., .)) and Suc((., .)) respectively. Since each edge in G respective vertex in G^* symbolizes a link including a queuing and scheduling unit, the edges in the graph G^* indicate dependencies that occur if traffic flows along the belonging turn in G. The transformation is summarized in Alg. 2.

Algorithm 2 Calculate dependency graph $G^* = (E, T)$

```
**Require:** $G = (V, E), T = \emptyset, \mathrm{Pre}((.,.)) = \emptyset, \mathrm{Suc}((.,.)) = \emptyset$
  **for all** $b \in V$ **do**
    **for all** $a \in \mathrm{Pre}(b), c \in \mathrm{Suc}(b), a \neq c$ **do**
      $T \leftarrow T \cup \{(a, b, c)\}$
      $\mathrm{Pre}((b, c)) \leftarrow \mathrm{Pre}((b, c)) \cup \{(a, b)\}$
      $\mathrm{Suc}((a, b)) \leftarrow \mathrm{Suc}((a, b)) \cup \{(b, c)\}$
```

Figure 2 shows the corresponding dependency graph for the example network in Fig. 1. The vertices are labeled by the source and destination vertices of the corresponding edges from Fig. 1 and whenever traffic can flow from one edge to another in Fig. 1, the corresponding turns are represented by edges in Fig. 2. The edges that correspond to the prohibited turns $(3, 2, 5)$ and $(5, 2, 3)$ are indicated by dotted lines. The dependency graph clearly shows the cyclic dependencies that are broken by prohibition of these turns.

While allowing for a demonstrative analysis of dependencies, the turn network can immediately be applied for routing. In [7] it is shown that routing algorithms, like Dijkstra's algorithm, might not find optimal paths in a network with prohibited turns whereas optimal paths are found in the corresponding turn network from where they can be transformed back to the original network.

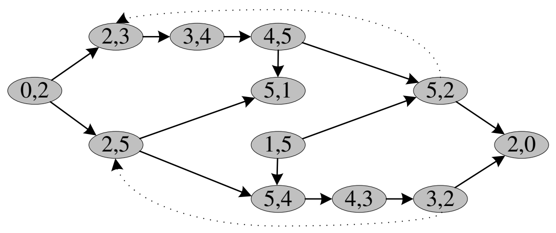

Fig. 2. Dependency graph after turn-prohibition

3 Unique Dependency Paths

Definition 2 (Unique Dependency Paths). *In a queueing network with unique dependency paths, traffic flows span at most one path from any queue i to any queue j. Flows span a path from queue i to queue j if a set of flows exists, where the first flow passes queue i, each pair of subsequent flows traverse at least one common queue, and the last flow passes queue j.*

A network which fulfills both, the feed-forward and the unique dependency path property, conserves the independence of initially independent flows until they are multiplexed. While the feed-forward property has been investigated for general topologies, we are not aware of corresponding methods regarding unique dependency paths. So far, many authors leave the problem open whereas few address the issue either by restricting the topology, for example applying sink-tree networks [4], or by assuming independence only at the ingress and dependence throughout the core of the network [3, 13].

Figure 3 (a) extends the example from Fig. 1 to show how indirect dependence can occur in a network without unique dependency paths. In the sequel we use the path of a flow as a synonym for the flow. The two flows $(0, 2, 5, 1)$ and $(0, 2, 3, 4)$ create a direct dependence at edge $(0, 2)$. Then, at edge $(3, 4)$ the flows $(0, 2, 3, 4)$ and $(3, 4, 5, 1)$ become dependent. Due to indirect dependence the flow $(3, 4, 5, 1)$ also depends on flow $(0, 2, 5, 1)$. Thus, when multiplexing flows $(3, 4, 5, 1)$ and $(0, 2, 5, 1)$ at edge $(5, 1)$ they are not independent. Obviously, the network does not have unique dependency paths since the chain of the two flows $(0, 2, 3, 4)$ and $(3, 4, 5, 1)$ – connected by the common edge $(3, 4)$ – spans a different path from edge $(0, 2)$ to edge $(5, 1)$ than flow $(0, 2, 5, 1)$.

An immediate though suboptimal solution to the problem is to use only edges that belong to a spanning tree where the shortest path spanning tree with root vertex 5 is shown in Fig. 3 (b). Removing the edges $(2, 3)$ and $(3, 2)$ solves the problem as displayed in (c). However, a considerably better solution shown in (d) can be obtained as derived in the sequel where only the turns $(2, 3, 4)$, $(4, 3, 2)$, $(3, 2, 5)$, and $(5, 2, 3)$ but not the edges $(2, 3)$ and $(3, 2)$ have to be prohibited.

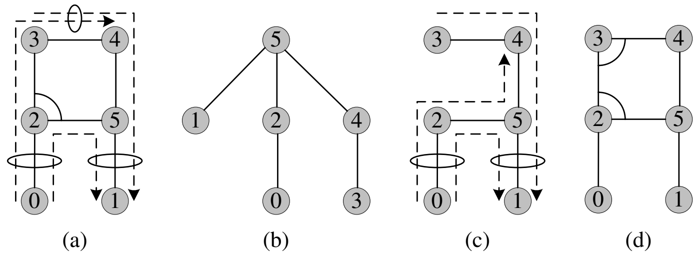

Fig. 3. Example application of extended turn-prohibition

3.1 Extended Turn-Prohibition Algorithm

Our solution applies the turn-prohibition algorithm [16] to ensure the feed-forward property in a first step before switching over to the dependency graph. Certainly, the feed-forward property could also be ensured applying cycle-breaking methods to the dependency graph, however, the turn-prohibition algorithm is assumed to be efficient [16]. Our extended turn-prohibition algorithm derives a set of turns T' which can be used without violating the feed-forward and the unique dependency path properties. It comprises the following steps:

1. Apply Alg. 1 to the graph $G = (V, E)$ to calculate a set of prohibited turns P that ensure the feed-forward property.
2. Construct the shortest path spanning tree $SPST = (V, E')$ from the last vertex visited by turn-prohibition without using any of the turns in P and compute the dependency graph $SPST^* = (E', T')$ with Alg. 2.
3. Substitute nonessential turns – that are turns for which edges exist – by the respective edges using Alg. 3.
4. Add all remaining edges from the set E to $SPST^*$.
5. Incrementally add all turns which are not in P to $SPST^*$ if they do not violate the unique dependency path property according to Alg. 4.

Step 1 applies the known turn-prohibition algorithm [16] to derive a set of prohibited turns $P \subset T$ to ensure the feed-forward property.

In step 2 we identify a set of critical turns $T' \subseteq \{T \setminus P\}$ which are required to provide full connectivity among all vertices while ensuring unique dependency paths. A valid set T' is given by all turns that belong to the shortest path spanning tree rooted from the last vertex visited by the turn-prohibition algorithm which trivially ensures unique dependency paths. The shortest path spanning tree can for example be computed using Dijkstra's algorithm on the dependency graph $G^* = (E, T \setminus P)$ [7]. Figure 4 shows the dependency graph $SPST^*$ where the gray vertices correspond to the edges of the spanning tree in Fig. 3.

In step 3, Alg. 3 is applied to substitute turns by edges if applicable. Consider the initial network graph $G = (V, E)$ and the dependency graph of the shortest path spanning tree $SPST^* = (E', T')$. If there exist edges $(b, d) \in E$ and $(b, d) \notin E'$ and corresponding turns $(b, c, d) \in T'$, then the turns (b, c, d) are replaced by

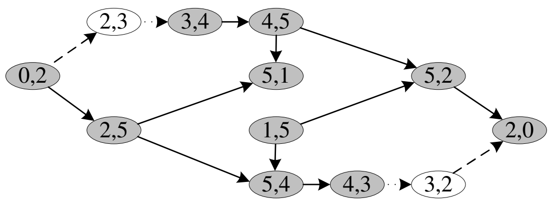

Fig. 4. Dependency graph after extended turn-prohibition

Algorithm 3 Substitute nonessential turns

```
Require: $G = (V, E), SPST^* = (E', T'), P$
  repeat
    $T'' \leftarrow \emptyset$
    for all $b, c, d \in V, (b, c, d) \in T'$ do
      if $(b, d) \in E \setminus E'$ then
        $T''' \leftarrow \emptyset$
        for all $a \in V, (a, b, c) \in T'$ do
          $T''' \leftarrow T''' \cup \{(a, b, d)\}$
        for all $e \in V, (c, d, e) \in T'$ do
          $T''' \leftarrow T''' \cup \{(b, d, e)\}$
        if $T''' \cap P = \emptyset$ then
          $T' \leftarrow T' \setminus \{(b, c, d)\}, E' \leftarrow E' \cup \{(b, d)\}, T'' \leftarrow T'' \cup T'''$
    $T' \leftarrow T' \cup T''$
  until $T'' = \emptyset$
```

the edges (b, d) and for each turn $(a, b, c) \in T'$ respective $(c, d, e) \in T'$ a turn $(a, b, d) \notin P$ respective $(b, d, e) \notin P$ is added to T' to connect the edge (b, d).

In Fig. 3 no such turns exist, thus we investigate the modified example in Fig. 5. In (a), all turns belonging to the tree are indicated by arrows around vertices and two edges that do not belong to the tree are given by dashed lines; all turns that include these edges are assumed to be permitted. The result of the first iteration is shown in (b). The edges $(4, 5)$ and $(5, 4)$ replace the turns $(4, 6, 5)$ and $(5, 6, 4)$. Further on, the new edges are connected by turns $(1, 4, 5)$, $(5, 4, 1)$, $(2, 5, 4)$, and $(4, 5, 2)$. In the second iteration, shown in (c), the turns $(2, 5, 4)$ and $(4, 5, 2)$ are then replaced by the edges $(2, 4)$ and $(4, 2)$ which are connected by turns $(0, 2, 4)$, $(4, 2, 0)$, $(1, 4, 2)$, and $(2, 4, 1)$.

In step 4, all edges are added to the dependency graph $SPST^*$ since including edges without connecting turns does not create dependencies. In Fig. 4 these are the white vertices $(2, 3)$ and $(3, 2)$ which at this step are still unconnected.

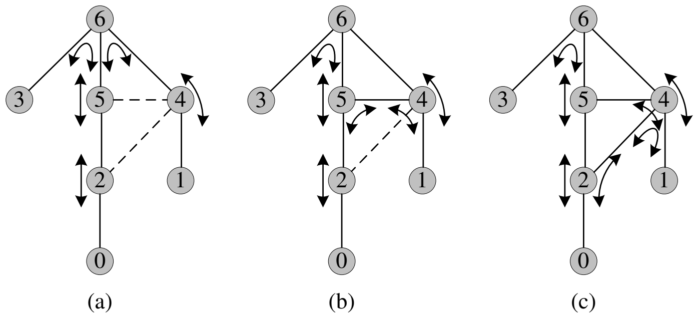

Fig. 5. Shortest path spanning tree and substitution of turns

Finally, in step 5 all turns not in P and not yet in T' are tested by Alg. 4 and added to T', if they do not violate the unique dependency path property. Preferably start with turns that are part of potential shortest paths between sparsely connected vertices. In Fig. 4 turns $(0, 2, 3)$ and $(3, 2, 0)$ can be added, whereas subsequently $(2, 3, 4)$ and $(4, 3, 2)$ fail the test and must not be used.

Algorithm 4 Test uniqueness of dependency paths

```
Require: $G^* = (E, T)$
  for all $i \in E$ do
    $A \leftarrow \{i\}, B \leftarrow \{i\}$
    repeat
      $C \leftarrow \emptyset$
      for all $j \in B$ do
        $C \leftarrow C \cup \mathrm{Pre}(j)$
      if $A \cap C \neq \emptyset$ then
        The graph does not have unique dependency paths.
      $A \leftarrow A \cup C, B \leftarrow C$
    until $C = \emptyset$
```

3.2 Proof of Correctness

Proposition 1. *The shortest path spanning tree $SPST$ from the last vertex visited by the turn-prohibition algorithm provides full connectivity among all vertices V without using any of the prohibited turns in P.*

A spanning tree that consists of bidirectional edges trivially provides full connectivity among all vertices. Note, however, that this is not an immediate result if certain turns are prohibited. Consider the spanning tree in Fig. 3 (b). If the turns $(5, 2, 0)$, $(0, 2, 5)$, $(5, 4, 3)$, and $(3, 4, 5)$ are permitted, the tree contains valid paths from the root vertex 5 to any other vertex and vice versa. This holds also if for example the turns $(1, 5, 2)$ and $(2, 5, 1)$ are prohibited, in which case the tree, however, does not contain any paths between vertices 1 and 2.

Proof. Let the vertices be numbered in the order of inspection by turn-prohibition starting with zero. Since the full version of the turn-prohibition algorithm preserves connectivity [16], the tree $SPST$ exists and there are valid paths in $SPST$ from the root vertex $n = |V| - 1$ to all other vertices and vice versa.

Yet, it remains to be shown that the tree $SPST$ provides connectivity among all of the remaining vertices. Clearly, since vertex n is the last vertex visited by turn-prohibition, there cannot be any prohibited turns around it. Thus, full connectivity among all direct children of the root vertex n is provided.

Consider a vertex i and a direct child of i with label j. If $j < i$ vertex j was removed from the graph G by the turn-prohibition algorithm before vertex i was inspected. Consequently, turns around vertex i originating from or destined for vertex j cannot be prohibited. Thus, as long as children have smaller label values than parents, full connectivity among all vertices is ensured.

Let us now assume that a vertex with label i has a child with label j and $j > i$. In this case vertex i has been inspected by the turn-prohibition algorithm before vertex j and also before the root vertex n. If vertex i is a direct child of the root vertex n this, however, means that the turns (n, i, j) and (j, i, n) have been prohibited which means that vertex j cannot be a child of vertex i in contradiction to the assumption.

If vertex i is not a direct child of the root vertex, then either the parent vertex of i has a higher label $k > i$ in which case the same argumentation applies immediately, or it has a lower label $k < i$ in which case the above argumentation applies recursively for the parent vertex k. □

Proposition 2. *The substitution of nonessential turns by edges does not introduce new dependencies to the dependency graph* $SPST^* = (E', T')$ *and hence preserves unique dependency paths.*

Proof. Consider a turn $t = (b, c, d) \in T'$ that is substituted by an edge $s = (b, d) \in E \setminus E'$ where for each turn of type $(a, b, c) \in T'$ respective $(c, d, e) \in T'$ a turn $(a, b, d) \notin P$ respective $(b, d, e) \notin P$ is added to T'.

The substitution is one-to-one. Any path that includes the turn (b, c, d) with or without further connected turns of type (a, b, c) and (c, d, e) can be realized using the edge (b, d) or the turns (a, b, d) and (b, d, e) and vice versa.

Because there exists at most one dependency path between any two edges, all dependency paths that included t have to include s after substitution and all dependency paths that include s had to include t before substitution, provided s was not used before which is ensured by the condition $s \notin E'$.

Since all dependency paths that formerly included t and only these paths include s instead, the substitution does not introduce new dependencies. Again there exists at most one dependency path between any two edges and the proof applies inductively to all following iterations of the algorithm. □

4 Application to the DFN G-WiN Topology

In this section, we give an example applying our algorithm to the DFN G-WiN topology as of 2000 [10] that is shown in Fig. 6 (a). More recent topologies are available, see for example [9], however, the chosen topology reveals considerable complexity while still being useful for demonstrative purposes.

The level one nodes have been investigated by Alg. 1 in order of their numbering. Note that the unnumbered level two nodes fulfill the feed-forward property trivially. The resulting 7 bidirectional prohibited turns are marked by arcs in Fig. 6 (b). Further on, the edges that belong to the considered shortest path spanning tree rooted at the vertex with the highest label are indicated by solid lines compared to dashed lines that are edges which do not belong to the tree.

Figure 7 shows the respective dependency graph where level two nodes and corresponding edges and turns are left out for clearness. Note, however, that

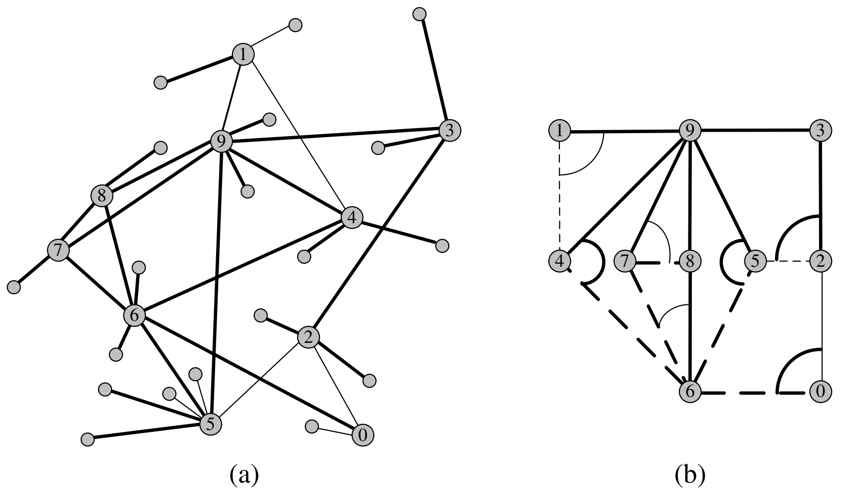

Fig. 6. G-WiN topology as of 2000, turn-prohibition, and spanning tree

given a level one node b, all edges (b, c) may share common predecessors (a, b) that are not displayed, where a is a level two node, respective all edges (a, b) may share common successors (b, c), where c is a level two node. While we take the first into account to ensure independence in the level one core, we ignore the latter such that flows when leaving the level one core may be dependent.

Solid lines in Fig. 7 indicate turns that belong to the graph after substitution of nonessential turns by edges. In the first iteration of Alg. 3 turns $(1, 9, 4)$ and $(4, 9, 1)$ have been substituted by edges $(1, 4)$ and $(4, 1)$ and turns $(8, 9, 7)$ and $(7, 9, 8)$ have been replaced by edges $(8, 7)$ and $(7, 8)$ which have been connected by turns $(6, 8, 7)$ and $(7, 8, 6)$. In the second iteration turns $(6, 8, 7)$ and $(7, 8, 6)$ have been substituted by edges $(6, 7)$ and $(7, 6)$.

The vertices in Fig. 7 that are marked grey correspond to edges that are part of the spanning tree in Fig. 6. The remaining vertices are marked white and all further turns that did not violate the unique dependency path property according to Alg. 4 are marked by dashed lines. The incremental procedure to add these turns started with turns that are part of potential shortest paths and processed these in the order of the numbering of the vertices, that is turns starting or ending at the vertex with the lowest index were tested first.

We find that of the 86 possible directed turns 14 have to be prohibited to ensure the feed-forward property and further 22 have to be prohibited to ensure unique dependency paths, thus 50 permitted turns remain. To ensure unique dependency paths the following bidirectional turns are prohibited: $(0, 2, 5)$, $(0, 6, 8)$, $(1, 4, 9)$, $(2, 5, 9)$, $(4, 6, 5)$, $(4, 6, 8)$, $(5, 6, 7)$, $(5, 6, 8)$, $(6, 7, 9)$, $(6, 8, 7)$, $(7, 8, 9)$. Yet, only two paths with minimal hop count, $(0, 6, 8)$ and $(8, 6, 0)$, have to be replaced by the longer paths $(0, 6, 7, 8)$ and $(8, 7, 6, 0)$, due to the prohibition of turns $(0, 6, 8)$ and $(8, 6, 0)$ to ensure independence in the level one core.

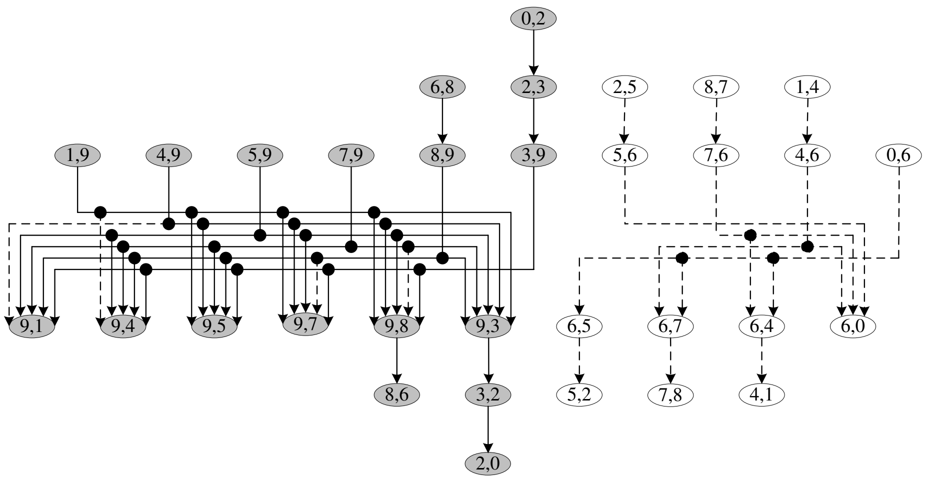

Fig. 7. G-WiN dependency graph

5 Concluding Remarks

A variety of powerful and elegant methods for network analysis rely on the assumption of independence of flows at multiplexing points. However, the correctness of this supposition is usually not investigated for specific topologies. To this end, we developed an extended turn-prohibition algorithm that ensures both the feed-forward and the unique dependency path property, under which flows that are independent at the ingress of the network retain this attribute throughout the core of the network until they are potentially multiplexed.

While the turn-prohibition approach is considered to be efficient [16], it clearly impacts routing performance. Yet, in case of the DFN G-WiN topology which can be considered to be challenging in this respect, we presented a solution where only one pair of shortest paths has to be replaced by paths with one additional hop to ensure independence in the level one core of the network.

Areas of application are for example emerging MPLS [14] networks with explicit routing and DiffServ [1] like aggregate service provisioning where turn-prohibition can be applied to compose a premium service with defined service guarantees. In this scenario, common best-effort traffic is unaffected and can still be routed along any path, thus providing an efficient solution.

A particular advantage of our analysis using dependency graphs is given in case of methods for performance analysis that make formulas available both for multiplexing independent and dependent flows, for example [6, 19]. In this case, individual decisions can be made, either to adapt the routing to ensure independence or to apply rules for multiplexing dependent flows, whenever the dependency graph indicates the necessity to do so, as in case of the outgoing links from level one to level two nodes in the G-WiN example.

Acknowledgements

This work was supported in part by an Emmy Noether grant of the German Research Foundation. The Q2S Centre of Excellence is appointed by the Research Council of Norway and funded by the Research Council, NTNU and UNINETT.

References

1. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. *An Architecture for Differentiated Services.* RFC 2475, 1998.
2. G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications.* Wiley, 1998.
3. R.-R. Boorstyn, A. Burchard, J. Liebeherr, and C. Oottamakorn. *Statistical Service Assurances for Traffic Scheduling Algorithms.* IEEE JSAC, 18(12):2651-2664, 2000.
4. C.-S. Chang. *Performance Guarantees in Communication Networks.* Springer, 2000.
5. J. Duato, S. Yalamanchili, and N. Lionel. *Interconnection Networks: An Engineering Approach.* Morgan Kaufmann, 2003.
6. M. Fidler. *Elements of Probabilistic Network Calculus Applying Moment Generating Functions.* Preprint Series of the Institut Mittag-Leffler, Sweden, 2005.
7. M. Fidler and G. Einhoff. *Routing in Turn-Prohibition Based Feed-Forwad Networks.* LNCS 3042, Springer, Proceedings of Networking, pp. 1168-1179, 2004.
8. B. R. Haverkort. *Performance of Computer Communication Systems: A Model-Based Approach.* Wiley, 1999.
9. O. Heckmann, M. Piringer, and R. Steinmetz. *On Realistic Network Topologies for Simulation.* Proceedings of the ACM Sigcomm Workshops, pp. 28-32, 2003.
10. G. Hoffmann. *G-WiN - the Gbit/s infrastructure for the German scientific community.* Elsevier Computer Networks, 34(6):959-964, 2000.
11. F. Kelly. *Notes on Effective Bandwidths*, Stochastic Networks: Theory and Applications, Royal Statistical Society Lecture Notes Series, 4:141-168, 1996.
12. J.-Y. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queueing Systems for the Internet.* Springer, 2001.
13. J. Liebeherr, S. D. Patek, and A. Burchard. *Statistical Per-Flow Service Bounds in a Network with Aggregate Provisioning.* Proceedings of IEEE Infocom, 2003.
14. E. Rosen, A. Viswanathan, and R. Callon. *Multiprotocol Label Switching Architecture.* RFC 3031, 2001.
15. M. D. Schroeder, A. D. Birrell, M. Burrows, H. Murray, R. M. Needham, and T. L. Rodeheffer. *Autonet: A High-speed, Self-configuring Local Area Network Using Point-to-point Links.* IEEE JSAC, 9(8):1318-1335, 1991.
16. D. Starobinski, M. Karpovsky, and L. Zakrevski. *Application of Network Calculus to General Topologies using Turn-Prohibition.* IEEE/ACM ToN, 11(3):411-421, 2003.
17. D. Starobinski and M. Sidi. *Stochastically Bounded Burstiness for Communication Networks.* IEEE TIT, 46(1):206-216, 2000.
18. D. Wischik. *The output of a switch, or, effective bandwidths for networks.* Queueing Systems, 32(4):383-396, 1999.
19. Q. Yin, Y. Jiang, S. Jiang, and P. Y. Kong. *Analysis of Generalized Stochastically Bounded Bursty Traffic for Communication Networks.* Proceedings of IEEE LCN, pp. 141-149, 2002.