

ILP Formulation for Monitoring-Cycle Construction Using Segment Routing

Xiaoqian Li and Kwan L. Yeung

Dept. of Electrical and Electronic Engineering

The University of Hong Kong, Hong Kong

E-mail: {xqli, kyeung}@eee.hku.hk

Abstract—A monitoring-cycle can be easily implemented using segment routing and subject to a given maximum segment list size. In this paper, we propose the first ILP formulation (ILP1) to optimally solve the problem of covering every link in the network using monitoring-cycles and with minimum cycle cover length. To further conserve network bandwidth, we extend ILP1 to jointly minimize the total segment list size needed (ILP2). Since the time required to detect a network failure is affected by the longest cycle, we also extend ILP1 to jointly minimize the length of the longest cycle (ILP3). Finally, since all the existing network monitoring algorithms need to maintain a dedicated monitoring topology, we evaluate the gain brought by the monitoring topology using our ILPs. We found that the gain is limited, therefore we argue that maintaining a dedicated monitoring topology for monitoring-cycle construction is unnecessary.

Keywords—segment routing; integer linear program (ILP); network monitoring; cycle cover; minimum cycle cover problem.

I. INTRODUCTION

Network monitoring is about detecting network component failures and is important in providing reliable end to end communications. Existing monitoring techniques can be divided into two types, protocol-based and probe-based. A protocol-based scheme uses control plane message (e.g. BFD [1]) to infer link failures. But failures only affecting data plane, such as packet drops due to data plane bugs and misconfigurations, cannot be detected. A probe-based monitoring scheme addresses above problems by directly sending probe packets onto the data-plane to detect failures. We call the device that sends/receives probe packets as a monitor and the monitor is placed at a *source node*. At a given source node, a cycle-based monitoring scheme [2, 3] is implemented by using a monitor to send probe packets along a set of pre-determined monitoring cycles to cover/traverse every link in the network. By comparing the probe packets sent into and received from the cycles, the monitor can detect/infer a network failure if there is any discrepancy.

Throughout the paper, we consider a network that can be modelled as a *symmetric directed graph*, i.e., if two nodes are connected, there always exists a pair of directed links (in two directions) between them. In Fig. 1 (and others), for simplicity, each pair of directed links is shown as an undirected link. We focus on designing efficient cycle-based monitoring scheme. Let the number of links in a cycle be its length. Then the set of cycles, originating from the given source node and covering every directed link in the network, form a *cycle cover*. Let the *cycle cover length* be the total length of all the cycles found. An efficient cycle-based monitoring scheme should find a cycle cover with minimum cycle cover length; we call it *minimum*

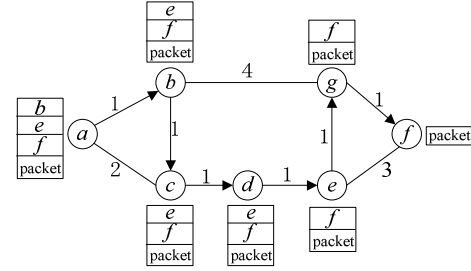


Fig. 1: A network to illustrate segment routing. The number next to each link is its IGP link cost.

cycle cover problem. Note that an Euler cycle [4] is a cycle that covers every (directed) link in the network exactly once. If an Euler cycle exists and can be found, it provides the optimal solution to the minimum cycle cover problem. But in a network with segment routing, due to the maximum segment list constraint (see below), Euler cycle usually does not exist.

A cycle-based monitoring scheme requires the network ability of forwarding packets along any arbitrary cycle. This is difficult to implement in conventional IP networks because of the shortest path routing. The new segment routing [5, 6] makes it easy. Segment routing is a routing mechanism that facilitates traffic engineering [7]. It can be easily deployed on an MPLS (Multiprotocol Label Switching) [8] data plane while keeping an Interior Gateway Protocol (IGP) such as OSPF (Open Shortest Path First) [9], IS-IS (Intermediate System to Intermediate System) [10] on its control plane. Segment routing relies on source routing to forward packets along any arbitrary path or cycle without keeping per-flow state at intermediate nodes. In segment routing, a forwarding path consists of concatenated segments. Each segment is identified by a *segment identifier* (SID). The forwarding path is thus identified by a list of SIDs, or a *segment list*. At an ingress node of a segment routing domain, a segment list is added to the packet and the packet is forwarded along a shortest path (node-SID) or a link (adjacency-SID) towards the network element represented by the topmost SID in the list. At the *penultimate node* of each segment [8], the topmost SID is popped and the packet continues its journey based on the *new* topmost SID until the destination is reached. As shown in Fig. 1, a packet is sent from node *a* to *f* along path (*a, b, c, d, e, g, f*) using segment list {*b, e, f*}. Similarly, if node *a* wants to forward a probe packet along, e.g. cycle (*a, b, c, d, e, g, f, g, e, d, c, b, a*), segment list {*b, e, f, e, b, a*}, with a size of 6 SIDs, can be added to the packet at node *a*. (Please refer to Section II for a more detailed operation of segment routing.)

Note that the segment list for identifying a path/cycle is usually not unique. A shorter list is always preferred because it reduces per packet overhead. The process of finding a smallest segment list to identify a path is called path encoding [11]. Note that there is a practical limit on the maximum segment list size supported by commercial routers. If we denote this limit by K , then a path/cycle must be identified/encoded by no more than K SIDs. In a cycle-based monitoring scheme, if a cycle needs to be identified by more than K SIDs, it must be split into *smaller* ones, each can be identified with no more than K SIDs, such that the links of the original cycle can be jointly covered by the set of smaller cycles. The total cycle length of those smaller cycles will be no smaller than that of the original one. But if the splitting fails, no feasible solution can be found.

With segment routing, the minimum cycle cover problem becomes: From a given source node, find a cycle cover such that its length is minimized and each cycle is encoded by K or less SIDs. To the best of our knowledge, SCMon [2] is the first algorithm to solve the minimum cycle cover problem. SCMon consists of two phases. In Phase 1, a dedicated *monitoring topology* is constructed by designing a set of new link costs. The goal is to eliminate any non-shortest-path links (i.e., links that do not belong to any shortest path) and equal-cost-multi-paths (ECMPs) in the monitoring topology. It is believed that a network free of non-shortest-path links and ECMPs can utilize the segment list more efficiently (see Section II). As an example, Fig. 2(a) is a monitoring topology (with new link costs) obtained using SCMon on a 3×3 grid network. In Phase 2, a greedy algorithm aiming at minimizing the number of cycles required is designed to find a cycle cover for the monitoring topology. With source node at a and $K=10$, Fig. 2(b) shows the cycle cover found by SCMon, which consists of C_1 and C_2 , and with a cycle cover length of 32 links. (Please refer to [2] for the details of SCMon.)

Following the same two-phase approach, the performance of SCMon is enhanced by a series of four algorithms in [3], which are denoted as algorithms $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$. Among them, $\mathcal{D}$ features a parallel-and-sequential cycle-construction process and gives the best overall performance. Applying $\mathcal{D}$ to the same monitoring topology in Fig. 2(a), the cycle cover found is shown in Fig. 2(c), which has a (reduced) cycle cover length of 24. Since the total number of directed links in Fig. 2(a) is also 24, the solution found by $\mathcal{D}$ is optimal. Indeed, C_1 and C_2 in Fig. 2(c) can be connected at node a to form an Euler cycle (assume K is large enough). Note that two-cycle solution here is preferred because each cycle has a shorter segment list than the single Euler cycle. (Please refer to [3] for the details of algorithm $\mathcal{D}$.)

All existing algorithms [2, 3] for solving the minimum cycle cover problem are based on heuristics. In this paper, we propose the first ILP formulation (ILP1) to optimally solve the minimum cycle cover problem. Since a segment list is added to each packet, a smaller segment list is preferred. A second ILP formulation (ILP2) is then designed to jointly minimize the cycle cover length and the total segment list size of all cycles found. Since the time required to detect a network failure is determined by the length of the cycle it traversed, we propose the third ILP (ILP3) while aiming at jointly minimizing the cycle cover length and the length of the longest cycle found. Finally, as all existing algorithms require a dedicated monitoring topology, we study the performance gain brought by the monitoring topology using our

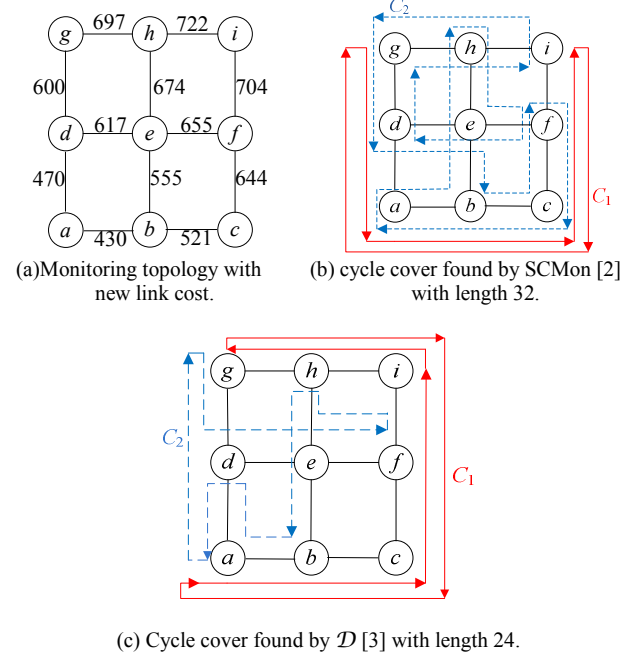


Fig. 2: Monitoring-cycle construction with source at node a and $K=10$.

ILPs. Our numerical results show that the gain is limited, therefore we argue that maintaining a dedicated monitoring topology for probe packets only is unnecessary.

The rest of paper is organized as follows. In Section II, we highlight some salient features of segment routing, which can be used to justify the need for a monitoring topology. In Section III, ILP1 is formulated for solving the basic minimum cycle cover problem. In Section IV, ILP2 and ILP3 are designed to jointly consider the total segment list size or the length of the longest cycle. In Section V, two key ILP parameters, the minimum value of K to ensure a feasible solution and the number of cycles required by a feasible cycle cover, are discussed. In Section VI, numerical results are presented and we conclude the paper in Section VII.

II. SEGMENT ROUTING

There are two basic types of SIDs in segment routing, *node-SID* and *adjacency-SID*. A node-SID identifies a *shortest path* ending at a node, e.g. SID = d identifies a segment (shortest path) ending at node d . An adjacency-SID identifies a *single link* in the network, and we use $\langle a, b \rangle$ as the adjacency-SID for identifying the link from nodes a to b . Consider the network shown in Fig. 1. Suppose ingress node a wants to forward a packet to destination f . There are three shortest paths, or *Equal Cost Multi-paths* (ECMPs), from a to f , i.e., (a, b, g, f) , (a, b, c, d, e, g, f) and (a, c, d, e, g, f) . If a single-element segment list $\{f\}$ is added to the packet at node a , the packet will choose one of the three paths based on the specific load balancing algorithm (or hashing function) implemented at a . In this case, the packet is guaranteed to arrive f , but there is no guarantee a specific path in the ECMP group will be used. Note that in network monitoring, every link must be unambiguously covered.

If the packet is required to follow, e.g. the arrowed path (a, b, c, d, e, g, f) , extra SIDs are needed in the segment list. For

example, with segment list $\{b, e, f\}$, the packet will be first forwarded to b along shortest path (a, b) , and the topmost SID = b will be popped at node a using the *penultimate hop popping* [8]. Then the packet will be forwarded to e (with SID = e popped at d) along shortest path (b, c, d, e) , and finally to destination f (with SID = f popped at g) along shortest path (e, g, f) . From this example, we can see that a node-SID can only unambiguously identify a *unique shortest path* (a shortest path that does not belong to any ECMPs). To unambiguously identify a path in an ECMP group, extra SIDs are needed.

Since a segment list is added to every packet at the ingress node, a shorter segment list can reduce per packet overhead. To find the smallest segment list for encoding/identifying a given path [11], node-SIDs are generally preferred over adjacency-SIDs, because an adjacency-SID can only identify a single link. But there are two cases that adjacency-SID must be used. First, to identify a link that does not belong to any shortest path, or a *non-shortest-path link*. For example, if we want a packet to follow path (a, b, c, d, e, f) in Fig. 1, since link (e, f) is a non-shortest-path link, a segment list including adjacency-SID $\langle e, f \rangle$, or $\{b, e, \langle e, f \rangle\}$, is needed. Another case that requires an adjacency-SID is to identify a link who is a member of an ECMP group; we call it an *ECMP link*. In Fig. 1, link (b, g) and path (b, c, d, e, g) are two ECMPs from nodes b to g , and thus link (b, g) is an ECMP link. If b wants to send a packet to g using link (b, g) , segment list $\{\langle b, g \rangle\}$ must be used.

From the examples above, we can see that if a network has ECMPs, non-shortest-path links or ECMP links, a longer segment list is needed to unambiguously identify a path/cycle. Therefore, if a network is free of ECMPs (thus no ECMP links) and non-shortest-path links, SIDs in the segment list can be better utilized. This is the rationale behind the two-phase approach [2, 3] of designing a cycle-based network monitoring scheme. As a result, a dedicated monitoring topology, free of both non-shortest-path links and ECMPs, is constructed (in the first phase) by designing and maintaining a new set of links costs for forwarding/routing probe packets only.

III. ILP FOR MINIMIZING CYCLE COVER LENGTH

Assume network $G = (N, E)$ is a symmetric directed graph, where N is the set of all the nodes in the network, E is the set of all the directed links in the network. We use $|N|$ to denote the total number of nodes in the network, and $|E|$ to denote the total number of directed links in the network. In a symmetric directed graph, if two nodes are connected, there always exists a pair of directed links between them. That means every node has equal in-degree and out-degree. In this case, an Euler cycle must exist. If the maximum segment list size allowed K is large enough to encode/identify the Euler cycle, or its multi-cycle equivalent (see Fig. 2(c)), the minimum cycle cover length is the number of directed links in the network (i.e., $|E|$). But if K is not large enough, the minimum cycle cover length cannot achieve $|E|$ because the SIDs in the segment list will be exhausted before the Euler cycle (or its multi-cycle equivalent) is represented. In other words, when K increases, the minimum cycle cover length will converge to $|E|$.

For a given source node and K , we aim at using ILP to find a cycle cover with the minimum cycle cover length and every

cycle can be represented by K or less SIDs. We denote this ILP as ILP1. ILP1 will be extended to also consider either the total segment list size (ILP2) or the longest cycle (ILP3) in Section IV. In the following, we first define the notations to be used by all three ILPs. Like [2, 3], a monitoring cycle can visit a node multiple times but not a link. In other words, a cycle can be either simple or non-simple.

A. Notations

- s : The source node to monitor all the directed links in the network, where $s \in N$.
- K : The maximum segment list size allowed.
- P : The set of all the *one-segment paths* in the network. A one-segment path is a path that can be unambiguously identified by one SID.
- p : A specific path in P , i.e., $p \in P$.
- s_p : The source node of path p .
- t_p : The end node of path p .
- l_p : The number of links in path p .
- e_p : A precomputed constant. It takes the value of 1 if link e ($e \in E$) is traversed by path p , and 0 otherwise.
- J : The maximum number of cycles allowed in the solution.
- j : Cycle index where $j \in \{1, 2, \dots, J\}$.
- C_p^j : Binary variable. It takes the value of 1 if p is in the j -th cycle, or C_j , and 0 otherwise.
- x_a^j : Binary variable. It takes the value of 1 if p is in C_j with $s_p = a$ or $t_p = a$, where $a \in N$, and 0 otherwise. This variable is to generate a new graph G'_j based on C_j by adding the source node and end node of path p in C_j to the new graph.
- e'_{ab}^j : Binary variable. It takes the value of 1 if p is in C_j with $s_p = a$ and $t_p = b$, where $a, b \in N$, and 0 otherwise. This variable is to generate the new graph G'_j based on C_j by adding a *directed* link from the source node to the end node of path p in C_j to the new graph.
- $f_{uv|ab}^j$: Binary variable. This variable assumes that node pair $[a, b]$ in the new graph G'_j is checked with a flow-based analysis [12]. It takes the value of 1 if there is a flow traverses link (u, v) in G'_j , and 0 otherwise.
- r_1 : A predefined constant only used in ILP2 (Section IV). Its value determines the relative importance between the cycle cover length and the total segment list size of all cycles.
- r_2 : A predefined constant only used in ILP3 (Section IV). Its value determines the relative importance between the cycle cover length and the length of the longest cycle.
- L : General integer variable only used in ILP3 (Section IV). It is the length of the longest cycle.

B. ILP1 Formulation

An important input to our ILP1 is P , the set of *one-segment paths* in the network. A one-segment path is a path that can be unambiguously identified by one SID, which can be either a node-SID or an adjacency-SID. Accordingly, P contains all the unique shortest paths between any node pairs in the network, all the non-shortest-path links, and all the ECMP links (see Section II). If a cycle is formed by, e.g. three one-segment paths, it can be identified by a segment list with 3 SIDs (or less; see later).

With P , the constraint of no more than K SIDs in a segment list can be easily enforced by making sure each cycle is formed by K or less one-segment paths.

Another important input to ILP1 is J , the maximum number of cycles allowed in our ILP solution. The running time of ILP1 is sensitive to the value of J . The smallest/optimal value of J is the number of cycles in the cycle cover found by ILP1, or an output. But it is not known until ILP1 is solved. Some suggestion on setting J is given in Section V.B. In the following, we assume J is appropriately set.

Given source s , the maximum segment list size K , all the one-segment paths P , and the number of cycles allowed in the solution J , we can find a cycle cover with minimum length by solving ILP1 below.

Objective:

$$\text{minimize } \sum_{j \in J} \sum_{p \in P} l_p \cdot C_p^j \quad (1)$$

Subject to:

$$\sum_{s_p=n} C_p^j - \sum_{t_p=n} C_p^j = 0, \forall n \in N, \forall j \in J \quad (2)$$

$$\sum_{p \in P} C_p^j \leq K, \forall j \in J \quad (3)$$

$$\sum_j \sum_{p \in P} e_p C_p^j \geq 1, \forall e \in E \quad (4)$$

$$\sum_{s_p=s} C_p^j \geq \frac{1}{K} \sum_{p \in P} C_p^j, \forall j \in J \quad (5)$$

$$\frac{1}{K} \sum_{s_p=a} C_p^j \leq x_a^j \leq \sum_{s_p=a} C_p^j, \forall j \in J, a \in N \quad (6)$$

$$\frac{1}{K} \sum_{t_p=a} C_p^j \leq x_a^j \leq \sum_{t_p=a} C_p^j, \forall j \in J, a \in N \quad (7)$$

$$\frac{1}{K} \sum_{s_p=a, t_p=b} C_p^j \leq e_{ab}^j \leq \sum_{s_p=a, t_p=b} C_p^j, \forall j \in J, \forall a, b \in N \quad (8)$$

$$f_{uv} |_{sb}^j \leq e_{uv}^j, \forall b, u, v \in N : b \neq s, \forall j \in J \quad (9)$$

$$\sum_v f_{uv} |_{sb}^j - \sum_w f_{wu} |_{sb}^j = 0, \forall j \in J, \forall b, u \in N : u \neq s, u \neq b, b \neq s \quad (10)$$

$$\sum_v f_{sv} |_{sb}^j - \sum_w f_{ws} |_{sb}^j \leq 1, \forall b \in N : b \neq s, \forall j \in J \quad (11)$$

$$\sum_v f_{bv} |_{sb}^j - \sum_w f_{wb} |_{sb}^j \geq -1, \forall b \in N : b \neq s, \forall j \in J \quad (12)$$

$$1 + \sum_v f_{sv} |_{sb}^j - \sum_w f_{ws} |_{sb}^j \geq x_s^j + x_b^j, \forall b \in N : b \neq s, \forall j \in J \quad (13)$$

Constraint (2) defines the j -th cycle, or C_j , by requiring each node to have equal outgoing and incoming links incident on C_j . Constraint (3) ensures every C_j is formed by K or less one-segment paths. The cycle cover consists of all the cycles C_j ($j=1, 2, \dots, J$) and constraint (4) needs every directed link in the network be covered at least once. Constraint (5) guarantees every C_j must originate from source node s . Since J is set large enough to find the minimum cycle cover, it is possible that C_j is *empty* (not containing any nodes/links), or $\sum_{p \in P} C_p^j = 0$. When a cycle is non-empty, we have $0 < \frac{1}{K} \sum_{p \in P} C_p^j \leq 1$ from (3), and $\sum_{s_p=s} C_p^j \geq 1$ from (5) to ensure the cycle must be originated

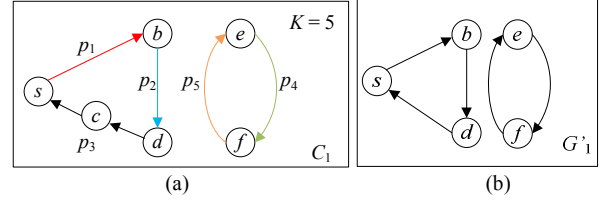


Fig. 3. An example to illustrate the problem of disjoint cycles. (a) C_1 is formed by two disjoint cycles. (Nodes and links not traversed by C_1 are omitted.) (b) A new graph generated based on C_1 .

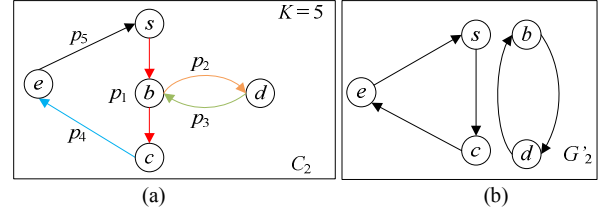


Fig. 4. An example to illustrate the problem of split-segments. (a) C_2 contains a split-segment p_1 . (b) A new graph generated based on C_2 .

from source s . In short, constraints (2), (3) and (5) tries to make sure every non-empty cycle C_j found is valid, and constraint (4) wants to ensure all the cycles found form a cycle cover.

If ILP1 is formulated by (1) to (5) only, *there are three potential problems*. First, cycle C_j may contain multiple disjoint cycles, e.g., C_1 in Fig. 3(a) satisfies (2), (3) and (5), but it consists of two disjoint cycles. C_1 is invalid because a packet generated by source node s can never reach (disconnected) e and f . To address this disjoint cycle found problem, we need extra constraints to make sure C_j is a single-connected cycle.

The second problem with C_j found using constraints (2) to (5) only is that it may contain *split-segments*, causing the *actual* SIDs required to encode the cycle to be larger than K . An example is shown in Fig. 4(a) with $K=5$. Here $C_2 = (s, b, d, b, c, e, s)$ is formed by five one-segment paths, p_1, p_2, p_3, p_4 and p_5 , and the segment list size constraint in (3) is met. But when a packet is forwarded along C_2 , the sequence of paths traversed is: $(s, b), p_2, p_3, (b, c), p_4, p_5$. Note that the two subpaths of p_1 , (s, b) and (b, c) , are *not* visited consecutively. We call p_1 a *split-segment*. Since each subpath of p_1 needs a different SID for encoding it, this will increase the actual segment list size (from 5) to 6, rendering C_2 invalid. Note that if (s, b) and p_2 can be adjoined to form a one-segment path (s, b, d) , one SID will be saved and the *actual* segment list size for encoding C_2 remains 5. But there is no guarantee that (s, b, d) is a one-segment path. To solve the problem of not being able to guarantee the *actual* segment list size be no bigger than K , extra constraints are needed to ensure C_j does not contain split-segments.

The third problem with C_j is that the segment list obtained, by transforming each one-segment path in C_j to one SID, may not be the shortest/optimal. Without loss of generality, assume C_j is formed by connecting K one-segment paths (with no split-segments), $p_1, p_2, \dots$, and p_K , sequentially. Obviously, C_j can be identified by K SIDs. If the path formed by adjoining two consecutive one-segment paths, say, p_1 and p_2 , is also a one-segment path, then this adjoined path can be encoded using a

single SID. (Note that any subpath of a one-segment path is also a one-segment path.) So only $K - 1$ SIDs are needed to encode C_j . In other words, constraint (3) ensures no more than K one-segment paths in C_j , but not the smallest number of one-segment paths being used. As the smallest number of one-segment paths or not does not affect the cycle cover length in (1), this problem will not break the ILP. Besides, the inefficiency due to longer-than-needed segment list size can be solved by re-encoding C_j using a path encoding algorithm [11]. (Note that ILP2 does not have this problem by taking into the total SIDs into account.)

In the following, we focus on solving the first two problems. Our approach is to generate a new graph G'_j based on C_j and conduct a *flow-based analysis* [12] to ensure G'_j is connected. G'_j is generated by keeping the source and destination nodes of every one-segment path in C_j once, and linking each pair of source and destination nodes by a directed link. For example, G'_1 in Fig. 3(b) is generated from C_1 in Fig. 3(a), and G'_2 in Fig. 4(b) is generated from C_2 in Fig. 4(a). In ILP1, the generation of G'_j is implemented by constraints (6) to (8). In particular, when C_j is empty, no nodes and links will be added to G'_j ; otherwise, G'_j is generated by setting $x_a^j=1$ and $e'_{ab}^j=1$ in (6)-(8).

In G'_j , if C_j contains multiple disjoint cycles, G'_j will not be connected (as shown in Figs. 3(a) & 3(b)). Similarly, if C_j contains split-segments, G'_j will not be connected (as shown in Figs. 4(a) & 4(b)). Based on the observations above, if G'_j is connected, the problems of disjoint cycles and split-segments are simultaneously solved. To examine if G'_j is connected, a flow-based analysis is conducted to ensure there is a flow/path between every node pair in G'_j . Consider an arbitrary node pair $[a, b]$. When node a generates a unit flow to b , if they are connected, all nodes in G'_j will obey the flow conservation except a and b , and b must receive this unit flow; otherwise, they are not connected. Take node pair $[s, b]$ in Fig. 4(b) as an example. When one unit flow is generated by node s , this unit flow will finally return to s (not b). That implies, nodes s and b are not connected in Fig. 4(b).

Constraints (9)-(13) are designed to conduct the flow-based analysis. To minimize the variables needed, only node pairs originating from source s are examined. Note that according to (2), if there exists a path from s to b in G'_j , there must be a path from b back to s . Constraint (9) guarantees that the flow must move along the links in G'_j . Constraint (10) enforces the flow conservation constraint except at nodes s and b . Constraints (11) and (12) imply that there is at most one unit flow generating from source s , and at most one unit flow receiving by node b . Finally, (13) says that if both s and b are in G'_j , a unit flow will be generated by s but will not be received by s . Combining (9) to (13), if both nodes s and b are in G'_j , there must be a flow (or a path) from nodes s to b ; otherwise, they cannot both exist in G'_j . This guarantees G'_j found is connected, and thus C_j found contains neither disjoint cycles nor split-segments.

IV. TWO EXTENDED ILPS

Based on ILP1, two new ILPs, ILP2 and ILP3, are formulated in this section. ILP2 aims at jointly minimizing the cycle cover length and the total segment list size of all the cycles

found. ILP3 aims at jointly minimizing the cycle cover length and the length of the longest cycle found.

A. ILP2: minimizing cycle cover length and total SIDs

In the previous section, ILP1 is formulated to minimize the cycle cover length. No doubt that minimizing cycle cover length can save network bandwidth. Since a segment list is added to every packet, network bandwidth is also affected by the total segment list size of all the cycles found. As an example, the two cycles found by SCMon in Fig. 2(b) can be expressed as:

$$C_1=(a, b, c, \underline{f}, \underline{i}, \underline{f}, c, b, \underline{a}, d, \underline{g}, d, \underline{a});$$

$$C_2=(a, b, e, \underline{h}, e, \underline{f}, e, d, \underline{g}, h, \underline{i}, h, \underline{g}, d, \underline{e}, \underline{b}, c, \underline{f}, c, b, \underline{a}).$$

(For each cycle, the segment list is formed by the underlined nodes.) The cycle cover length is 32 links and the total SIDs needed is 13 (4 SIDs for C_1 , 9 SIDs for C_2). Although the topmost SID in the segment list will be popped as a packet is being forwarded along its path, for simplicity we only focus on the overhead due to the initial segment list size.

In ILP1, the segment list size required for each cycle is bounded by K but the total segment list size of all cycles found has not been considered. In this section, we extend ILP1 to jointly consider the cycle cover length and the total segment list size required. In our new ILP formulation (ILP2), the following objective function is used:

$$\text{minimize } \sum_{j \in J} \sum_{p \in P} l_p \cdot C_p^j + r_1 \cdot \sum_{j \in J} \sum_{p \in P} C_p^j, \quad (14)$$

where r_1 in (14) is used to determine the relative importance between the cycle cover length (first term) and the total SIDs needed (second term). Note that (14) can only be achieved if the number of one-segment paths in each cycle is minimized, i.e., the segment list size is minimized. In other words, ILP3 does not have the (third) problem of using longer-than-needed segment list size (see Section III), thus no path encoding algorithms [11] are required for the cycle found by ILP2.

If segment routing is implemented using the MPLS data plane, each SID in the segment list will be mapped to a 4-byte MPLS label [8]. If the segment list size added to the packet is significantly smaller than the size of a whole probe packet, minimizing the cycle cover length is of paramount importance. To make sure the cycle cover length found is minimized *before* minimizing the total segment list size, we can set

$$r_1 = \frac{1}{J \cdot K}. \quad (15)$$

Since the total segment list size is bounded by $J \cdot K$, r_1 in (15) makes sure that the maximum contribution of the second term in (14) is less than one (link).

B. ILP3: minimizing cycle cover length and the longest cycle

In addition to minimizing network bandwidth consumption, it is also important for a network monitoring scheme to detect any network fault as quickly as possible. A cycle-based network monitoring scheme operates by comparing the probe packets sent into and received from all the cycles (in the cycle cover). Since different cycles have different lengths, e.g. C_1 and C_2 in Fig. 2(b) have lengths of 12 and 20 respectively, the fault detection time is determined by the longest cycle. To minimize

the fault detection time, the length of the longest cycle should be minimized. Accordingly, ILP3 is formulated using the new objective function in (16) and an additional constraint in (17).

$$\text{minimize } \sum_{j \in J} \sum_{p \in P} l_p \cdot C_p^j + r_2 \cdot L \quad (16)$$

$$\sum_{p \in P} l_p \cdot C_p^j \leq L, \forall j \in J \quad (17)$$

Note that L in (16) is the maximum cycle length for all the cycles found, which is guaranteed by constraint (17). Since r_2 in (16) determines the relative importance between the cycle cover length and L , if we want to minimize the cycle cover length *before* the length of the longest cycle, we can set $r_2 = 1/|E|$. Since the maximum cycle length can never be larger than $|E|$, setting $r_2 = 1/|E|$ makes sure the maximum contribution of the second item in (16) is less than one.

V. DISCUSSION

In this section, two key ILP parameters, the minimum value of K to ensure a feasible solution and J the number of cycles required for a feasible cycle cover, are discussed.

A. Minimum value of K

The fundamental problem in this paper is to find a cycle cover such that every cycle can be represented by K or less SIDs. It is interesting to ask: For an arbitrary topology (modelled by a symmetric directed graph), what is the minimum value of K such that a feasible solution is guaranteed? Obviously, if $K=1$, no feasible solution can be found because no cycle can be formed/identified by a segment list with only one SID. If $K=2$, a cycle can be formed from source node s to another node x (along the shortest path) and back, using a two-element segment list $\{x, s\}$. In doing so, all links on the shortest path tree rooted at s can be covered, but other links cannot. Therefore, $K=2$ also cannot guarantee a feasible solution.

Next we show that if $K=3$, a feasible solution always exists. Consider a naïve way of finding a cycle cover: Each directed link is covered by a dedicated cycle. Without loss of generality, to cover link (x, y) , a segment list with 3 SIDs, $\{x, \langle x, y \rangle, s\}$, is sufficient. Specifically, node-SID x ensures the probe packet will reach node x . This is true even if there are ECMPs between nodes s and x , since we do not care which path in the ECMPs will be traversed (see Section II). Then, adjacency-SID $\langle x, y \rangle$ ensures that the probe packet will traverse link (x, y) to reach node y . Finally, node-SID s ensures that the probe packet will be routed back to source s .

Note that in the three ILPs formulated by (1) to (17), there is a catch that cycles must be constructed using one-segment paths stored in P . Recall that a one-segment path is a path that can be unambiguously identified by one SID. Since paths in an ECMP group cannot be unambiguously identified by a single SID, they are excluded from P . So if K is small, e.g. $K=3$, our ILPs may not be able to find a feasible cycle cover solution even if such a solution exists. This (corner) problem can be easily fixed by extending P to include two subsets, the original set of one-segment paths P_1 , and the new set of ECMPs (excluding ECMP links) P_2 . Note that all the paths in the extended P can be identified by one SID though the identification of an ECMP path

is not unambiguous. To form a cycle cover, every link in the network must be unambiguously covered. That means when the extended P is used in our ILPs, constraint (4) needs to be modified to ensure every link is covered at least once using the paths stored in P_1 (not P or P_2).

B. How to choose J

In our ILPs, the number of ILP variables and constraints soars exponentially with J , the *maximum number of cycles allowed in a cycle cover*. This is because the flow-based analysis (via constraints (9)-(13)) introduces $O(|N|^3 J)$ variables and constraints. If J is set too small, ILP cannot find a feasible solution. If J is set too big, ILP will take too long to complete. In the following, we provide some suggestions on how to choose a just-enough value for J .

To cover all the directed links in the network using the naïve approach of a dedicated cycle per link, the maximum number of cycles J is $|E|$. We can reduce J to $|E|/2$ if the network is free of parallel directed links. This is because when the network is free of parallel directed links, each cycle found must cover at least two unique links. (Consider two cycles C_1 and C_2 . Assume C_1 differs with C_2 by one link, or link e in C_1 is different from e' in C_2 , and other links are the same. Then e and e' must be two parallel directed links.)

To further reduce the value of J , we propose a simple algorithm called SR- J . In SR- J , the value of J is derived based on a shortest path tree rooted at source node s , in particular, this shortest path tree only contains all the unique shortest paths starting from s . In every iteration, we form a cycle by finding the longest uncovered path in the tree (i.e., the path with the largest number of uncovered links); if possible, extending this path by adding a single uncovered link that does not belong to this tree; and then routing back to source s along the opposite direction of the current path. When all the links in this shortest path tree are covered by the cycle construction above, SR- J stops.

Consider the topology in Fig. 2(a), and assume node a is the source. The shortest path tree rooted at a is shown in Fig. 5(a). In the first iteration, we find the longest uncovered path (a, b, c, f, i) , then extend this path by link (i, h) , and finally route back to a along (h, i, f, c, b, a) , we get the first cycle $C_1=(a, b, c, f, i, h, i, f, c, b, a)$. Similarly, C_2 and C_3 can be found in the next two iterations. Note that each cycle found can be identified by 4 SIDs.

When SR- J stops, let the number of cycles found by SR- J be J_1 . Let the remaining number of uncovered links in the network

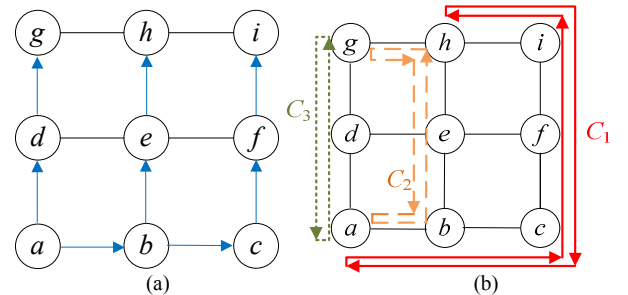


Fig. 5. An example to illustrate SR- J using the topology in Fig. 2(a) with source at node a . (a) A shortest path tree rooted at a . (b) Three cycles found by SR- J .

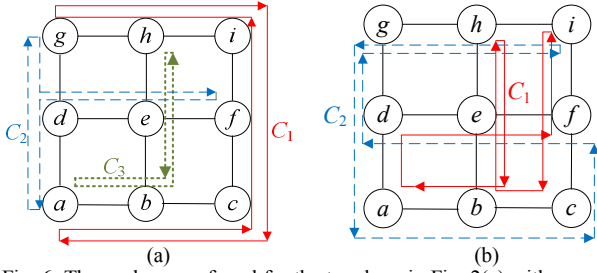


Fig. 6. The cycle cover found for the topology in Fig. 2(a) with source at node a and $K = 5$. (a) Cycle cover found by $\mathcal{D}$ with length 26. (b) Cycle cover found by ILP1 with (optimal) length 24; running time: 0.61 sec.

be J_2 . With SR- J , we can see that all the uncovered links form pairs, e.g. links (e, f) and (f, e) in Fig. 5(b). To cover each pair of uncovered links, a single cycle with a 4-SID segment list such as $\{e, \langle e, f \rangle, \langle f, e \rangle, a\}$ is sufficient. To cover all J_2 links, we need $J_2/2$ cycles. Combining it with the J_1 found from SR- J , we can set $J = J_1 + J_2/2$.

Note that we can also take advantage of the existing heuristic algorithms [2, 3] to set J . If a cycle cover is found by a heuristic algorithm, the value of J can be set to the number of cycles in the cycle cover found. But algorithms in [2, 3] require a dedicated monitoring topology, and our SR- J above does not.

VI. NUMERICAL RESULTS

A. Choosing the value of J

We first use algorithm SR- J to find an appropriate value of J , the maximum number of cycles allowed in a cycle cover, which will be used as input to our ILPs. Table I compares the value of J obtained using SR- J , and the two other approaches, $|E|$ and $|E|/2$, based on four topologies: Grid 3×3 , Grid 4×4 , US network [13] and German network [13]. For each topology, two link cost models are adopted, one [2] is for constructing the monitoring topology, and the other is a unit link cost model. To distinguish the two models in Table I, we use superscript 1 to denote the monitoring topology model, and superscript 2 to denote the unit link cost model.

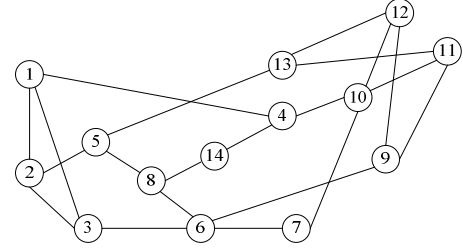
From Table I, we can see that SR- J significantly reduces the value of J , as compared to both $|E|$ and $|E|/2$. This is important in shortening the ILP running time. Besides, the value of J found based on the monitoring topology is smaller than unit link cost model. This is because when the network contains ECMPs, additional SIDs are needed to distinguish ECMPs, which will increase the number of cycles needed to form a cycle cover.

TABLE I. VALUE OF J

J	Grid ¹ 3×3	Grid ² 3×3	Grid ¹ 4×4	Grid ² 4×4	US ¹	US ²	Ger- man ¹	Ger- man ²
$ E $	24	24	48	48	42	42	52	52
$ E /2$	12	12	24	24	21	21	26	26
SR- J	5	8	10	18	8	10	12	15

B. ILP1

We use ILOG CPLEX 12.6 to implement the ILPs on a computer with a 3.2 GHz Intel CPU 6500. First, we use our ILP1 to solve the minimum cycle cover problem on a 3×3 grid network. We only compare ILP1 with algorithm $\mathcal{D}$ [3] since it is the most efficient algorithm. As $\mathcal{D}$ can only function on a



Cycles found by $\mathcal{D}$ [3]:

$C_1 = (1, 3, 6, 7, 10, 12, 10, 7, 6, 3, 1)$	$C_2 = (1, 2, 5, 8, 6, 9, 6, 8, 5, 2, 1)$
$C_3 = (1, 4, 10, 11, 13, 5, 13, 11, 10, 4, 1)$	$C_4 = (1, 4, 14, 8, 14, 4, 1)$
$C_5 = (1, 4, 10, 11, 9, 12, 9, 11, 10, 4, 1)$	$C_6 = (1, 2, 3, 2, 1)$
$C_7 = (1, 4, 10, 12, 13, 12, 10, 4, 1)$	Cycle cover length: 58

Cycles found by ILP1

$C_1 = (1, 4, 10, 12, 13, 11, 10, 7, 10, 11, 13, 5, 2, 1)$	$C_2 = (1, 3, 2, 3, 6, 7, 6, 3, 1)$
$C_3 = (1, 2, 5, 8, 5, 13, 12, 9, 11, 9, 12, 10, 4, 1)$	$C_4 = (1, 4, 14, 8, 6, 9, 6, 8, 14, 4, 1)$
Cycle cover length: 44	Total SIDs needed for C_1 - C_4 : 19
Maximum cycle length: 13 links	Running time: 34.8 sec

Cycles found by ILP2 with $r_1=0.025$

$C_1 = (1, 3, 6, 7, 10, 11, 13, 12, 10, 4, 1)$	$C_2 = (1, 2, 3, 2, 5, 13, 11, 10, 7, 6, 3, 1)$
$C_3 = (1, 4, 10, 12, 9, 11, 9, 12, 13, 5, 8, 5, 2, 1)$	$C_4 = (1, 4, 14, 8, 6, 9, 6, 8, 14, 4, 1)$
Cycle cover length: 44	Total SIDs needed for C_1 - C_4 : 18
Running time: 259.8 sec	

Cycles found by ILP3 with $r_2=0.02$

$C_1 = (1, 4, 10, 12, 13, 11, 10, 7, 6, 3, 2, 1)$	$C_2 = (1, 2, 5, 13, 12, 9, 11, 9, 12, 10, 4, 1)$
$C_3 = (1, 4, 14, 8, 6, 9, 6, 8, 5, 2, 3, 1)$	$C_4 = (1, 3, 6, 7, 10, 11, 13, 5, 8, 14, 4, 1)$
Cycle cover length: 44	Maximum cycle length: 11 links
Running time: 368.7 sec	

Fig. 7. The cycle cover found for US network with source at node 1 and $K = 5$. The link costs are based on the monitoring topology model [2] and are not shown for simplicity.

monitoring topology, we adopt the monitoring topology in Fig. 2(a). The results are shown in Fig. 6 with source at node a and $K = 5$. In ILP1, we set $J = 5$ (from Table I). From Fig. 6, we can see that the cycle cover found by ILP1 consists of two cycles with a length of 24 (obtained in 0.61 sec). The cycle cover found by $\mathcal{D}$ also consists of two cycles, but with a length of 26. In this case, a saving of 7.7% in cycle cover length is obtained.

We then apply ILP1 (with $J = 8$) to find a cycle cover for the US network shown in Fig. 7. Note that the link costs are obtained from the monitoring topology model [2] and are not shown for simplicity. With source node at node 1 and $K = 5$, cycle covers found by different algorithms are shown in Fig. 7. We can see that the solution of ILP1 cuts down the cycle cover length by 24% as compared to the solution found by algorithm $\mathcal{D}$.

Tables II-IV further compare the cycle cover length obtained by ILP1 and algorithm $\mathcal{D}$ with varying K and on 3 different topologies: a 4×4 grid network ($|E|=48$), US network ($|E|=42$) and German network ($|E|=52$). In each topology, the source node is set to be the *center* node, i.e., the node whose shortest path to the farthest away node is the smallest (in terms of number of links). Besides, all link costs are set based on the monitoring topology model. From Tables II-IV, we can see that the performance gap between ILP1 and algorithm $\mathcal{D}$ varies in different topologies. In general, when K is small, the performance gain brought by ILP1 is more significant.

TABLE II. CYCLE COVER LENGTH FOR 4×4 GRID NETWORK

K	4	5	6	7	8	9	10
$\mathcal{D}$	54	54	52	52	48	48	48
ILP1	50	48	48	48	48	48	48
Gap	8%	12.5%	8%	8%	0%	0%	0%

TABLE III. CYCLE COVER LENGTH FOR US NETWORK

K	4	5	6	7	8	9	10
$\mathcal{D}$	58	58	46	44	42	42	42
ILP1	46	44	42	42	42	42	42
Gap	26%	32%	10%	4.7%	0%	0%	0%

TABLE IV. CYCLE COVER LENGTH FOR GERMAN NETWORK

K	4	5	6	7	8	9	10
$\mathcal{D}$	58	56	54	54	52	52	52
ILP1	56	54	52	52	52	52	52
Gap	3.5%	3.7%	3.8%	3.8%	0%	0%	0%

C. Extended ILPs

We also apply ILP2 and ILP3 to US network with source at node 1 and $K = 5$, and the results are shown in Fig. 7. With $J = 8$ (from Table I) and $K = 5$, we set $r_1 = 0.025$ in ILP2 to focus more on reducing the cycle cover length. Similarly, we set $r_2 = 0.02$ in ILP3 since $|E| = 42$ for US network. From the solutions shown in Fig. 7, ILP1, ILP2 and ILP3 all provide the same cycle cover length of 44. But the total SIDs required is reduced by one, from 19 SIDs to 18 SIDs, when ILP2 is used. With ILP3, the maximum cycle length is reduced from 13 links to 11 links.

D. Monitoring topology or not

TABLE V. CYCLE COVER LENGTH FOR 4×4 GRID NETWORK

K	4	5	6	7	8	9	10
Mon	50	48	48	48	48	48	48
Unit	68	52	50	48	48	48	48
Random	52	50	48	48	48	48	48

TABLE VI. CYCLE COVER LENGTH FOR US NETWORK

K	4	5	6	7	8	9	10
Mon	46	44	42	42	42	42	42
Unit	58	48	44	44	42	42	42
Random	54	46	44	42	42	42	42

TABLE VII. CYCLE COVER LENGTH FOR GERMAN NETWORK

K	4	5	6	7	8	9	10
Mon	56	54	52	52	52	52	52
Unit	64	56	52	52	52	52	52
Random	60	54	52	52	52	52	52

Existing heuristic algorithms [2, 3] all need to rely on a monitoring topology (see Section I), but maintaining a monitoring topology only for probe packets is inefficient. To evaluate the performance gain brought by the monitoring topology, we consider three topologies (4×4 grid, US and German networks) with three link cost models: monitoring topology model, unit link cost model and random link cost model (by randomly choosing a link cost from 1 to 100).

Tables V-VII compare the cycle cover length found by our ILP1 under three link cost models: ILP1 with monitoring topology (Mon), ILP1 with unit link cost model (Unit) and ILP1 with random link cost model (Random). Note that Unit and Random do not use the monitoring topology. The difference in cycle cover length, as compared with Mon, can be regarded as the performance gain of using monitoring topology. From Tables V-VII, we can see that the gain (in cycle cover length) brought by monitoring topology (Mon) reduces quickly with K . When there is a gain, the amount is limited. As such, we conclude that maintaining a dedicated monitoring topology is not worthwhile.

VII. CONCLUSION

In this paper, we formulated the first ILP (ILP1) to optimally solve the minimum cycle cover problem in segment routing. To further cut down bandwidth consumption due to segment list, we extended ILP1 to jointly minimize the total segment size required by all cycles (ILP2). Since the network fault detection time is determined by the longest cycle in the cycle cover found, we further extended ILP1 to jointly minimize the length of the longest cycle (ILP3). Numerical results showed that our ILPs can significantly reduce the cycle cover length of existing heuristic algorithms especially when K , the maximum segment list size supported, is small. Since all existing algorithms rely on a dedicated monitoring topology, but it is not clear how much gain can be brought by monitoring topology. In this paper, we used ILPs to study the potential gain under different link cost models. We found that the reduction in cycle cover length is not only limited but also disappeared quickly as K increases. Therefore, we argued that maintaining a dedicated monitoring topology for probe packets only is unnecessary.

REFERENCES

- [1] D. Katz and D. Ward, "Bidirectional forwarding detection (bfd)," IETF RFC 5880, June 2010.
- [2] F. Aubry, et al., "SCMon: Leveraging segment routing to improve network monitoring," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1-9.
- [3] X. Li and K. L. Yeung, "Designing network monitoring schemes based on segment routing," in *Proc. IEEE GLOBECOM*, Dec. 2017, pp.1-6.
- [4] B. Bollobás, *Modern graph theory*. New York: Springer, 1998, pp. 215-300.
- [5] C. Filsfils, et al., "The segment routing architecture," in *Proc. IEEE GLOBECOM*, Dec. 2015, pp. 1-6.
- [6] C. Filsfils, et al., "Segment routing architecture," Internet Draft, draft-ietf-spring-segment-routing-14, Dec. 2017.
- [7] R. Bhatia, F. Hao, M. Kodialam and T. V. Lakshman, "Optimized network traffic engineering using segment routing," in *Proc. IEEE INFOCOM*, May 2015, pp. 657-665.
- [8] C. Filsfils, et al., "Segment routing with MPLS data plane," Internet Draft, draft-ietf-spring-segment-routing-mpls-11, Oct. 2017.
- [9] P. Psenak, et al., "OSPF extensions for segment routing," Internet Draft, draft-psenak-ospf-segment-routing-extensions-24, Dec. 2017.
- [10] S. Previdi, et al., "IS-IS Extensions for Segment Routing," Internet Draft, draft-ietf-isis-segment-routing-extensions-15, Dec. 2017.
- [11] A. Giorgetti, et al., "Path encoding in segment routing," in *Proc. IEEE GLOBECOM*, Dec. 2015, pp. 1-6.
- [12] B. Wu, K. L. Yeung and P. H. Ho, "Monitoring cycle design for fast link failure localization in all-optical networks," *IEEE Journal of Lightwave Technology*, vol. 27, no. 10, pp. 1392-1401, May, 2009.
- [13] R. Hulsermann, et al., "A set of typical transport network scenarios for network modelling", *ITG Fachbericht*, no. 182, pp. 65-72, 2004.