FEAL: A source routing Framework for Efficient Anomaly Localization

Mohamed Rahali, Jean-Michel Sanner, Gerardo Rubino

in ICC'20

FEAL: A source routing Framework for Efficient Anomaly Localization

Mohamed Rahali
IRT B<>Com, Rennes, France
Mohamed.Rahali@b-com.com

Jean-Michel Sanner
IRT B<>Com, Rennes, France
Jean-Michel.Sanner@b-com.com

Gerardo Rubino
INRIA Rennes – Bretagne Atlantique, France
Gerardo.Rubino@inria.fr

***Abstract*—Source routing represents a good opportunity to enhance monitoring solutions, particularly probing techniques. This technique allows deploying customized probing schemes to fulfill different monitoring needs like troubleshooting or Service Level Agreement (SLA) supervision. In this context, the use of probing cycles is a promising monitoring method. The deployment of such probing schemes becomes easier thanks to source routing since it allows constraining the traffic to follow specific paths.**

In this paper we propose the FEAL monitoring framework (Framework for Efficient Anomaly Localization) based on source routing probing cycles. The framework is mainly composed of two parts: the "Probing Cycles" and the "Anomaly Detection" modules. The first one defines the probing strategy by deploying the needed monitors and finding the probing cycles to cover the network topology. The "Anomaly Detection" module is based on our previously proposed statistical algorithm for the inference of link metrics named ESA (Evolutionary Sampling Algorithm) [1], here extended to more general classes of metrics. We prototype and evaluate the FEAL framework with a `P4` implementation of source routing over a `Mininet` emulator. The results show that our framework detects and localizes efficiently the failure points in the network.

I. Introduction

Internet networks are continuously evolving to satisfy the increasing clients' demands on connectivity offers. The emergent new services usually have high SLA requirements that must be respected (e.g. high-quality video streaming, augmented and virtual reality, connected cars). Operators and service providers should respect these requirements and must have suitable and efficient tools to detect SLA violations and make the needed remediation actions. Thus, the network monitoring tools should follow the network and services evolution to fulfill these new demands.

Classic monitoring procedures, based for instance on `SNMP` or `traceroute`, rely on the direct metrics measurement on network devices. These methods introduce an important overhead and do not help in the design of customized monitoring solutions. Probing techniques, where monitoring traffic is exchanged between specific nodes, i.e. monitors, are not well adapted to large backbone networks. For example, to check the operation of a specific router or link, it is very difficult to constrain the monitoring traffic to follow a specific path or to pass through the required network nodes due to mechanisms like multi-path routing, aggregated links and backup links [2].

Recently, more attention has been accorded to apply source routing [3] in network monitoring [4] [5]. The main idea is that the traffic source encodes the forwarding path, or some parts of the path, in the packet header. Segment Routing (SR) [6] is an efficient and flexible method of applying source routing in MPLS networks [7]. In SR, the path is encoded in the packet header as a list of segment ID (SID). Then, the packet is forwarded according to the instructions corresponding to the stacked SIDs.

Source routing has been well explored from a traffic engineering point of view [8]. It provides fast and efficient tools to deploy traffic forwarding strategies along arbitrary paths without any additional protocol or signaling procedure. It enables to constrain the traffic to pass throw particular places and thus, to build specific probing schemes. Some works focus on a specific approach of source routing for probing where only cycles are used [4]. The probing cycle schemes have a significant advantage compared to regular paths. When the same node sends and receives the monitoring packets (the probes), it is not necessary to synchronize with the other nodes. This reduces synchronization issues between source and destination nodes. But in spite of the flexibility offered by source routing in monitoring, the technology presents multiple challenging issues. For example, most of the commercial switches support only a limited number of SIDs in the packet headers. This adds a constraint on the length of the probed cycles.

In this paper, we propose a monitoring framework called *FEAL* (Framework for Efficient Anomaly Localization). Our solution is composed of two main components, the "Cycle Probing" and the "Anomaly Detection" modules. For the first one, we consider minimizing the number of monitors as the main objective for the monitors' placement strategy. Meanwhile, we compare two conditions to be satisfied: covering the network topology, as in most previous works, and adding more conditions on the probed cycles to enhance the anomaly localization, following an algebraic modeling of the problem. The "Anomaly Detection" module is based essentially on a statistical approach for additive metrics inference that we proposed in [1], here adapted to a more general case.

The remainder of the paper is organized as follows. Section II overviews the work on network monitoring based on end-to-end measurement, especially the use of probing cycles. Sections III and IV present respectively the problem formulation and the framework components. Section V gives a description of the implemented proof of concept and the

performance results. Finally, Section VI concludes the work and outlines some perspectives for future developments.

II. Background and related works

In [4], the authors propose a Segment Routing monitoring solution. A single monitor is deployed in a graph centroid and the probes are forwarded through the probing cycles starting and ending at that point. The paper proposes a set of algorithms to minimize the probing cycles covering the topology and to encode them efficiently. Using only one monitor reduces the monitoring operation cost and discards the synchronization problems. However, this requires very long cycles to cover the network topology which can introduce additional bias in the measurements. Authors in [9] propose a method similar as in [4] since they use only one monitor for the probing. They focus on minimizing the cycle cover length, which is the total length of all the probed cycles, to reduce the consumed bandwidth. An Integer Linear Programming (ILP) formulation of the same problem is proposed in [5]. Other works studied the monitors' placement problem while enabling the deployment of regular paths between different nodes. Similarly, the same optimization methods are still applicable. A well detailed classification of these different approaches has been presented in [10] and [11].

In [12], the authors give necessary and sufficient conditions to identify an additive metric on all the network links for a given topology. The proposed solution identifies the minimum number of monitors to exchange probes with regular paths or cycles. However, no constraint on the length of the paths is considered. In [13] the authors propose a solution to find the optimal monitors placement for the inference of additive metrics from end-to-end measurements made on regular paths between different nodes. They give an algebraic formulation of the problem. The proposed algorithms are based on the graph decomposition into strongly connected components and find an efficient placement to guarantee the identifiability of all the links. [14] studied the same problem. Meanwhile, only a subset of links is privileged to guarantee their identifiability. This leads to the objective of finding the minimal number of monitors and their right placement in order to identify an additive metric on all the considered links. In [15], the authors focus on the dual problem where the number of monitors is fixed first, and the objective is to find their effective placement to identify the highest number of links.

III. General context and problem description

A. General context

A network monitoring operation usually has two main steps. First, the monitoring data is collected from the network via specific tools and protocols. Second, the collected information is examined and interpreted to detect and localize the existence of malfunctioning, in order to take the needed remediation actions. Thus, the *FEAL* framework is composed of two main components. The first one, the "Cycle Probing", is responsible for deploying the probing strategy and collecting the monitoring data. The second one, the "Anomaly Detection" module, is mainly based on a statistical algorithm that enables the localization of the most likely failure points from the collected end-to-end monitoring measures. As shown in Fig. 1, the *FEAL* framework can be deployed, for instance, as an SDN application over a network controller since it needs a global view of the network to deploy the probing strategy and adapt it according to the network updates.

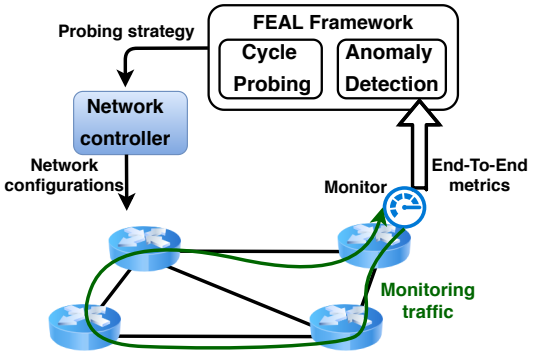

Fig. 1: Framework components

B. Network model and notation

The network topology is modeled by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{L})$, with set of vertices $\mathcal{V} = \{1, 2, \ldots, V\}$ and set of links $\mathcal{L} = \{1, 2, \ldots, L\}$. Let us denote by $\mathcal{C} = \{1, 2, \ldots, C\}$ the set of given probed cycles. A is a $C \times L$ Boolean matrix whose rows code the probed cycles vectors: $A(c, \ell)$ is equal to 1 iff cycle c includes link ℓ. The monitors send and receive the probes through these cycles and measure the end-to-end performances. Y denotes the metrics vector of the selected probed cycles. Hence, its size is C and the cth element $Y(c)$ represents the metric on cycle c. X denotes the vector of the unknown link metrics. Thus, its size is L and its ℓth element $X(\ell)$ is the metric value corresponding to link ℓ.

C. Introducing two placement strategies

Consider the case of additive metrics, such as the delay, or the probabilities of not loosing a packet. Using the notation described above, the process of producing the end-to-end measurement linear operation

$$Y = AX. \tag{1}$$

Assume that the objective of collecting end-to-end performances is to supervise the general network state and localize the failure points. Problems are simply defined by the fact that the considered metric value on a path/link (think, for instance, of delays or losses) are higher than some beforehand specified threshold. See that the metric whose value is 1 on path/link z iff there is an anomaly on z, is not anymore additive.

Covering all the network's links by the probing cycles allows the detection of any anomaly. From this observation, the first strategy for the monitors' placement, called **LinkCovering**, has simply the goal of covering the maximum number of links without any additional constraint. However,

if a breakdown is detected, it is difficult to localize exactly the failed link. This requires computing the link metrics vector X. Now, to solve (1) for X, matrix A should be square and non singular. This adds new constraints to the probing cycles: we need L different cycles for which their vector representations are linearly independent. This constraint can be hard or impossible to satisfy in realistic conditions. A possible strategy is then to search the maximum of linearly independent equations to compute a good approximation of X. This leads to our second strategy for the monitors' placement, called **MaxRank**, whose goal is to choose the cycles such as the rank of A is maximized.

IV. *FEAL* FRAMEWORK

Before searching the best monitors placement, the "Cycle Probing" module starts by exploring the possible cycles that can be generated from each feasible node v. Thus, we propose a resolution algorithm that explores the graph network to find all the possible cycles formed by less than k links, for a given k. In this way, the monitors' placement task can be seen as a set covering problem. For the LinkCovering strategy, the objective is to cover the set of links $\mathcal{L}$. For the MaxRank strategy, the objective is to cover the network topology with cycles that give the maximum possible rank of matrix A to enhance the anomaly localization.

A. Additional notation

Let $\mathcal{C}_k$ denote the set of probing cycles formed by less than k links. The subset that includes vertex v is denoted by $\mathcal{C}_k^v$. Let us denote by $\mathcal{B}_k^v$ a basis of linearly independent vectors in $\mathcal{C}_k^v$, with dimension $\dim(\mathcal{B}_k^v)$. For each vertex v, the set of links covered by $\mathcal{C}_k^v$ is denoted by R_k^v. The selected nodes of $\mathcal{G}$ for the monitors' placement are denoted by $\mathcal{M}$.

B. Probing cycles encoding

Each probe stacks the forwarding path in the packet header as a list of SIDs. Most network equipment have a limitation on the number of SIDs which limits the probing cycle length.

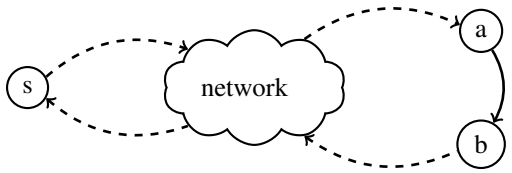

Fig. 2: To reach a link between nodes a and b starting from s, it is possible to forward the probe to node a with a node SID, then to b with an adjacency SID and finally to s with a node SID.

There are two types of segments: adjacency segments, when the router forward the probe to an intermediate link before achieving the final destination, or node segments, when the router sends the probe to an intermediate node.

It is possible to reach any point in the network with long paths encoded with node SIDs as proposed in [4]. However, this can affect the measurement accuracy by introducing multiple sources of bias. If we consider the example illustrated in Fig. 2, to forward the probes from s to a or from b to s, the routers use the shortest paths as the regular traffic. However, it is difficult to know exactly the taken paths due to the dynamic mechanisms deployed for traffic load balancing. In our proposal, the cycles are encoded with only adjacency SIDs. The complete path is described in the packet header, so all the crossed links and nodes are well known beforehand. It is possible to make a trade-off by combining node and adjacency SIDs to extend the coverage of each monitor. However, the use of node SIDs should be well controlled to avoid uncertainty and bias in the collected end-to-end measurements.

In Algorithm 1 we describe a simple recursive method to generate the possible probing cycles starting and ending at a given vertex v.

Algorithm 1 Cycles construction

INPUT: Topology $\mathcal{G}$, length k, vertex v.
OUTPUT: $\mathcal{C}_k^v$.

```
1: startNode ← v
2: C_k^v ← ∅
3: function FINDCYCLES(vertex, visited)
4:     if |visited| > k then
5:         Return
6:     if vertex = startNode then
7:         C_k^vertex.add(visited)
8:         Return
9:     for v ∈ vertex.adjacentNodes do
10:        if v ∉ visted then
11:            FINDCYCLES(v, visited ∪ {v})
```

C. LinkCovering algorithms

1) Optimal algorithm: We consider the monitors' placement task as a set covering problem. Thus, the objective is to minimize the number of selected monitors in $\mathcal{M}$ to cover $\mathcal{L}$ using the branch and bound technique. Finding the optimal solution for the set covering problem is NP-complete [16]. The branch and bound technique is an efficient method to deal with this complexity level.

Before exploring a branch of a solution space, the algorithm approximates the largest set of links that can be covered. The first step of the algorithm is the computation of the associated bounds of the methodology. If we consider the ordered set of vertices $\mathcal{V} = \{1, 2, ..., V\}$, $boundLinks^v$ returns the largest set of links that can be covered by any subset of vertices included in $\{v, v+1, ..., V\}$.

The MLC algorithm (Maximum Links Covering) is implemented with a recursive function as described in Algorithm 2. At each node v, the algorithm tests if all the non covered links are included in $boundLinks^v$. Then, there are two possible recursive calls: either node v is selected or not. When the algorithm reaches the final step and all the links are covered, the set of selected nodes corresponds to a possible solution.

Algorithm 2 Maximum Links Covering algorithm (MLC)

INPUT: Topology $\mathcal{G}$, $\mathcal{R}_k$.
OUTPUT: Monitors placement $\mathcal{M}$.

$boundLinks^{V+1} = \emptyset$
for $v \in \mathcal{V}$ **do**
  $boundLinks^{V-v+1} \leftarrow boundLinks^{V-v+2} \cup \mathcal{R}_k^{V-v+1}$
$\mathcal{M} \leftarrow \emptyset$
$maxLinks \leftarrow |boundLinks^1|$
function MLC($coveredLinks, usedNodes, vertex$)
  if $vertex = V$ **then**
    if $|coveredLinks| = maxLinks$ **then**:
      if $|usedNodes| < |\mathcal{M}|$ **then**:
        $\mathcal{M} \leftarrow usedNodes$ ▷ Save Solution
    Return
  $newBound \leftarrow coveredLinks \cup boundLinks^{vertex}$
  if $|newBound| = maxLinks$ **then**
    MLC($coveredLinks \cup \mathcal{R}_k^{vertex}, usedNodes \cup \{vertex\}, vertex + 1$)
    MLC($coveredLinks, usedNodes, vertex + 1$)

Then, it is compared to the last saved solution to choose the best between them.

2) Greedy algorithm: The branch and bound method can reach a high accuracy, but its computational complexity is still very high, becoming useless in case of large topologies. Thus, we propose an alternative greedy approach described by Algorithm 3. The GMLC algorithm (Greedy Maximum Link Covering) makes multiple iterations. At each one, the network nodes are browsed to select the probing node that covers the maximum of new links. This process is repeated until the maximum possible number of covered links is reached.

3) Complexity analysis: The worst case complexity of the optimal algorithm is $O(2^V L)$, when all the solution space branches are explored. There are 2^V possible combinations of monitors' placements. The complexity verification of the optimality of each one is $O(L)$. The worst case complexity of the greedy algorithm is $O(V^2 L)$. At each iteration, all the nodes are browsed to find the one that covers the maximum number of links which can be done in $O(VL)$ steps. These iterations are repeated until covering all the links.

Algorithm 3 Greedy Maximum Link Covering (GMLC)

INPUT: Topology $\mathcal{G}$, $\mathcal{R}_k$.
OUTPUT: Monitors placement $\mathcal{M}$.

$maxLinks \leftarrow |BoundLinks^1|$
$\mathcal{M} \leftarrow \emptyset$
$links \leftarrow \emptyset$
while $|links| < maxLinks$ **do**
  $s \leftarrow s \in \mathcal{V} | \forall v \in \mathcal{V}, |\mathcal{R}_k^v \setminus links| \leq |\mathcal{R}_k^s \setminus links|$ & $s \notin \mathcal{M}$
  $links \leftarrow links \cup \mathcal{R}_k^s$
  $\mathcal{M}.add(s)$

D. MaxRank algorithms

In this section, we focus on the second strategy for monitors placement whose goal is to maximize the rank of matrix A.

If we consider the global set of cycles $\mathcal{C}_k$, their associated vectors can be represented in a basis denoted by $\mathcal{B}_k$ with dimension D. Therefore, the objective is to find D probing cycles, whose representative vectors are linearly independent to form the basis $\mathcal{B}_k$. Using the notation described before, the objective is to cover the basis $\mathcal{B}_k$ with the minimal set of basis $\mathcal{B}_k^v$ which maximizes the rank of the obtained matrix A.

1) Optimal algorithm: The Maximum Rank algorithm (MR) applies the same approach as the optimal algorithm for LinkCovering. Meanwhile, the condition becomes maximizing the rank of the matrix A.

Algorithm 4 Maximum Rank algorithm (MR)

INPUT: Topology $\mathcal{G}$, Probing cycles $\mathcal{C}_k$.
OUTPUT: Monitors placement $\mathcal{M}$.

$boundDim^{V+1} = \emptyset$
for $v \in \mathcal{V}$ **do**
  $boundDim^{V-v+1} \leftarrow bounDim^{V-v+2} \cup \mathcal{B}_k^{V-v+1}$
$\mathcal{M} \leftarrow \emptyset$
$D \leftarrow dim(boundDim^1)$
function MR($basis, usedNodes, vertex$)
  if $vertex = V$ **then**
    if $dim(basis) = D$ **then**:
      if $|usedNodes| < |\mathcal{M}|$ **then**:
        $\mathcal{M} \leftarrow usedNodes$ ▷ Save Solution
    Return
  $newBound \leftarrow dim(basis \cup boundDim^{vertex})$
  if $dim(newBound) = D$ **then**
    MR($basis \cup \mathcal{B}_k^{vertex}, usedNodes \cup \{vertex\}, vertex + 1$)
    MR($basis, usedNodes, vertex + 1$)

The algorithm starts by computing the bounds used for the branch and bound technique. The objective is to cover the basis of linearly independent vectors $\mathcal{B}_k$. The bound for a subset of monitors is a basis of vectors representing the probing cycles of these nodes. $boundDim^v$ represents the basis of maximum dimension that can be formed with any subset of monitors included in $\{v, v+1, ..., V\}$. The remainder of the algorithm is very similar to the LinkCovering strategy as described in Algorithm 4.

2) Greedy algorithm: The greedy approach of the MaxRank procedure uses the same concept as the LinkCovering strategy. The GMR algorithm (Greedy Maximum Rank) browses the graph nodes and chooses the best placement to maximize the rank of A. It makes similar iterations until covering the global basis $\mathcal{B}_k$. This process is described in Algorithm 5.

3) Complexity analysis: The worst case complexity of the optimal algorithm can be written $O(2^V L^3)$. In fact, there are 2^V of monitors' placement combinations. Then, the verifica-

Algorithm 5 Greedy Maximum Rank (GMR)

INPUT: Topology $\mathcal{G}$, Probing cycles $\mathcal{C}_k$.
OUTPUT: Monitors placement $\mathcal{M}$.

$D \leftarrow dim(boundDim^1)$
$\mathcal{M} \leftarrow \emptyset$
$basis \leftarrow \emptyset$
while $dim(basis) < D$ **do**
  $s \leftarrow s \in \mathcal{V} | \forall v \in \mathcal{V}, dim(basis \cup \mathcal{B}_k^v) \leq dim(basis \cup \mathcal{B}_k^s)$ & $s \notin \mathcal{M}$
  $basis \leftarrow basis \cup \mathcal{B}_k^s$
  $\mathcal{M}.add(s)$

tion of the optimality of each one requires to compute the rank of a $L \times L$ matrix (generally with smaller dimensions) with a complexity $O(L^3)$.

The worst case complexity of the greedy algorithm is $O(V^2L^3)$. At each iteration, all the nodes are browsed to find the best one that maximizes the rank of A with a complexity $O(VL^3)$. These iterations are repeated until maximizing the rank of A.

E. Anomaly Detection module

The "Anomaly Detection" module aggregates the end-to-end performances from the monitors to make a deeper analysis of the collected data. The FEAL framework integrates our previously proposed algorithm for the inference of additive metrics called ESA (Evolutionary Sampling Algorithm) [1]. The ESA algorithm computes the probability distribution for each link metric. Then, an alarm can be activated if the failure probability on a link exceeds a fixed threshold, let's say if $\Pr(X_\ell > V_{\text{fail}}) > p^*$. More details are available in [1].

V. Evaluation

A. Proof of Concept and testing environment

To evaluate the performance of our solution, we implement a test-bed over the `Mininet` emulator. We experiment with topologies taken from [17] and [18]. We use a `P4` [19] software switch named "behavioral model" (bmv2) [20] with a source routing implementation from [21] to build the topologies. The monitors are implemented as `Python` scripts that are running over hosts connected to the switches.

Each host constructs the probes using the `Scapy Python` library [22]. The forwarding path is stacked is the packet header. Before sending each probe, the host saves the sending timestamp with the associated packet identifier. When the probe comes back to its starting node, the host saves the returning timestamp to compute the end-to-end delay.

B. Results

1) Monitors placement evaluation: Firstly, we compare the performances of the optimal and the greedy approaches of the monitors' placement with different topologies with characteristics described in Table I.

TABLE I: Network topologies

Toplogy	**A-GridNet**	**B-Polska**	**C-NewYork**	**D-India**	**E-Pioro**
Vertices	9	12	16	35	40
Edges	20	20	51	80	89

For each studied topology, we compute the minimal number of monitors to cover it, or to maximize the path matrix rank A. We vary the maximal number of labels k for path encoding.

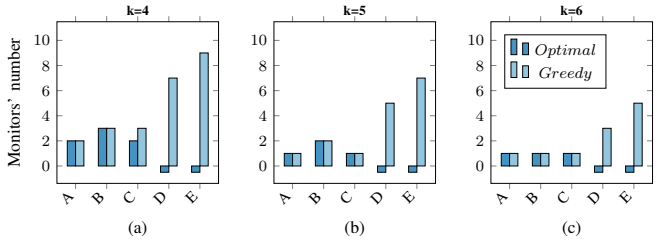

Fig. 3: k vs number of monitors for LinkCovering strategy.

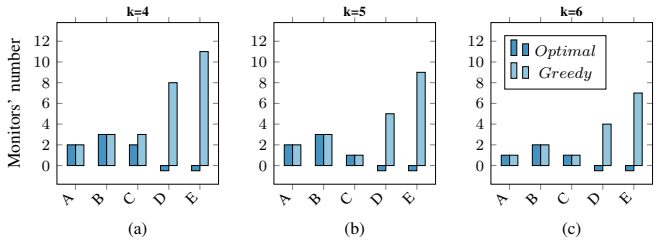

Fig. 4: k vs number of monitors for MaxRank strategy.

Fig.3 and Fig.4 illustrate the number of needed monitors for different value of k respectively for LinkCovering and MaxRank strategies.

For small and medium topologies like A, B and C, the optimal and the greedy solution usually give the same results except for a few infrequent cases like topology C with k equal to 4. With larger topologies like D and E, the optimal algorithm cannot give results in reasonable computing time.

Comparing the two strategies for the monitors' placement, maximizing the rank of A usually requires more monitors than only covering the topologies, an obvious result.

Fig. 5 and Fig. 6 illustrate the computing time for the different experiences. The computing time increases exponentially according to the number of links only with the optimal algorithm as shown with topology C compared to A and B. Remark: the computing time of the second algorithm is too small compared to that of the first; this makes often hard to represent both in the same graphics.

2) Anomaly detection evaluation: We choose an anomaly detection use case to test the efficiency of the FEAL framework with its two probing strategies. For each experience, we choose randomly a link to generate an anomaly and the framework tries to localize it. Since we are focusing on additive metrics, the chosen scenario is to increase the delay on the selected link. We use the testbed described in Section V-A to make multiple tests on the two topologies A

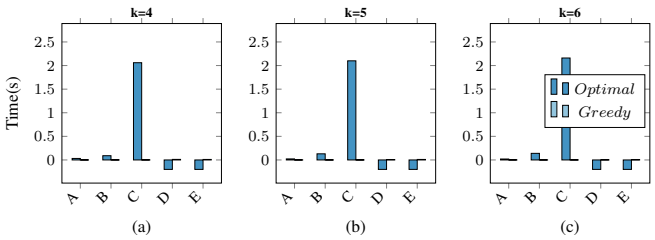

Fig. 5: k vs computing time for LinkCovering strategy.

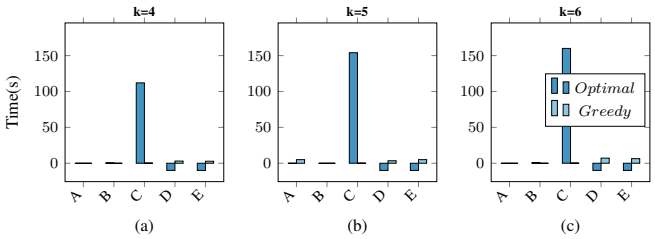

Fig. 6: k vs computing time for MaxRank strategy.

and B. Note that we cannot test with larger topologies since the used network emulator needs much more resources (CPU and memory) to emulate the needed scenarios in good conditions. However, the proposed algorithms are well adapted for larger ones.

We make 30 different experiences for each topology. In each one, the "Anomaly Detection" module computes the probability of failure on each link. Then, the alarm can be activated if this probability exceeds a fixed threshold denoted p^*. As a result, this task can be considered as a binary one, and the results for each fixed threshold can be summarized by four metrics that are described in Table II.

TABLE II: Binary classification parameters

Variable	Description
True Positives (TP)	Successfully detected link failure
False Positives (FP)	False warning
True Negatives (TN)	Good prediction of normal links
False Negatives (FN)	Not detected link failure

There is an important feature in our case that must be taken into consideration. The two classes that we are considering, normal and failed links, are heavily imbalanced as shown in Fig. 7. The precision and recall metrics are well adapted to evaluate a binary classification model with imbalanced data [23]. These metrics focus more on the positive class and is not affected by the large number of TN as described in (2).

$$\text{Precision} = \frac{TP}{TP+FP}, \text{Recall} = \frac{TP}{TP+FN}. \quad (2)$$

Fig.7 shows the probabilities dispersion of 30 tests made with topology A. The prediction-recall curve is a model to evaluate the binary classification model. It is obtained by computing multiple pairs of precision and recall for different thresholds p^* (represented by the dotted red line).

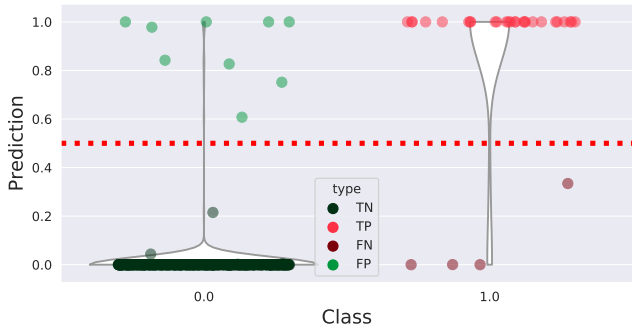

Fig. 7: Class distribution probabilities for 30 tests made with topology A. The MaxRank probing strategy is used with k equal to 4. The threshold p^* is 0.5.

These values are used to plot the precision-recall curve presented in Fig.8 for topology A. We use the AUC score (Area Under the precision-recall Curve) as a global indicator to evaluate the performances: the higher the area, the better the model accuracy. We make similar tests with the two strategies of probing. The MaxRank probing strategy outperforms the LinkCovering since its curve is always on the top. Besides, the AUC for MaxRank is 0.8, while it is 0.67 for LinkCovering.

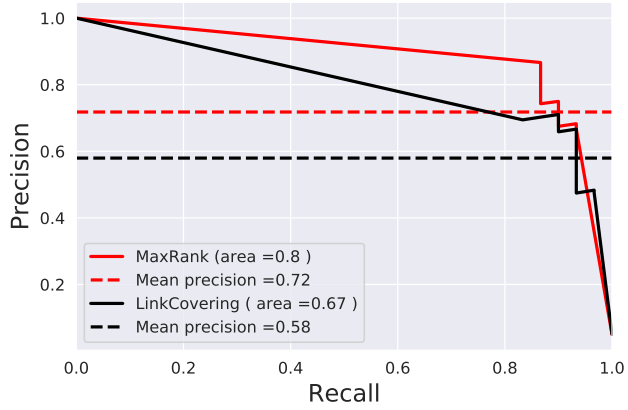

Fig. 8: Precision-Recall curve for 30 tests made with topology A. The two probing strategies are tested with k equal to 4.

Fig. 9 represents the same tests made with topology B. The MaxRank gives always better results as in the first example. For this topology, the MaxRank probing requires 3 hosts to achieve the maximum rank, while the LinkCovering requires only one host. However, with topology A, both of the two probing strategies requires two monitors, but with different placements. This case shows that even with identical resources (number of monitors), the right placement can make a significant difference.

VI. Conclusion

In this paper, we study the use of source routing, particularly probing cycles, in network monitoring, from different points

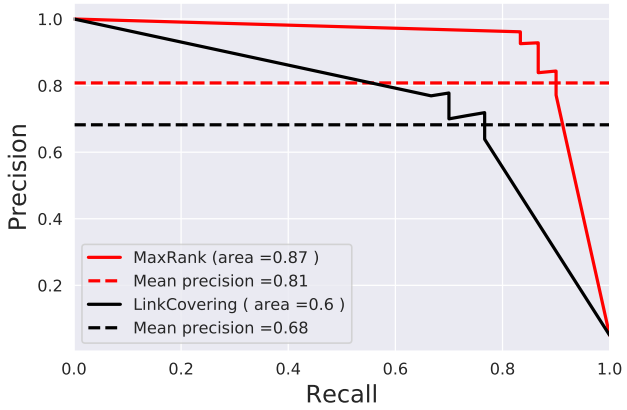

Fig. 9: Precision-Recall curve for 30 tests made with topology B. The two probing strategies are tested with k equal to 6.

of view. We explore two different approaches for monitors placement, called LinkCovering and MaxRank, both modeled as a set covering problem, and we propose two alternatives each of them: an optimal branch and bound solution and a much faster greedy algorithm. The entire solution is implemented and tested over the `Mininet` emulator to evaluate its performances.

The paper compares the two approaches, because most of existing solutions consist of following the same ideas as in LinkCovering. The result is that MaxRank outperforms LinkCovering and enhances the anomaly localization performances. The proof of concept with `P4` switches and a source routing implementation shows that the proposed ideas can be applied in realistic conditions. One constraint that can limit their applicability is the maximum of labels in the packet header. In this work, we consider a simple strategy for path encoding. An enhanced strategy of path encoding is possible to extend the coverage of each monitor which can reduce the cost of the probing operation and the efficiency of the anomaly detection.

The applicability to real networks of this approach relies essentially on the flexibility and the programmability offered by SDN networks. Problem sizes don't appear as a serious limitation.

References

[1] M. Rahali, J. Sanner, and G. Rubino, "Unicast inference of additive metrics in general network topologies," in *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Oct 2019, pp. 107–115.

[2] C. Pelsser, L. Cittadini, S. Vissicchio, and R. Bush, "From paris to tokyo: on the suitability of ping to measure latency," in *Proceedings of the 2013 Internet Measurement Conference, IMC 2013, Barcelona, Spain, October 23-25, 2013*, 2013, pp. 427–432.

[3] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. François, "The segment routing architecture," in *2015 IEEE Global Communications Conference, GLOBECOM 2015, San Diego, CA, USA, December 6-10, 2015*, 2015, pp. 1–6.

[4] F. Aubry, D. Lebrun, S. Vissicchio, M. T. Khong, Y. Deville, and O. Bonaventure, "Scmon: Leveraging segment routing to improve network monitoring," in *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*, 2016, pp. 1–9.

[5] X. Li and K. L. Yeung, "ILP formulation for monitoring-cycle construction using segment routing," in *43rd IEEE Conference on Local Computer Networks, LCN 2018, Chicago, IL, USA, October 1-4, 2018*, 2018, pp. 485–492.

[6] Z. N. Abdullah, I. Ahmad, and I. Hussain, "Segment routing in software defined networks: A survey," *IEEE Communications Surveys and Tutorials*, vol. 21, no. 1, pp. 464–486, 2019.

[7] A. Bashandy, C. Filsfils, S. Previdi, B. Decraene, S. Litkowski, and R. Shakir, "Segment Routing with MPLS data plane," Internet Engineering Task Force, Internet-Draft draft-ietf-spring-segment-routing-mpls-22, May 2019, work in Progress.

[8] E. Moreno, A. Beghelli, and F. Cugini, "Traffic engineering in segment routing networks," *Computer Networks*, vol. 114, pp. 23–31, 2017.

[9] X. Li and K. L. Yeung, "Designing network monitoring schemes based on segment routing," in *2017 IEEE Global Communications Conference, GLOBECOM 2017, Singapore, December 4-8, 2017*, 2017, pp. 1–6.

[10] A. Dusia and A. S. Sethi, "Recent advances in fault localization in computer networks," *IEEE Communications Surveys Tutorials*, vol. 18, no. 4, pp. 3030–3051, Fourthquarter 2016.

[11] M. Steinder and A. S. Sethi, "A survey of fault localization techniques in computer networks," *Science of Computer Programming*, vol. 53, pp. 165–194, 11 2004.

[12] A. Gopalan and S. Ramasubramanian, "On identifying additive link metrics using linearly independent cycles and paths," *IEEE/ACM Transactions on Networking - TON*, vol. 20, pp. 906–916, 06 2012.

[13] T. He, A. Gkelias, L. Ma, K. K. Leung, A. Swami, and D. Towsley, "Robust and efficient monitor placement for network tomography in dynamic networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, pp. 1732–1745, June 2017.

[14] W. Dong, Y. Gao, W. Wu, J. Bu, C. Chen, and X. Li, "Optimal monitor assignment for preferential link tomography in communication networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 1, pp. 210–223, Feb 2017.

[15] L. Ma, T. He, K. K. Leung, A. Swami, and D. Towsley, "Link identifiability in communication networks with two monitors," in *2013 IEEE Global Communications Conference (GLOBECOM)*, Dec 2013, pp. 1513–1518.

[16] A. Caprara, P. Toth, and M. Fischetti, "Algorithms for the set covering problem," *Annals of Operations Research*, vol. 98, no. 1, pp. 353–371, Dec 2000.

[17] S. Knight, H. X. Nguyen, N. Falkner, R. A. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[18] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, "Sndlib 1.0—survivable network design library," *Netw.*, vol. 55, no. 3, pp. 276–286, May 2010.

[19] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014. [Online]. Available: http://doi.acm.org/10.1145/2656877.2656890

[20] "Behavioral-model," https://github.com/p4lang/behavioral-model, 2019.

[21] "Source routing implementation," https://github.com/p4lang/tutorials/tree/master/exercises/source_routing, 2019.

[22] "Scapy python library," https://github.com/secdev/scapy/, 2019.

[23] T. Saito and M. Rehmsmeier, "The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets," in *PloS one*, 2015.