Multipoint passive monitoring in packet networks

Multipoint Passive Monitoring in Packet Networks

Mauro Cociglio, Giuseppe Fioccola, Guido Marchetto, Amedeo Sapio, and Riccardo Sisto

***Abstract*—Traffic monitoring is essential to manage large networks and validate Service Level Agreements. Passive monitoring is particularly valuable to promptly identify transient fault episodes and react in a timely manner. This paper proposes a novel, non-invasive and flexible method to passively monitor large backbone networks. By using only packet counters, commonly available on existing hardware, we can accurately measure packet losses, in different segments of the network, affecting only specific flows. We can monitor not only end-to-end flows, but any generic flow with packets following several different paths in the network (multipoint flows). We also sketch a possible extension of the method to measure average one-way delay for multipoint flows, provided that the measurement points are synchronized. Through various experiments we show that the method is effective and enables easy zooming in on the cause of packet losses. Moreover, the method can scale to very large networks with a very low overhead on the data plane and the management plane.**

***Index Terms*—Passive network monitoring, alternate marking, multipoint flows.**

I. Introduction

NETWORK measurements are crucial to manage today's large networks. Providers of backbone networks rely on accurate measurements to validate their performance targets from *Service Level Agreements* (SLAs), since failing to meet SLA guarantees results in significant revenue losses. **Packet loss ratio** and average **one-way delay** (OWD) are two important *Key Performance Indicators* (KPIs) used for the definition of performance targets. In fact, a large number of losses heavily affects a wide range of Internet applications and large end-to-end delays have a substantial impact on the performance of TCP- and UDP-based communication.

While lossless networks are available in datacenters [1], in backbone networks packet losses cannot be completely avoided due to the intrinsic dynamics of packet switched networks, which heavily rely on buffers to deal with bursty traffic and congestion. Similarly, network congestion heavily affects the time packets spend in buffers and, as a consequence, the perceived OWD.

Typically, traffic measurements methods are classified in active and passive approaches. With active measurements additional probe traffic is injected into the network (e.g., the ubiquitous `PING` utility) and the characteristics of this traffic are measured, while passive methods are non-intrusive, as no additional traffic is introduced and the quality of actual customer traffic is directly evaluated. With passive methods, live traffic features can also be monitored with different granularities, from single packets to traffic summaries and statistics. Moreover, to reduce the size of these summaries, traffic data are commonly aggregated in network flows [2], [3] (i.e., sets of packets defined by common values in certain header fields). Flow monitoring allows to identify problems affecting only packets with certain characteristics (e.g. involving individual applications). Packets of a single flow can follow multiple, dynamic, paths, therefore flow monitoring requires a relatively large per-flow state (e.g., packet digests) to track the trajectory of the packet to accurately monitor the flow [4]. As a result, it is challenging to monitor a large number of flows in large networks without having a huge reporting traffic.

In this paper we present a new approach for accurate passive network monitoring of live traffic in backbone networks. Our method has been designed around three objectives. First comes flexibility, as we can tune the monitoring resolution in terms of time interval, monitored packets and network devices. Second is easiness of implementation on existing networking hardware currently deployed in large networks. Third is the ability to monitor not only end-to-end flows (i.e., packets with the same 5-tuple), but any generic flow, which can include packets from different sources and to different destinations, following several different paths in the network (**multipoint flow**). The ability to monitor multipoint flows in backbone networks is of great interest for service providers, which want to evaluate SLA compliance for single customers. A single multipoint flow can identify all the traffic towards an Online Service Provider data center, the traffic coming from one eNodeB in an LTE network or all the traffic of one VPN with multiple terminators throughout the network. This would instead require to monitor a large number of point-to-point flows, generating significant reporting traffic and post-processing overhead.

Our method is inspired by the *flow conservation rule* in a portion of the network (segment): in a lossless network the amount of packets entering any network segment is equal to the amount of packets leaving the same network segment, as long as the network segment does not contain packet sources and sinks (e.g., hosts). In case of packet losses, the difference between the number of packets entering the segment and leaving the segment is equal to the amount of lost packets. This rule stands for unicast flows, while it is not valid for multicast flows (packets can be duplicated by network devices). Multicast flows can be measured using existing methods [5], therefore are not subject of the present paper.

Leveraging interface counters, available ubiquitously from routers through SNMP measurement points (MPs), we can measure packet losses in any network segment by applying the flow conservation rule. If the MPs are accurately synchronized (e.g., via low cost GPS receivers [6]) we can also apply the same principle to measure the average OWD. This method is well-suited to monitor (*i*) a network segment of any size,

M. Cociglio and G. Fioccola are with TIM, Torino, Italy.
G. Marchetto, A. Sapio, and R. Sisto are with Politecnico di Torino, Italy.

Manuscript received ...; revised ...

from a single link to the entire network, and (*ii*) a generic multipoint flow, entering/leaving the segment from/to several different points.

However, to compare counters correctly across different devices, a form of synchronization is required. In fact, in one measurement period, all the devices must count the same set of packets, upon entering and leaving the segment. A simple and effective solution to this problem is given by the Alternate Marking method [5], [7], which splits the flow into consecutive blocks by marking the packets of the same block in the same way. The duration of a block is a parameter that can be adjusted to the required temporal resolution of the measurement. This work leverages the same synchronization technique to monitor multipoint flows, while the original method can only be applied to a single point-to-point flow.

Considering that (*i*) one counter is dedicated to a single monitored flow, (*ii*) there are potentially a large number of monitored flows and (*iii*) the number of counters that one device can deploy is limited, we expect that not all the flows can be measured by all the devices. Therefore, our method considers that only a subset of devices/interfaces in the network are actively counting packets of one specific flow. As a result, we propose a new approach to model the monitored network, i.e., the network containing only the monitored interfaces. If counters are not deployed on all the interfaces, it is not possible to identify the exact link or device causing the measured losses or delay. To address this problem, we propose a novel algorithm that can be used to converge on the smallest identifiable subnetworks where losses or OWD can be measured. We name these subnetworks *Clusters*.

The contributions of this paper are: (*i*) the design of a novel solution for passive monitoring of multipoint flows in packet networks, (*ii*) a technique to model the monitored network, (*iii*) an algorithm to identify the clusters, (*iv*) the validation of the method for packet loss measurements through experiments and (*iv*) the analysis of the features and recurrent patterns of these subnetworks in real backbone networks.

The rest of the paper is organized as follows: Section II briefly describes the alternate marking method. Section III presents the modeling approach (III-A), the computation of the amount of lost packets per block (III-B), the clustering algorithm (III-C), and the evaluation of average OWD (III-D). Section IV summarizes one possible method to implement this technique on existing network devices. Section V validates the method (V-A), analyzes the size of the model (V-B), the features of the clusters (V-C) and the execution time of the proposed algorithms (V-D). Section VI surveys related works and Section VII concludes the paper. Finally, Appendix A delves into the algorithm to generate the monitored network model.

II. Alternate Marking method

This section gives an overview of the alternate marking method, upon which our work is based. This method is aimed at coordinating live traffic monitoring to measure the amount of lost packets in a given interval, the one-way and round-trip forwarding delays, and the interarrival jitter. Considering a single unicast flow, a straightforward method to measure lost packets is to compare the number of packets entering and leaving the monitored network. Additionally, packets can be counted by multiple network devices along the path to identify the specific devices or links that are causing the losses. This comparison can be performed periodically and requires some form of synchronization in order to count at any measurement point the same sets of packets.

The alternate marking method provides this synchronization by splitting a long-lived flow into consecutive blocks, where each block represents a measurable entity unambiguosly recognizable by all the network devices along a path, so that they can count packets belonging to the same block and compare the measurements to define the number of lost packets among the various measurement points. The ingress network devices define these blocks by equally marking all the packets in the same block, while consecutive blocks are characterized by a different mark. To simplify the implementation on existing network devices, a single bit is used for packet marking. This bit has the same value in packets belonging to the same block and a different value in packets of consecutive blocks. In common routers, packet marking can be easily implemented, without software modifications, using bits of the DSCP field of the IP header, as demonstrated by the operational experiment presented in [5] (see Section IV).

According to the alternate marking method, the ingress devices switch the value of the marking bit at regular intervals of *T* seconds, requiring a loose synchronization of the clock of such devices (e.g., NTP-based synchronization). Specifically, two endpoints, whose measurements are compared, must have a maximum clock displacement of *A* seconds, such that:

$$T > 2(A + D_{max} - D_{min}) \quad (1)$$

where D_{min} and D_{max} denote respectively the lower and upper bounds of the network delay between the two network devices. As a result, the duration *T* of a block is chosen, according to equation (1), considering the maximum synchronization error and the maximum and minimum delay between the endpoints. This delay must be considered to account for out of order packets. In fact, the values of packet counters are not read immediately after the end of a block, since there could be delayed packets arriving out of order. On the contrary, the counters are read after half a block period ($T/2$ seconds), to minimize the possibility that a packet of that block arrives afterwards. The estimation of lower/upper bound of delay between nodes is required only at setup time, so it can be performed using ad-hoc methods (active probing, GPS-based time synchronization). Since this delay can change depending on network conditions (e.g., time that packets spend in buffers), the duration of a block should be chosen conservatively. Moreover, the block duration must be chosen as a trade-off between the resolution of the measurement and the size of the reporting traffic. A larger block interval reduces the frequency of reports, but it also causes a higher delay in identifying loss events.

This technique adds a very small overhead to network devices. In fact, each measuring node needs to keep only 2 counters per interface, one for each value of the marking bit. All packet losses happening within 2 different counters are detected, given the difference between the counters. This comprises losses occurring on links, in packer buffers and within network devices. Additionally, the alternate marking method can also be used to measure the one-way delay between 2 network devices. However, this requires that the network devices are synchronized, so that the timestamps recorded by different nodes are comparable. To perform one-way delay measurements, the two endpoints save a timestamp when they receive the first packet of a block. The difference between the two timestamps corresponding to the same block provides the one-way delay between the two nodes. However, in this case a measurement is valid only if the first packet of a block is not lost and is not received out of order. Alternatively, to have a one-way delay measurement that tolerates lost and out-of-order packets, network devices can perform mean delay measurements. In this case they take the timestamp of all the packets in a block and compute the mean timestamp, which is used to measure the mean delay between the 2 endpoints. Moreover, in case of symmetric paths, the mean delay in the 2 directions can be summed to measure the round-trip delay. In this last case, the constraint on the devices synchronization is relaxed, since the synchronization error (which is equal and opposite in the two directions) is canceled by the sum. To measure the round-trip time the requirement on number of counters per interface is increased to 4, since two counters are needed for each direction. Similarly, the measurements of the one-way delay can be used to measure absolute and average interarrival jitter, by evaluating the delay variation of consecutive samples.

Since this method is based on packet counters, it relies on the assumption that packets are not split or aggregated between two measurement points. As a result, fragmentation should be carefully addressed. This is not a concern in IPv6 networks (where fragmentation is performed by the source) and is practically irrelevant in IPv4 networks of a single operator (where fragmentation most commonly happens at the network edge [8]). In networks managed by multiple operators, fragmentation can be addressed simply by adding 2 measurement points where fragmentation can occur (e.g., at the border of networks with different MTUs). The management plane is responsible for adding counters whenever there is a configuration change that could cause fragmentation on a link.

The correlation of counters and timestamps of different nodes can be performed by a centralized entity in the *Network Operation Center* (NOC) using a Network Management System, or using a different protocol to distribute the measurements.

III. Multipoint Performance Monitoring

The alternate marking method presented in the previous section can be applied only to unicast flows (e.g., one TCP connection) because it assumes that all the packets of the flow measured on one node are measured again by a single second node. In this section we generalize this method to measure any kind of flow, whose packets can follow several different paths in the network, as shown in Figure 1. In fact, the definition of the term ’flow’ present in the IPFIX standard [2] (i.e., all the packets having a set of common properties) is more general than a single TCP connection. As an example, we can define as a flow all the packets sharing the same source IP address. Clearly, packets belonging to this flow follow different paths in the network to reach their intended destination.

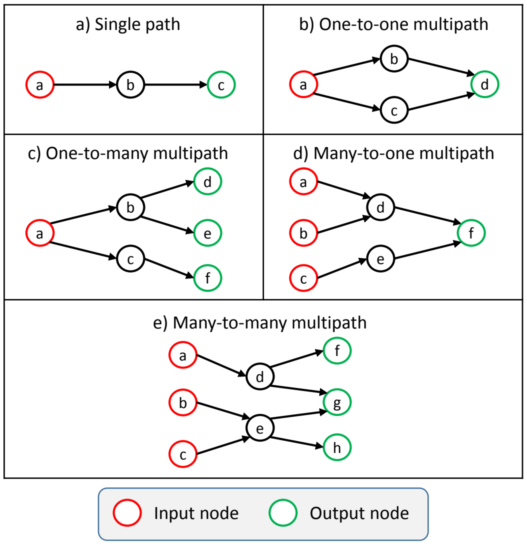


Fig. 1. Examples of graphlets resulting from flow paths.

Following this definition, we consider that a flow can be defined by a set of selection rules used to match a subset of the packets processed by the network device. These rules specify a set of headers fields (*Identification Fields*) and the relative values that must be found in matching packets. The choice of the identification fields directly affects the type of paths that the flow would follow in the network. In fact, it is possible to relate a set of identification fields with the pattern of the resulting graphs, as listed in Figure 1. A TCP/IP 5-tuple usually identifies flows following either a single path or a one-to-one multipath (in case of load balancing). On the contrary, a single source address most probably selects flows following a one-to-many multipath, while a many-to-one multipath can be the result of a matching on a single destination address. It is worth noting that a selection rule and its reverse are used for bidirectional measurements, therefore they can correspond to a one-to-many multipath in one direction and a many-to-one multipath in the opposite direction. While the alternate marking method is applicable only to a single path (and partially to a one-to-one multipath), the extension proposed in this paper is suitable also for the most general case of many-to-many multipath, which embraces all the other patterns.

These selection rules can be implemented using commonly available features of network devices, such as ACLs (see Section IV), which guarantee packet matching at line rate. The number of rules that can coexist, and thus the number of possible monitored flows, depends on the capability of the device. However, it is worth noting that the use of few match-action rules to monitor multipoint flows is by far more scalable than existing solutions, which require one rule for each point-to-point flow [2]. When needed, multipoint flows can be defined to aggregate multiple rules with a common feature (e.g., same source IP) to reduce the number of counters deployed, at the cost of coarser measurement granularity.

A. Monitored network

In this section we provide a model of the monitored network based on the assumption that each network interface can separately count packets in both the incoming and the outgoing directions. We model the entire monitored network as a directed network $G = (N, A)$ defined by a set N of n *nodes* and a set A of m *directed arcs*. Each network interface i is modelled by two nodes, $i_i \in N$ and $i_o \in N$, corresponding to the incoming and the outgoing directions, respectively.

Each physical link connecting two interfaces i and j, is modelled by two directed arcs, $(i_o, j_i) \in A$ and $(j_o, i_i) \in A$, corresponding to the two directions. To model the routing process, we also consider one directed arc $(m_i, n_o) \in A$ for each ingress node m_i and egress node n_o corresponding to two interfaces m, n on the same physical device. This comprises also directed arcs $(m_i, m_o) \in A$ connecting the ingress and egress nodes corresponding to the same physical interface, in order to model the possibility that a packet enters and leaves from the same interface.

As a result, if there are p links in the network and q network devices in total, each one with r_i active interfaces, with $i \in \{1, \ldots q\}$, the number of nodes n and the number of arcs m are:

$$n = 2 \times \sum_{i=1}^{q} r_i \qquad (2) \qquad\qquad m = 2 \times p + \sum_{i=1}^{q} r_i^2 \qquad (3)$$

As a requirement, all the border interfaces of the monitored network must be configured to separately count incoming and outgoing packets matching the chosen identification fields. On the other hand, only a subset of the internal network interfaces can be selected to monitor the traffic. As a consequence, we consider a directed network $\bar{G} = (\bar{N}, \bar{A})$ where:

- $\bar{N}$ is the subset of N containing only the nodes corrisponding to a monitored interface;
- $\bar{A}$ is a set of arcs, where an arc $(i, j) \in \bar{A}$ corresponds to a possible directed path in G between two nodes $i \in \bar{N} \subseteq N$ and $j \in \bar{N} \subseteq N$, not crossing any other node in $\bar{N}$.

Figure 2 shows the directed network corresponding to a simple network with only 3 devices and 3 monitored interfaces.

To find all the arcs in $\bar{A}$, it is important to avoid the search for all the simple paths in G. In fact, the number of simple paths in G can be very large, $O(n!)$ in the worst case [9]. However, we can check the existence of a single path using Dijkstra's algorithm with worst-case performance $O(m + n \log n)$ [10]. Therefore, to verify the existence of a path in G between $i \in \bar{N}$ and $j \in \bar{N}$ that does not contain inner nodes in $\bar{N}$, we search for the shortest path in the graph G^* with nodes $N^* = N - \bar{N} + \{i, j\}$ and as arcs A^* all the arcs in A connecting nodes in N^*. This algorithm, described in the Appendix A, has complexity polynomial in m,n and $\bar{n}$, as shown in Section V-D.

We define the set of *input nodes* (or sources) of $\bar{G}$ as the set of nodes in $\bar{N}$ without incoming arcs:

$$I = \{j \in \bar{N} \,|\, \nexists(i, j) \in \bar{A} \quad \forall i \in \bar{N}\} \tag{4}$$

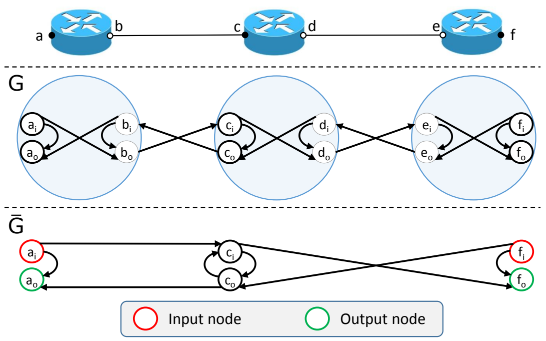


Fig. 2. A simple network model. The monitored interfaces are illustrated as full dots.

and the set of *output nodes* (or sinks) of $\bar{G}$ as the set of nodes in $\bar{N}$ without outgoing arcs:

$$O = \{i \in \bar{N} \,|\, \nexists(i, j) \in \bar{A} \quad \forall j \in \bar{N}\} \tag{5}$$

In the example presented in Figure 2 these sets are $I = \{a_i, f_i\}$ and $O = \{a_o, f_o\}$.

B. Packet loss monitoring

According to the alternate marking method, every monitored physical interface has to keep four counters, two counters for incoming packets with the two different marks, and similarly two counters for outgoing packets. After some time following the mark switch (e.g. $T/2$ time units if T is the duration of each block with the same mark), it is possible to read the still counters associated with the previous block. Consequently we associate with each node $i \in \bar{N}$ a non-negative integer number $b(i) \geq 0$ representing the number of packets counted on the corresponding interface and direction. Considering the example in Figure 2, $b(c_i)$ is the value of the still counter measuring the incoming packets in the interface c, while $b(c_o)$ is the value of the still counter measuring the outgoing packets in the same interface.

Since all the packets of the considered flow leaving the network have previously entered the network, the number of packets B_{in} counted by all the input nodes is always greater or equal than the number of packets B_{out} counted by all the output nodes:

$$B_{in} = \sum_{i \in I} b(i) \geq \sum_{i \in O} b(i) = B_{out} \tag{6}$$

In particular, in a lossless network, we would have

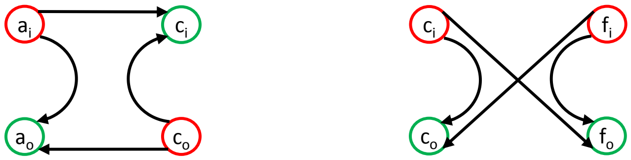
$$B_{in} = B_{out} \tag{7}$$

while, in presence of packet loss, the number of packets lost in the monitored network (of the monitored flow in the last period T) would be:

$$L = B_{in} - B_{out} \tag{8}$$

C. Network Clustering

Using the equation (8), we can determine the number of packets lost globally in the monitored network, exploiting only the data provided by the counters in the input and output nodes. In addition we can also leverage the data provided by



(a) Cluster C_1.

(b) Cluster C_2

Fig. 3. Clusters deriving from the network in Figure 2.

the other counters in the network to converge on the smallest identifiable subnetworks where the losses occur (clusters).

A cluster $\tilde{G} = (\tilde{N}, \tilde{A})$ is a minimal subnetwork of $\bar{G}$ that still satisfies the equation (8) where L in this case is the number of packets lost in the cluster. For this reason a cluster must contain all the arcs emanating from its input nodes and all the arcs terminating at its output nodes. This ensures that we can count all the packets (and only those) exiting an input node again at the output node, whatever path they follow.

In a completely monitored network (a network where every network interface is monitored), each network device corresponds to a cluster and each physical link corresponds to two clusters (one for each direction). In fact, when two monitored interfaces i and j are the endpoints of a physical link, one cluster would be made of $\tilde{N} = \{i_o, j_i\}$ and $\tilde{A} = \{(i_o, j_i)\}$, while a second cluster would have $\tilde{N} = \{i_i, j_o\}$ and $\tilde{A} = \{(j_o, i_i)\}$. This is reasonable, since, on a physical link, all the packets leaving one interface either reach the other endpoint or are lost. However, in general, two monitored interfaces can be far apart and divided by one or multiple non-monitored devices, thus making more complex the task of identifying a cluster (i.e., a subnetwork where all the packets can be counted exactly twice, upon entering and leaving). Figure 3 shows the 2 clusters deriving from the network in Figure 2. Considering, as an example, the cluster C_1 in Figure 3a, if:

$$B_{in} = b(a_i) + b(c_o) > b(a_o) + b(c_i) = B_{out}$$

there is a packet loss:

$$L = b(a_i) + b(c_o) - b(a_o) - b(c_i)$$

but it is not possible, by looking at the counters, to identify exactly the link where the losses occur, therefore the cluster is minimal.

A simple algorithm to identify the clusters is defined by the following steps:

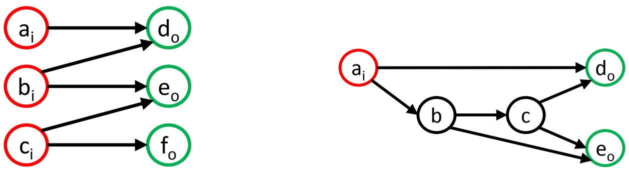
1) Compose the sets $\tilde{A}_i \quad \forall i \in \bar{N}$ containing all the arcs $(i, j) \in \bar{A}$ with the common tail $i \in \bar{N}$;
2) Join all the sets $\tilde{A}_i$ and $\tilde{A}_j$ containing at least one arc $(i, z) \in \tilde{A}_i$ and one arc $(j, z) \in \tilde{A}_j$ with a common head $z \in \bar{N}$;
3) Each one of the composed sets of arcs, together with the nodes that acts as their endpoints, constitutes a cluster.

The corresponding pseudocode is presented in the Algorithm 1. Note that the line 5 must iterate also on the arcs of $\tilde{A}_i$ added in the process.

It is worth noting that, while the most common cluster is the one with only one hop between ingress and egress nodes, it is possible to have clusters with multiple hops. Figure 4a shows the typical one-hop cluster layout, while Figure 4b shows an example of multi-hop cluster. The value of the counters of intermediate nodes in a multi-hop cluster (e.g., nodes *b* and *c* in Figure 4b) are unusable because they count an unknown fraction of the packets from/to the ingress/egress nodes.

Algorithm 1: Iterative clustering algorithm

```
Input: $\bar{N}$ and $\bar{A}$
Output: Clusters $\tilde{A}_i$
1  $\bar{n} \leftarrow |\bar{N}|$
2  forall $(i, j) \in \bar{A}$ do                          ▷ Group arcs by the tail
3      $\tilde{A}_i \leftarrow \tilde{A}_i \cup \{(i, j)\}$
4  forall $\tilde{A}_i$ do
5      forall $(i, j) \in \tilde{A}_i$ do
6          for $i' \leftarrow i + 1, \bar{n}$ do
7              forall $(i', j') \in \tilde{A}_{i'}$ do
8                  if $j = j'$ then
9                      $\tilde{A}_i \leftarrow \tilde{A}_i \cup \tilde{A}_{i'}$
10                     $\tilde{A}_{i'} \leftarrow \emptyset$
11                     break                               ▷ Go to the next $\tilde{A}_{i'}$
```



(a) Common one-hop cluster.

(b) Multi-hop cluster.

Fig. 4. Cluster layouts.

In the worst case, when each cluster is made of only one arc (the network $\bar{N}$ is a *path*), Algorithm 1 needs $\bar{m}$ iterations (where $\bar{m}$ is the number of arcs in $\bar{A}$) to compose the groups according to the arc tail, and

$$\sum_{i=1}^{\bar{m}-1} (\bar{m} - i)$$

iterations to test if any groups should be joined. As a result the total number of iterations in the worst case is:

$$\bar{m} + \sum_{i=1}^{\bar{m}-1} (\bar{m} - i) = \bar{m} + \frac{\bar{m}(\bar{m} - 1)}{2} \tag{9}$$

and the complexity of the algorithm is $O(\bar{m}^2)$.

Leveraging the *node-node adjacency matrix representation* [11], it is also possible to apply a recursive algorithm to identify the clusters in the network $\bar{G}$. The node-node adjacency matrix representation stores the network $\bar{G}$ as an $\bar{n} \times \bar{n}$ matrix $M = \{h_{ij}\}$. The matrix has a row and a column corresponding to every node, and its entry h_{ij} equals 1 if $(i, j) \in \bar{A}$ and equals 0 otherwise. The matrix has $\bar{n}^2$ elements, only $\bar{m}$ of which are nonzero. We can obtain the arcs emanating from node i by scanning the *i-th* row: if the *j-th* element in this row has a nonzero entry, (i, j) is an arc

Algorithm 2: Recursive clustering algorithm

```
Input: $\bar{N}$ and $M = \{h_{ij}\}$    ▷ Node-node adjacency matrix
Output: Clusters $\tilde{A}_i$
forall $i \in \bar{N}$ do
    $\tilde{A}_i \leftarrow \emptyset$
    InspectRow($i$,$\tilde{A}_i$)
Function InspectRow($i$,$C$)
    forall $j \in \bar{N}$ do                    ▷ Scan row $i$
        if $h_{ij} = 1$ then
            $C \leftarrow C \cup \{(i, j)\}$
            $h_{ij} \leftarrow 0$
            forall $i' \in \bar{N}$ do          ▷ Scan column $j$
                if $h_{i'j} = 1$ then
                    $C \leftarrow C \cup \{(i', j)\}$
                    $h_{i'j} \leftarrow 0$
                    InspectRow($i'$,$C$)
```

of the network $\bar{G}$. Similarly, we can obtain the arcs entering node j by scanning the *j-th* column: if the *i-th* element of this column has a nonzero entry, (i, j) is an arc of the network $\bar{G}$. With these steps we can to identify all the outgoing or incoming arcs of a node in time proportional to $\bar{n}$.

Using the adjacency matrix, for each arc $(i, j) \in \bar{A}$ we can form a cluster by grouping all the outgoing arcs of i, all the incoming arcs of j and, recursively, all the outgoing and incoming arcs of the newly added nodes. The detailed pseudocode is presented in the Algorithm 2. This algorithm requires a full scan of the adjacency matrix M ($\bar{n}^2$ operations) and a row or column scan for every arc $(i, j) \in \bar{A}$. As a result the total number of iterations is, in any case:

$$\bar{n}^2 + \bar{n} \times \bar{m} \tag{10}$$

In the event that $\bar{m} \gg \bar{n}$, the complexity of Algorithm 2 is $O(\bar{n} \times \bar{m})$, which makes this algorithm preferable to Algorithm 1. If $\bar{m}$ and $\bar{n}$ are comparable in size, Algorithm 1 is faster on average.

By applying to each cluster $\tilde{G} = (\tilde{N}, \tilde{A})$ the equations (4) and (5) (to select the input and the output nodes of the cluster) and the equation (8), we can determine how many packets are lost in the cluster. By identifying the cluster(s) that have a relevant packet loss, we can select the corresponding devices and links that need maintenance.

While it is not possible to identify exactly what link or device in a cluster is causing the losses, we can assign to each non-ingress node the number of packets probabilistically lost in the paths from the ingress nodes towards the node itself. First we can define the *loss probability* of a cluster as the number of packets lost in the cluster over the total number of packets received by the cluster:

$$\Pr(loss) = \frac{L}{B_{in}} = \frac{B_{in} - B_{out}}{B_{in}} \tag{11}$$

Since in a cluster we do not have specific measurements for a single arc entering a node, we assume that all of the possible paths in the cluster have the same probability $\Pr(loss)$ to lose packets. As a result, we can associate with each non-ingress node i in a cluster C a non-negative value $l_{us}(i)$ corresponding to the number of packets probabilistically lost in the paths leading to that node (upstream to the node). The value of $l_{us}(i)$ is given by the product between the loss probability and the number of packets sent towards that node. The latter number corresponds to both the packets counted by the node and probabilistically lost in the paths upstream:

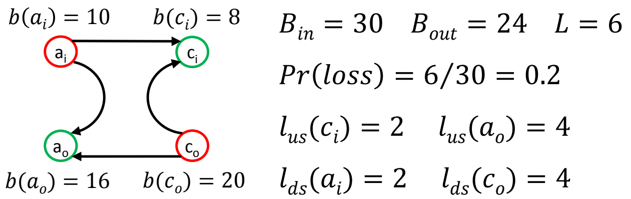

$b(a_i) = 10$
$b(c_i) = 8$
a_i
c_i
a_o
c_o
$b(a_o) = 16$
$b(c_o) = 20$
$B_{in} = 30$ $B_{out} = 24$ $L = 6$
$Pr(loss) = 6/30 = 0.2$
$l_{us}(c_i) = 2$ $l_{us}(a_o) = 4$
$l_{ds}(a_i) = 2$ $l_{ds}(c_o) = 4$

Fig. 5. Example of loss probabilities.

$$\begin{aligned} l_{us}(i) &= [b(i) + l_{us}(i)] \times \Pr(loss) \implies \\ l_{us}(i) &= b(i) \times \frac{\Pr(loss)}{1 - \Pr(loss)} = b(i) \times \frac{B_{in} - B_{out}}{B_{out}} \\ &= b(i) \times \frac{L}{B_{out}} \end{aligned} \tag{12}$$

Equation (12) is valid for $B_{out} \neq 0$, while for $B_{out} = 0$ the cluster is losing all the packets, thus all the paths are dropping all the packets (all the paths have the same loss probability $\Pr(loss) = 1$). Moreover, the upper bound of $l_{us}(i)$ is L which is reached when $b(i) = B_{out}$ (all the packets are routed towards the same egress node, thus the paths to that node are causing all the losses). As expected, the sum of the probabilistic losses in the paths towards all the egress nodes is equal to the total number of losses:

$$\sum_{i \in O} l_{us}(i) = \frac{L}{B_{out}} \times \sum_{i \in O} b(i) = L \tag{13}$$

Symmetrically, we can define for each non-egress node i in a cluster C a non-negative value $l_{ds}(i)$ corresponding to the number of packets probabilistically lost in the paths originated from that node (downstream to the node):

$$l_{ds}(i) = b(i) \times \Pr(loss) = b(i) \times \frac{L}{B_{in}} \tag{14}$$

Figure 5 shows a numerical example of the upstream and downstream loss probabilities for the cluster C_1 in Figure 3a. The values of $l_{us}(i)$ and $l_{ds}(i)$ can be used to provide a ranking of probable causes for the losses, driving the maintenance operation.

D. Average delay

While this paper focus is primarily on packet loss measurements, in this section we present briefly how also average delay measurements can be generalized to the case of multipath flows. Similarly to the original alternate marking method, the network devices must be synchronized and they must be able to extract a timestamp when they receive a packet. Delay measurements relative to a single packet cannot be performed in this scenario, since they would not be representative of the entire flow, whose packets can follow different paths with various delays. However, we can compute the average one-way delay $\bar{D}$ of packets, in one block, either in a cluster or in the entire monitored network. In fact, for each node, we can consider the mean timestamp $\bar{t}(i)$ of all the packets in one block:

$$\bar{t}(i) = \frac{\sum\limits_{k=1}^{b(i)} t_k(i)}{b(i)} \tag{15}$$

where $t_k(i)$, $k \in \{1, \ldots, b(i)\}$ are the timestamps of the $b(i)$ packets in a block.

The average delay $\bar{D}$ can be measured as the difference between the weighted averages of the mean timestamps of the sets of output and input nodes:

$$\begin{aligned} \bar{D} &= \frac{\sum\limits_{i \in O} [b(i) \times \bar{t}(i)]}{\sum\limits_{i \in O} b(i)} - \frac{\sum\limits_{i \in I} [b(i) \times \bar{t}(i)]}{\sum\limits_{i \in I} b(i)} \\ &= \frac{\sum\limits_{i \in O} \sum\limits_{k=1}^{b(i)} t_k(i)}{\sum\limits_{i \in O} b(i)} - \frac{\sum\limits_{i \in I} \sum\limits_{k=1}^{b(i)} t_k(i)}{\sum\limits_{i \in I} b(i)} \end{aligned} \tag{16}$$

We briefly presented how the proposed technique can be extended to average delay measurements. However, given the space limit, an in-depth analysis of average delay measurements using this method is outside the scope of the current paper, which focuses on packet loss measurements. We will address this aspect in future work.

IV. Implementation on existing devices

As demonstrated by the operational experiment presented in [5], the alternate marking method can be implemented using features already available in common network devices. In this section we summarize one possible method to implement this technique. More details can be found in [5].

Packet marking requires only a single bit in the packet header. We assume that one bit of the DSCP field of the IP header can be reserved for this purpose, and the service provider can still use the remaining 5 bits for QoS classification. Per-flow packet marking using the DSCP field can be implemented by configuring access lists (ACLs) on the device output interfaces [12]. The ACL matches the monitored flow(s) and applies a policy that sets the DSCP field. The policy is periodically updated by an automatic script running on the device to change the value of the marking bit for the next block. This operation must be performed by devices at the edge of the network (i.e., where the monitored flows enter and leave the network), so that all the traffic in the monitored network is correctly marked.

The operator must select the devices inside the network that count marked packets. They can be configured by using an ACL that matches specific DSCP values and counts packets of the flow(s) being monitored. Additionally, network flow monitoring, such as provided by IPFIX [2], can be used to extract timestamps of the first and last packet of a batch in order to perform delay measurements. The deployment of MPs can be incremental. In fact, only routers at the border of the monitored network are strictly required to mark and count packets. Counters on internal routers can instead be deployed progressively to reduce the size of the measuring clusters. Moveover, the monitored network can be progressively extended when counters are deployed on new routers.

A script running on the network device can periodically collect counters and timestamps and send them to the NOC for processing. The NOC is responsible for packet loss calculations, performed by comparing the values of counters from different devices. It is worth noting that the use of the DSCP field for marking implies that the method would reliably work only within a single management and operation domain, which is within the scope of this work.

V. Experimental evaluation

In this section we evaluate the effectiveness of our approach and analyze the dimensions of the networks G, $\bar{G}$ and the clusters corresponding to real world network topologies.

Our main results are as follows:

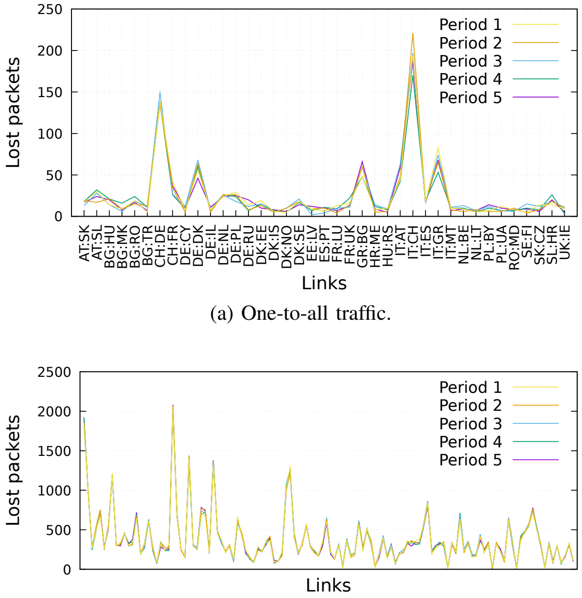
- We show how our approach can be used to passively monitor multipoint flows in real topologies. The results show that, using only packet counters, we can provide very detailed measurements that allow to identify the exact packet loss between multiple measurement points, while active methods require ad-hoc agents at the edge of the network and provide measurements per path.
- We assess the sizes of the networks G and $\bar{G}$ for multiple real topologies.
- We evaluate how the number of clusters and their size change with the number of monitored interfaces. These results show how the monitoring resolution increases, together with our ability to identify the cause of packet losses, when we increase the number of monitored nodes.
- We measure the execution time of the presented algorithms to evaluate how they scale with an increasing number of monitored interfaces.

All the details to reproduce our tests, together with the required software, are available in our public repository[1]. In our tests we use real world network topologies surveyed in [13] leveraging data that network operators make public. We run the experiments on a workstation with an Intel Core i7-4770 CPU (8 logical cores at 3.40GHz) and 32 GB of RAM running Linux Fedora 25.

A. Packet loss measurements

To test our approach in a realistic scenario with realistic traffic, we emulate two network topologies: GEANT [14], the pan-European network for the research and education community with 40 nodes and 61 links, and BICS [15], the backbone network managed by its namesake company with 33 nodes and 48 links. We choose these 2 topologies because they have different shapes and they are the largest ones that our testbed can manage. Additionally, they are real backbone networks with publicly available topologies, often

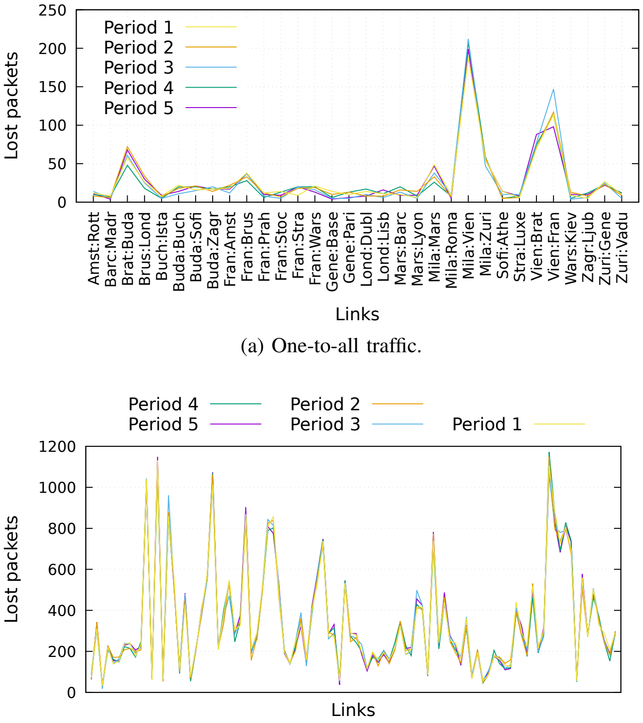
[1] https://github.com/netgroup-polito/Multipoint-monitoring



(a) One-to-all traffic.

(b) All-to-all traffic.

Fig. 6. Packet loss measurements on the GEANT topology.



(a) One-to-all traffic.

(b) All-to-all traffic.

Fig. 7. Packet loss measurements on the BICS topology.

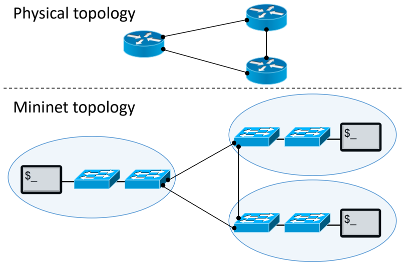


Fig. 8. Example of physical and emulated topology.

used in networks research [13], [16], [17]. Larger topologies are analyzed with simulations in the subsequent sections.

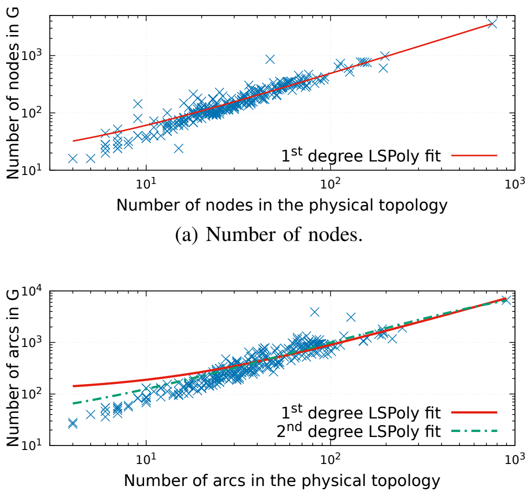
Using Mininet [18] and Open vSwitch [19], we deploy two bridges and one host (a linux namespace) for each node in the topology (see Figure 8). We need two bridges for each node because one is used for packet marking and one to count the packets. We observed that if we use a single bridge for both operations, the newly marked packets are occasionally not counted because of Open vSwitch flow caching [20]. One bridge is connected only to the host and to the second bridge to mark packets from the host, while the second bridge is connected according to the topology. We also use `tc` to artificially drop 1% of the packets on each link to emulate packet losses.

An OpenFlow controller connected to all the bridges takes care of: (*i*) pushing the forwarding rules according to the shortest path for each source and destination pair, (*ii*) periodically push the rules on the bridges directly connected to the hosts to mark external packets that match the chosen Identification Fields with an alternating value (we use one bit in the TOS IP field for the mark), (*iii*) push the rules to count internal marked packets on each interface in both directions, and (*iv*) periodically read and reset the still counters and correlate the measurements from multiple nodes. To validate the method, in this test we are deploying one counter per direction on each interface (i.e., all the interfaces are monitored) , while in Section V-C we evaluate the clustering algorithm varying the ratio of monitored nodes.

Since we are interested in measuring multipoint traffic, in a first experiment we use as identification field the source IP to measure one-to-all flows. In a second experiment we count all the packets to measure all-to-all traffic. To have a complete mesh background traffic, we run a packet generator in every host that sends/receives UDP packets to/from all the other hosts. At the end of the test every packet generator instance sends a report to a central collector with the number of packets sent and received for each source and destination pair. We use these reports to measure the amount of lost packets on each path (from one host to the other).

We observed that, in every path, the number of lost packets reported by the traffic generator is equal to the sum of the losses reported by the controller for all the links in the path for values of the period T greater than 20 seconds. Below this value, it is possible that packets are miscounted and the measurements are not accurate. This parameter must be adapted to the specific deployment scenario considering: (*i*) the lower and upper bounds of the network delay between two network devices, (*ii*) the synchronization error among the devices, and (*iii*) the overall number of measurements that the NOC can process per minute. Considering that there are two measurements (one per direction) per monitored interface per period, the number of measurements to process can be large for sizable topologies (see Section V-B).

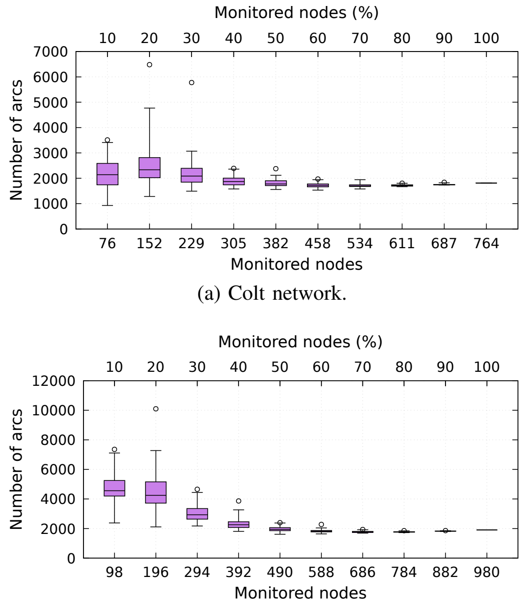
Figure 6 and Figure 7 show the measured packet loss per



(a) Number of nodes.

(b) Number of arcs.

Fig. 9. Relation between the physical topology and G.



(a) Colt network.

(b) Cogent network.

Fig. 10. Relation between the number of monitored interfaces (i.e., $|\bar{N}|$) and the number of arcs in $\bar{A}$.

link and per period as reported by the controller for GEANT and BICS, respectively. The x-labels report the names of the links in the topology. The number of losses on each link is proportional to the amount of traffic on that link, which depends on how many shortest paths between nodes contain that link.

Figure 6a shows the measurements obtained when selecting the flow with source IP corresponding to the `IT` node in GEANT, while Figure 7a is obtained measuring packets generated by the `Milan` node in BICS. These figures show only the links with nonzero reports, which are the ones in the single-source shortest paths from the selected node to all the other nodes. Figure 6b and Figure 7b show the measured losses of the all-to-all flow in all the links in the two topologies. While the packet generator provides only the total number of packets lost in a path between one source and one destination, our method provides a fine grained measurement per link per time period.

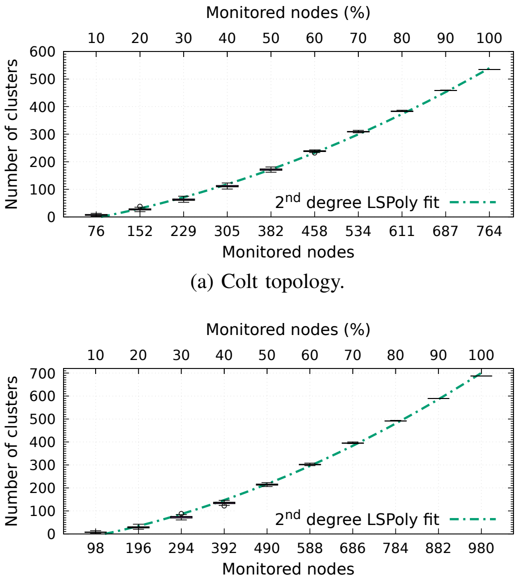
B. Graph size

In this section we evaluate the size of the network G and $\bar{G}$ for real topologies. The number of nodes n in G depends on the number of devices in the real topology and on the number of interfaces per device (Equation 2), while the number of arcs m in G depends also on the number of links (Equation 3). The nodes $\bar{N}$ in $\bar{G}$ correspond to the interfaces that are chosen to count packets, while the arcs $\bar{A}$ correspond to the paths between monitored interfaces, and are computed using Algorithm 3.

To evaluate the trend of n and m we compute the network G for all the 261 topologies in [13]. The smallest topology in this set has 4 nodes and 4 arcs and the largest has 754 nodes and 899 arcs. Figure 9 shows the resulting values of n and m. Specifically, in Figure 9a we show the value of n compared to the number of nodes in the original topology. We observe that the relation is close to linear. In fact, the first degree least squares polynomial (*LSPoly*) fit has a small root-mean-square error (*RMSE*), equal to 57.3. Similarly, in Figure 9b we show the value of m compared to the number of arcs and we measure that the linear fit has a RMSE of 279.9, while the quadratic fit has a RMSE of 269.98. Given the very small improvement, the linear fit can be considered a valid approximation, proving that the model scales well and an increase in the size of the physical topology will cause a comparable increase in the size of the model.

To evaluate the size of $\bar{G}$, that depends on the choice of monitored interfaces, we consider two real IP topologies, (*i*) Colt [21], a European network with 153 nodes and 191 links and (*ii*) Cogent [22], a multinational ISP network with 197 network nodes and 245 links. The interface-level network G generated from the Colt topology has 764 nodes and 1814 arcs, while the one generated from Cogent has 980 nodes and 1906 arcs. To build the network $\bar{G}$, we randomly select, with different sampling ratios, subsets of nodes in G as monitored interfaces and build the corresponding network $\bar{G}$ using Algorithm 3. We repeated the test 100 times for each ratio of monitored nodes to evaluate how the size of $\bar{A}$ varies with different selections of monitored interfaces.

Figure 10 shows the number of arcs in $\bar{G}$ for different ratios of monitored nodes for the Colt and Cogent networks. The whiskers in all the boxplots in this paper span the 99% of the points, while the outliers are shown separately. In a completely monitored network (with 100% monitored nodes) the networks $\bar{G}$ and G are equivalent, therefore the number of arcs in $\bar{G}$ is exactly equal to the above mentioned values for G in all the tests. With few monitored interfaces we observe that there is a large variability. The number of paths between monitored interfaces (i.e., $|\bar{A}|$) can be both lower or higher than the number of arcs in A. However, $|\bar{A}| > |A|$ is more frequent because with fewer monitored interfaces the paths between them are closer to a full-mesh, thus it is common that the number of paths exceeds the number of underlying arcs.



(a) Colt topology.

(b) Cogent topology.

Fig. 11. Number of clusters for different ratios of monitored nodes.

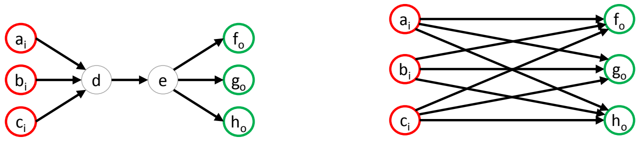
We show an example of this behavior concerning clusters in Section V-C.

C. Clusters dimensions

In this section we study the dimensions and shape of the clusters resulting from the algorithms 1 or 2 (both algorithms provide the same results). We run the same tests described in the previous section and, in all the executions, we apply to $\bar{G}$ the clustering algorithm to compute all the clusters $\tilde{G}$. Figure 11 shows the number of clusters for obtained for different sampling ratios and the second degree LSPoly fit, which shows a quadratic relation between the number of monitored nodes and the number of clusters.

We observe that the variability is very low and decreases when increasing the number of monitored nodes. This suggests that the different positioning of monitored nodes has a relatively low effect on the resulting clusters, therefore, in most cases, a random selection (or a selection based on external constraints) can be as effective as a more complex, intelligent selection aimed at finding the optimal counters distribution that provides the maximum number of smallest clusters. Additionally, a simple and effective strategy is to iteratively add counters in the portion of the network corresponding to the larger clusters, until the required monitoring resolution (i.e., maximum cluster size) is reached. Similarly, small clusters can be joined together to compose larger clusters, removing the counters from shared nodes. The same small variability can be seen in the other dimensions of the clusters, although with a larger tail.

Figure 13 shows the features of the clusters, obtained by running the test until 20000 clusters for each sampling ratio are generated. We show only the results for the Cogent network. However, we also performed these tests with different topologies, obtaining comparable results. In fact, our tests show that clusters have similar features regardless of the initial topology. Figure 13a and Figure 13b show the number of nodes and arcs in the clusters respectively. They show that a cluster with only two nodes and one arc is the most recurrent (corresponding to both the first quartile and the median). However, clusters can rarely reach hundreds of nodes and thousands of arcs . To further study the shape of the clusters we show in Figure 13c the diameter of the cluster. Given that clusters are feed-forward, non-connected graphs, we consider a slightly relaxed definition of diameter. We define diameter as the greatest **finite** distance between any pair of vertices. We observe that more than 75% of the clusters have the layout shown in Figure 4a (diameter equal to one) and multi-hop clusters (up to 19 hops) rarely occur. Multi-hop clusters are usually undesirable because the counters in the intermediate nodes are useless, as described in Section III-C.



(a) Physical cluster.

(b) Logical cluster.

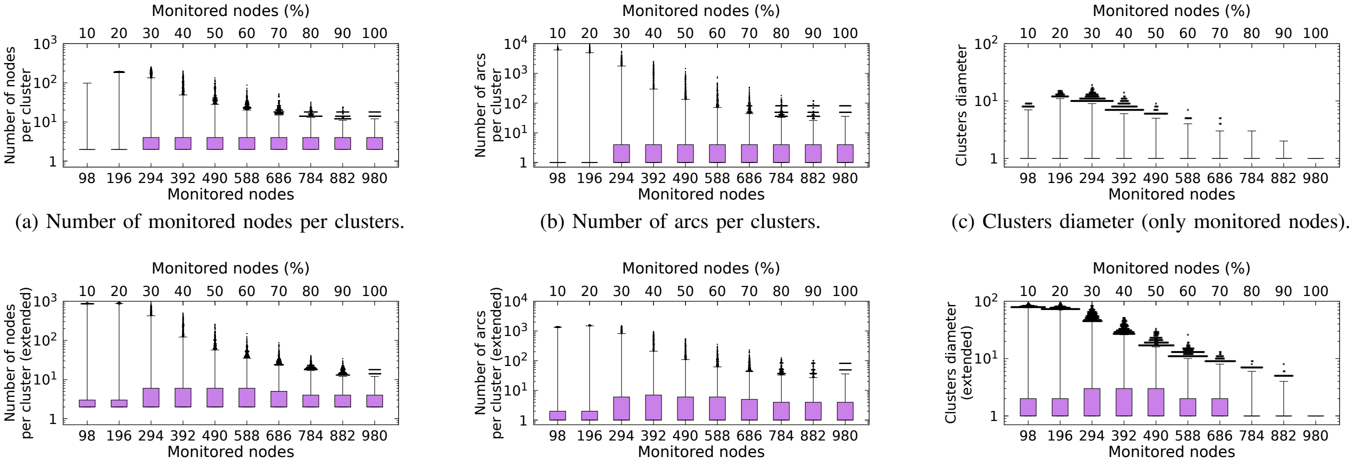
Fig. 12. Example of cluster where the number of paths between monitored nodes is greater than the number of arcs making those paths (nodes d and e are not monitored).

Next, we analyze the physical size of the clusters, namely the size of the clusters considering also the non-monitored interfaces that are in the paths between monitored ones. In general, all the physical nodes (i.e., monitored and non-monitored interfaces) can be responsible for packet losses in a cluster. Figure 13d shows the total number of nodes (i.e., physical interfaces) per cluster while Figure 13e shows the total number of arcs (i.e., physical links). Even considering non-monitored nodes, the most common clusters have 2-6 nodes and 1-7 arcs. However, if we consider the 99th percentile, the number of nodes per cluster increases when considering non-monitored nodes. Counterintuitively, the number of arcs decreases. This happens because there are cases where the number of paths between monitored nodes (i.e., arcs in $\bar{A}$) is greater than the number of arcs in A making those paths (see Figure 12).

Finally, Figure 13f shows the diameter of the clusters considering also non-monitored nodes. As with logical clusters, physical clusters have one hop in the majority of cases, while rarely they can have up to 3 physical hops. This result suggests that, even with only 10% monitored nodes, most of the clusters correspond to a single link or device, therefore this deployment allows to pinpoint where the losses occur in most of the cases. The maximum amount of hops in a cluster decreases when we increase the ratio of monitored nodes, as expected. In fact, if all the interfaces are monitored, all the clusters have only one hop and correspond to either one link or one network device. In a real deployment, the number of monitored nodes has to be chosen according to the capabilities of the devices and the granularity of the measurement required.

D. Execution time

In this section we report on the execution time of the proposed algorithms. We evaluate the execution time to provide



(a) Number of monitored nodes per clusters.

(b) Number of arcs per clusters.

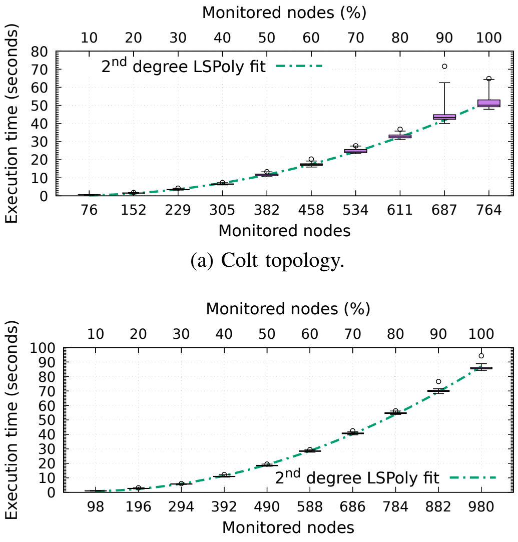
(c) Clusters diameter (only monitored nodes).

(d) Number of nodes per clusters considering non-monitored interfaces.

(e) Number of arcs per clusters considering non-monitored interfaces.

(f) Clusters diameter considering non-monitored interfaces.

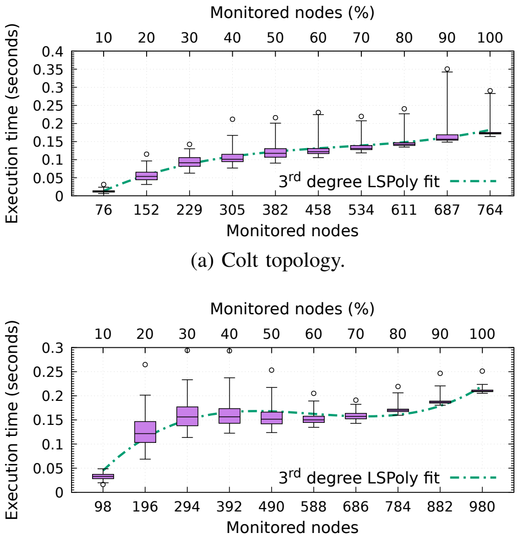
Fig. 13. Clusters dimensions with the Cogent topology.



(a) Colt topology.

(b) Cogent topology.

Fig. 14. Execution time of Algorithm 3.



(a) Colt topology.

(b) Cogent topology.

Fig. 15. Execution time of Algorithm 1 (iterative clustering).

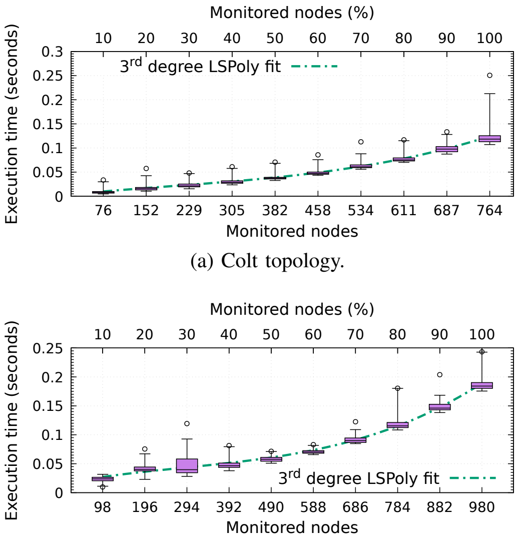
a general understanding of the timescale and how these novel algorithms scale with an increasing number of nodes. As mentioned in Section III-A, the computation of the arcs $\bar{A}$ can be expensive. However, Algorithm 3 described in Appendix A can find all the arcs in $\bar{A}$ with polynomial complexity. In fact, in our tests the execution time of Algorithm 3, reported in Figure 14, shows a quadratic trend relative to the number of monitored nodes $\bar{n}$.

In our implementation we use the single-threaded *networkx* pure-python library [23] to perform basic graph processing, such as finding the shortest path. Given the sequential nature of the Dijkstra's shortest path algorithm, a parallel implementation would provide very limited improvements [24], while a more optimized C implementation can be significantly faster [25]. The results show that, even with a commodity workstation and a non-optimized implementation, the clustering algorithms take less than a second, and the model is computed in few minutes. These are reasonable values for any practical deployment, since these algorithms can be executed offline, in the NOC, and only when the location of the monitored interfaces changes, which we expect to be a rare event.

Both clustering algorithms take substantially less time than Algorithm 3. In fact, in our tests they require always less than 0.5 seconds. Figure 15 and Figure 16 show the execution time for the iterative (Algorithm 1) and recursive (Algorithm 2) clustering algorithm, respectively. As showed in Figure 10, $\bar{m}$ can be up to 8 times greater than $\bar{n}$ therefore the recursive algorithm is faster than the iterative in all the tests.

VI. Related Work

PING is the most popular tool for network troubleshooting and to measure packet losses and round trip time. It is a basic tool that works by transmitting specific ICMP packets, so the



(a) Colt topology.

(b) Cogent topology.

Fig. 16. Execution time of Algorithm 2 (recursive clustering).

main drawbacks are that it is invasive, probe packets can be dropped or rate-limited, works only between two hosts (end-to-end) and thus cannot measure multipoint flows.

Many previous works propose methods for active measurements of packet losses and end-to-end delay [26]–[29]. The IETF has also standardized active methods for loss and delay measurements through the IPPM working group [30]–[32]. `Badabing` [26] and `Zing` [28] estimate end-to-end packet losses sending UDP probe packets between two cooperative hosts. The former sends packet at specified intervals while the latter uses poisson-modulated intervals with fixed mean rate. Poisson-modulated probes have been shown to provide unbiased network measurements in theory [31], [33], while in practice this implies that measurements must be taken either in a long period of time or with high average rate [26]. A different approach is taken by `Sting` [29] which measures packet losses between a client and any existing TCP-based server by first sending a series of TCP data packets and then analyzing the loss recovery algorithms of TCP. While these tools enhance the basic functionality of `PING`, none of them solves the abovementioned drawbacks. In particular, due to the discrete sampling nature of the probe process, active monitoring methods should run frequently to measure bursty and occasional packet losses, generating a large number of packets (especially considering that losses on the Internet are typically rare events [34]). High probe rates sensibly change the state of the network, therefore the measurement becomes non-representative of the real network dynamics. As a result, these methods must find a trade-off between measurement accuracy, impact on the network (probe rate) and timeliness of results.

Passive measurements are usually based on full traffic collection [35], [36] or statistical sampling [3], [37]. `Tstat` is a tool to passively monitor packets that flow on a link. It provides traffic statistics, obtained through offline data correlation between incoming and outgoing traffic. The analysis of network traces is usually time-consuming, therefore cannot be used to identify and rapidly react to problems threatening SLA compliance. As a result, statistical sampling is used to reduce the amount of data to process. As an example, the tool proposed in [37] analyzes a sample of TCP packets and applies a heuristic to the sequence numbers to detect losses, within a certain error. As with active probes, sampling necessarily introduces a measurement error, thus occasional losses can be missed. The method proposed in this paper counts all the packets of the measured flow, avoiding the sampling error.

Methods leveraging packet counters have been proposed [38], [39], particularly in the contest of high speed Data Center Networks (DCNs). `PcktLoss` [38] measures packet losses by comparing counters in different parts of the network only when the considered flow is expired. A flow is considered expired if no packet has arrived for that particular flow within a specified timeout. With this method it is not possible to explicitly tune the measurement interval. Moreover, it cannot be applied to multipoint flows that, in general, can be always active, since they can correspond to different end-to-end connections.

Other solutions [39]–[41] leverage the different nature of DCNs, due to the different type of network equipments used, the limited geografic distribution and the availability of cheap and large bandwidth. These methods are based on specialized data structures (Invertible Bloom Filter, CM-Sketch) to reduce the memory required to monitor a large number of flows. They benefit from the introduction of data plane programmability [42] on new network devices [43]. As an example, `LossRadar` [39] is a loss detection system for DCNs which leverages programmable dataplane devices to deploy counters to collect data on the forwarded packets. These data are aggregated in digests that are sent to an external collector. The collector correlates the digests to measure packet losses and the affected flows. Our solution, while it can be easily implemented on programmable switches, is tailored for backbone networks, which are usually characterized by heterogeneous, legacy devices and high bandwidth cost. As a result, we reduce the amount of collected data to only values of packet counters, we consider that not all the devices in the network can deploy such counters and we exploit functionalities already available in widely deployed network devices.

VII. Conclusion

This paper has presented a novel method to passively monitor backbone networks. One chief contribution is the ability to monitor multipoint flows including packets from different sources and to different destinations, following several paths in the network. By deploying counters commonly available on existing hardware, it is possible to accurately measure packet losses within the monitored network. If the network devices are synchronized (using commodity equipment) it is also possible to measure average OWD for multipoint flows.

We introduced a technique to model the monitored network considering the location of the measurement points. This model can be used by the proposed clustering algorithm to extract the smallest subnetworks where performance can be measured. Within these subnetworks, the measurement can be statistically split among the links and devices, to find the cause of a fault, driving the maintenance process.

The presented experiments validated the method and showed that the model grows linearly with the size of the physical network. We determined that, in a real scenario, clusters present recurrent patterns and that, with high probability, they are simple one-hop networks. By distributing the loss probability among the nodes of the cluster, the identification of the causes of the losses becomes a simple task. The experiments showed also that a random distribution of the measurement points usually generates small clusters, and thus a more complex distribution algorithm would provide limited benefits. Such algorithm and an evaluation of the benefits will be subject of future work. Moreover, the distribution of MPs can be driven by specific policies, e.g., in networks spanning multiple ASes, it can be useful for troubleshooting to have MPs at the AS borders.

The considerably greater complexity of Algorithm 3 compared to the clustering algorithms suggests that future work should consider it for possible improvements. One possible solution would be an incremental version of this algorithm. Starting from a distribution plan for the counters, a reasonable strategy is to determine the clusters and then move or add only a small number of counters to reduce the size of the large clusters. An incremental version of the algorithm could leverage the previous execution and reuse the partial results of the computation that are not affected by the new changes.

We are currently working, within the IETF IPPM Working Group, to push this method toward standardization [44].

Appendix A
Algorithm to build $\bar{A}$

To build the monitored network $\bar{G}$, we have to find all the paths in G between two monitored nodes $i \in \bar{N}$ and $j \in \bar{N}$. These paths must not contain inner nodes in $\bar{N}$. Since the number of simple paths in G is $O(n!)$, we cannot iterate on all the paths and ignore the ones with inner nodes in $\bar{N}$. Instead, for each pair of nodes $i \in \bar{N}$, $j \in \bar{N}$ we find the shortest path in the graph G^* with nodes $N^* = N - \bar{N} + \{i, j\}$. The arcs of G^* are all the arcs in A connecting nodes in N^*. As a result, the shortest path between $i \in \bar{N}$ and $j \in \bar{N}$ (if exists) does not traverse other nodes in $\bar{N}$.

This solution, detailed in Algorithm 3, has complexity polynomial in m,n and $\bar{n}$. First, for each node in $\bar{N}$ we save the incoming and outgoing arcs in A. We also initialize $N^*_{init} \leftarrow N - \bar{N}$ and A^*_{init} with all the arcs in A connecting nodes in N^*_{init}. Then for each pair $i \in \bar{N}$, $j \in \bar{N}$ we build G^* by adding the pair to N^*_{init} and by adding to A^*_{init} all the arcs incident to nodes in N^*. Finally, we add (i, j) to $\bar{A}$ if one path between them exists in G^*.

Algorithm 3: Algorithm to build $\bar{A}$

```
Input: $N$, $A$ and $\bar{N}$
Output: $\bar{A}$
$\bar{A} \leftarrow \emptyset$
$N^*_{init} \leftarrow N - \bar{N}$
$A^*_{init} \leftarrow \emptyset$
▷ Save all the arcs in $A$ with at least a node of $\bar{N}$ at one end.
forall $(i, j) \in A$ do
    if $i \in \bar{N}$ then
        $out\_arcs[i] \leftarrow out\_arcs[i] \cup \{(i, j)\}$
    if $j \in \bar{N}$ then
        $in\_arcs[j] \leftarrow in\_arcs[j] \cup \{(i, j)\}$
    if $i \in N^*_{init}$ and $j \in N^*_{init}$ then        ▷ Build $A^*_{init}$
        $A^*_{init} \leftarrow A^*_{init} \cup \{(i, j)\}$
forall $i \in \bar{N}$ do
    forall $j \in \bar{N}$ do
        if $i \neq j$ then
            $N^* \leftarrow N^*_{init} \cup \{i, j\}$
            $A^* \leftarrow A^*_{init}$
            forall $(i', j') \in in\_arcs[i]$ do
                if $i' \notin \bar{N}$ or $i' = j$ then
                    $A^* \leftarrow A^* \cup \{(i', j')\}$
            forall $(i', j') \in out\_arcs[i]$ do
                if $j' \notin \bar{N}$ or $j' = j$ then
                    $A^* \leftarrow A^* \cup \{(i', j')\}$
            forall $(i', j') \in in\_arcs[j]$ do
                if $i' \notin \bar{N}$ or $i' = i$ then
                    $A^* \leftarrow A^* \cup \{(i', j')\}$
            forall $(i', j') \in out\_arcs[j]$ do
                if $j' \notin \bar{N}$ or $j' = i$ then
                    $A^* \leftarrow A^* \cup \{(i', j')\}$
            $G^* \leftarrow (N^*, A^*)$
            if $\exists$ $shortest\_path(G^*, i, j)$ then
                $\bar{A} \leftarrow \bar{A} \cup \{(i, j)\}$
```

References

[1] A. Scarfò, "The evolution of data center networking technologies," in *CCP*. IEEE, 2011.

[2] B. Claise, B. Trammell, and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information," IETF RFC 7011, 2013.

[3] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow monitoring explained: From packet capture to data analysis with netflow and ipfix," *IEEE Communications Surveys & Tutorials*, 2014.

[4] N. G. Duffield and M. Grossglauser, "Trajectory sampling for direct traffic observation," *IEEE/ACM Transactions on Networking*, 2001.

[5] G. Fioccola, A. Capello, M. Cociglio, L. Castaldelli, M. Chen, L. Zheng, G. Mirsky, and T. Mizrahi, "Alternate-Marking Method for Passive and Hybrid Performance Monitoring," IETF RFC 8321, 2018.

[6] Synergy Systems LLC, Motorola M12+Timing Starter kit. http://www.synergy-gps.com/. [Online; accessed 29-March-2018].

[7] T. Mizrahi, G. Navon, G. Fioccola, M. Cociglio, M. Chen, and G. Mirsky, "AM-PM: Efficient Network Telemetry using Alternate Marking," *IEEE Network*, 2019.

[8] C. Shannon, D. Moore, and K. C. Claffy, "Beyond folklore: observations on fragmented traffic," *IEEE/ACM Transactions on Networking*, 2002.

[9] S. Robert, *Algorithms in C, Part 5: Graph Algorithms*. Reading, MA: Addison-Wesley, 2002.

[10] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM (JACM)*, 1987.

[11] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993, ch. 2.3.

[12] Implementing Quality of Service Policies with DSCP. https://www.cisco.com/c/en/us/support/docs/quality-of-service-qos/qos-packet-marking/10103-dscpvalues.html. [Online; accessed 31-March-2019].

[13] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet Topology Zoo," *IEEE Journal on Selected Areas in Communications*, 2011.

[14] GEANT topology map. https://www.geant.org/Networks/Pan-European_

network/Pages/GEANT_topology_map.aspx. [Online; accessed 4-May-2019].

[15] BICS Global Network Infrastructure. https://bics.com/our-network/. [Online; accessed 4-May-2019].

[16] D. Rossi and G. Rossini, "On sizing ccn content stores by exploiting topological information," in *INFOCOM Workshops*. IEEE, 2012.

[17] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, "Achieving sub-second igp convergence in large ip networks," *ACM SIGCOMM Computer Communication Review*, 2005.

[18] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *HotNets*. ACM, 2010.

[19] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The Design and Implementation of Open vSwitch," in *NSDI*. Usenix, 2015.

[20] [ovs-discuss] Issue with modify rule. https://mail.openvswitch.org/pipermail/ovs-discuss/2017-December/045865.html. [Online; accessed 29-March-2018].

[21] Colt Technology Services. https://www.colt.net/. [Online; accessed 29-March-2018].

[22] Cogent Communications. http://www.cogentco.com. [Online; accessed 29-March-2018].

[23] NetworkX library. networkx.github.io. [Online; accessed 29-March-2018].

[24] N. Jasika, N. Alispahic, A. Elma, K. Ilvana, L. Elma, and N. Nosovic, "Dijkstra's shortest path algorithm serial and parallel execution performance analysis," in *MIPRO*. IEEE, 2012.

[25] Graph-tool performance comparison. https://graph-tool.skewed.de/performance. [Online; accessed 29-March-2018].

[26] J. Sommers, P. Barford, N. Duffield, and A. Ron, "Improving accuracy in end-to-end packet loss measurement," *ACM SIGCOMM Computer Communication Review*, 2005.

[27] ——, "Accurate and efficient SLA compliance monitoring," *ACM SIGCOMM Computer Communication Review*, 2007.

[28] A. Adams, J. Mahdavi, M. Mathis, and V. Paxson, "Creating a scalable architecture for internet measurement," *IEEE Network*, 1998.

[29] S. Savage, "Sting: A TCP-based Network Measurement Tool," in *Symposium on Internet Technologies and Systems*. USENIX, 1999.

[30] J. Fabini and A. Morton, "Advanced Stream and Sampling Framework for IP Performance Metrics," IETF RFC 7312, 2014.

[31] G. Almes, M. Zekauskas, S. Kalidindi, and A. Morton, "A One-Way Delay Metric for IP Performance Metrics," IETF RFC 7679, 2016.

[32] ——, "A One-Way Packet Loss Metric for IP Performance Metrics," IETF RFC 7680, 2016.

[33] R. W. Wolff, "Poisson Arrivals See Time Averages," *Operations Research*, 1982.

[34] Y. Zhang and N. Duffield, "On the constancy of Internet path properties," in *SIGCOMM*. ACM, 2001.

[35] M. Mellia, R. Lo Cigno, and F. Neri, "Measuring IP and TCP behavior on edge nodes with Tstat," *Computer Networks*, 2005.

[36] A. Sapio, M. Baldi, F. Risso, N. Anand, and A. Nucci, "Packet Capture and Analysis on MEDINA, A Massively Distributed Network Data Caching Platform," *Parallel Processing Letters*, 2017.

[37] P. Benko and A. Veres, "A passive method for estimating end-to-end TCP packet loss," in *GLOBECOM*. IEEE, 2002.

[38] A. Friedl, S. Ubik, A. Kapravelos, M. Polychronakis, and E. P. Markatos, "Realistic passive packet loss measurement for high-speed networks," in *TMA*. Springer, 2009.

[39] Y. Li, R. Miao, C. Kim, and M. Yu, "LossRadar: Fast detection of lost packets in data center networks," in *CoNEXT*. ACM, 2016.

[40] ——, "Flowradar: a better netflow for data centers," in *NSDI*, 2016.

[41] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic sketch: Adaptive and fast network-wide measurements," in *SIGCOMM*. ACM, 2018.

[42] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," *ACM SIGCOMM Computer Communication Review*, 2013.

[43] Tofino. https://barefootnetworks.com/products/brief-tofino/. [Online; accessed 4-May-2019].

[44] G. Fioccola, M. Cociglio, A. Sapio, and R. Sisto, "Multipoint Alternate Marking method for passive and hybrid performance monitoring," IETF Internet-Draft draft-fioccola-ippm-multipoint-alt-mark-02, 2018.

Mauro Cociglio graduated in Computer Science in 1988 at the University of Torino. Since 1990 he has been working at CSELT (now TIM Innovation department), the research centre of TIM, following the packet networks evolution. Since 2005 his main reaserch interests have been in the area of performance monitoring on IP based networks. In 2008, with his team, he invented the Alternate Marking methodology (IETF RFC 8321) and on this issue and other topics he has received 10 international patents.

Giuseppe Fioccola received the M.Sc. Degree in Electronic Engineering in 2008 at the Università di Napoli Federico II. He attended in 2009 a Postgraduate Master Course in Telecommunication Engineering at the Politecnico di Torino in partnership with TIM. Since 2010 he has been working for TIM in the Innovation Department as a Research Network Engineer. Most of his activities have been in the area of IP and MPLS Networking, SDN Technologies and OAM (Operations, Administration and Management). In 2015 he received an award as Youngest Serial Inventor from TIM. He is currently the TIM Delegate at IETF and is an active Contributor with several published RFCs and drafts.

Guido Marchetto is an associate professor at the Department of Control and Computer Engineering of Politecnico di Torino. He got his Ph.D. in Computer Engineering in April 2008 from Politecnico di Torino. His research topics cover distributed systems and formal verification of systems and protocols. His interests also include network protocols and network architectures.

Amedeo Sapio is a researcher and software engineer in the Cisco Data Center Switching group. His main research interests are high-speed packet processing, data plane programming, innovative network services and in-network computing. He received a M.Sc. degree cum laude and a Ph.D. in Computer Engineering at Politecnico di Torino. He has been visiting researcher at Narus, Inc. (California) and a postdoctoral researcher at KAUST (Saudi Arabia).

Riccardo Sisto graduated in Electronic Engineering in 1987, and received a Ph.D degree in Computer Engineering in 1992, both from Politecnico di Torino. Since 1991 he has been working at Politecnico di Torino, in the Computer Engineering Department, first as a researcher, then as an associate professor and, since 2004, as a full professor of computer engineering. Since the beginning of his scientific activity, his main research interests have been in the area of formal methods, applied to software engineering, communication protocol engineering, computer security, and computer networks. On these and related topics he has authored and co-authored more than 100 scientific papers. Riccardo Sisto has been a member of the ACM since 1999. He is currently Senior Member of the ACM.