Self-Adaptive Decentralized Monitoring in Software-Defined Networks

Gioacchino Tangari, Daphne Tuncer, Marinos Charalambides, Yuanshunle Qi, and George Pavlou

Abstract—**The Software-Defined Networking (SDN) paradigm can allow network management solutions to automatically and frequently reconfigure network resources. When developing SDN-based management architectures, it is of paramount importance to design a monitoring system that can provide timely and consistent updates to heterogeneous management applications. To support such applications operating with low latency requirements, the monitoring system should scale with increasing network size and provide precise network views with minimum overhead on the available resources. In this paper we present a novel, self-adaptive, decentralized framework for resource monitoring in SDN. Our framework enables accurate statistics to be collected with limited burden on the network resources. This is realized through a self-tuning, adaptive monitoring mechanism that automatically adjusts its settings based on the traffic dynamics. We evaluate our proposal based on a realistic use case scenario, where a content distribution service and an on-demand gaming platform are deployed within an ISP network. The results show that reduced monitoring latencies are obtained with the proposed framework, thus enabling shorter reconfiguration control loops. In addition, the proposed adaptive monitoring method achieves significant gain in terms of monitoring overhead, while preserving the performance of the services considered.**

Index Terms—**Network monitoring, Software-Defined Networks, Self-adaptation.**

I. Introduction

Efficient resource monitoring is a fundamental requirement for any network management system. Accurate and timely updates are needed to support resource reconfigurations and to warrant precision when troubleshooting failures or detecting anomalies. Over the last few years existing practices in network management have been challenged by the advent of Software Defined Networks (SDNs). SDN technologies have emerged as promising solutions to improve and simplify the operator's tasks [2] as they enable the development of applications that reconfigure the network automatically [4][5]. These advances pose new requirements on the monitoring functionality especially in terms of measurement frequency and information granularity.

Recent research on SDN monitoring focused on the implementation of task-specific measurements [14][15] and investigated how to allocate finite hardware resources (in particular switch memory) to different monitoring operations [16][17][6] under heterogeneous traffic workloads and for different operator's objectives. These monitoring solutions can however fall short in supporting management applications with short latency requirements [8][12].

First, these approaches mainly rely on the assumption of a centrally-managed network, which is an important limiting factor for the case of large-scale networks, *i.e.,* with a large number of geographically dispersed nodes. As the network diameter grows, the associated latencies can become considerable, penalizing the responsiveness of the network management system. Also, monitoring a large number of nodes can generate unsustainable loads on the central controller/manager due to the increasing amount of measurement traffic converging to it, with the effect of inflating the network configuration times [7]. Decentralized solutions for SDN have been proposed in the literature, *e.g.,* [7][11][10], but their main focus is on the control plane (*i.e.,* routing functionality), devoting less attention to monitoring, which is reduced to periodically synchronizing topology databases. In addition, monitoring solutions for SDN generally extract information from the network devices based on regular measurement intervals, *e.g.,* based on a fixed switch query period. As such, they can fail in detecting short-lived network episodes, especially if the switch query rate is too low, or can saturate the switch control bandwidth when measurements are too frequent. Although adaptive SDN monitoring approaches exist in the literature [17][26][18], they all require some complex parameter tuning and need continuous adjustments under dynamic traffic patterns.

In this paper we extend our previous work presented in [1] in which a decentralized monitoring architecture for SDN was proposed to satisfy the monitoring needs of large-scale networks. Our solution was designed to overcome the limitations of existing distributed monitoring frameworks which are not directly applicable to the domain of software-defined networks due to *i)* the shift towards new measurement enablers, and *ii)* the monitoring requirements of applications that reconfigure the network at a faster pace and at a finer granularity (*e.g.,* up to a single TCP flow).

Compared to [1], we provide a comprehensive description of the proposed modular monitoring architecture and complement it with new functionality. In particular, we introduce a configurable interface for the synchronization of monitoring data between *local managers* operating within a distributed management environment. We show how our architecture can support the monitoring requirements of a wide range of management applications and effectively aggregate measurement tasks to reduce the amount of resources consumed at the switches.

This research was partly funded by the EPSRC KCN project (EP/L026120/1). The work of Daphne Tuncer is supported by the Imperial College Research Fellowship Scheme.

Gioacchino Tangari, Marinos Charalambides, Yuanshunle Qi, and George Pavlou are with the Department of Electronic and Electrical Engineering, University College London, London WC1E 7JE, UK. Daphne Tuncer is with the Department of Computing, Imperial College London, London SW7 2AZ, UK.

To enable efficient extraction of monitoring information, we also extend our previous work by proposing SAM (*Self-tuning Adaptive Monitoring*), a novel adaptive monitoring method that guarantees timely and accurate reconfigurations of the switch query rate. As opposed to previous solutions, such as [26] and [17], SAM requires minimal tuning effort as the algorithm parameters are automatically updated based on the evolution of the traffic shape. Through a comparative evaluation, we show that SAM always provides more predictable/reliable results in terms of both monitoring precision and resource consumption with respect to existing approaches. Moreover, we show that SAM can always match – and in some cases even outperform – the best-case accuracy of previous methods [26] [17], *i.e.,* the one obtained with the optimal parameter settings, while using less amount of resources. SAM can be used in a wide range of monitoring systems, both with a centralized or distributed structure, as long as they rely on explicit switch polling. In this paper, we focus on applying SAM to the proposed decentralized architecture, and on the performance benefits derived from its efficient switch polling scheme.

In our previous work [1], the benefits of the proposed monitoring system were investigated using a realistic use case, where a distributed management application coordinates a content distribution service in an ISP network. In this paper, we further extend the work by investigating an additional scenario, in which the operator runs an on-demand gaming service by offering processing resources (*e.g.,* specialized hardware) as part of the network infrastructure. To evaluate the performance of the decentralized monitoring approach in terms of monitoring latency, as well as traffic overhead, we compare against a centralized solution based on two realistic network topologies. In addition, we extensively evaluate the effect of monitoring operations on the two use case services by focusing on the impact of monitoring data extraction (through the use of SAM), as well as monitoring information synchronization. The results show that our decentralized monitoring approach can reduce the monitoring delays by up to 60% compared to a centralized one, which translates to more reactive control loops. They also highlight that SAM can produce significant benefits on the use case services at a reduced cost in terms of utilization of the switch resources. Finally, the evaluation shows that although relaxing the synchronization of monitoring information can impact the service performance, it is possible to achieve substantial reductions in monitoring overhead while minimizing potential service disruption.

The remainder of this paper is organized as follows. Sec. II provides background information on the distributed network management framework considered in the paper and presents the main SDN monitoring approaches. In Sec. III, we describe in detail the design of the proposed architecture. In Sec. IV, the SAM approach is presented and extensively evaluated. Sec. V describes the use case services considered for the evaluation of our solution. Experiment setup and evaluation results are presented in Sec. VI. Sec. VII describes related work and Sec. VIII concludes the paper.

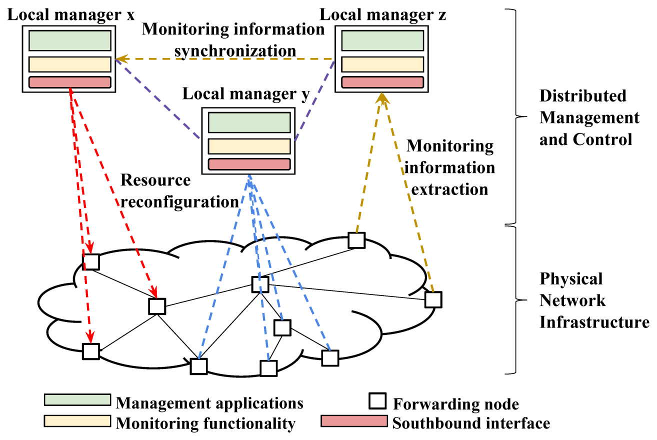


Fig. 1: Distributed resource management framework.

II. Background

In this section, we provide background information about the SDN-based resource management framework considered for the design of our monitoring solution, as well as an overview of the techniques used for performing measurements in SDN infrastructures.

A. Distributed Resource Management Framework

In [8], co-authors of this paper present a novel SDN-based network management and control framework that supports dynamic resource management applications in fixed backbone infrastructures. In this paper we adopt the design principles of the relevant architecture, which separates management and control functionality, allowing the two to evolve independently. A set of *local managers* (LMs), distributed over the network, hosts various management applications (MAs) that implement the necessary logic to decide on network (re)configurations. MAs are instantiated on the local managers as modules embedding information data structures and running on a common execution environment offered by the LMs. Each MA can execute in all LMs or in a subset of them (e.g. ones operating at edge network nodes). Configuration decisions taken by LMs are translated into sets of commands, transmitted to the forwarding hardware through a *southbound* interface (*e.g., OpenFlow*), which defines the sequence of actions to be enforced for updating the network parameters.

Monitoring is an essential component of LMs. First, it is concerned with extracting raw statistics from the physical resources and generating useful information for applications. In this context, each LM needs to implement the necessary capabilities to collect the status of variables (*e.g.,* links, traffic flows) within its local scope and make this information available to local MA instances. Second, since MA instances operating at different locations may need monitoring data gathered from outside their local scope, the monitoring functionality is also concerned with disseminating network state updates to remote LMs. In a SDN environment such a synchronization phase is essential for reconfiguring the network parameters based on a global, unified network view. This information can be exchanged between instances of a distributed MA through the signaling framework proposed in [36], which provides a communication protocol and the necessary primitives to share the monitoring information and to coordinate decisions between two or more MA instances.

Fig. 1 depicts a simplified representation of the resource management framework considered in this paper. The forwarding nodes are partitioned in clusters, each under the control of a LM. The monitoring functionality initially retrieves raw data from the forwarding nodes, which is subsequently processed (*e.g.,* filtered, aggregated) to form knowledge, and is made available to local MA instances, *i.e.,* the ones operating on the same LM. In addition, a subset of the generated knowledge can be shared with a remote manager (*e.g.,* between *LM z* and *LM x* in Fig. 1) on a communication channel established using the signaling protocol in [36]. Using the synchronized information, a remote manager (*LM x* in Fig. 1) can reconfigure its own partition of the network infrastructure by interacting with the forwarding nodes in the cluster.

B. Monitoring Software-Defined Networks

Compared to traditional computer networks, where monitoring solutions require ad-hoc software installation / configuration and low-level tools, SDN has introduced a set of simple and reusable primitives for the collection of network variables at different granularity levels, which make them suitable to a wide range of management tasks. SDN flow-based switches (*e.g.,* OpenFlow enabled devices) allow network operators to flexibly specify the flows to monitor based on different packet fields (*e.g.,* source and/or destination IP addresses), and to count the number of bytes or packets for these flows. Counters are fetched by polling a switch using ad-hoc *Read State* messages. The switch flow rules can be adapted or replaced depending on the analysis performed on the corresponding counters or according to predefined expiration timeouts.

This measurement approach is affected by several hardware technology issues. First, flow-based counters are maintained in expensive and power-hungry TCAMs and, as such, only a limited number of entries can be used for measurements. Another issue is the limited bandwidth between the switch and the SDN controller, which limits flow fetching to no more than a few thousand per second [30]. Finally, SDN-enabled switches may also exhibit inaccuracies when updating the flow counters. For example, as discussed in [33], some devices do not update the counters every time a new packet matches a rule, but perform the updates periodically instead. Furthermore, devices from different vendors introduce different biases in measurements and may even present some limitations in terms of protocol support. Despite these open issues, our proposal relies on the *counting* approach – *i.e.,* polling the network devices for raw counters – due to its implementation simplicity, the wide support by different vendors, and configuration flexibility in terms of information granularity and measurement frequency. Alternative methods, which implement hashing techniques (*e.g., sketches*) on the network hardware [21], or require enhanced programmability (*i.e.,* beyond OpenFlow) of the forwarding plane [22] [23] [24], still have very limited support on devices, which makes their applicability uncertain. At the same time, solutions based on stream processing [40] [39] can pose a much higher processing burden on local managers, *e.g.,* in the case of large-scale networks with a limited number of LMs, and are unsuitable for many management applications due to the adoption of packet sampling [31].

III. System Architecture

This section presents the proposed monitoring system and motivates the design principles of the associated architecture. Our solution leverages a decentralized approach where each of the local managers described in Sec. II hosts a monitoring entity, called the *monitoring module* (MM), which is responsible for gathering information within the scope of the LM. Scalability for coping with a large number of network devices and their geographical span was the main driver for selecting a distributed approach.

A. Design Requirements

Effective design of distributed monitoring functionality has to take into account a number of key issues. If very intrusive, monitoring operations can adversely affect the network performance. At the same time, these operations need to be frequent and fast to enable management applications to operate at short timescales. In addition, they should provide accurate and high-granularity information to support configuration decisions. The impact of these issues is amplified in the case of large-scale SDNs, since configuration decisions might be taken far away from the locations where monitoring is performed. We identify below the three main requirements that have been taken into account for the design of the proposed monitoring approach.

Scalability: The monitoring system should be able to cope with a large number of information sources. As the number of physical resources under the scope of a single MM increases, the monitoring traffic converging to it, as well as the associated computational load, could drastically impact the system reactivity, as was shown in [10][30]. While in dense networks with small diameter (*e.g.,* data centers) this drawback can be mitigated through replication or by investing more CPU cycles and memory, in wide area networks (WANs) the monitoring responsiveness is significantly affected by network latencies. Based on the same motivation for distributing the SDN control plane [7], *i.e.,* switch-to-controller latency reduction, we consider a decentralized monitoring solution for reducing monitoring delays and avoiding processing bottlenecks.

Programmability: The frequency and granularity of measurements have to be highly configurable based on the requirements of heterogeneous management applications. While some applications, such as elephant-flow detection, need fine-grained flow-based measurements, others only require aggregate statistics. In such a case, low-granularity measurements, which can be retrieved at a lower cost are preferable (*e.g.,* switch port measurements as opposed to individual flow measurements).

Responsiveness: MAs can change their monitoring requirements based, for example, on the analysis of measured metrics. The MM should be responsive in adapting measurement parameters, such as the polling frequency or the flow-level granularity, according to new requirements. Fast adaptations, as argued in [26][17], are essential for warranting acceptable information accuracy and can additionally reduce the monitoring overhead.

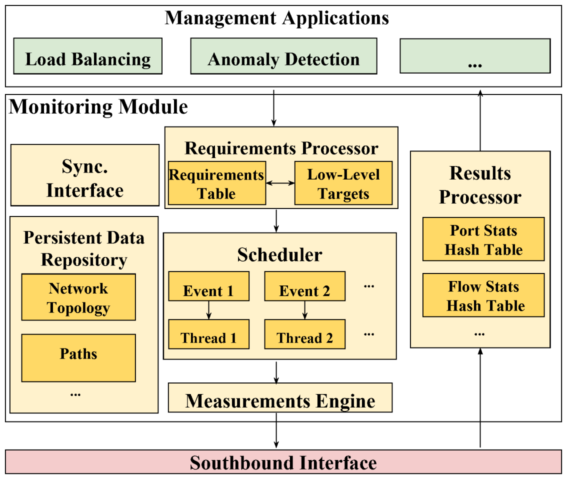


Fig. 2: Monitoring module architecture.

B. Monitoring Module

Fig. 2 presents the architecture of the MM, which sits between MA instances and the southbound interface. Applications use a common interface, *e.g.,* a RESTful interface, offered by the MM for both injecting new monitoring requirements and receiving the corresponding measurement results.

Each MM relies on a modular composition to maximize the system extensibility and improve the overall flexibility of the solution. The modular structure allows to decouple the logic involved in the processing of the application requirements from the one operating on the raw measurement primitives. This reduces the deployment effort when new types of requirements need to be supported, or new measurement mechanisms become available. The various components of the MM are described below.

1) Persistent Data Repository This component maintains network information which is not updated frequently, such as the topology graph representation (*e.g.,* switches and links) and the current setup of paths between pairs of edge nodes. Such information can be represented through transactional databases and can be flexibly accessed/modified by a SQL-like querying mechanism.

2) Requirements Processor The first task of this component is to parse new monitoring requirements received from applications. These are registered in a local data store (*Requirements Table*, *c.f.* Fig. 2), with each requirement represented as a tuple:

⟨ `Req_id, MA_id, Task, HL_targets, Mon_times` ⟩

Req_id and *MA_id* are the unique identifiers of the monitoring requirement and the requesting management application. *Task* represents the overall goal of the measurements, for example the utilization of one or a set of links. *HL_targets* is the list of targets (high-level identifiers in the application's abstract view of the network) of the monitoring task, for instance, in case of a path utilization request, the corresponding list of paths. *Mon_times* can be a single parameter, *i.e.,* the polling period, or an explicit sequence of measurement intervals. In Sec. IV, we present an adaptive polling mechanism (*SAM*) where the measurement rate is continuously adjusted based on the network traffic behaviour. When this mechanism is enabled, the MA instance only needs to provide, under the attribute *Mon_times*, the boundaries for the measurement rate, *i.e.,* the interval of acceptable monitoring frequencies $[f_{min}, f_{max}]$.

The next procedure performed by this component is a translation routine, based on the *Task* specification, that maps each new table entry into one or more *Low-level targets*. Each low-level target (*LL_target*) can identify a specific physical resource, *e.g.,* a switch port, or map a set of flow rules, *i.e.,* a specific subset of the switch flow table. These entities are stored in the *Low-level targets* table with the following format:

⟨ `LL_target, Op_type, [Req_id], Sched_state` ⟩

Op_type indicates what type of measurement operation should be performed, for example collecting the average traffic rate of a specific switch interface. [*Req_id*] is the list of pointers to the corresponding application requirements, used for the reverse translation. *Sched_state* is a flag indicating whether the low-level target refers to a new monitoring requirement (*i.e.,* measurement operations have to be scheduled from scratch), or to a previous task for which some adaptation is required (*i.e.,* operations have to be re-scheduled).

The acquisition of statistics from a switch poses a substantial burden on the device in terms of both processing and control bandwidth [30] [33]. As a result, for each time unit only a limited amount of statistics can be reported by the switch to the MM. Any data exceeding the limit can be lost or considerably delayed, which penalizes the accuracy of MA operations. To mitigate this issue, our solution aggregates different monitoring tasks, when possible. In other words, before the insertion of a new low-level target, the table is looked up for similar entries. An existing entry is *similar* if the target is equivalent or included, *e.g.,* two targets with the same *Task* and *Mon_times* attributes are similar if one refers to the flows matching source IP address 128.40.200.1 and the other corresponds to the flows for any source IP in the subnet 128.40.200.0/24. In such a case, the MM merges the two low-level targets and the corresponding measurement times is updated accordingly to satisfy both requests. Such a feature reduces the consumption of switch resources, especially for different MAs requiring flow measurements at similar times, and/or for similar portions of the switch flow space. The result is a more sustainable monitoring (reduced monitoring load) when measurements compete with other control plane operations, such as new flows setup, for accessing the switch resources.

Resource saving through aggregation. To show the gain that can be achieved through aggregation, we consider a scenario where 3 monitoring requirements m_1, m_2, m_3 are registered at the same time and on the same MM by three different MAs. The execution of the measurements associated with each m_i results in fetching a fixed number of flow table entries k at a period $p_i \in [1, 2, ..7, 8]$ from the same switch. Two parameters α and β are associated with each set. Parameter α represents the level of temporal concurrency of the required measurements, which ranges from 0 (lowest level of concurrency, *e.g.,* $[p_1, p_2, p_3] = [6, 7, 8]$) to 4 (maximum level of concurrency, *i.e.,* $p_1 = p_2 = p_3$). Parameter β

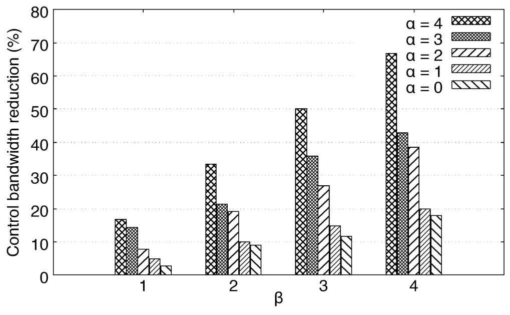


Fig. 3: Reduction of switch control bandwidth based on aggregation.

represents the level of overlap in terms of similar flow entries required from the switch, and also ranges from 0 to 4. $\beta = 0$ represents the case of no overlap. For $\beta = 1$, there exists an overlap of 50% between the requirements of two MAs. For $\beta = 2$, the three MAs share 50% of requests. For $\beta = 3$, two MAs have completely overlapping requirements and share 50% with the third one. $\beta = 4$ represents an overlap of 100% between the three MAs. Fig. 3 shows the resulting gain in terms of switch control bandwidth. The case $\beta = 0$ is not depicted as it does not result to any savings.

Significant reductions can be achieved when $\beta \geq 2$. For example, for $\beta = 2$, an average reduction close to 20% is obtained. Such savings can allow the overall measurement rate to increase, thus enhancing the resource reconfiguration reactivity. For instance, assuming a fully saturated bandwidth between the switch CPU and the local manager, aggregation can allow an increase of the monitoring rate by up to a factor of 3 (*i.e.,* $\alpha = \beta = 4$).

3) Scheduler This component is in charge of generating and managing the individual measurement procedures (*e.g.,* the ones requiring a single message exchange with a switch), which are executed as threads. It is called on every insertion in the *Low-level targets* table, and on any modification of existing tuples involving the measurement times. Scheduling new measurements indiscriminately can lead to none of them getting enough switch resources. As such, once invoked, the scheduler executes an admission control routine, in which it verifies, depending on the current measurement load, whether the measurement procedures for the new low-level target can be performed. In case there are not enough resources available to accommodate the new measurement procedures, these are rejected and the corresponding MAs are notified so that monitoring requirements can be re-negotiated. The measurement load for a specific switch is defined by the expected monitoring bandwidth, which is estimated on a time-window basis given the list of the low-level targets already scheduled. This metric depends on the current measurement rates (*i.e.,* the average polling frequency) and on the number of flow (or switch port) records returned for each measurement procedure in the corresponding OpenFlow *Statistics Reply* messages. In this respect, our approach differs from recent proposals, which focus on the limited flow table TCAM space [16] rather than the control bandwidth. The amount of resources required by an incoming measurement procedure is quantified using the *Mon_times* indication from the monitoring requirements. If the adaptive polling mechanism presented in Sec. IV (*SAM*) is enabled, maximum and minimum resource requirements are obtained using the query rate boundaries f_{min}, f_{max}.

Once accepted, the new low-level target is mapped onto a set of events, each one associated with a timer to trigger the new measurement thread. The way this mapping is executed depends on the switch polling algorithm used by the MM. In case of fixed frequency measurements, the mapping is executed all at once, and all timers are recorded in the scheduling table. When *SAM* is enabled, the measurement events and timers are generated one by one, based on the feedback (*e.g.,* the new value of a low-level target) provided by the *Results Processor* upon receiving fresh monitoring data.

4) Measurements Engine This component, called every time a new measurement thread is triggered, operates as an interface between the measurement thread under execution and the measurement mechanisms implemented on the southbound interface. It assigns individual measurement procedures to one of the available primitives offered on the interface and supported by the underlying device. Such an interface is essential to allow most of the monitoring operations to remain independent of specific implementations.

5) Results Processor This component receives the raw measurement results, for example messages of type OpenFlow statistic reply. These are parsed (*e.g.,* into JSON format) and the *Low-level targets* table is looked up for the corresponding target(s). Based on the operation type specified in matching table entries, the measurements are filtered to select the required counters. These are stored in corresponding data structures (hash-tables) and used for computing the metrics of interest. Finally, the processed results are associated to the relevant high-level targets and delivered to MAs through update messages. In case the MM adopts an adaptive monitoring scheme like the one presented in Sec. IV, the Results Processor also generates a call to the Scheduler, so that new measurement events can be adaptively scheduled based on variations of the relevant metrics over time.

6) Synchronization Interface In addition to the aforementioned components, the MM offers an extensible interface for the synchronization of monitoring information in a distributed management plane. Using the methods of this interface, instances of MAs can forward monitoring reports through signaling channels as described in [36]. The interface provides different solutions for the exchange of monitoring data. In the simplest case, this can take place periodically or every time new statistics are available locally. More advanced solutions consist in exchanging new values only when they differ substantially from the previously reported ones. These techniques can improve the synchronization efficiency, as they allow management applications to strike the right tradeoff between accuracy and overhead in monitoring data dissemination.

IV. Adaptive query mechanism

Querying a network switch for updated statistics involves a tradeoff between accuracy and overhead. On one hand, continuous pulls of fresh statistics impose a considerable burden on the switch hardware, *e.g.,* on its scarce control channel bandwidth [30], and can overflow the network capacity with monitoring overhead. On the other hand infrequent measurements fail in capturing transient events, such as short-lived congestion, due to sampling and averaging bias. To reduce the monitoring load while ensuring timely and precise reports, adaptive monitoring mechanisms by which the query frequency is dynamically reconfigured are needed. In this section, we propose a novel approach that produces precise reconfigurations of the measurement rate, between boundaries f_{min} and f_{max}, with minimal parameter tuning effort. The use of boundaries f_{min} and f_{max} is essential as it allows the operator to maintain some control on resource utilization, thus influencing the resource/accuracy tradeoff associated with the measurements. Provided by the operator in the specification of monitoring requirements, these values are used by the *Scheduler* at run time as constraints for the generation of measurement events/threads.

A. State-of-the-Art and Limitations

Two main techniques have been proposed in the literature to adapt the switch query rate: i) threshold-based approaches [26] and ii) prediction-based approaches [17]. The objective of these approaches is to adapt the period at which switch variables are queried based on network traffic behavior. While in the case of threshold-based approaches, the adjustment is based on threshold conditions, a linear prediction of the evolution of the variable value is used in the case of prediction-based approaches to update the period. The advantage of these techniques is that they can easily apply to different monitoring operations – for instance, flow size and link utilization estimation. Other methods exist but they are bound to specific measurement tasks, *e.g.,* flow autocorrelation [18].

Threshold-based adaptive monitoring The principle of Threshold-based Adaptive Monitoring (TAM) is to adapt the monitoring period (*i.e.,* rate at which the switch is queried) based on the variation of the variable values between two consecutive measurements. If the difference is above a threshold th_1, the monitoring period is divided by a constant d, while if it is below a threshold th_2, the period is multiplied by a second constant m. In essence, the sharper the variation, the shorter the period and hence the more intense the polling.

Prediction-based adaptive monitoring In contrast to the threshold-based approach, Prediction-based Adaptive Monitoring (PAM) uses an history of the last set of collected measurements (not only the last one) to decide how to adjust the monitoring period. More specifically, based on the previous N collected values of monitored variable x, denoted as $x_1, ..., x_n$, a predictor x_p of the next value of x is computed. When the actual new value x_{n+1} of x is fetched, x mean and standard deviation are updated and the *real* variation $(x_{n+1} - x_n)$ is compared to the *predicted* one $(x_p - x_n)$. If the predicted variation is substantially higher than the real one, *i.e.,* for $x_p > mean(x) + \alpha \cdot std(x)$, the monitoring period is increased by a factor equal to d. In this case, the prediction is overestimating the change. Otherwise, if the value of x is changing much faster than predicted, *i.e.,* for $x_p < mean(x) - \alpha \cdot std(x)$, the period is divided by d. In both cases, α is a small integer constant. In all other cases, the period remains unchanged.

The main issue with TAM and PAM is that to efficiently adapt the period, they both require some complex parameter tuning, *i.e.,* thresholds th_1, th_2 and multipliers m, d for TAM, and selection of sample queue size N and factor α for PAM. This is specifically evident under periods of bursty traffic, where accurate reconfiguration of the query rate is essential. In particular, changes in the traffic burst patterns (*e.g.,* burst amplitude, duration, inter-arrivals) are likely to require new settings given that for a specific pattern, only a small subset of setups guarantees efficient statistics collection.

To illustrate this issue we implemented the two approaches to measure the size of a bursty flow over a period of 5 minutes and compared the obtained results with the actual flow size, denoted as *ground-truth*, measured every millisecond. To emulate a bursty profile, the size of the flow is modulated between 0 and 10 Mbps by injecting traffic bursts with arrivals modeled as $Poisson(0.1)$, height (in Mbps) and duration (in seconds) as $Uniform[0, 5]$. Fig. 4 shows the performance of the two approaches for different parameter setups with $th1, th2$ in $[10\%, 20\%, ..100\%]$; m, d in $[2, 4, ..10]$; N in $[2, 3, ...30]$; α in $[1, 2, 3]$. For each configuration, the accuracy is quantified using the Root Mean Square Error (RMSE) between the value collected by monitoring and the ground-truth, while the resource consumption is given by the average switch query rate. The results are shown in Fig. 4.

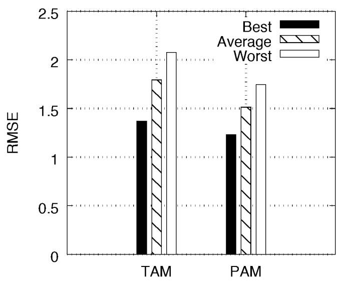
The first main observation is that, for both approaches, different setups can produce widely different outcomes in terms of precision, as depicted in Fig. 4.a. More surprisingly, Fig. 4.b shows that different parameter configurations can lead to significantly different accuracy levels even when consuming exactly the same amount of resources. For instance, with a mean polling rate 0.5 Hz, the TAM approach can either produce RMSE=1.9 or RMSE=1.5. At the same time, different resource usages can result in the same monitoring precision, *e.g.,* RMSE=1.4 can be obtained with both 1 and 2 Hz average query rate with the PAM approach. In all cases, the performance with respect to the accuracy *vs.* query rate tradeoff strictly depends on how well parameters are tuned. In practice, however, determining the optimal setups is hard. This not only requires long (preliminary) periods of traffic observation but also necessitates adjusting the setups to match emerging traffic characteristics (*e.g.,* new traffic burst patterns).

In the next subsection, we present a novel self-tuning adaptive monitoring approach that addresses the limitations of existing solutions.

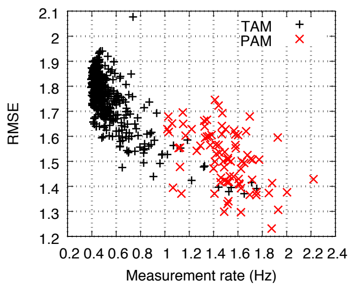
B. Self-Tuning Adaptive Monitoring

The proposed Self-tuning Adaptive Monitoring (SAM) approach allows to achieve the right tradeoff between accuracy and consumed resources, without requiring complex parameter tuning as it automatically refines the algorithm parameters based on the evolution of the traffic shape.

More specifically, the objective of the proposed solution is to continuously adapt the timeout T, *i.e.,* the time to the next measurement. In a similar fashion to prediction-based approaches [17], SAM uses linear prediction to predict the



(a) Monitoring accuracy (RMSE)



(b) Accuracy(RMSE) vs. average query rate

Fig. 4: Performance of state-of-the-art approaches.

next value x_p of a variable x based on the value of its previous measurements. When the new value x_{n+1} of x is collected, the normalized deviation D of the predicted variation $(x_p - x_n)$ from the real one $(x_{n+1} - x_n)$ is computed. When the value of D is negative, the prediction of the variation of x is underestimated. For instance $D = -0.5$ indicates that the predicted variation is 50% of the real one. In this case, the behavior of x is more dynamic than expected and T is reduced proportionally to D, so that the larger the deviation, the faster the query rate. In contrast, when D is positive, the prediction is overestimating the variation of x, *e.g.,* $D = 1.0$ corresponds to 100% overestimation. In this case x is changing less (or less quickly) than expected (the behavior of x tends to become more stable) and T is increased proportionally to D.

After each measurement, the linear prediction is automatically reconfigured by tuning the length N of the sample queue used to compute x_p based on an Additive Increase Multiplicative Decrease (AIMD) scheme. In particular, N is increased by one when $T_{new} > T$ and halved otherwise. Intuitively, the length is shortened when traffic becomes more dynamic (*e.g.,* when a traffic burst starts) so that the next decision on T only involves the most recent history of x. It is progressively expanded as the behavior of x becomes more stable.

The pseudo-code of the proposed algorithm is shown in Alg.1. It takes as input the latest timeout T and the current sample queue length N, and returns as output the time to the next measurement T_{new} and the new sample queue length N_{new}. Compared to the TAM and PAM approaches, the proposed solution requires minimal tuning effort. The only parameter needed for the algorithm setup is the initial value of N, whose impact is strictly limited to the algorithm startup phase. In addition, unlike previous approaches [26] [17] for which the switch query period is only modified when large variations of x are observed, our solution continuously adjusts the timeout T. Although this may lead to rescheduling measurement tasks more frequently (incurring thus increased burden on the monitoring module scheduler), it can prevent situations where the variations of x are undetected, e.g., when the fluctuations of x are periodic and "phase-locked" [41] to the switch polling rate (*i.e.,* same frequency but out of phase).

C. Evaluation

We evaluate the performance of SAM by implementing it to measure the bitrate (in Mbps) of traffic generated using both

Algorithm 1: COMPUTE NEXT QUERY TIMEOUT

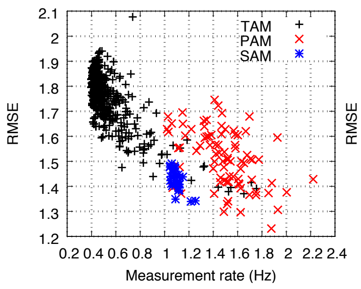
```
Input: Current timeout $T$, Current sample queue length $N$
Output: Next timeout $T_{new}$, New sample queue length $N_{new}$
1  Compute prediction $x_p$: $x_p = x_n + \frac{t_n - t_{n-1}}{n-1} \sum_{i=1}^{N-1} \frac{x_{i+1}-x_i}{t_{i+1}-t_i}$
2  Retrieve value $x_{n+1}$
3  Compute deviation of $(x_p - x_n)$ from $(x_{n+1} - x_n)$:
       $D = \frac{(x_p - x_n) - (x_{n+1} - x_n)}{x_{n+1} - x_n} = \frac{x_p - x_{n+1}}{x_{n+1} - x_n}$
4  if $D < 0$ then
5      $T_{new} = max(\frac{1}{f_{max}}, T_{old} - D \cdot T_{old})$
6  else
7      $T_{new} = min(\frac{1}{f_{min}}, T_{old} + D \cdot T_{old})$
8  Add $x_{n+1}$ to sample queue
9  Update sample queue $N_{new} = AIMD(N)$
10 return $T_{new}, N_{new}$
```

synthetic and real traffic traces and compare the results to the ones obtained with TAM and PAM. To test the performance under different traffic conditions, we use two synthetic traces with the same duration (5 minutes) and bitrate range (between 0 and 10 Mbps) but different levels of burstiness.

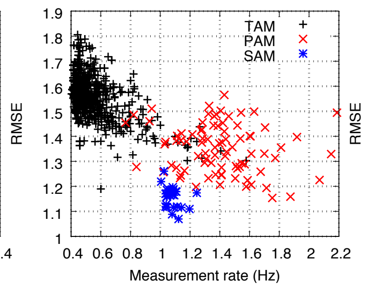
The first trace, referred to as *Highly Bursty Traffic*, emulates a bursty traffic flow with burst arrival modeled as $Poisson(0.1)$ (on average one in 10 seconds), burst height in Mbps as $Uniform[0, 10]$ and burst duration in seconds as $Uniform[0, 1]$. The second trace, referred to as *Slightly Bursty Traffic*, corresponds to a more stable traffic flow with burst arrival modeled as $Poisson(0.02)$, burst height as $Uniform[0, 5]$ and bust duration as $Uniform[1, 10]$. Compared to the first profile, bursts in the second trace are less frequent, have a longer duration and exhibit smaller variation of the traffic rate. In addition to the synthetic traces, we also use two real 15 minute traffic-packet traces from a 100 Mbps link of a Japanese operator [42], representing peak time (JP-2pm) and off-peak time (JP-2am) traffic, respectively.

For all traces we evaluate the performance based on a wide range of parameter setups. In particular, for the TAM approach we use all combinations of thresholds $th1, th2$ in $[10\%, 20\%, ..100\%]$, with $th1 > th2$, and query rate constant factors m, d in $[2, 4, ..10]$. For the PAM approach, we select the sample queue length N in $[2, 3, ...30]$ and the constant α in $[1, 2, 3]$. Finally, for the proposed self-tuning approach we vary the initial value of N in the range $[2, 3, ...30]$. The same minimum (f_{min}) and maximum (f_{max}) query rate boundaries are applied for the three approaches, with $f_{min} = 0.1Hz$ and $f_{max} = 10Hz$. In a similar fashion to the example in Fig. 4, we use the monitoring precision, given by the RMSE with respect to the *ground-truth* traffic rate, and the resource consumption, represented by the average switch query rate over each experiment run, as the two main performance indicators.

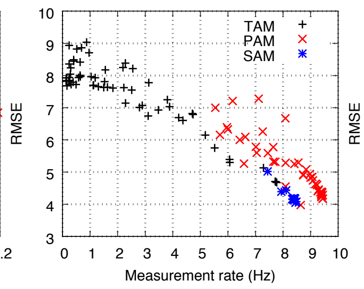
The results are depicted in Fig. 5 and 6. Fig. 5 shows that the performance of the self-tuning approach is in general more predictable in terms of precision and resource consumption



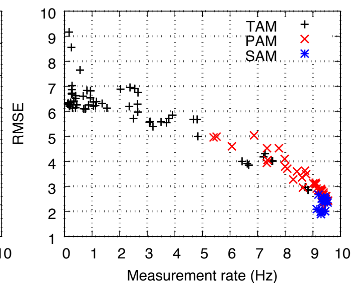
(a) Highly Bursty Traffic



(b) Slightly Bursty Traffic

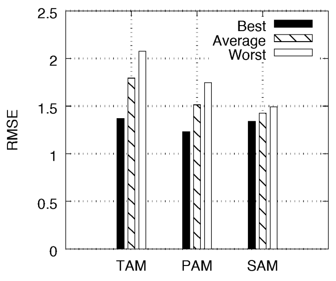


(c) JP-2pm

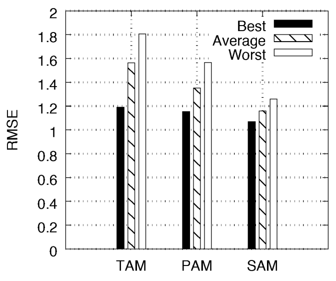


(d) JP-2am

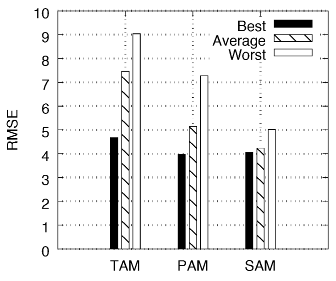
Fig. 5: Adaptive monitoring precision and resource consumption.



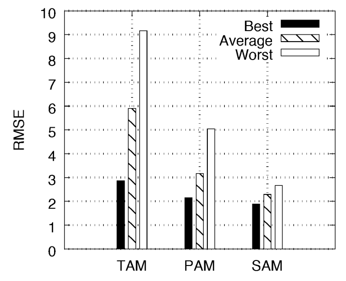
(a) Highly Bursty Traffic



(b) Slightly Bursty Traffic



(c) JP-2pm



(d) JP-2am

Fig. 6: Best-case, Worst-case and Average monitoring precision.

compared to the two state-of-the-art solutions. For example, the RMSE obtained with our approach only oscillates between 1.35 and 1.5 for the first trace (Fig. 6a), and between 1.05 and 1.25 for the second one (Fig. 6b), while for PAM the distance between the best and worst case accuracy is twice the one obtained with our algorithm, and it can even be three times greater for TAM. The same applies to the average switch query rate. For example, for the first trace (Fig. 5a) our solution queries the switch with mean rate between $1Hz$ and $1.5Hz$, while the two state-of-the-art approaches operate with an average query frequency between $0.5Hz$ and $2.5Hz$. Similar observations can be made for the experiments with real traffic profiles (Fig. 6c and Fig. 6d), where SAM approach achieves more predictable results compared to previous solutions.

Another important observation is that our algorithm can achieve a good tradeoff between accuracy and resource consumption under different traffic profiles. In particular, the average monitoring accuracy obtained by our solution generally lies close the *best-case* precision of the PAM approach, always using a lower or, in the worst case, similar amount of resources. For the Slightly Bursty Traffic trace (Fig. 5b), lower values of RMSE with respect to TAM and PAM are obtained with a reduced average query rate, while for the case of the Highly Bursty Traffic trace shown in Fig. 5a, PAM can still achieve a slightly lower error but by consuming substantially more resources (by 50% or more). In the case of the peak time real packet trace (JP-2pm – Fig. 5c), our algorithm obtains similar performance in terms of precision compared to the best results of PAM but by consuming approximately 10% less resources. Finally, in the case of the off-peak time real packet trace (JP-2am – Fig. 5d), results are in line with the most precise results obtained with PAM.

V. Use case scenario

To demonstrate the capabilities of the monitoring architecture presented in this paper, we consider a distributed SDN environment on which two different services are deployed by the network operator. The first one is a *content distribution* service, for which a set of content items is cached within the network. The network management system periodically updates (*e.g.,* in the order of hours) the content placement and the paths between user locations and content servers. Following an approach similar to the one proposed in [34], it reconfigures the routing of user requests in real-time by selecting an appropriate *path* between the user location and one of the available content servers, based on the current path utilization.

The second service provided by the operator is an on-demand gaming (*cloud gaming*) service, in a similar fashion to well-known platforms such as Gaikai [43]. The network provider offers specialized hardware resources, such as GPUs and fast memory, to support the computation required for the user game experience, which is offloaded from the end-user devices. From the network perspective, this service implies a continuous interaction between the user device and the server, where the client sends new input data, and in response it receives chunks of the video stream to be reproduced on the user device. The network management system can tune this service at run time by reconfiguring the routing of client and server traffic, which is performed by selecting a suitable path from a set of available options. While doing so, the operator objective is to avoid congestion in the network and, as such, to prevent potential Quality of Experience (QoE) degradation. Network congestion will increase the content delivery times and can lead to user dissatisfaction due to unresponsive client-

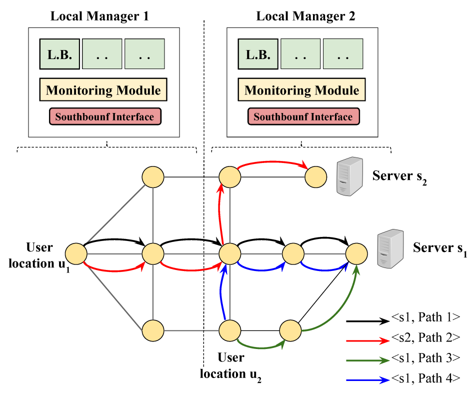


Fig. 7: Use case illustration.

server interactions in the on-demand gaming service.

When reconfigurations of the two services are generated in response to congestion episodes, the latest decisions on server and path selection are enforced using a method similar to the one proposed in [29]. The programmable (*e.g.,* OpenFlow-enabled) forwarding hardware at the edge of the network is instructed in real-time to rewrite fields of the IP packet header (*e.g.,* the destination address) in order to redirect traffic transparently to clients and servers. The enforcement of the path selection decisions is part of the header rewriting operations. For example, the path selection can be encoded in the packet ToS field.

Fig. 7 exemplifies the use case. The forwarding nodes are partitioned in clusters, each under the control of a local manager. Each manager hosts an instance of a distributed Load Balancing (LB) application, which implements the necessary logic for reconfiguring, on a per user location, the content server from which a requested content is retrieved (for content distribution) and the path through which client/server traffic is delivered (for content distribution and cloud gaming). For simplicity, we assume clients and servers to communicate on symmetric paths. For each network edge switch mapped to a user location, the application keeps a list of available setups $\langle Server, Path \rangle$ indicating how traffic should be routed. Each LB instance operates periodically on a short timescale, *i.e.,* every few seconds, and at each execution it obtains two statistics from the monitoring system.

The first statistic is the *average link utilization* for the links included in the local paths, *i.e.,* the paths emanating from a client location within the scope of the relevant LM. Statistics for these links are collected and exposed by the underlying MM at each execution of the application. The monitoring of remote links is delegated to the LB instance operating on the corresponding network partition. This registers the relevant monitoring requirements on its MM, and periodically synchronizes the results with the other LB instances.

The second statistic is the *average rate* of traffic originated by users in the local network partition. More specifically, each LB instance obtains the average throughput of all the flows matching the source IP address of one of the clients and the destination IP address of one of the servers, or vice versa. This measurement is used to determine the volume of traffic by which congested paths can be offloaded.

In case of link congestion (*e.g.,* average utilization exceeding a predefined threshold), the LB application is responsible for offloading part of the traffic from the congested link in order to bring its utilization below the threshold. Some flows are removed from the congested paths (paths including the congested link) and are (equally) assigned to alternative, non-congested, options represented by the 2-tuple $\langle Server, Path \rangle$. The new configurations are enforced on the ingress OpenFlow switches. If a congested path spans multiple network partitions, the corresponding LB instances operate iteratively, as in the solution presented in [35]. The first decision is taken by the LB instance directly associated with the congested link based on the bandwidth availability on the alternative paths. The result is then communicated to the next LB instance until the process terminates.

TABLE I: Network characteristics

	Network	# Nodes	# Bidirectional Links
Topo1	Geant [46]	23	37
Topo2	Germany50 [47]	50	88

VI. Evaluation

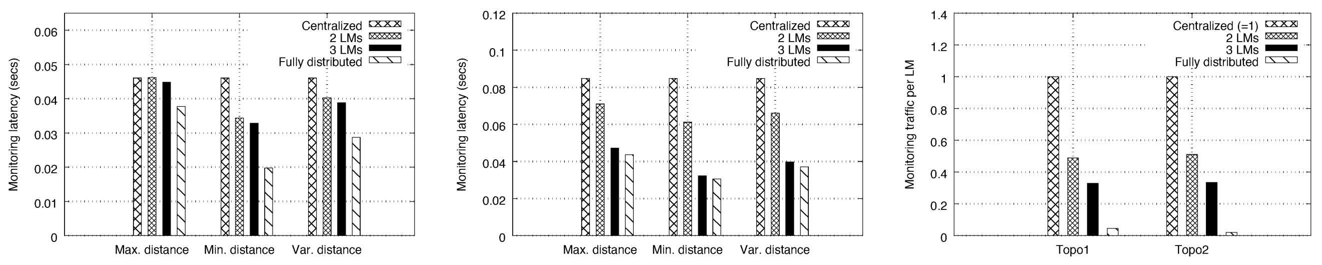
We evaluate our monitoring system based on the use case described in Sec. V and focus on the performance in terms of latency and traffic overhead, as well as on the impact on the two different services. In addition, we investigate the gain that can be achieved by applying SAM, the self-tuning adaptive monitoring solution presented in Sec. IV. Experiments are performed using *Mininet* to emulate the network topology, including hosts, *i.e.,* clients and content servers, and OpenFlow switches. The LM, including the monitoring module and the LB application logic, is implemented as a set of Python modules. Finally, we reuse a small set of APIs from the SDN controller POX [38] to implement the southbound interface functionality.

A. Experiment Setup

Experiments are performed on the two network topologies, *Topo1* and *Topo2*, summarized in Table I, where each node is an OpenFlow-enabled switch, and all links have 10Mbps bandwidth. In *Topo1* the average link latency is $5ms$, while the end-to-end latencies (round-trip) fall in the range $[25ms, 70ms]$. In the case of Topo2, the link latencies are artificially tuned in a way that allows us to experiment with increased end-to-end delays, with round trip latencies between $100ms$ and $150ms$.

In both topologies, clients are distributed over five user locations. Each client can reach two servers, each being accessed using three alternative paths. By default, all clients are initially assigned to the shortest path in terms of hop count. Each experiment has a duration of 5 minutes and is preceded by a short startup phase in which paths are installed and the MM and LB application initialized. The placement of LMs is provided as an input and used to compute the relevant hop-count and corresponding latencies between pairs of LMs.

For each experiment, only one of the two services is emulated. In the case of *content distribution*, each client generates content requests following a pattern derived from the one used in [37]. The content size is scaled down in accordance to the reduced link bandwidth. In the case of *cloud*



(a) Monitoring latency, Topo1 (b) Monitoring latency, Topo2 (c) Monitoring traffic

Fig. 8: Performance of the decentralized monitoring approach.

gaming we directly emulate the corresponding network traffic by generating separate client and server-originated packet streams. In particular, we use the results reported in [45] to configure the average packet size and packet rate for both upstream (*i.e.,* client-originated) and downstream (*i.e.,* server-originated) traffic.

The LB application reconfigures the flow routing in case of link congestion, as described in Sec. V, based on local knowledge, including the utilization of local links and the throughput of local traffic flows (if any), as well as information about remote links, which is accessed through periodic synchronization. For simplicity, all LB instances run simultaneously (same frequency and clock reference). For each experiment we configure two parameters: p_l is the period of local measurements performed by each MM, and p_s the synchronization period of link status between the LB instances, with $p_s \geq p_l$. For both services, link congestion is generated by creating spikes of user demand, which is achieved by increasing the number of clients over the experiment time. Another key parameter is the congestion threshold, which has a direct impact on the flow (re)scheduling operated by LB. We set it to 85% of the link capacity in accordance to [25]. Such settings allow to avoid excessive route flapping and to keep the average period of route reconfigurations at least one order of magnitude higher than the content download times. These values are in line with traditional end-user redirection practices [32].

B. Performance of Decentralized Monitoring

In this subsection, we compare our decentralized monitoring approach with a centralized solution where the full state of the network is collected by a single management entity. We first focus on the monitoring latency, a measure of reactivity, defined as the delay between the time the measurement starts (*e.g.,* the corresponding procedure is selected by the scheduler) and the time the requested information is made available to the LB instance performing the flow routing reconfigurations. We evaluate the monitoring latency for the link utilization measurements in 4 different setups: *Centralized* (single manager), *2 LMs*, *3 LMs* and *Fully distributed*, in which one LM is assigned to every switch. For the centralized, 2 LMs and 3 LMs cases, we perform 10 experiments, each with a different manager allocation, and average the results. We fix $p_s = p_l$, *i.e.,* the link status is synchronized between LB instances with every new measurement, and we configure each LM to synchronize with every other LM in the topology.

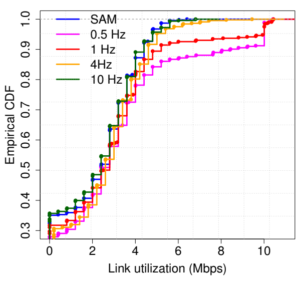
Fig. 8 depicts the average monitoring latency for 3 cases: *i) Minimum distance*: reconfigurations are computed close to where raw statistics are extracted, *i.e.,* by the closest LM; *ii) Maximum distance*: reconfigurations are computed by the farthest LM from where the statistics are collected and *iii) Variable distance*: reconfigurations are computed with the same probability by any of the available LMs. As can be observed, the performance obtained with the *Centralized* setup (baseline scenario) is almost constant as the monitoring information is always processed at the central manager independently from where the statistics are gathered. For the decentralized setups, we observe a significant delay reduction for *minimum distance* in comparison to the centralized scenario. The reduction is up to 57% for *Topo1*, and even more evident (61%) for *Topo2*, where paths generally span higher latencies and number of hops. Smaller latency reductions can be noticed for *maximum distance*, up to 17% for $Topo1$ and 40% for *Topo2*. As expected, the higher the percentage of reconfigurations computed close to where the relevant knowledge is collected, the higher the reduction in terms of control-loop delays achieved with the decentralized approach.

In addition, we evaluate the total amount of monitoring traffic handled by an individual LM in the different decentralized setups. Results of Fig. 8c are normalized to the ones obtained in the *Centralized* case. As observed, the decentralized approach can drastically reduce the burden on the single LM since the incoming monitoring traffic decreases, as expected, proportionally to the number of LMs deployed in the network.

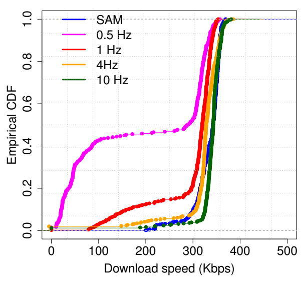
C. Monitoring Information Extraction

In this subsection, we investigate how the extraction of monitoring information by the monitoring modules can affect the performance of the use case services (content distribution and cloud gaming) and the LB application. In particular, we compare a baseline solution where the switches are queried at a fixed rate with the self-tuning adaptive monitoring algorithm (SAM) introduced in Sec. IV. In this section we extend the analysis of the performance of the SAM approach by focusing on the tradeoff between application/service performance and monitoring overhead.

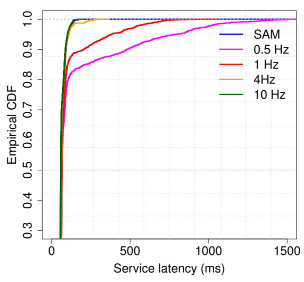
To quantify the performance of the LB application, as well as the impact on content distribution and cloud gaming, we take into account three different metrics. The performance of the LB management application is represented by the *utilization* of the congested link l, obtained by sampling it at rate $1/p_l$ (*i.e.,*, every 1 second) throughout the duration of



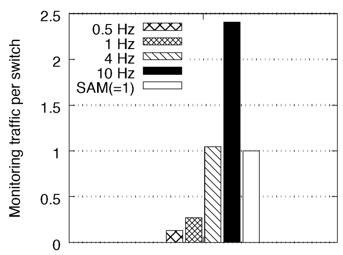
(a)



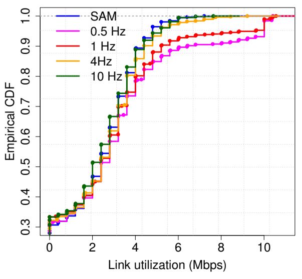
(b)



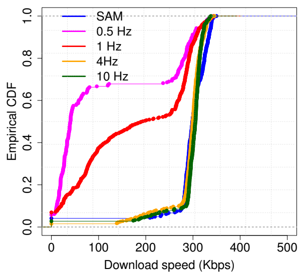
(c)



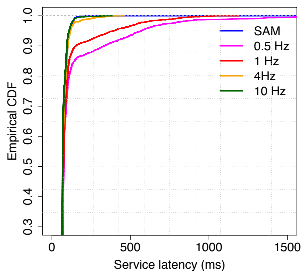
(d)



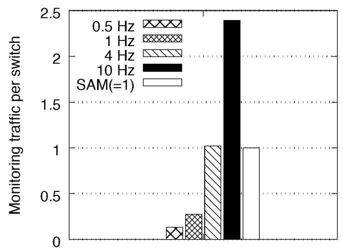
(e)



(f)



(g)



(h)

Fig. 9: Extraction of monitoring information ([a,b,c,d]:Topo1; [e,f,g,h]:Topo2)

the experiment. For the content distribution service, we collect the *download speed* as a measure of the user's QoE. Finally, the performance of cloud gaming is evaluated by measuring the *service latency*, *i.e.,* the response time of each individual user request, which determines how responsive the gaming service is. To guarantee acceptable performance, the service latency should not exceed $80ms$ for highly interactive games and $150ms$ for slow-paced games [44].

To represent the monitoring overhead, we consider the *mean monitoring traffic* rate produced by individual switches. This metric is in line with [30] and [33], which show that only a limited amount of monitoring data can be reported by SDN-enabled switches for each time unit.

The evaluation is performed by running experiments with the *Fully Distributed* setup in which we generate congestion episodes on a specific network link. For each experiment, monitoring is configured to query the switch either at a fixed rate (with frequency $0.5Hz$, $1Hz$ or $10Hz$) or using SAM with query rate varying in the range $[0.1Hz, 10Hz]$. We also fix $p_s = p_l$, so that measurement results are always synchronized between the LMs. For each test run, download speed and cloud gaming latency values are recorded for all clients in the network. Also, for the results of different clients not to be affected by the different path latencies, the selected users are served through paths of equal length.

Fig. 9 shows the empirical CDF of *link utilization* of the congested link for the different monitoring configurations, as well as the one of the *download speed* of content distribution and of the cloud gaming *service latency*. In addition, the mean monitoring traffic (*i.e.,* the monitoring overhead) is reported, normalized to the one obtained with SAM. As depicted in the figure, the values of the three performance indicators improve when increasing the measurement rate as congestion episodes can be detected more reactively. Interestingly, we observe that the performance obtained with SAM can generally match the one obtained with fixed $10Hz$ monitoring. For example, as shown in Fig. 9c and 9g, the service latency never exceeds the $150ms$ threshold in both the SAM and fixed $10Hz$ monitoring cases. The SAM approach does however produce less than 50% of monitoring traffic compared to fixed $10Hz$ monitoring as shown in Fig. 9d and 9h. This is because the adaptive monitoring logic of SAM increases the query rate up to $10Hz$ only when needed, *i.e.,* when congestion arises. The only case in which $10Hz$ monitoring can outperform SAM is for the download speed but yet the difference in performance is never substantial. For instance in *Topo1*, where the difference is more noticeable, the download speed obtained with SAM is lower in only 20% of the cases, and the speed reduction never exceeds $40Kbps$, which represents less than 12% of the maximum speed. This small benefit of fixed $10Hz$ monitoring comes furthermore at a huge cost as 150% more monitoring traffic is produced. Finally, we also consider the case where the operator adopts a fixed 4Hz measurement frequency, thus *guessing* the same average query rate of SAM (Fig. 9d,9h). In practice, this is unlikely to happen, as no prior knowledge on a suitable measurement rate is usually available to the operator. As shown in Fig. 9, even in this case SAM is not outperformed.

D. Monitoring Information Distribution

In the considered decentralized management framework [8], instances of a distributed application can take decisions based on information extracted at a remote location. In this subsection we investigate the effects of the dissemination of monitoring information between LMs on the performance of the LB application and its impact on the content distribution and cloud gaming services. We run experiments with the *Fully Distributed* setup in which congestion episodes occur on a specific link l, located under the scope of a specific LB instance (local LB), while the offloading decisions are

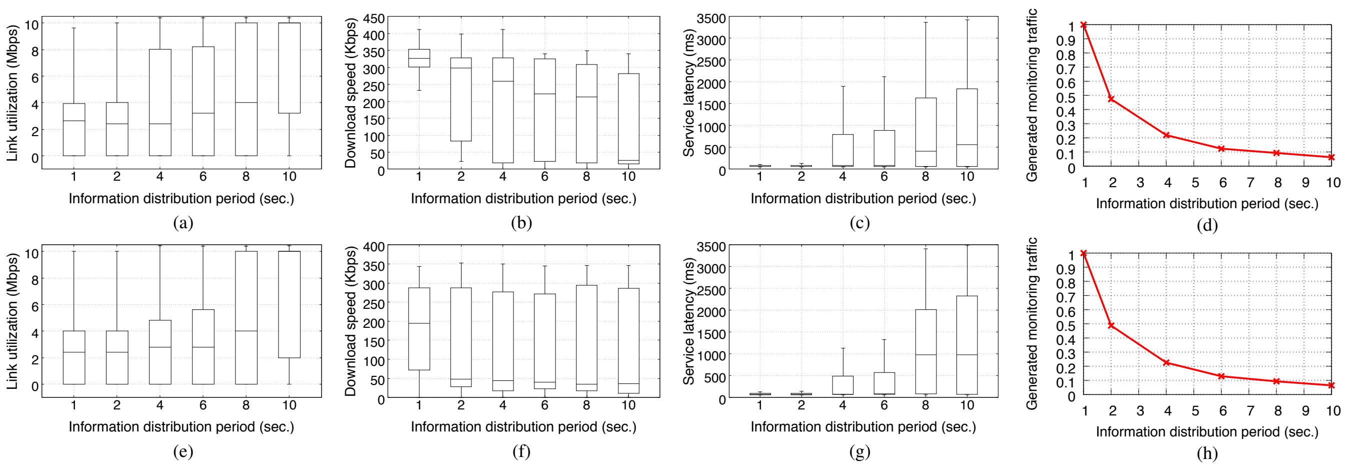


Fig. 10: Synchronization of monitoring information ([a,b,c,d]:Topo1; [e,f,g,h]:Topo2)

made outside the local partition by another application instance (remote LB).

In these experiments, we fix $p_l = 1$ second and let p_s vary in the range $(1, 10)$ seconds in order to increase the inconsistency between the views of the two LB instances. We quantify the effects of relaxed synchronization based on the three metrics presented in Sec. VI-C, *i.e., link utilization*, *download speed* and cloud gaming *service latency*.

Results are shown in Fig. 10 in the form of boxplots, with the whiskers extending from the box (first and third quartile boundaries) to the 95 percentiles. Fig. 10a and 10e show that the utilization of l is significantly affected by the synchronization period. For low values, such as $p_s = 1, 2$, the utilization is below the congestion threshold for more than half of the total duration of the experiment. Starting from $p_s = 4$, the inconsistency of views between local and remote LB brings a noticeable increase to the utilization median, in both *Topo1* and *Topo2*. For example, in the case $p_s = 10$, the median utilization is beyond the congestion threshold (8.5 Mpbs) and coincides with the link capacity (10Mbps), *i.e.,* the link is in a congested state for at least half of the experiment duration.

This trend is also reflected on the user perceived quality of the two use case services. In the content distribution case a decrease of the median download speed can be observed, which can be more or less progressive, *e.g.,* based on the duration of congestion episodes. The impact of relaxed synchronization is particularly evident in *Topo2*, where the download rates are lower than in *Topo1* due to the higher end-to-end latencies in this network. For instance, increasing p_s from 1 second to 2 seconds leads to a substantial reduction of the median throughput that drastically decreases from 300 to 50 Kbps. In the cloud gaming case (Fig. 10c and 10g), the effect of infrequent synchronization can be even more disruptive. In particular, while an acceptable gaming experience can be guaranteed with $p_s = 1, 2$ throughout all the experiment time (the service latency rarely exceeds $100ms$), under $p_s = 8, 10$ the latency is above $500ms$ in *Topo1* and above $1sec$ in *Topo2* for approximately half of the cases. This would in practice make the gaming service inaccessible to clients.

In addition, we evaluate the monitoring overhead that we define for each experiment as the generated monitoring traffic, *e.g.,* sum of the size of each packet multiplied by the path length (number of hops). The overhead is the sum of two components: i) the *measurements* overhead, *i.e.,* the traffic incurred by the collection of the raw statistics from the physical devices, and ii) the traffic incurred by the distribution of the link status between the LB instances that linearly increases with the frequency of the information distribution. Only the latest is plotted in Fig. 10d and 10h since, as shown in our previous work [1], the generated traffic is dominated by the dissemination of monitoring information. As expected, increasing the synchronization period can lead to substantial reductions of the overhead that decreases proportionally to p_s.

We finally observe that significant overhead reductions can be obtained by trading off service performance a little bit. For instance, the overhead can be approximately halved by choosing $p_s = 2$ instead of $p_s = 1$, while incurring negligible disruption on the cloud gaming performance, as shown in Fig. 10c and 10g.

VII. Related Work

The de-facto monitoring standard of today's IP network is NetFlow, which is based on packet sampling. Netflow samples packets with the same probability and aggregates them into flows. However, as discussed in [31], several studies have shown the limitations of packet sampling to perform fine-grained monitoring (*e.g.,* biases toward sampling larger flows) making it unsuitable for many management applications.

The advent of SDN has empowered network monitoring with new measurement enablers as OpenFlow switches can keep track of active flows in the network and update per flow counters. A number of proposals have recently exploited this feature to provide direct and precise flow measurements without resorting to packet sampling. In OpenTM [27] the SDN controller pulls, at fixed intervals, the switch counters collected by explicitly polling the switches in order to periodically generate traffic matrices. In [28] the authors propose FlowSense, an approach where the network utilization is measured using a different, push-based, approach. This uses the messages generated during the setup and eviction of flows from the switch flow table. Compared to the technique used in our paper that takes advantage of explicit switch polling, the solution in [28] can reduce the measurement overhead but suffers from limited flexibility since it only works with short-lived flows.

While most of the recent proposals have focused on specific measurements or on a very limited set of measurement tasks, the approaches presented in [21] and [26] provide a measurement API for supporting a wide range of tasks. OpenSketch [21] relies on a clean-slate approach where a novel processing pipeline is used on the switch to support many different measurement tasks. In addition, a library is developed for the control-plane to reconfigure the pipeline. Payless [26] resembles more the approach adopted in this paper as it provides an API to serve different monitoring requests, all executed through pull-based measurements.

Adaptive monitoring in SDN has recently attracted several research efforts. In Payless [26], monitoring adaptations are performed based on fixed thresholds. In [18], a model for dynamically updating the switch query timeout is presented, but it is exclusively tailored to flow covariance measurements. In a similar fashion to SAM, our proposal, the approach in [17] reconfigures the query rate based on the outcome of a linear prediction. It does however require some complex parameter tuning, an issue that our scheme overcomes through automatic reconfigurations of the algorithm parameters.

In contrast to our work, all the aforementioned proposals are mainly tailored to early SDN solutions that rely on a physically centralized control infrastructure. This assumption has been questioned in [10] and [7] where distributed control planes have been proposed to overcome scalability issues such as processing bottlenecks at the central controller and large control latencies. However, the main focus of these papers is on how distributed controllers can unify their local views of the network, paying little attention to measurement issues. In [7] a controller-to-controller communication mechanism based on a pub/sub paradigm is presented. In [10] a distributed database for the dissemination of slowly changing network state and a distributed hash table for exchanging volatile information are proposed. Another important work is [12], which investigates the main issues posed by state distribution in a logically centralized, physically distributed SDN architecture. One of these issues, *i.e.,* the tradeoff between performance optimality and state distribution overhead, has been considered in Sec. VI-D, where we have evaluated how the timeliness of synchronized monitoring information impacts the MA performance and the overhead in terms of additional monitoring traffic. In a less recent work [13], the authors introduce a model, called A-Gap, for adaptive reduction of the traffic overhead in distributed monitoring based on filtering. However, this technique addresses a different, hierarchical, monitoring architecture where the information is aggregated and transmitted on a spanning tree toward a central management station.

VIII. Conclusion

In this paper we have presented a novel monitoring approach for software-defined networks that can provide heterogeneous management applications with frequent and consistent network state updates, thus enabling fast and effective resource reconfigurations. Our solution relies on a decentralized architecture satisfying the requirements of networks with a large number of geographically dispersed devices. To reduce the consumption of the switch control bandwidth, it performs frequent adaptations of the switch query rate using SAM, our novel self-adaptive monitoring method. As opposed to existing approaches, for which complex parameter tuning is needed under highly dynamic network traffic, the proposed algorithm can automatically reconfigure itself without any intervention from the operator.

The evaluation, based on realistic topologies and demanding use case services, has shown that our decentralized monitoring framework can improve the reconfiguration reactivity by significantly reducing the control-loop delays, in particular when a large portion of reconfiguration decisions are taken close to where the relevant statistics are collected. In addition, we have demonstrated that service performance can significantly improve by enabling SAM, without incurring additional switch resource consumption. Finally, although decentralizing the monitoring functionality involves additional communication overhead, this can be mitigated by slightly relaxing the synchronization of monitoring data, while maintaining acceptable service performance.

In future work, we plan to explore new use case applications, as well as different network topologies. We also plan to investigate how to dynamically allocate switch resources, in terms of monitoring bandwidth, to different monitoring tasks under limited resources. This will involve developing an online monitoring accuracy estimation tool and its use in conjunction with SAM query period adaptations.

References

[1] G. Tangari, D. Tuncer, M. Charalambides, and G. Pavlou. Decentralized Monitoring for Large-Scale Software-Defined Networks. In *Proc. IEEE/IFIP IM*, Lisbon, Portugal, May 2017, pp. 289-297.

[2] H. Kim and N. Feamster. Improving network management with software defined networking. In *IEEE Communication Magazine*, vol. 51, no. 2, Feb. 2013, pp. 114-119.

[3] Y. Yuan, R. Alur, and B. T. Loo. NetEgg: Programming network policies by examples. In *Proc. ACM Hotnets*, USA, Oct. 2014, pp. 20-27.

[4] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: dynamic flow scheduling for data center networks. In *Proc. NSDI*, San Jose, CA, USA, Apr. 2010.

[5] S. Agarwal, M. Kodialam, and T. Lakshman. Traffic engineering in software defined networks. In *Proc. IEEE INFOCOM*, Turin, Italy, Apr. 2013, pp. 2211-2219.

[6] X. Liu, M. Shirazipour, M. Yu, Y. Zhang. MOZART: Temporal Coordination of Measurement. In *Proc ACM SOSR*, USA, Mar 2016.

[7] A. Tootoonchian and Y. Ganjali. HyperFlow: a distributed control plane for OpenFlow. In *Proc. USENIX INM/WREN*, USA, 2010, pp. 3-9.

[8] D. Tuncer, M. Charalambides, S. Clayman, and G. Pavlou. Adaptive resource management and control in software defined networks. In *IEEE TNSM*, vol. 12, no. 1, Mar. 2015, pp. 18-33.

[9] Shan-Hsiang Shen, Aditya Akella. DECOR: A distributed coordinated resource monitoring system. In *Proc. IEEE IWQoS*, Coimbra, Portugal, Jun. 2012, pp. 1-9.

[10] T. Koponen et al. Onix: a distributed control platform for large-scale production networks. In *Proc. USENIX OSDI*, Vancouver, BC, Canada, Oct. 2010, pp. 351-364.

[11] P. Berde, et al. ONOS: towards an open, distributed SDN OS. In *Proc. ACM HotSDN*, Chicago, Illinois, USA, Aug. 2014, pp. 1-6.

[12] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann. Logically centralized?: State distribution trade-offs in software defined networks. In *Proc. ACM HotSDN*, Helsinki, Finland, Aug. 2013, pp. 1-6.

[13] A. Gonzales, and R. Stadler. Adaptive distributed monitoring with accuracy objectives. In *Proc. ACM INM*, Pisa, Italy, Sep. 2006, pp. 65-70.

[14] N. Van Adrichem, C. Doerr, and F. Kuipers. OpenNetMon: network monitoring in openflow software-defined networks. In *Proc. IEEE/IFIP NOMS*, Krakow, Poland, May 2014, pp. 1-8.
[15] C. Yu et al. Software-defined latency monitoring in data center networks. In *Proc. PAM*, New York, NY, USA, Mar. 2015, pp. 360-372.
[16] M. Moshref, M. Yu, R. Govindan, and A. Vahdat. Dream: dynamic resource allocation for software-defined measurement. In *Proc. ACM SIGCOMM*, Chicago, IL, USA, Aug. 2014, pp. 419-430.
[17] T. Zhang. An adaptive flow counting method for anomaly detection in SDN. In *Proc. ACM CoNEXT*, Santa Barbara, USA, Dec. 2013, pp. 25-30.
[18] Z. Bozakov, A. Rizk, D. Bhat, and M. Zink. Measurement-based Flow Characterization in Centrally Controlled Networks. In *Proc. IEEE INFOCOM*, San Francisco, CA, USA, Apr. 2016, pp.1-9.
[19] Y. Yu, C. Quien, X. Li. Distributed collaborative monitoring in software defined networks. In *Proc. ACM HotSDN*, USA, Aug. 2014, pp. 85-90.
[20] N. McKeown et al. OpenFlow: enabling innovation in campus networks. In *ACM SIGCOMM CCR*, vol. 38, no. 3, Apr. 2008, pp. 69-74.
[21] M. Yu et al. software defined traffic seasurement with OpenSketch. In *Proc. USENIX NSDI*, Lombard, IL, USA, Apr. 2013, pp. 29-42.
[22] C. Kim et al. In-band network telemetry via programmable dataplanes. In *Proc. ACM SIGCOMM*, London, UK, Aug. 2015, Demo Session.
[23] P. Bosshart et al. P4: programming protocol-independent packet processors. In *ACM SIGCOMM CCR*, vol. 44, no. 3, Jul. 2014, pp. 87-95.
[24] P. Bosshart et al. Forwarding metamorphosis: fast programmable match-action processing in hardware for SDN. In *Proc. ACM SIGCOMM*, Hong Kong, China, Aug. 2013, pp. 99-110.
[25] D. Tipper et al. An analysis of the congestion effects of link failures in wide area networks. In *IEEE JSAC*, vol. 12, Jan. 1994, pp. 179-191.
[26] S. Chowdhury, M. Bari, R. Ahmed, and R. Boutaba. PayLess: a low cost network monitoring framework for software defined networks. In *Proc. IEEE/IFIP NOMS*, Krakow, Poland, May 2014, pp. 1-9.
[27] A. Tootoonchian, M. Ghobadi, and Y. Ganjali. OpenTM: traffic matrix estimator for OpenFlow networks. In *Proc. PAM*, Zurich, Switzerland, Apr. 2010, pp. 201-210.
[28] C. Yu et al. FlowSense: monitoring network utilization with zero measurement cost. *Proc. PAM*, Hong Kong, China, Mar. 2013, pp. 31-41.
[29] M. Wichtlhuber, R. Reinecke, and D. Hausheer. An SDN-based CDN/ISP collaboration architecture for managing high-volume flows. In *IEEE TNSM*, vol. 12, no. 1, Mar. 2015, pp. 48-60.
[30] J. C. Mogul et al. Devoflow: cost-effective flow management for high performance enterprise networks. In *Proc. ACM Hotnets*, Monterey, CA, USA, Oct. 2010, pp. 1-6.
[31] V. Sekar, M. K Reiter, and Hui Zhang. Revisiting the case for a minimalist approach for network flow monitoring. In *Proc. ACM IMC*, Melbourne, Australia, Nov. 2010, pp 328-341.
[32] A. Su, D. Choffnes, A. Kuzmanovic, and F. Bustamante. Drafting behind Akamai. In *IEEE/ACM TON*, vol 17, no. 6, Dec. 2009, pp. 1752-1765.
[33] L. Hendriks, R. Schmidt, R. Sadre, J. Bezerra and A. Pras. Assessing the quality of flow measurements from OpenFlow devices. In *Proc. TMA*, Louvain La Neuve, Belgium, Apr. 2016.
[34] I. Poese et al. Enabling content-aware traffic engineering. In *ACM SIGCOMM CCR*, vol. 42, no. 5, Oct. 2012, pp. 21-28.
[35] D. Tuncer, M. Charalambides, G. Pavlou, N. Wang. DACoRM: a coordinated, decentralized and adaptive network resource management scheme. In *Proc. IEEE/IFIP NOMS*, Westin Maui Maui, HI, USA, Apr. 2012, pp. 417-425.
[36] D. Valocchi et al. Extensible signaling framework for decentralized network management applications. In *Proc. IEEE/IFIP NOMS*, Istanbul, Turkey, Apr. 2016, pp. 153-161.
[37] M. Claeys, D. Tuncer, J. Famaey, M. Charalambides, S. Latre, G.Pavlou, F. De Turck. Hybrid multi-tenant cache management for virtualized ISP networks. In *Journal of Network and Computer Applications*, vol. 68, issue C, Jun. 2016, pp. 28-41.
[38] *POX* OpenFlow controller. *http://www.noxrepo.org*.
[39] *OpenConfig* project. *http://www.openconfig.net*.
[40] T. Jirsik, M. Cermak, D. Tovarnak, P. Celeda.Toward Stream-Based IP Flow Analysis. In *IEEE Communications Magazine*, Volume 55, Issue 7, 2017, pp 70-76.
[41] F. Baccelli, S. Machiraju, D. Veitch, J. Bolot. The Role of PASTA in Network Measurement. In *Proc. ACM SIGCOMM*, Italy, Sep. 2006.
[42] 100 Megabit Ethernet anonymized packet traces without payload: WIDE-TRANSIT link. *http://mawi.wide.ad.jp/mawi/ditl/ditl2007/*.
[43] Gaikai video game streaming platform. *https://www.playstation.com/en-gb/explore/playstation-now/?smcid=psnow-vanityurl*.
[44] S. Choi, B. Wong, G. Simon, C. Rosemberg. A hybrid edge-cloud architecture for reducing on-demand gaming latency. In *Journal of Multimedia Systems*, Issue 5/2014, pp. 503-519.
[45] M. Manzano, J.A. Hernandez, M. Uruena, E. Calle. An empirical study of Cloud Gaming. In *Proc. Netgames*, Venice, Italy, Nov. 2012, pp. 1-2.
[46] The GEANT Topology, 2004. *http://www.dante.net/server/show/nav.007009007*
[47] The Germany50 Topology, 2004. *http://sndlib.zib.de*

Gioacchino Tangari is a PhD student in the Department of Electronic and Electrical engineering at University College London. His main research interests include network monitoring in the context of programmable networks, and high-speed packet processing on commodity hardware. He has been working as a research intern in Nokia Bell Labs in 2014, Paris, and Telefonica Research, Barcelona, in 2017.

Daphne Tuncer is a Research Fellow in the Department of Computing at Imperial College London, UK. She received her Ph.D. from University College London (UK) in 2013 and a Diplome d'ingenieur de Telecom SudParis (France) in 2009. Her research interests are in the areas of software-defined and programmable networks, adaptive network resource management and multimedia content distribution.

Marinos Charalambides is a senior researcher at University College London. He received a BEng in Electronic and Electrical Engineering, a MSc in Communications Networks and Software, and a Ph.D. in Policy-based Network Management, all from the University of Surrey, UK, in 2001, 2002 and 2009, respectively. His research interests include network programmability, adaptive resource management, content delivery and network monitoring.

Qi Yuanshunle is a BEng student in the Department of Electronic and Electrical Engineering at University College London. His research interests are in computer networks.

George Pavlou is Professor of Communication Networks in the Department of Electronic and Electrical Engineering, University College London, UK. He received a PhD in Computer Science from University College London, UK. His research interests focus on networking and network management, including aspects such as autonomic networking and software defined networks. He is the chief editor of the bi-annual IEEE Communications network and service management series and in 2011 he received the Daniel Stokesbury award.