Rhaban Hark, Nieke Aerts, David Hock, Nils Richerzhagen, Amr Rizk, and Ralf Steinmetz. *Reducing the Monitoring Footprint on Controllers in Software-defined Networks* in IEEE Transactions on Network and Service Management, Special Issue on Novel Techniques for Managing Softwarized Networks, Oct. 2018, ISSN: 1932-4537

Reducing the Monitoring Footprint on Controllers in Software-defined Networks

Rhaban Hark, *Student Member, IEEE*, Nieke Aerts, David Hock, Nils Richerzhagen, *Student Member, IEEE*, Amr Rizk, *Member, IEEE*, and Ralf Steinmetz, *Fellow, IEEE*

***Abstract*—A decisive advantage of Software-defined Networking is its support for flexible network reconfigurations. Considering that, Software-defined Networks require accurate and timely data-plane state information. Network monitoring mechanisms usually require considerable resources on SDN controllers as well as on the data-plane elements. In this work, we propose an optimization of the statistic transmission to reduce costs on both control- and data-plane regardless of the used monitoring application and statistic provisioning tool. To this end, we intercept the statistic message exchange and *(i) aggregate multiple requests* coming from different monitoring applications/controllers, *(ii) filter irrelevant statistic messages* with respect to their information gain before delivering them to the control applications, and *(iii)* deploy *statistic caching*. The proposed system, denoted STATISTIC REQUEST RELAY (SRR), forms a logically centralized statistic relay between controllers and the managed data-plane network. Our evaluation shows that the number of statistics processed on controllers as well as statistic requests on switches is reduced significantly while the performance penalty is negligible when using statistic aggregation and filtering as proposed here.**

***Index Terms*—Software-defined Networking, Monitoring, Middleware, Distributed Control-Planes**

I. INTRODUCTION

MONITORING the state of forwarding entities and events occurring in the network is a cornerstone of network management. Management functions such as load balancing, resource provisioning, accounting and even routing require accurate monitoring information. Numerous measurement tasks are required to regularly obtain accurate monitoring data, which introduces considerable resource overhead [1].

With the advent of Software-defined Networking (SDN) new techniques for network monitoring arise, i.e. the broadly accepted OpenFlow protocol includes techniques such as resource-friendly statistic collection using built-in packet/byte/lifetime counters or per-flow traffic mirroring [2]. Equipped with the aforementioned techniques, a number of works propose improvements to monitoring efficiency through approaches such as adaptive statistic polling [3], [4] and mechanisms that leverage control messages to access statistics in a resource-neutral manner [5]. A common theme of such approaches in SDNs is the involvement of the logically centralized network controllers in the monitoring process.

SDN incorporates centralized controllers in order to perform management tasks. Monitoring is a prominent application typically run on the SDN controller and, as such, it consumes computing and networking resources [1], [6]. Even utilizing non-SDN specific monitoring techniques, such as sFlow [7] or IPFIX [8], require a monitoring collector potentially placed in the controller, to gather and process statistics captured in the network. Basic monitoring approaches such as periodic queries often lead to a resource utilization overhead. In general, we find that in all existing approaches the controller is involved in collecting, processing, and/or analyzing monitoring statistics, constituting a potential bottleneck in the network management process. Hence, for scalability and fail-safety purposes, the logically centralized control needs to be implemented in a distributed manner [6], [9], [10]. However, it remains unclear how to architecture the different controller entities and their integration to optimize the network management process. For example, a higher number of coexisting controllers that share the monitoring workload does not guarantee optimized monitoring overhead.

Overall we find that existing approaches which aim to reduce monitoring costs still substantially involve the SDN controllers [4], [5], [11], [12]. Krishnamurthy et al. [13] as well as Yeganeh et al. [6] argue that even powerful controllers can not handle the load when the full network state must be observed or frequent events must be handled. From the fact that a smooth controller operation in terms of load and failure-resilience is of major importance for network performance, we extract the requirement *to lighten the load on the controllers by removing monitoring associated tasks whenever possible*.

In this work, we propose to push monitoring functionality out of the controllers while retaining the benefits SDN entails. We relieve the controller completely of monitoring tasks which can potentially be performed elsewhere without weakening the controller's capabilities. Nevertheless, we ensure that controllers receive all information they require, such that the network state representation does not downgrade in completeness or timeliness. The system we propose, denoted STATISTIC REQUEST RELAY (SRR), is especially designed to support multi-controller environments, which is a scenario in which monitoring is rarely researched. Making use of multiple controllers performing measurement tasks, we furthermore add

This work has been performed in the framework of the CELTIC EUREKA project SENDATE-PLANETS (Project ID C2015/3-1), and it is partly funded by the German BMBF (Project ID 16KIS0471). The authors alone are responsible for the content of the paper. This work has been partially funded by the German Research Foundation (DFG) within the Collaborative Research Center (CRC) 1053 - MAKI as part of subprojects B1 and B4.

Rhaban Hark, Nils Richerzhagen, Amr Rizk, and Ralf Steinmetz are with the Department of Electrical Engineering and Information Technology of the Technische Universität Darmstadt, Germany; {first.last}@kom.tu-darmstadt.de.

Nieke Aerts and David Hock are with Infosim GmbH & Co. KG, Würzburg, Germany; {last}@infosim.net

Rhaban Hark, Nieke Aerts, David Hock, Nils Richerzhagen, Amr Rizk, and Ralf Steinmetz. *Reducing the Monitoring Footprint on Controllers in Software-defined Networks* in IEEE Transactions on Network and Service Management, Special Issue on Novel Techniques for Managing Softwarized Networks, Oct. 2018, ISSN: 1932-4537

mechanisms not only to reduce controller load but also to unburden the data-plane entities.

The remainder of the paper is structured as follows: Section II describes the principles SRR is based on to reduce load. In Section III we outline the scenario SRR is designed for and network architecture. Subsequently, in Section IV we highlight the architecture of the component itself and show the interplay of included subcomponents. Section V shows a quantitative evaluation of monitoring cost reduction using SRR. In Section VI, we include relevant related works and Section VII recaps the paper and outlines future extensions.

II. Monitoring Cost Reduction Principles

In this section we describe how the proposed relay targets to mask the complexity and overhead of monitoring for controllers with three different principles. In general the system acts as an intermediator (relay) between controllers and switches for monitoring tasks, as depicted in Figure 1. The figure shows the controllers of a distributed SDN control-plane connected on their southbound interface to the relay that is itself connected on its southbound interfaces to the network elements, precisely, the switches. The system does not restrict the controller capabilities in any way: its use is optional and, as controllers are connected to switches directly, every task devolved to SRR can still be executed directly on the controller if preferred or required. However, if a control application needs to monitor the network, instead of executing statistic requests or registering for statistics, it can delegate this task to SRR. The system performs the measurement on behalf of the controller and relays the respective measurements back to the controller. For single request-response message exchanges between controllers and switches there is no immediate gain when using a relay, however, the costs for periodic tasks can be reduced through filtering and aggregation.

A. Statistic Filtering

The relay takes over the responsibility to execute periodic tasks such as measuring the used bandwidth for switch ports of interest. Traditionally, the controller would request those statistics at the end of every period, which continuously allocates resources . In contrast, using SRR, a controller devolves such a periodic task to the proposed monitoring service and specifies conditions that must be fulfilled in order to forward statistics to the controller. SRR uses the defined conditions to filter the periodic statistic data received from the switches. For example, a port, whose load does not change in successive periods does not lead to interruptions of the controller as SRR disturbs the controller only if a significant change in the measurement which requires a management applications reaction is observed. The controller can decide on different types of filtering: *(i)* SRR forwards only measurements if it observes outlier values with respect to historic measurements. *(ii)* SRR forwards only measurements exceeding a fixed absolute value (threshold). *(iii)* A combination of both: after exceeding a fixed threshold, relay only measurements when significant changes occur. More complex filtering rules are surely possible [14] The filtering allows reducing the load on the controllers' computational and networking resources, as their involvement is limited to situations where a reaction of control applications is required.

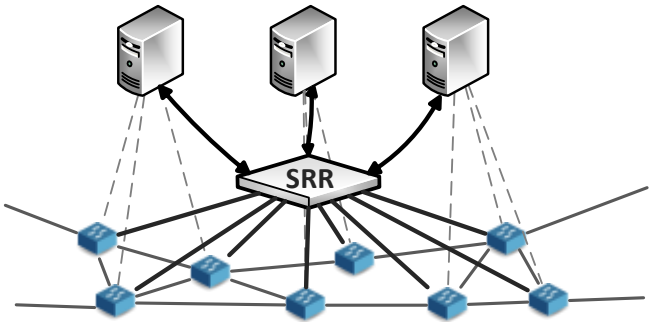


Fig. 1: Conceptual SRR deployment as logically centralized component between the distributed control-plane consisting of the controllers (*here three*) and the data-plane consisting of the switches forming the network.

B. Statistic Request Aggregation

Consider two control applications measuring the same information with different frequencies, e.g. the bandwidth of a shared link or a shared flow. In such cases, they capture the similar information redundantly. SRR compares incoming monitoring task registrations with existing tasks and if an existing task matches a new request, SRR aggregates both monitoring tasks. During aggregation, the system tries to find an optimal measurement point to capture this information. In particular, it decides on one of the measurement points originally given in the task registrations. Furthermore, it combines the update frequencies of periodic measurements. For example, using the highest frequency is of choice, i.e. updates can be skipped for controllers requesting lower update rates. Note that this feature potentially leads to distortions, therefore, it has to be allowed explicitly while registering a task.

Example: Flow F traverses the adjacent network parts controlled by controller A and controller B. Controller A registers to measure F's byte counter at any arbitrary switch S_A on the flow's path in its network every two seconds ($f_A = 1/2s$) and controller B registers for the measurement at any arbitrary switch S_B in his network every four seconds ($f_B = 1/4s$). If the byte counter can be assumed to remain stationary over the path, SRR aggregates both measurements. It selects one measurement point on F's path, S_A or S_B and uses the higher frequency, in this case $f_A = 1/2s$. It stands to reason that only measurements of flows that traverse networks of at least two involved controllers (or applications within a single controller) can be aggregated. In this example, the system omits the full execution of one task taking load from the switches, while both controllers get their measurement results as requested.

C. Statistic Caching

SRR furthermore reduces monitoring costs on the southbound interface through caching of statistics. The results of all conducted measurements are stored in a statistic cache.

Rhaban Hark, Nieke Aerts, David Hock, Nils Richerzhagen, Amr Rizk, and Ralf Steinmetz. *Reducing the Monitoring Footprint on Controllers in Software-defined Networks* in IEEE Transactions on Network and Service Management, Special Issue on Novel Techniques for Managing Softwarized Networks, Oct. 2018, ISSN: 1932-4537

These values can be forwarded if a similar request reaches SRR without the need for new measurements. The service provides the possibility to define a staleness limit for the measurement values turning this cache to a time-to-live controlled cache. Caching is only conducted for non-periodic measurement tasks since the aggregation mechanism there replaces time aggregation, i.e., caching, with spatial aggregation. It is reasonable to use the cache for metrics that do not vary on short timescales, e.g., derived metrics such as the average bandwidth utilization.

III. Scenario and Deployment Modes

Before outlining different possible manifestations of SRR instances, we briefly describe the scenario it can be applied to. As Figure 1 depicts, SRR is an intermediate layer between the control-plane and the data-plane, comparable to existing middleware approaches such as FlowVisor [15] and OpenVirteX [16]. Hence, the scope covers multiple controllers setting up a distributed control-plane managing a variety of underlying switches and other data-plane entities. In such generic scenarios, SRR exploits its full potential including aggregation of requests monitoring the same flow from different sources (controllers or control applications), filtering of periodic measurements, and statistic caching. Note, that the introduction of an additional intermediate entity (causing CAPEX) has to be evaluated against the operational gain (saving OPEX) in very small networks.

Although SRR could execute the filtering also in the controllers themselves, we design the system as an additional component to gain further advantages: *(i)* The load on the controllers network interface can be reduced significantly, giving other control traffic in critical situations more bandwidth. This load is shifted to the new component that does not handle other management traffic than that on its southbound interface. *(ii)* The aggregation of statistic tasks from multiple controllers is much easier to handle in a logically centralized manner, which had to be implemented distributively if the controller themselves contain this functionality; *(iii)* Preprocessing of information or even whole analysis functions can be devolved relieving the resources on the controllers using SRR on a dedicated entity with separate processing capabilities.

In the first place, we design SRR as a centralized entity. We leave the transparent physical distribution of a logically centralized system for future work as the actual deployment does not affect the principles investigated in this paper. However, in this section we take fail-safety and scalability concerns into account and point out different deployment modes.

Single-Server Deployment: In this deployment mode SRR is placed in only one physical entity. It reduces complexity and deployment costs, however, it lacks mechanisms to scale or fallback possibilities in case of faults.

Cluster Deployment: In order to enable high availability, SRR can be deployed in multiple different physical entities, as cluster deployment. Such a mode is comparable to the use of a distributed control-plane: If the resources of a single component do not suffice, a physical decentralization is desired. Levin et al. [10] investigate trade-offs between logical centralized and physical distribution. A first instantiation is a high-availability solution, i.e. one SRR deployment runs actively in one of the entities, while the other deployment is in standby mode. Once a failure occurs and the active SRR is not operable anymore, a standby entity takes over the monitoring responsibilities. While providing higher availability this approach introduces considerable deployment and operation costs: Despite failure detection capabilities, either an additional proxy service must enable seamless switching to the active service or the controllers must update the service endpoint information. However, state distribution between the active and passive SRR deployments affects the deployment costs to an even higher degree as all backup entities require identical state to transparently take over tasks. While the former problem of seamless switching can be solved intuitively, a number of works, e.g. located in the context of distributed control-planes [17], tackle the latter problem of state distribution/migration.

A second instantiation of a cluster deployment is a load-balanced cluster that enables load balancing between multiple SRR deployments. A proxy redirects incoming request to one of the available active service endpoints. The mode enables higher availability while reducing the load on single deployments. However, it leads to considerable costs for state distribution.

IV. SRR Architecture Design

This section covers the SRR architecture and component logic including the interplay of subcomponents and the communication with control and data-plane entities.

A valid abstraction of the potentially distributed deployment of SRR (as discussed in Section III) is a single server providing the monitoring services. Figure 2 shows the inner structure of such an instance. Roughly speaking, three layers assemble the architecture: *(i)* The service interface layer communicates with controllers and provides the northbound API. *(ii)* The inner parts compose the components logic. *(iii)* The lower layer consists of monitoring agents that communicate through their southbound interfaces with the network elements. The remainder of this section covers first the component logic and then the north- and southbound interfaces.

A. Relay Component Logic

The relay consists of a small number of subcomponents building the system's inner logic (cf. Figure 2). We picture the interplay and responsibilities of the subcomponents in detail using workflows of executed subtasks. Figure 3 sketches the principles of the described workflows.

a) Task Registration Workflow: Requests, such as task registrations, sent from controllers that use the service reach the system through the *Service Interface*. This interface maintains connection endpoints to all controllers that initially contact the component. The *Task Inventory Manager* decides how to handle incoming requests. Controllers may devolve different types of tasks to the service resulting in different request types. A task can either be a single measurement request, which is performed immediately and only once. The service answers immediately after obtaining the result, either

Rhaban Hark, Nieke Aerts, David Hock, Nils Richerzhagen, Amr Rizk, and Ralf Steinmetz. *Reducing the Monitoring Footprint on Controllers in Software-defined Networks* in IEEE Transactions on Network and Service Management, Special Issue on Novel Techniques for Managing Softwarized Networks, Oct. 2018, ISSN: 1932-4537

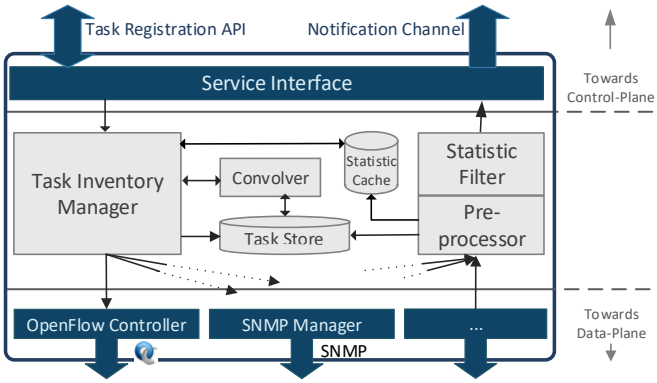


Fig. 2: SRR component inner architecture.

from the *Statistic Cache* or through conducting a measurement. In order to find out whether a measurement can be obtained from the cache, the manager asks the *Convolver* to look for a similar measurement stored in the cache. If the cache contains a matching measurement that is not yet outdated, that is, inside of the allowed staleness tolerance, the manager forwards the result directly to the controller. More interesting are periodic tasks, as they leave more space for optimization. The task manager asks the *Convolver* for similar periodic tasks and, if found, maps them to one common associated *aggregated task*. A new task can also be mapped to an existing aggregated task. The southbound agents execute all measurements included in a task after the *Task Inventory Manager* advices them to do so.

Both, the *Task Store* and the *Statistic Cache* are small databases holding information about tasks registered at the service and conducted measurements, respectively. The *Task Store* maintains registered tasks and task aggregations, including their full specification. Furthermore, it holds information on whether a task is *active* or not. A measurement might be inactive as it is part of a task aggregation. The data structure to store a task includes the following fields {`taskId`, `active`, `referenceTaskId`, `controller`, `additionalControllers`, `metric`, `periodic`, `period`, `threshold`, `thresholdType`, `entity`}. The `taskId` is used to identify tasks, e.g. when measurements are forwarded to a controller, it may use this id. If a task is not active as it is aggregated with another task, the `active` flag is set to false. Once the associated aggregation is stopped a task might become active again. The associated task from an aggregation is referenced using the `referenceTaskId`. The `controller` field identifies the controller that registered the task and is used to forward measurements. Furthermore, the `additionalControllers` list is filled with information on controllers that are aggregated with the current task as they need to be updated as well. The `metric` stores the metric to be measured, e.g. the bandwidth in combination with the `entity` field that defines the entity. SRR's implementation allows a simple extension of the list of supported metrics. The set of exemplary metrics include in particular the bandwidth for links and for flows. If the `periodic` field is set, the `period` defines how often a measurement must be taken. Lastly, the filter conditions are stored in the `threshold` and `thresholdType` field, respectively. The *Statistic Cache* stores contemporary measurement values which point to their respective task in the *Task Store*. SRR clears outdated cached values after a fixed idle timeouts.

b) Task Aggregation Workflow: The *Convolver* induces one essential part of the resource-saving capabilities to the system: aggregating tasks based on their similarity. A task can be aggregated with another task if the desired metric, e.g. bandwidth consumptions or loss ratio, and the resource that is investigated, e.g. the flow or switch port, match. The control applications can specify in the task registration to what extend the service is allowed to modify the measurement. Typical modifications are the measurement point and frequency. Thus, the service might use a different switch to measure a flow's byte count than specified or instead of updating the measurement every three seconds do so every two seconds. Once the *Convolver* decides to use a higher frequency for a measurement (it never uses lower frequencies to retain timeliness) it tries to use the switch specified in the task with the selected frequency. This way, a switch is never loaded more than it would be without aggregation. Note that SRR can only aggregate tasks that request the same metric on the same entity, i.e. if the *meaning* does not alter, e.g., by modifying the specified measurement point. The *Convolver* reads and writes the *Task Store* and interacts with the *Task Inventory Manager* for aggregation.

c) Forwarding of Measurement: Measurements performed by the southbound agents traverse multiple stages before they eventually trigger a message to the controller. Note that SRR is designed to take over the measurement process and is not responsible for the monitoring logic and analysis. However, despite executing simple measurements, the system includes a small set of simple precalculation capabilities. To avoid forcing controllers to receive meaningless measurements solely to calculate a derived metric, the *Statistic Preprocessor* is capable of computing simple derived metrics composed of multiple sub-measurements. In particular, this includes simple derived metrics, such as the loss ratio calculated from two counter values from nodes on a certain path or link and the consumed bandwidth calculated from two timewise subsequent counter values. For this, the *Statistic Preprocessor* identifies connected tasks for a conducted measurement and checks if the value is part of a derived metric, so that it can perform the precalculation or hold back the value until all necessary values are present. The *Statistic Preprocessor* saves directly captured and, if available, derived metrics into the statistic cache including a timestamp to determine their lifetime.

After potentially passing preprocessing steps, measurements traverse the *Statistic Filter*. The *Statistic Filter* is the second essential part to achieve resource savings in SRR. It filters measurements based on task specifications given from the controllers (cf. Section II-A), thus, leading to an event-based communication with the controllers. As described in Section II-A, controllers can request the monitoring system to forward measurements only if they differ significantly from previous measurements, exceed a fixed threshold, or a combination of both. Hence, the monitoring system compares

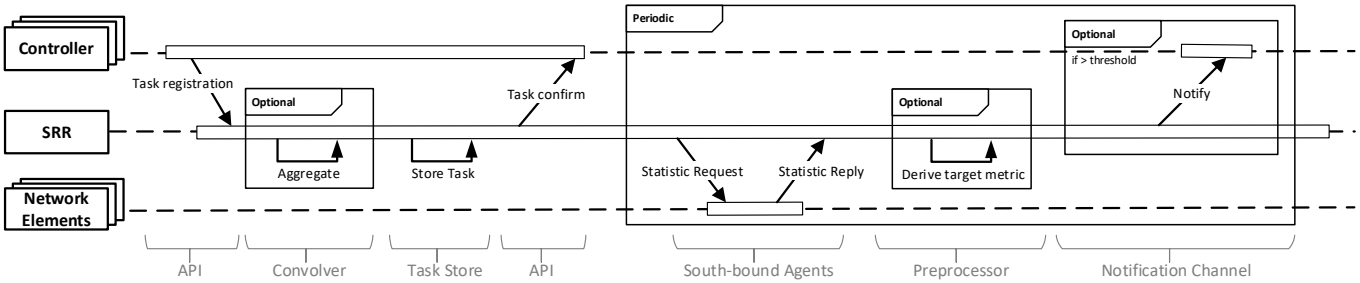


Fig. 3: Typical workflow for a periodic task registration with potential aggregation, metric derivation, and threshold-based statistic event provisioning.

the measurement with the previous value and/or the fixed threshold and either drops the measurement or forwards it to the *Service Interface* towards the control-plane. An example of a task registration including parameters that define the filter conditions are described later in Listing 1. The workflow is also depicted in Figure 3: After a task has been registered, potentially aggregated with another task, saved, and confirmed, the periodic statistic provisioning phase starts. Following the given measurement period, SRR requests the statistics from the data-plane elements, receives the statistics and performs the precalculation if required. A precalculation could be, e.g., the calculation of the bandwidth based on two consecutive byte counter values. Subsequently, the final value traverses the filter stage, which either discards it or forwards it to the controller. During the task registration, the controller defines what type of filtering is applied and the threshold parameter used for the filter. Hence, the notification of the controller is optional depending on the measurement value and its information gain.

B. Southbound Monitoring Agents

In order to conduct measurements and receive information from the forwarding network, SRR uses different agents connected to data-plane elements (cf. Figure 2). The list of usable agents is extensible in order to add new measurement functionality or support alternative mechanisms to perform measurements. Note that SRR does not require the data-plane to support customized functionality as it relies only on existing protocols. However, in order to connect to SRR's southbound agents, the data-plane elements have to be configured to start and enable corresponding end-points. The *Task Inventory Manager* selects the agent used to execute the respective tasks. For some tasks, multiple agents may be usable. In such cases the manager optimizes the costs using the most efficient agent. Especially in the context of SDN, it seems obvious to exploit SDN specific monitoring mechanisms. Hence, in the first place we propose using an OpenFlow agent to be able to send statistic requests as defined in OpenFlow. An OpenFlow agent can be implemented as an OpenFlow slave controller [18], which is an additional controller connected to switches. Slave controllers can only read state from switches, like the flow packet counter, but they cannot modify a the state of a switch. OpenFlow supports a large number of measurements especially regarding traffic metrics, but also, for instance, port-based throughput and loss rate metrics. Please note, that only a controller can request to change its role so that SRR will never be considered as master controller in master-failover cases.

Although OpenFlow already supports a large variety of measurement mechanisms, the SRR system allows to add further agents like SNMP [19], sFlow [7], IPFIX [8] / NetFlow [20] or custom solutions. In particular, we propose to use an SNMP manager as additional monitoring agent. SNMP allows the collection of information from network elements that OpenFlow does not support, such as hardware performance metrics of the devices like the CPU utilization, memory state, etc.

C. Northbound Service Interfaces

The API of the *Service Interface* is an integral part of system, as it defines to what extend the service is supposed to understand the monitoring tasks.

Notes on the Information Model: The information model, including the understanding of monitoring tasks, defines which role the service takes in the control-loop. In order to develop a viable monitoring service, it is crucial that the service users, here control applications, and the service itself have a common understanding of the shared information. Thus, a primary challenge is a common understanding of the monitoring information model. A monitoring service either serves only measurements to control applications without profound understanding of the metrics, or understands the measured information and provides derived state information to management applications, which do not derive the information further. Roughly speaking, there are two alternatives: either the monitoring system serves the management application the raw data without profound understanding of the metrics, or it is responsible for understanding the measured information and provides derived state information, in which case the management application does not derive the information further. In the latter alternative, when the monitoring system is responsible for analyzing the information on behalf of the control applications, this becomes an exhausting task, as for each task a bulletproofed definition is required which leaves no space for interpretation. In order to avoid such error-prone complexity, we limit the service in our work to conduct only simple measurements on behalf of the applications. However, we include simple preprocessing steps, which require a clear definition of the performed calculations, nevertheless, with very limited complexity. Furthermore, we support handing

Rhaban Hark, Nieke Aerts, David Hock, Nils Richerzhagen, Amr Rizk, and Ralf Steinmetz. *Reducing the Monitoring Footprint on Controllers in Software-defined Networks* in IEEE Transactions on Network and Service Management, Special Issue on Novel Techniques for Managing Softwarized Networks, Oct. 2018, ISSN: 1932-4537

over original OpenFlow statistic requests as they stand, without further interpretation.

As shown in Figure 2, SRR has two northbound interface endpoints to the control-plane: *(i)* a legacy synchronous request-response interface that supports task registration and their confirmation, *(ii)* a further interface that provides asynchronous event-based statistics for periodic measurements.

Task registration messages contain a full description of the task, including information such as the type (single or periodic measurement), the targeted metric, where to measure - the measurement point, the respective entity (flow specification, port/link description, ...), and the period time if applicable. As mentioned previously, a controller can also relay OpenFlow or SNMP requests without interpretation and modification. Furthermore, there are several flags and tolerances specified regarding the degree of freedom the *Convolver* has. At first, they specify if the measurement point is allowed to be changed to another. Despite that, it specifies the allowed modification of the measurement period time. Single measurement tasks specify a maximal staleness in order to allow cached measurements. We use JSON[1] as payload in UDP/IP packets for all communication between the controllers and SRR. An example of a task registration is shown in Listing 1:

Listing 1: Task registration message format.

```
{
 "controller": "10.10.10.100",
 "controller_port": 7702,
 "task": {
  "entity": {
   "dpid": 144115188075855874,
   "port": 1
  },
  "metric": "bandwidth",
  "period": 1000,
  "periodic": true,
  "threshold": 10000,
  "threshold_type": "delta",
  "aggregation_freedom": {
      "min_period": 500,
      "max_period": 1500,
  }
 },
 "type": "register_task"
}
```

In the listing above we see that every message specifies the sender, the port where its interface can be reached (`controller` and `controller_port`), and the type of the message (`type`). In the case above the message registers a new task. The `task` field contains task-specific properties such as the entity that should be measured. Here the `entity` field selects a switches port, thus, a link. However, an entity could also be a flow depending on the metric that is selected in the `metric` field. The fields `periodic` and `period` specify whether the task must be performed periodically and the corresponding period. As tasks can be aggregated, the `aggregation_freedom` defines to what extend SRR is supposed to change the task (here the degree of freedom defines an allowed range for the period length). The filtering mechanisms uses the `threshold` and `threshold_type` fields. In this task only values that differ relatively (`delta`) more than `10kBps` should be forwarded. Task registration confirmations and measurement result messages are constructed analog.

[1]http://json.com, accessed October 22, 2018

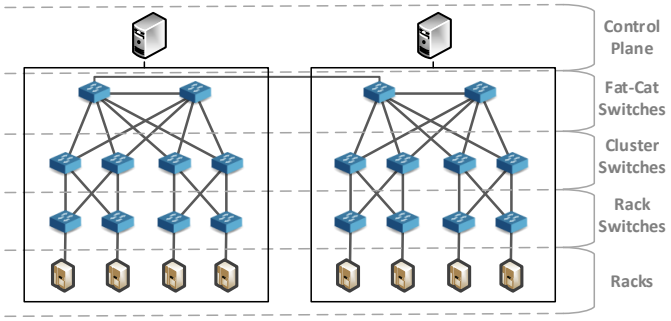


Fig. 4: Distributed datacenter topology used for the evaluation oriented at [22]. Each datacenter contains two *Fat-Cat* switches, four cluster switches for two clusters and two racks including two rack switches per cluster.

V. Evaluation

We developed a prototype of SRR to evaluate the expected resource saving goals. In the following, we first describe the prototype and the evaluation environment, then briefly the evaluated methodology, and, finally, the results of our investigation on the performance of SRR in different scenarios.

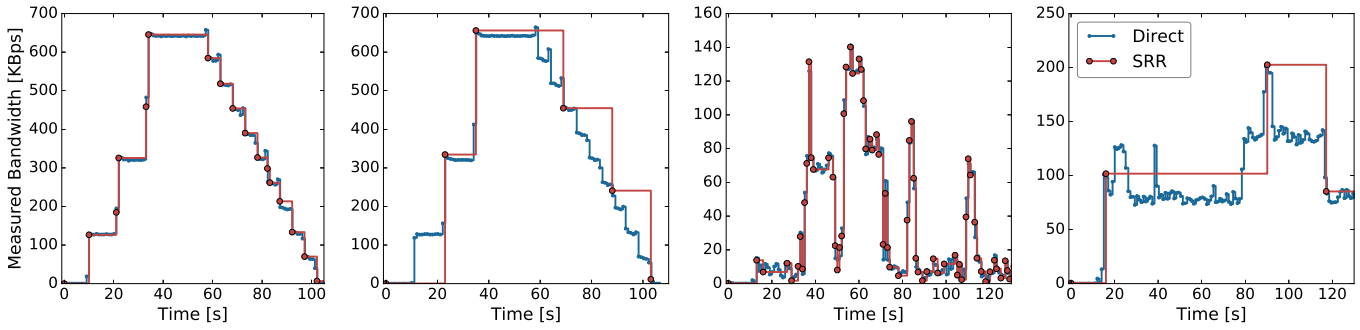
A. Prototype and Evaluation Environment

We conduct the evaluation using a simulative environment consisting of a virtual Mininet [21] network, two Floodlight[2] controllers, and the SRR prototype. The controllers each manage one part of our network exclusively and provide all prerequisites such as inter-domain routing. We encapsulate the developed prototype of SRR in an additional controller, which is capable of performing OpenFlow requests to the data-plane. The SRR controller registers as slave OpenFlow controller (cf. [18]), having only read capabilities.

The topology of choice imitates a distributed datacenter (DC) as depicted in Figure 4: the two subnetworks follow a typical, simplified DC topology following [22]. Each DC has two so-called *Fat-Cat* switches, which connect two clusters in each DC. All clusters have two cluster switches, and two racks where a single host abstracts the servers of a rack. A link between two Fat-Cat switch connects both datacenter networks.

For traffic generation, we rely on iperf[3]. In a first preliminary evaluation, we show a proof-of-concept using synthetic traffic, which increases step-by-step and again decreases step-by-step. In that scenario, we use the first rack as sender, and the last rack, placed in the opposite datacenter network, as receiver and vice versa. Furthermore, all other evaluations use more realistic datacenter traffic similar to [22], [23]. For this, we randomly select a sender and a receiver in the network. We pick a flow duration out of the zeta distribution with a scale parameter $s = 1.6$ with a maximum of 10^3 seconds to mimic

[2]http://www.projectfloodlight.org/, accessed October 22, 2018
[3]https://iperf.fr, accessed October 22, 2018



(a) Step-function; T = 25 KBps (b) Step-function; T = 200 KBps (c) DC traffic; T = 5 KBps (d) DC traffic; T = 100 KBps

Fig. 5: Comparison of bandwidth measurement using direct measurements of controllers versus measurements performed with SRR. Figures 5a and 5b show measurements for synthetically traffic following a step-function. In Figures 5c and 5d measurements with datacenter traffic as described in Section V-A.

roughly the flow duration shown in [23]. We use a normal distributed bandwidth with a mean of 0.5 Mbps and a standard deviation of 0.1 Mbps, so that the we observe light traffic mimicking application heartbeat or cyber-physical control traffic. Note that our experiments showed that increasing the absolute bandwidth does not oddly influence the metrics captured in this evaluation but rather results in a linear scaling of system parameters: the filtering threshold. The flow inter-arrival time is selected from an exponential distribution with factor $\lambda = 0.1$ in seconds inspired by [22]. Roughly summarized, we have a Poisson based flow arrival of randomly placed flows with a friendly bandwidth consumption around 0.5 Mbps while most of the flows are mice flows (duration slightly larger than 10ms) and few elephant flows (resulting from the zeta distribution) as common in datacenter networks. We use rather light rates to avoid biased measurements due to overloaded evaluation environments, however, the qualitative behavior remains the same. Despite the absolute flow throughput, the number of flows and the dynamics in the traffic affect the effectiveness of SRR, i.e. measurements of friendly traffic get filtered much more than bursty traffic using the same relative threshold. This effect can also be faced adapting the threshold system parameter according to the needs of the operator.

B. Measurement Methodology

The evaluation relies mainly on two aspects: *(i)* Accuracy of the measurements conducted when using SRR and *(ii)* the monitoring costs both at the controller and at the data-plane elements, respectively. We measure the first aspect using the absolute difference denoted *error* between the measurements performed by the controllers themselves and the measurements forwarded by SRR. We concatenate all measurements of all simulations to calculate the error in each measurement point. For this, during the evaluation the controllers continue measuring in addition to devolving measurement tasks to SRR. For the second aspect, the costs are measured using the number of statistic requests, which either the controllers or the data-plane elements, respectively, have to process in each simulation.

C. Event-based Statistic Provisioning Showcase

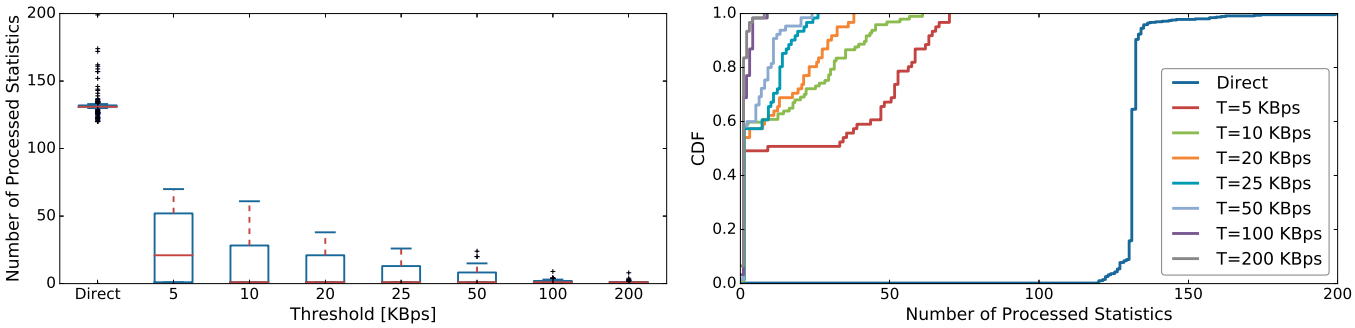
In the first evaluation, we compare measurements of the SRR system with measurements performed directly on the controllers (which we assume to have negligible inaccuracy). Figures 5a and 5b show exemplary measurements of the bandwidth on the inter-datacenter link. Figure 5a indicates that small thresholds lead to frequent measurement updates, which closely follow the directly performed measurements. Accordingly, Figure 5b shows for a comparably large threshold of 200 KBps that the measurements performed with SRR differ up the chosen threshold from the measurements performed on the controllers. However, the number of interruptions of the controller reduces in this synthetic example from 17 to 6 times.

Next, we consider light DC-like traffic and measurements at a randomly selected link, Figures 5c and 5d depict a comparable impression. On the one hand, depicted in Figure 5c, for a small threshold (5 KBps) both test series follow an almost identical curve. On the other hand, using a threshold of 100 KBps (Figure 5d), the measurements from SRR differ up to the given threshold from the direct measurements, however, require a significantly smaller number of updates.

In the following, we investigate this performance of SRR statistically. All measurements were conducted using DC-like traffic as described, measuring the bandwidth of a randomly selected link every second (frequency $f = 1/s$) for a duration of 130 seconds in each DC. All simulations were repeated at least 30 times.

a) Influence of Task Devolvement and Pre-Filtering on the Measurement Costs: The following analysis investigates the measurement costs in terms of statistic messages a controller needs to process. Figure 6a shows the number of statistic messages a controller needs to handle. If SRR is not used, thus, the controller directly measures the bandwidth, a number of around 130 statistic messages need to be processed. In addition, the controller needs to *request* statistics periodically from the network elements, which is not shown here. This is not the case when it devolves the monitoring task to SRR which performs the requests mandatory in OpenFlow. If we use SRR with a threshold of 5 KBps, we can observe a significant reduction of processed statistics on the controller

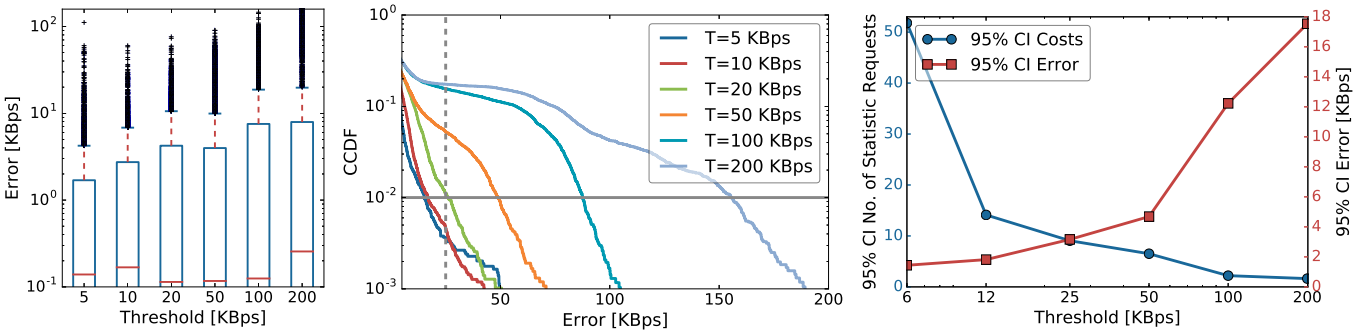
Rhaban Hark, Nieke Aerts, David Hock, Nils Richerzhagen, Amr Rizk, and Ralf Steinmetz. *Reducing the Monitoring Footprint on Controllers in Software-defined Networks* in IEEE Transactions on Network and Service Management, Special Issue on Novel Techniques for Managing Softwarized Networks, Oct. 2018, ISSN: 1932-4537



(a) Costs in terms of statistic requests conducted by the controllers with varying threshold.

(b) Distribution of costs in terms of statistic requests conducted by the controllers with varying threshold (T).

Fig. 6: Cost reduction analysis when using SRR's filtering mechanism creating an event-based stream of statistics.



(a) Difference between direct and SRR measurements.

(b) Distribution of difference between direct and SRR measurements

(c) 95% CI of mean cost and 95% CI of mean error as depicted in Figures 6a and 7a (logarithmic x-axis).

Fig. 7: Performance analysis when using SRR's filtering mechanism creating an event-based stream of statistics.

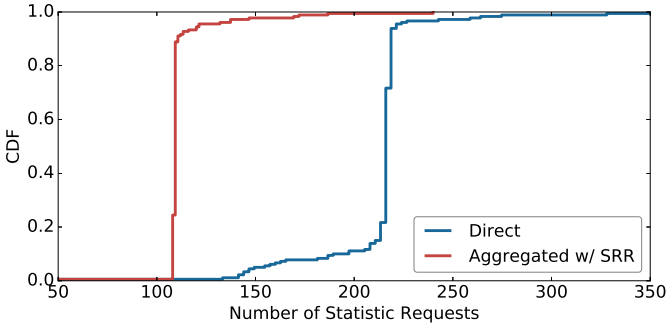
with a median around 25 and an upper quartile at ~70. If we further increase the threshold, hence, allow a higher difference between the current measurement value on the controller and the actual measurement, the number of statistics processed by controllers further decreases. Note that the median reaches a value close to 1 statistic message very quickly, because we select the measured link randomly from all available links in the DC. In combination with shortest-path routing and the used traffic model, the link utilization is low. If a link is not used, none of the measured values differs more than the threshold from the first measurement. This interesting case especially shows the strength of the system: If a monitoring task leads to measurements with no further information gain, SRR does not interrupt the controller for statistic processing. However, for all cases where there is traffic on the link, thus, changing bandwidth, the number of statistic messages decreases with higher thresholds as depicted in Figure 6a. Values of 100, or even 200 KBps have statistic numbers close to 0, thus, showing that bandwidth changes will not be captured by the monitoring if it specifies too high thresholds.

We note that in general tuning this threshold is a straightforward manner as it involves the trade-off of information accuracy to message costs (overhead). E.g. administrators could set up a threshold to adhere to some maximum deviation that is tolerated.

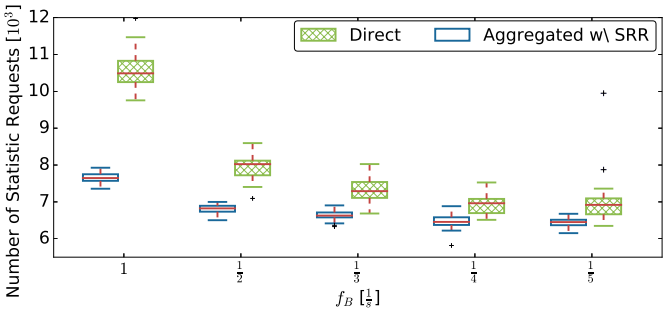
Figure 6b shows that behavior more detailed. For thresholds (T) above 5 KBps, more than 50% of the values are at one. Increasing the threshold shifts the distribution of numbers of statistic messages towards lower values, yielding less controller disturbance.

b) Influence of Pre-Filtering on the Measurement Accuracy: Applications using SRR give away accuracy limited by the allowed threshold. We analyze the accuracy using the difference between the measurements conducted with and without SRR, which we denote as *error* here. Figure 7a depicts the error in KBps for changing thresholds. It is observable that most measurements all have a median close to zero indicating minimal error. Analog to our cost analysis we find that this behavior is caused by the measurements of links with low utilization. Hence, the third quartile and the box's upper whisker become more interesting. We see an increasing manner of those values (not continuous, but generally observable) with an increasing threshold. Hence, the lower the threshold the less often SRR's measurements diverge highly from the *real* measurements. If we take, for instance, a threshold of 5 KBps, 75% of all values have an error below 2 KBps and most outliers below an error of 60 KBps. In contrast, a threshold of, e.g., 100 KBps has an upper quartile at 8 KBps and most outliers below 100 KBps.

In Figure 7b we see the distribution of the error in more



(a) Distribution of the number of statistic requests occurring on the data-plane elements when aggregating two monitoring tasks devolved to SRR versus directly measured from controllers.



(b) Cost in terms of statistic requests towards the data-plane: Controller A requesting with a fixed frequency of $f_A = 1/s$ while controller B has a varying request frequency f_B.

Fig. 8: Cost reduction analysis when using SRR's aggregation mechanism.

detail. Despite the behavior analyzed already in Figure 7a, it shows the reasonable thresholds for fixed errors. For example the mark at 20 KBps (vertical dashed gray) shows that thresholds below 20 KBps have in ~99.0% smaller errors than the selected threshold. For a fixed threshold of 50 KBps, we observe that still ~94.5% of all values are below an error of 20 KBps Thresholds of 100 KBps or 200 KBps lead to errors below 20 KBps in ~82% of all cases.

Furthermore, for this case marked with a solid gray line, we see, for instance, that for a threshold of 100 KBps, 99% of all values are under ~87 KBps. For thresholds of 5, 10, and 20 KBps, we see that 99% of all values are below 21, 13, and 11 KBps, respectively. This investigation shows that the measurement error in rare cases exceeds the given thresholds. Our analysis shows that this happens due to asynchronous measurements between the controller and SRR consulted for the error calculation, hence, it is an artifact of the evaluation methodology as the selected threshold limits the error.

D. Statistic Request Aggregation

The second part of the evaluation covers the performance of SRR when using the monitoring task aggregation mechanism. We investigate the performance in terms of saved statistic requests on the data-plane elements.

In the following, we examine two scenarios that have different impact on the expected aggregation benefit. In the first scenario, the two controllers each measure their outbound link, i.e. the shared link between their Fat-Cat switches. The system detects redundant measurement tasks and convolves them into a single task. Both controller advise SRR to measure with a period of $1s$, thus $f_A = f_B = 1/s$. Hence, we expect to halve the number of statistic requests on the data-plane elements.

In Figure 8a the distribution of statistic requests on data-plane elements accumulated over each run is shown. Thus, the distribution of the number of requests processed by the switches shown in the figure shifts to be centered around a smaller number, approximately half the number of requests required in the direct measurement case. Note that also the tails of the distribution are smaller indicating less variance.

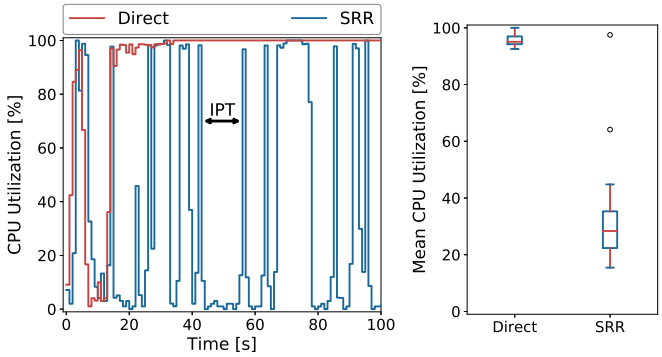
In a second analysis we consider a scenario, where both controllers collect information about all flows traversing their networks. Whenever a new flow arrives, the controllers register a flow bandwidth measurement task at the SRR interface. While doing so, they allow SRR to change the point of measurement such that the tasks from both controllers can be conducted at a single switch. It stands to reason that SRR aggregates measurements only for flows that traverse both networks. For the simulations the measurement frequency f_A is fixed to $1/s$ while f_B is varied equidistantly between $1/s$ to $1/5s$.

Figure 8b shows the results of the described experiments. The blue translucent boxes show the number of statistic requests dispatched to the data-plane elements when SRR aggregates statistic requests whenever possible, while the green hatched boxes show the number of requests when SRR is not used, thus, the controllers measure directly. When $f_A = f_B = 1/s$, namely the leftmost boxes, we observe a sharp falloff in the median of the statistic requests. Increasing the measurement period on the second controller, i.e., $f_B = 1/2s$, decreases on the one hand the number of statistic requests when measuring directly to ~¾ of its previous value as SRR dispatches only half the number of requests for controller B with lower measurement frequency (those that match requests of controller A). The number of requests when aggregating requests in SRR decreases accordingly as the number of requests decreases in total. Recall that the aggregation through SRR only impacts the flows that traverse both networks. Increasing the measurement period of controller B further lets the aggregation gain diminish. Hence, the aggregation mechanism performs better when the measurements periods are alike for all controllers as the absolute number of aggregated measurements increases.

E. Influence on Computational Overhead at Controllers

SRR targets to relieve the computational resources of the controllers as they do not have to process irrelevant information. Consider a scenario in which a controller executes a resource exhausting task whenever a new measurement arrives. In our evaluation we exemplary consider a network traffic prediction application based on the bandwidth consumption using Auto Regressive Integrated Moving Average (ARIMA) forecasting together with historic measurements. This forecasting application is for example helpful for dynamic and

Rhaban Hark, Nieke Aerts, David Hock, Nils Richerzhagen, Amr Rizk, and Ralf Steinmetz. *Reducing the Monitoring Footprint on Controllers in Software-defined Networks* in IEEE Transactions on Network and Service Management, Special Issue on Novel Techniques for Managing Softwarized Networks, Oct. 2018, ISSN: 1932-4537



(a) Exemplary measurement run.

(b) SRR reducing CPU utilization.

Fig. 9: Evaluation results for computational resources when conducting ARIMA-based bandwidth forecasting using statistic updates: CPU utilization sequence using direct measurements of the controller versus prefiltered information provided from SRR with threshold T=20 KBps.

network assisted over the top applications such as network assisted streaming solutions [24].

The forecasting application utilizes an ARIMA time series model based on the measured bandwidth values over a historic window. The application provides a prediction for the utilized bandwidth for an upcoming window of size 5 values. The ARIMA(o, d, n) model contains an autoregressive part of order o, a differencing, and hence removing trends, part with parameter d and a moving average part with n noise terms. For the sake of simplicity we resort here to a standard ARIMA model using the R forecast library [25] which automatically decides on the model parameters using an Information Criterion.

As mentioned, each controller calculates a forecast whenever a new measurement arrives. Given this resource exhausting exemplary processing of measurements, we show how SRR relieves the computation resources by filtering information. Figure 9a shows that the processing resources (*here:* CPU) of a controller[4] are fully utilized when each measurement is processed using a default frequency of once per second $f = 1/s$. In contrast, when we provide only bandwidth measurements that differ from the previous more than 20 KBps, we can observe that the controller only sporadically utilizes the CPU. Most of the time, the CPU remains unused, thus, the resources are available for other management tasks. In the figure we see that the inter-peak times, marked with IPT, follow a random pattern depending on the frequency of measurements forwarded to the controller. This times get smaller when SRR forwards more statistics, hence, for instance when setting a small threshold value between consecutive measurements.

Figure 9b depicts the mean CPU utilization performing direct measurements and using SRR. We find that SRR reduces the utilization from above 90% to a median of close to 30% and an upper quartile of ~45%.

[4]The controller was running on a virtual machine with *Ubuntu Xenial 16.04* with 2 GB memory and two CPU cores running up to 3.4GHz. Floodlight was running using *OpenJDK Runtime Environment* 1.8.0_181.

F. Limitations

In this subsection we describe limitations of our approach as partially mentioned already in previous sections. First, SRR does not provide a robust mechanism against failures of the system itself or it adds so far an unknown overhead to distribute its state if running in high-availability cluster deployment. Hence, it is not meant to fully replace monitoring capabilities of controllers, but provides an system that allows increasing the efficiency and we argue that it can be activated optionally.

Furthermore, the aggregation mechanism and the caching mechanism provide an increase of efficiency only under given assumptions: *(i)* Aggregation requires a certain degree of freedom when measuring. In the previous sections we mentioned some examples, e.g., aggregating flow bandwidth measurements may require the need to alter the measurement point. For the example of link bandwidth measurements, there is a higher chance of successful aggregation if the measurement period can be modified within an acceptable range. *(ii)* For metrics that are time-critical the caching mechanism is not reasonable. Yet, metrics that are stationary over longer time spans, in particular derived metrics that take averages, can use the caching mechanism to reduce monitoring costs.

Regarding the performance, the accuracy of the conducted measurements will not suffer using SRR as it uses the same measurement techniques as the controllers themselves. However, it introduces additional latency to retrieving the measurements, which we assume to be negligible. For single measurements this latency increases by the time needed for the task registration message, its processing at SRR, and the delivery latency of the measurement values from SRR to the controllers. Looking at periodic measurements, the latency only increases by the preprocessing time in SRR and the delay induced for measurement transmission from SRR to the controllers.

Last, using SRR requires no additional functionality on the data-plane, however, it requires the data-plane elements to be configured according to the agents that SRR uses. If SRR uses its OpenFlow agent, the switches need to accept SRR as an additional controller. For future work it is interesting to investigate to what extent further agents such as SNMP, but also other SDN protocols such as P4runtime in combination with P4 can be used to improve the system.

G. Evaluation Summary

Concluding the evaluations, we see first that we are able to significantly reduce the number of statistics processed by the controllers by devolving tasks to SRR, which only sends updates when required. At the same time, we conclude that the accuracy suffers only very little and in a reasonable range. The method is particularly strong when the measured values do not change significantly and SRR serves only outlier values with respect to historic measurements.

Furthermore, we highlight the proof-of-concept of the aggregation mechanism, which helps avoiding duplicate measurements of the same metric of an entity. We show that, de-

pending on the number of aggregated tasks, SRR significantly reduces the number of statistic requests to data-plane elements.

Lastly, we show how SRR reduces the computational resource consumption when each measurement must be processed with a non-trivial task, i.e. forecasting based on all given measurements.

VI. Related Work

The majority of approaches to unburden controllers focus on the distribution of the control-planes [10]. In this context, many works outline sophisticated approaches on how management can be given to a network of controllers running in parallel: [17], [26] highlight approaches where separate controllers work in collaboration while being responsible for different networks part. Furthermore, [26]–[28] propose systems for parallel controllers which can overtake control in times of failures. To further improve the scalability, Hassas Yeganeh et al. focus in [6] on hierarchical approaches to avoid flat control-planes. Such approaches enable scalability of a previous single point of failure. However, scaling out requires additional costly resources. Therefore, intelligent mechanisms to reduce monitoring costs, as proposed in this work, should be preferred.

Considering monitoring in the context of SDN, most monitoring approaches are built as application on top of the controllers communicating through the controllers interfaces with the data-plane devices (cf. [4]). These monitoring applications can leverage, for example, flow counter information to characterize data flows and enable fine-grained traffic engineering [29]. MonSamp [30] moves the monitoring endpoint out of the controller letting it communicate directly with the data-plane elements. Despite the collection, MonSamp stores also the analysis into an additional architecture element. Hence, the system completely moves monitoring capabilities out of the controller and provides fully processed information to the applications. SRR in contrast does not target to do the analysis of statistics, but reduces monitoring related costs by, e.g., providing only relevant information to monitoring applications. Rasley et al. propose an approach in [31] to devolve the collection and analysis of monitoring information to a dedicated collector. Subsequently, the collector creates an event-based information stream to the controller relieving its monitoring resource consumption. This approach takes the analysis of information out of the controllers, which we, again, explicitly retain in the control applications.

Cheikhrouhou et al. [32] proposed in 2000 that redundant measurement tasks (or with comparable characteristics) should be aggregated into single measurement tasks for legacy networks using SNMP. They show that the number of polls required for monitored reduces significantly. Extending this idea, we add a task aggregation mechanism for *different* types of monitoring tools, including SNMP, to the event-based reporting of measurements in our system. Furthermore, [14] proposes to aggregate the results of measurements, but not the requests. Hence, based on different measurements, an aggregated report is produced. Besides that, their system can report objective-based measurements, e.g., SLA violations.

The SUMA [33] approach is the closest to our approach. SUMA, also published as SuVMF [34], is a middlebox that overtakes management tasks for controllers acting as a proxy for various tasks. The system consists of a common processing module, a transformation and adaptation module, and a dynamic event monitor module. The modules allow the mitigation of packets, events, alarms, statistic, and further messages. While we can compare the statistic filtering mechanism with the filtering in this work, we note that SUMA takes the complexity of controllers and moves them into an additional element. In contrast to our system, SUMA *shifts most aspects of the scalability problem* of controllers to the new entity, as it overtakes a large set of management tasks to its new component. Additionally to many other tasks, SUMA removes in particular the monitoring analysis out of the controllers and places it into the new component. Hence, also in contrast to our system, SUMA incorporates network control and analysis, thus, actively influences the network management, which we retain explicitly on the controllers in this work.

The OpenFlow 1.5 specification provides optional functionality for event-based statistic collection. It supports pushing statistics from the switches to the controller whenever a certain field reaches a given threshold (either once or on every multiple). This functionality allows reducing monitoring costs further as requests can be neglected in future OpenFlow scenarios. However, this is limited to the fields OpenFlow supports (e.g. flow entry duration or byte count) and does not yet support correlating thresholds, i.e. periodically asking if a byte count reached a threshold in the mean time.

VII. Conclusion

In this work, we developed a system to reduce monitoring overhead on controllers and data-plane elements. The system, denoted SRR, optimizes statistic retrieval by optimizing statistic requests and statistic provisioning. On the one hand, the optimization of statistic requests achieved through aggregation of similar monitoring tasks reduces the number of requests that the data-plane elements have to process, thus, it reduces the load on elements such as switches. On the other hand, SRR optimizes the provisioning of statistics through filtering, i.e., it only forwards relevant information that require a management applications' reaction to controllers. We discuss different deployment modes of the logically centralized system, the architecture of the component, and its interfaces. In our evaluation that simulates datacenter behavior, we show that the filtering mechanisms included in the system allows a significant reduction of the number of statistics requiring processing by a controller. At the same time, we show the trade-off between system control parameters and the accuracy of the monitored information. Furthermore, the evaluation shows that we are able to reduce the number of statistic requests processed by switches when SRR aggregates monitoring tasks.

We plan to investigate an automatic adaptation of the monitoring system parameters, e.g., using online machine learning tools, for future work.

Rhaban Hark, Nieke Aerts, David Hock, Nils Richerzhagen, Amr Rizk, and Ralf Steinmetz. *Reducing the Monitoring Footprint on Controllers in Software-defined Networks* in IEEE Transactions on Network and Service Management, Special Issue on Novel Techniques for Managing Softwarized Networks, Oct. 2018, ISSN: 1932-4537

REFERENCES

[1] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling Flow Management for High-performance Networks," in *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM) Conference*, 2011, pp. 254–265.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[3] R. Hark, N. Richerzhagen, B. Richerzhagen, A. Rizk, and R. Steinmetz, "Towards an Adaptive Selection of Loss Estimation Techniques in Software-defined Networks," in *Proceedings of the IFIP Networking (NETWORKING)*. IEEE, 2017, pp. 1–9.

[4] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "PayLess: A Low Cost Network Monitoring Framework for Software Defined Networks," in *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2014, pp. 1–9.

[5] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "FlowSense: Monitoring Network Utilization with Zero Measurement Cost," in *Proceedings of the Passive And Active Measurement Conference (PAM)*, 2013, pp. 31–41.

[6] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications," in *Proceedings of the ACM SIGCOMMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2012, pp. 19–24.

[7] P. Phaal and M. Lavine, "sFlow Version 5," 2004. [Online]. Available: http://sflow.org/sflow_version_5.txt

[8] B. Claise, B. Trammell, and P. Aitken, "RFC 7011: Specification of the IP Flow Information Export (IPFIX)," 2013. [Online]. Available: https://tools.ietf.org/html/rfc7011

[9] D. Kreutz, F. M. Ramos, and P. Verissimo, "Towards Secure and Dependable Software-defined Networks," in *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2013, pp. 55–60.

[10] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically Centralized?: State Distribution Trade-offs in Software Defined Networks," in *Proceedings of the ACM SIGCOMMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2012, pp. 1–6.

[11] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, "OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks," in *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2014, pp. 1–8.

[12] Y. Zhang, "An Adaptive Flow Counting Method for Anomaly Detection in SDN," in *Proceedings of the ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2013, pp. 25–30.

[13] A. Krishnamurthy, S. P. Chandrabose, and A. Gember-Jacobson, "Pratyaastha: An Efficient Elastic Distributed SDN Control Plane," in *Proceedings of the ACM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2014, pp. 133–138.

[14] Y.-J. Lin and M. C. Chan, "A scalable monitoring approach based on aggregation and refinement," *Journal on Selected Areas in Communications*, vol. 20, no. 4, pp. 677–690, 2002.

[15] R. Sherwood, G. Gibb, K. kiong Yap, M. Casado, N. Mckeown, and G. Parulkar, "FlowVisor: A Network Virtualization Layer," Tech. Rep., 2009.

[16] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow, "OpenVirteX: Make Your Virtual SDNs Programmable," in *Proceedings of the ACM SIGCOMMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2014, pp. 25–30.

[17] K. Phemius, M. Bouet, and J. Leguay, "DISCO: Distributed multi-domain SDN controllers," in *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2014, pp. 1–4.

[18] B. Pfaff, B. Lantz, and B. Heller, *OpenFlow Switch Specification, Version 1.3.0*, Open Networking Foundation, 2012.

[19] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "RFC 1157: A Simple Network Management Protocol (SNMP)," 2004. [Online]. Available: https://tools.ietf.org/html/rfc1157

[20] B. Claise, "RFC 3954: Cisco Systems NetFlow Services Export Version 9," 2004. [Online]. Available: https://tools.ietf.org/html/rfc3954

[21] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-defined Networks," in *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)*, 2010, pp. 19:1–19:6.

[22] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the Social Network's (Datacenter) Network," in *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2015, pp. 123–137.

[23] T. Benson, A. Akella, and D. A. Maltz, "Network Traffic Characteristics of Data Centers in the Wild," in *Proceedings of the ACM SIGCOMM Internet Measurement Conference (IMC)*, 2010, pp. 267–280.

[24] D. Bhat, A. Rizk, M. Zink, and R. Steinmetz, "Network assisted content distribution for adaptive bitrate video streaming," in *Proceedings of the 8th ACM on Multimedia Systems Conference (MMSys)*, 2017, pp. 62–75.

[25] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, 2014. [Online]. Available: http://www.R-project.org/

[26] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A Distributed Control Platform for Large-scale Production Networks," in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010, pp. 351–364.

[27] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: Towards an Open, Distributed SDN OS," in *Proceedings of the ACM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2014, pp. 1–6.

[28] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. R. Kompella, "ElastiCon; an elastic distributed SDN controller," in *Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2014, pp. 17–27.

[29] Z. Bozakov, A. Rizk, D. Bhat, and M. Zink, "Measurement-based flow characterization in centrally controlled networks," in *Proc. of IEEE INFOCOM*, 2016, pp. 1–9.

[30] D. Raumer, L. Schwaighofer, and G. Carle, "MonSamp: A Distributed SDN Application for QoS Monitoring," in *Proceedings of the IEEE Federated Conference on Computer Science and Information Systems (FedCSIS)*, 2014, pp. 961–968.

[31] J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, and R. Fonseca, "Planck: Millisecond-scale Monitoring and Control for Commodity Networks," in *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2014, pp. 407–418.

[32] M. Cheikhrouhou and J. Labetoulle, "An efficient polling layer for SNMP," in *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2000, pp. 477–490.

[33] T. Choi, S. Song, H. Park, S. Yoon, and S. Yang, "SUMA: Software-defined Unified Monitoring Agent for SDN," in *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2014, pp. 1–5.

[34] T. Choi, S. Kang, S. Yoon, S. Yang, S. Song, and H. Park, "SuVMF: Software-defined Unified Virtual Monitoring Function for SDN-based Large-scale Networks," in *Proceedings of the ACM Conference on Future Internet Technologies (CFI)*, 2014, pp. 1–6.

Rhaban Hark (rhaban.hark@kom.tu-darmstadt.de) received the master's degree in electrical engineering and the master's degree in information system technology both in 2015 at the Technical University of Darmstadt with the thesis "Dynamic and Transparently Discoverable Software-Defined Multicast in ISP Environments". He is currently pursuing his Dr. degree with the Multimedia Communications Engineering Lab, Technical University of Darmstadt, where he has been a Research Assistant since 2015. His current research focuses on distributed monitoring approaches in the context of Software-defined Networks.

Nieke Aerts (aerts@infosim.net) is a Senior Consultant R&D at Infosim GmbH & Co. KG. Previously, she worked as a research assistant at the Chair of Distributed Mathematics at the Institute of Mathematics of TU Berlin, where she finished her Dr. degree in 2015. Her current main research interests are in the unified network and services management of mixed SDN/NFV, IoT, and legacy infrastructures.

David Hock (hock@infosim.net) is Director of Research at Infosim GmbH & Co. KG and is leading the Infosim(R) research activities. Previously, he worked as a research assistant at the Chair of Communication Networks at the Institute of Computer Science at the University of Würzburg, where he finished his Dr. rer. nat. degree in 2014. His current main research interests are in the unified network and services management of mixed SDN/NFV, IoT, and legacy infrastructures.

Nils Richerzhagen (nils.richerzhagen@kom.tu-darmstadt) is a PhD student at the Multimedia Communications Lab (KOM) at the Technische Universität Darmstadt. Under the guidance of Prof. Dr. Silvia Santini Nils obtained his Master's degree in 2014 with the thesis "Adaptive Monitoring for Mobile Networks in Challenged Environments". His current research interests are on distributed systems and adaptive monitoring systems including communication protocols within his work in the collaborative research centre MAKI.

Amr Rizk (amr.rizk@kom.tu-darmstadt.de) received his doctoral degree (Dr.-Ing.) from the Leibniz Universität Hannover, Germany, in 2013. He has been a Post-Doctoral research fellow with the Department of Computer Science at the University of Warwick, UK in 2014 and with the University of Massachusetts (UMass), Amherst in 2015. Currently, he heads the Adaptive Overlay Communications group at the Multimedia Communications Lab at the TU Darmstadt, Germany. Amr received the best paper award at ACM/USENIX Middleware 2017 and the excellence in DASH Award at ACM MMSys 2016. He has been TPC Area and Track Chair for multiple conferences such as IEEE MIPR and ACM MMSys. His research activities span the areas of performance evaluation of communication networks, stochastic modeling, software-defined networks and IoT.

Ralf Steinmetz (steinmetz.office@kom.tu-darmstadt.de) is full professor at Technische Universität Darmstadt, Germany. In Darmstadt, he is the head of the Multimedia Communications Lab (KOM) as well as the Hessian Telemedia Technology Competence Center httc; see www.kom.tu-darmstadt.de. Together with more than 30 researchers, he works towards his vision of "seamless adaptive multimedia communcations". He has contributed to over 900 refereed publications; he is a Fellow of the IEEE, the ACM, VDE ITG, and the GI.