Rhaban Hark, Divyashri Bhat, Michael Zink, Ralf Steinmetz, Amr Rizk. *Preprocessing Monitoring Information on the SDN Data-Plane using P4* In: Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), p: 1–6, IEEE, November 2019.

Preprocessing Monitoring Information on the SDN Data-Plane using P4

Rhaban Hark*, Divyashri Bhat†, Michael Zink†, Ralf Steinmetz*, Amr Rizk*‡
* Multimedia Communications Engineering Lab, Technische Universität Darmstadt, Germany
† Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, USA
‡ Institut für Mess-, Regel- und Mikrotechnik, Universität Ulm, Germany

***Abstract*—Network management applications such as routing, load-balancing, or traffic forecasting, require up-to-date state information about the underlying data-plane. However, it is well known that data-plane measurements contain redundant information. In this work, we propose an approach that estimates how informative data-plane measurements are for control-plane applications that operate on such information. Using programmable data-planes, we present a novel approach on how the decision on forwarding data-plane measurements can be taken at network switches, and how this aids in filtering irrelevant monitoring information to save the controller's computational and networking resources.**

I. Introduction

Monitoring the state of a communication network constitutes one of the major tasks of network management. It sets the basis for continuous adaptations that are inevitable in today's dynamic networks. In the context of Software-Defined Networking (SDN), a logically centralized controller takes care of continuously fetching statistics from the data-plane elements. Accordingly, the controller provides the measured information in a raw or aggregated form to control-plane (network management) applications through its northbound interface.

Due to the potentially large number of network elements and the diversity of useful network state information, the number of statistics that may be periodically requested is very large. Processing a large number of statistics can create massive loads on controllers [7], [8], leading to intuitive approaches such as scaling out the control-plane [9], [12]. However, adding more resources is costly and it essentially does not solve the inherent problem. A different approach is letting network monitors collect statistics using sampling or aggregation. Sampling allows to take representative snapshots into account and infer the full state from the available sub-information [5]. Aggregation may provide the full state of a network, but loses detailed information [13]. We claim that even when using sampling-based or aggregated statistic collection, existing approaches often lack efficiency by introducing unnecessary measurement overhead. Newly collected data does not always provide additional knowledge to network management applications or analysis functions. Therefore, in this work, we propose to preprocess requested statistics on the data-plane to ensure its importance/informativeness to the application (residing north of the SDN controller) before delivery. In information theory, we find a similar approach where estimates of the *age of information* allow avoiding the transmission of stale information [4].

Recently, the authors proposed a middlebox for SDNs denoted SRR [6] to filter out redundant network monitoring statistics. Filtering was conducted based on the relation of the newly measured statistic to some predefined threshold or to the previously delivered statistic (measurement point). However, in contrast to relying on predefined thresholds for individual applications, we propose a new approach that has the goal to ensure that the newly measured statistic is only delivered to the analysis application if it significantly impacts the quality of the analysis. We do so by using an estimation technique to learn how much a measurement improves the quality of the running network management application. Note that this estimation process is application specific, i.e., the process gets more involved when the monitoring information is relayed to different network management applications carrying out different tasks such as anomaly detection, load balancing, and traffic prediction. Based on the learned training set, our preprocessing step executed on the data-plane estimates the improvement for each requested statistic and transmits only data which represent a significant improvement.

With respect to efficiency, we aim to preprocess measurements at the earliest stage possible, thus, we tap into switches before they dispatch requested statistics updates where we rely on existing state-of-the-art tools. As programmable data-planes gained popularity in the recent years, programmable P4 switches [2] provide a suitable starting point for our approach. We implement the preprocessing using only mechanisms provided in the P4 language that can be applied to any P4 switch.

Several approaches that use preprocessing to optimize the data-plane to control-plane information stream have been published in the past. In the field of distributed control planes Kandoo [7] proposes to implement the control-plane in hierarchical layers, so that lower layers process information that is not relevant for higher layer controllers. Such an architecture can be used to preprocess monitoring information before delivering it to some top-level controller. The approach named SUMA in [3], introduces a middlebox between the data-plane and control-plane to preprocess raw statistical information to provide filtered, aggregated, classified, and prioritized information. In contrast to both approaches, we target to preprocess monitoring information at the earliest stage possible, thus, within the data-plane. A more related

Rhaban Hark, Divyashri Bhat, Michael Zink, Ralf Steinmetz, Amr Rizk. *Preprocessing Monitoring Information on the SDN Data-Plane using P4* In: Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), p: 1–6, IEEE, November 2019.

approach was presented by Popescu et al. [10] to involve programmable data-plane elements, however, to detect heavy hitters using P4 to offload the control-plane.

The remainder of the paper is structured as follows: Section II describes the architecture of our approach before showing the detailed design of the filtering mechanism that explicitly considers the network management application. Limitations of the approach are discussed in the end of that section. In Section III, we describe and evaluate our prototypical implementation before concluding the paper in Section IV.

II. Architecture and Filter Design

In the following, first, we briefly describe how we integrate our approach in existing networks before illustrating the designed filtering mechanism, and, finally, discussing limitations the system might face.

A. Architecture Overview

In this work we refrain from using additional network elements as proposed by the discussed related works. Consequently, we target to preprocess measurement information on the data-plane elements, so that we can filter out measurements with limited gain for the application as early as possible. Figure 1 shows a schematic of the architecture: The upper part of the figure shows the controller that is connected to the data-plane elements, namely the switches. On the control-plane, an *Auto Regressive Integrated Moving Average (ARIMA)* [11] bandwidth forecast application serves as exemplary application consuming bandwidth measurements from the switches via the controller. The *Learner* uses the input measurements and the forecast output to learn how certain measurements influence the quality of the forecast. This knowledge is transferred later as a set of parameters to the switch, which does a a-priori estimation of the quality improvement before a measurement is transferred to the control-plane. Consequently, it forwards only measurements which presumably improve the forecast's quality significantly. As a result, measurement with a small information gain *do not generate transmission costs and controller disturbance*.

B. Filter Design

In the following, we will continue to consider the forecasting application. However, note that the described procedures are easily mapped to different network management applications such as anomaly detection or load balancing. We consider that the quality of the bandwidth forecast application depends directly on the dynamics of the forecast value. Intuitively, if the bandwidth is rather stable, the quality of the forecast does not improve significantly when a new measurement is provided, however, if the bandwidth is volatile, the forecast will be closer to the true future bandwidths if the newest measurements are available. We use linear regression to learn the impact of measurements on the improvement of the forecast. For this particular application simply filtering measurements that are close to previous measurements is a viable alternative, yet, with an eye on generality, the regression is a flexible solution also applicable in other use cases.

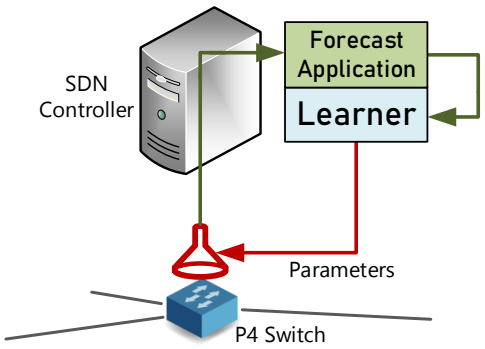


Fig. 1: Architecture of the preprocessing: A linear regression learns its parameters from historic values of a forecast and pushes them to the data-plane elements where they are used to estimate the importance of measurements and filter irrelevant measurements.

A linear regression is a method to estimate a dependent variable $\hat{i}$ (here, improvement of the forecast) from multiple independent variables $\mathbf{f} = [f_0 \ \dots \ f_n]$ (*features*, here, dynamics of the bandwidth). To do so, it seeks the parameter vector $\theta = [\theta_0 \ \theta_1 \ \dots \ \theta_n]^T$ (*output of the learner*), denoted bias term (θ_0) and features weights ($\theta_1 \dots \theta_n$), by minimizing the mean square error of Equation (1) for a given training set including sample features and corresponding improvements (*input to the learner*).

$$\hat{i} = \theta^T \cdot \mathbf{f} = \theta_0 \underbrace{f_0}_{=1} + \theta_1 f_1 + \dots + \theta_n f_n \tag{1}$$

After the learner generates θ, it can be used to estimate the improvement $\hat{i}$ using given features $\mathbf{f}$ also using Equation (1).

a) Finding suitable features:

Before showing our feature selection, we have to define the value we target to influence, namely the improvement of the forecast:

$$i = |\underbrace{|bw_{true} - bw_{fc_{n-1}}|}_{\text{error filtered}} - \underbrace{|bw_{true} - bw_{fc_n}|}_{\text{error normal}}| \cdot 1Bps^{-1} \tag{2}$$

The latter part "*error normal*" of Equation (2) includes the difference between the true bandwidth and the latest forecast (bw_{fc_n}). The first part "*error filtered*" shows the difference between the true bandwidth and the previous forecast ($bw_{fc_{n-1}}$), i.e., the forecast without taking the latest measurement into account. If the error of the forecast given the newest measurement is much smaller than the error of the forecast without the newest measurement, the improvement is significant. Note that there is only little improvement by taking the new measurement value if both errors are equal. The case that there is a negative improvement, thus, the old forecast was better than the current is possible. By always calculating the absolute improvement we take the conservative approach that forwards measurements also if their actual improvement

Rhaban Hark, Divyashri Bhat, Michael Zink, Ralf Steinmetz, Amr Rizk. *Preprocessing Monitoring Information on the SDN Data-Plane using P4* In: Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), p: 1–6, IEEE, November 2019.

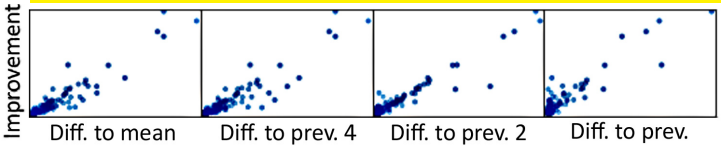


Fig. 2: Selected features to represent the dynamics of the bandwidth (x-axis) that linearly influence the application quality improvement (y-axis). Features describe differences between the new measurement and the mean of the previous as well as the sum of differences to the last four, two, and last measurement, respectively.

is unclear. Dividing the equation by $1Bps$ removes the unit as it is not meaningful for the improvement.

Next, we mainly concentrate on the differences between the latest bandwidth measurements to represent the dynamics of the bandwidth. Out of a pool of 12 potential features, we selected the following four features with the highest correlation factor with the improvement in preliminary tests (cf. Figure 2): *(i)* the difference of the latest bandwidth to the mean of the previous four samples; *(ii)* the sum of differences between the latest bandwidth and each of the previous four bandwidths; *(iii)* the sum of differences between the latest bandwidth and each of the previous two bandwidths; and *(iv)* the difference between the latest bandwidth and the previous bandwidth. Although these features partially depend on each other, each provides additional valuable information for the improvement estimation. Figure 2 shows the linear dependency between each of the features and the applications quality improvement. For each feature on the x-axis it is observable that the higher the difference to previous values is – what we consider dynamic – the higher is the absolute improvement.

b) Training phase and parameter injection:
In order to find the feature weights θ, the learner must be fed with a training set. To collect this set, whenever a new bandwidth measurement from a switch arrives at a controller, the controller pushes the value to the forecast application, which then calculates the next forecast. Furthermore, the value is used to check the quality of the previous forecast in terms of improvement i (cf. Equation (2)) against the forecast prior to that. Furthermore, based on the measured bandwidth, the controller generates all four features $\mathbf{f}$. The features $\mathbf{f}$ in combination with the improvement i serve as input to the learner (training set).

After the training phase, the controller calculates the feature weights as described earlier by minimizing the mean square error for Equation (1) using the available data. In order to estimate the forecast improvement for a new measurement, Equation (1) with the newly available feature weights $\theta_0 \ \ldots \ \theta_n$ must be calculated. As we decide to do this in the data-plane, the controller sends the weights to the switch.

Switches in SDN environments traditionally follow a match/action paradigm with a fixed set of parameterizable actions. As we require a custom functionality in the switch, this traditional match/action approach is not suitable. Therefore, we decided to take advantage of newly proposed programmable P4 switches [2] that allow us to program their behavior and processing pipeline. Note that we are able to save simple state information in the switches using registers, which we use to store the feature weights, the bias term, and a threshold. The controller sends the weights to the switch using a custom protocol that can be understood by the P4 program installed on the switch. The switch reads the weights and saves them in allocated registers. In addition to the feature weights, we store a threshold value that marks the significance of the calculated improvement estimation. If the improvement is estimated to be significant, the switch dispatches the measurement otherwise it is dropped. Note also that storing the threshold as register value allows it to be configurable.

c) Filtering phase:
After the controller injects the feature weights into the switch, it also activates the preprocessing, e.g., by modifying the performed action on a bandwidth statistic request. During the filtering phase, the switch first calculates the required features whenever a bandwidth statistic request reaches the switch. For this, it stores the last four historic bandwidth measurements as they are required to estimate the bandwidth dynamics represented with the features. Each feature is multiplied with the corresponding feature weight and added to the bias term as described in the beginning of Section II-B. The result of the linear regression is the estimated forecast improvement when including the latest measurement. The estimated improvement is compared to the configured threshold to decide whether to push the bandwidth measurement as a response to the request from the controller or to skip it and wait for the next request. If the mechanism decides that the measurement does not significantly improve the forecast, the controller assumes that the previous bandwidth forecast is still valid.

d) Linear regression in the data-plane using P4:
Next we describe how we implement the linear regression as a P4 program. P4 supports a very limited number of operations excluding basic operations such as multiplications and division.[1]

First, as our method requires preprocessing in the data-plane we cannot fetch statistics through the control channel of the switch as done traditionally with protocols such as OpenFlow and P4Runtime. As Figure 3 shows, messages coming through the control channel of the switch (shown as green arrow) are handled using the switches vendor-implemented fixed logic. Changing that behavior would lead to the development of a new switch with custom control logic, which we explicitly excluded. Our goal is to provide a method that is usable with all P4 switches using only out-of-the-box functionality provided by P4. Using P4, we can program the behavior of the switch for packets arriving through the data/forwarding channel. Therefore, we use a dedicated channel that connects the controller to a reserved data-plane port of the switch

[1]Note that version p4_16 supports multiplications, however, existing hardware switches running p4_16 only support multiplications with integers being the power of two, which is not usable in our context - we did not consider custom functionality of single switches implemented as P4 *extern objects*.

Rhaban Hark, Divyashri Bhat, Michael Zink, Ralf Steinmetz, Amr Rizk. *Preprocessing Monitoring Information on the SDN Data-Plane using P4* In: Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), p: 1–6, IEEE, November 2019.

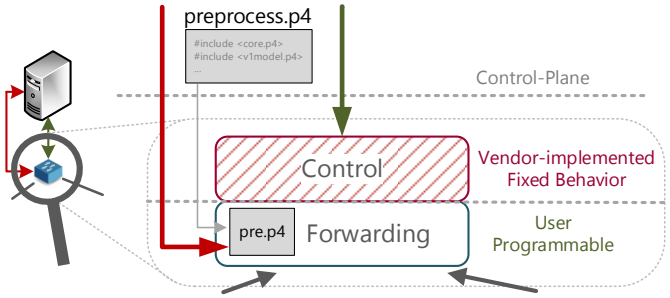


Fig. 3: Abstract switch architecture divided in control and forwarding layer. The control layer serves as interface to the control-plane and maintains that communication. The forwarding layer behaves according to the installed P4 program whenever packets arrive.

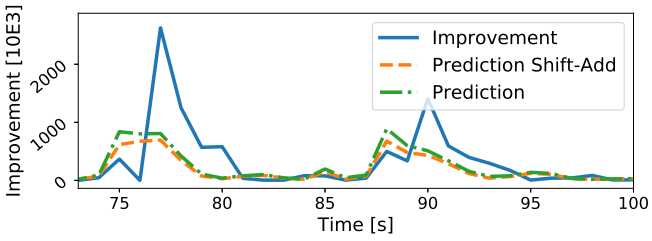


Fig. 4: The prediction using the target specific P4 (v16) multiplication is shown in the dashed dot-dashed green line. Estimates of the improvement are shown as solid blue. The prediction using only shift-and-add operations, that is shown as orange line, has comparable accuracy to the P4 (v16) case.

(shown in red). Packets arriving at that port are handled using the user programmable processing pipeline injected as a P4 program. Using the programmable logic to request statistics allows us *(i)* to directly send the bandwidth data instead of plain counters or registers and *(ii)* add preprocessing steps such as calculating a linear equation and filtering.

The processing pipeline and, thus, the behavior of the switch is programmed and injected in the switch during its startup. In the first phase, where no preprocessing is performed, the switch calls an action for incoming bandwidth requests that fetches the current counter for the inquired link as well as the previous. The switch sends the subtraction of the values plus timestamps for both values representing the bandwidth measurement. Note that the switch does not calculate the final bandwidth as it does not support divisions which we discuss in the limitations in Section II-C.

After the learning phase, the controller injects the feature weight parameters in the switch using our custom protocol. The switch stores the parameters in registers that can then be easily accessed. Now, the controller also configures the decision threshold to be reached by the estimated application improvement. The controller also changes the called action for new bandwidth requests so that preprocessing is activated.

With active preprocessing, the action called on a bandwidth request includes the following steps:

1) The action begins with reading of the current counter and historic counter values to calculate the bandwidths.
2) It shifts the historic counter registers so that it removes the oldest and stores the latest bandwidths.
3) The switch subtracts the counter values to estimate the latest and historic bandwidths.
4) Additionally, it reads the bias term and all feature weights from the registers.
5) In step 5, the switch calculates the features representing the bandwidth dynamics given in Paragraph *(a)* based on the previously fetched historic bandwidths.
6) With the bias term, feature weights, and the features, it calculates the linear regression using Equation (1). We provide details on this step in the following.
7) The result of the formula calculated in the previous step is the estimated forecast improvement for the current measurement. In this step, the action compares this improvement with the decision threshold configured by the controller. If the improvement is large enough, it forwards the measurement to the controller normally by pushing it to the port where the bandwidth requested originated from, otherwise, the action is finished.

In Step 6, the switch calculates Equation (1). As the support for multiplications is strongly target dependent in P4, we retain generality of the approach by using shift-and-add operations to map multiplications [1]. As known from processor implementations, shifting one bit left is a multiplication with two, shifting by 10 a multiplication with $2^{10} = 1024$. We use this method for each multiplication of a feature with its weight. Note however, that this method is only usable for integer multiplications. On the one hand, bandwidth values based on counters are solely integers, however, on the other hand, the bias term and feature weights are potentially decimals. A solution to this problem is to multiply the weight decimals by a factor and cut the tail. As a result, the estimated improvement is accordingly shifted. By shifting the decision threshold as well, the condition is still valid. In the following, we also investigate to what extent the estimation of the improvement becomes inaccurate. In Figure 4 we see that the estimated improvement using only integers and shift-and-add multiplication instead of real multiplications have only minor influence on the accuracy: The dot-dashed green line predicts the true improvement shown in solid, blue. The dashed orange line, showing the improvement estimation using solely the rough shift-and-add multiplications, follows the normal prediction closely. Note that this figure does not represent the overall accuracy of the prediction but only the similarity of both prediction approaches. Also note that we use a factor of 10 for the results in the figure so that larger factors provide even more accurate results. In the remainder of this work we use a multiplication factor of 1000. We did not experience a significant change in CPU utilization of the used P4 software switch using the built-in multiplication and the shift-and-add method: Using shift-and-add reduced the mean and CI_{95}

Rhaban Hark, Divyashri Bhat, Michael Zink, Ralf Steinmetz, Amr Rizk. *Preprocessing Monitoring Information on the SDN Data-Plane using P4* In: Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), p: 1–6, IEEE, November 2019.

utilization by 0.5% and 0.49%, respectively.

C. Limitations

The design we introduced in this section, faces some limitations. First of all, switches are designed to process packets at line-rate. Therefore, preprocessing on switches must be used with caution. The complexity of the features directly influences the processing complexity within the switch. Furthermore, when requesting a multitude of statistics, the computational resource usage similarly arises. In auxiliary investigations we did not find any impact on the switch performance (in terms of CPU utilization using a software switch) due to the negligible number of statistic requests compared to production traffic packets.

In addition, the proposed solution is vulnerable against slow, but constantly changing bandwidths as the number of considered previous measurements is limited. Potential approaches include the reflection of skipped measurements in the feature set or a fixed minimum update interval.

Note that the direct use of bandwidth data consumes two (byte-)counters per link. Furthermore, for every historic value we use in the features, we need one more counter. In addition to this, for every counter we enable preprocessing for, we need to store the bias term, feature weights and the decision threshold. P4 allows simple management of counters and registers, respectively, however, the memory consumption for preprocessing increases from two to eleven 64-bit counters per link (with four features).

As already described in the previous subsection, we do not calculate the bandwidth, but instead send the difference of counters plus the time difference. In our configuration we fetch counters every second, thus, we roughly handle Byte/s ignoring time differences due to jitter and changing processing time. Constantly fetching statistics faster or slower does not affect the solution, however, it is worth noting. Nevertheless, the design is currently not able to handle changing statistic fetching frequencies.

III. Prototype and Evaluation

In this section, we first describe the prototypical implementation of our preprocessing in P4. Furthermore, we describe the implementation and test environment and some evaluation results.

A. Prototypical implementation & test environment

The prototypical implementation consists of a P4 switch, the controller, and the forecast application. The controller in this case fills the switch with some routing rules for testing. It is based on Python and uses P4Runtime for the communication with switches. For the sake of simplicity, we implement the forecast application as a simple Python application within a *monitoring host*. Note that the application could also be placed within the controller. The monitoring host communicates with the switches through the data-plane interface as described in Section II-B using our custom protocol. The monitoring application calculates an ARIMA-based bandwidth forecast that uses *R*'s forecast library[2] and *rpy2* as adapter between Python and R. The core of the prototype is the P4 switch which provides generic L2 routing using routes injected from the controller. For each routing rule the switch maintains a cumulative byte counter that we use for our purposes. Despite that generic functionality, we add actions that are called whenever the ingress pipeline finds a valid packet with our custom protocol. Depending on a type-field in our protocol it either *(i)* simply forwards the bandwidth measurement value, *(ii)* sets the regression parameters and enables the filter, or *(iii)* it preprocesses the bandwidth measurement before answering the request. The test environment contains a single switch connecting two communicating hosts and the monitoring host within a *mininet* virtual network. The controller is a Python process on the physical hosting machine. We repeatedly generate traffic using *iperf* following a simple randomized pattern: UDP input flows have lengths that are uniformly distributed between 0 and 40 seconds and an input rate chosen uniformly at random between 0 and 40 MBps.

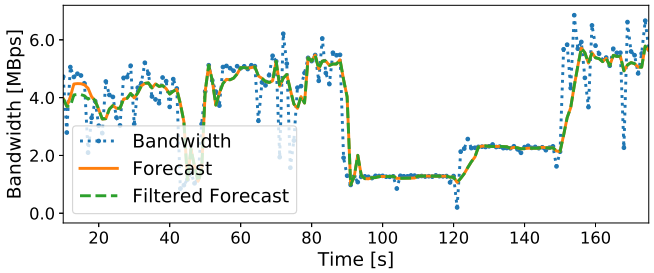
B. Evaluation results

Figure 5 shows the accuracy of the forecast with and without our preprocessing as well as the cost reduction achieved with early preprocessing. The two upper plots, Figures 5a and 5b, show the comparison between ground truth, i.e., the bandwidth measured at the monitor, its forecast, which was generated in the previous time step, and the forecast with active preprocessing - thus, filtering out measurements with small impact on the accuracy. The left figure (5a) shows that the forecast and the filtered forecast follow almost the same pattern that closely follows the ground truth. In this figure, we use a threshold for the forecast accuracy improvement to control the measurement filtering, which is set to a comparably small value of 25k. In contrast to this, we see in Figure 5b, that there is a difference between the forecast and the forecast using filtered information when this threshold is set to 500k. Now measurements with small improvements are not delivered and, thus, the accuracy slightly suffers.

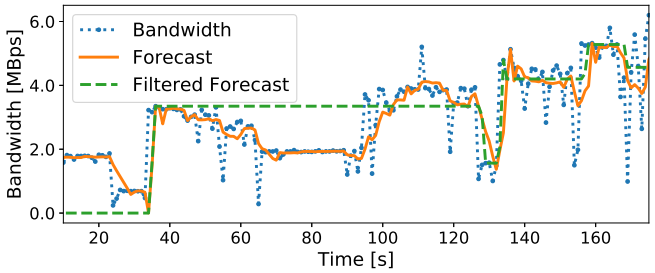
Figure 5c shows boxplots for the observed forecast error. The figure shows the gradual impact of the forecast threshold on the accuracy. In contrast, in Figure 5d we observe that the measurement costs in terms of the number of forecast update messages drop significantly using the filtering. In this case one may conjecture a linear scaling of the improvement on the long term, however, on smaller time scales - as the depicted ones - the improvement closely resembles the variations in the input traffic bandwidth. Together, Figure 5c and Figure 5d show that a significant reduction of the monitoring overhead can be obtained through a small sacrifice in accuracy when preprocessing measurements on the data plane. We observe that preprocessing statistic requests lets us successfully predict to which extent a measurement is of importance for the forecast application already at the switch level.

[2]https://cran.r-project.org/web/packages/forecast, accessed 01 Feb 2019.

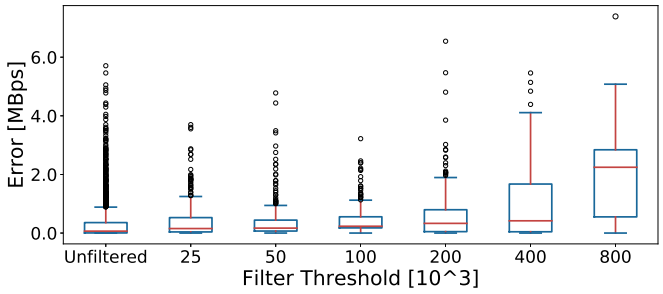
Rhaban Hark, Divyashri Bhat, Michael Zink, Ralf Steinmetz, Amr Rizk. *Preprocessing Monitoring Information on the SDN Data-Plane using P4* In: Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), p: 1–6, IEEE, November 2019.



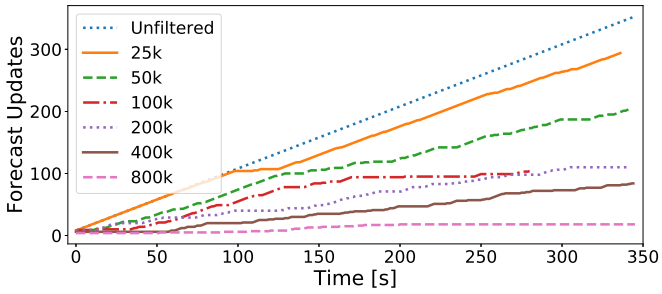
(a) Forecast with and without filtering compared to the measured bandwidth when filtering measurements with an estimated improvement below 25k.



(b) Forecast with and without filtering compared to the measured bandwidth when measurements are filtering with an estimated improvement below 500k.



(c) Number of transmitted measurements and, thus, number of forecast recalculations with and without filtering.



(d) Number of transmitted measurements and, thus, number of forecast recalculations with and without filtering.

Fig. 5: Performance evaluation: Subfigures 5a (Threshold 25k) and 5b (Threshold 500k) compare the accuracy of the forecast with and without filtering (preprocessing). Figure 5c shows the error with different threshold values, while Figure 5d show the corresponding costs. It is observable that the costs can be decreased much earlier than the accuracy suffers.

IV. Conclusion

In this work, we show how the information stream between the data- and control-plane can be tailored to the applications' requirements. We consider the example of a network management application that computes a bandwidth forecast for certain links and estimate the importance of a measurement with respect to the quality of the forecast. We use this measurement importance metric to only forward the *informative* measurements that will significantly impact the forecast outcome. To filter information as early as possible, we show how to use recently proposed programmable data-planes based on P4 to implement a regression on switches that essentially decides on the importance of a measurement. Finally, we show how we can preprocess information efficiently on the data-plane such that we retain sufficient accuracy while significantly reducing the monitoring costs given in terms of forecast updates.

In future work, we will investigate if and how more sophisticated learning algorithms that map the filtering of measurement information to network management application accuracy can also be implemented in the data-plane using P4.

Acknowledgment

This work has been funded by the German Research Foundation (DFG) within the Collaborative Research Center (CRC) 1053 - MAKI as part of subprojects B1, B4 and supported in parts by the project SPINE.

References

[1] Z. F. Baruch, *Structure of Computer Systems*. U.T. Pres, 2002.
[2] P. Bosshart, D. Daly, G. Gibb *et al.*, "P4: Programming Protocol-independent Packet Processors," *SIGCOMM CCR*, vol. 44, no. 3, pp. 87–95, 2014.
[3] T. Choi, S. Song, H. Park *et al.*, "SUMA: Software-defined Unified Monitoring Agent for SDN," in *Proc. of IEEE NOMS*, 2014, pp. 1–5.
[4] M. Costa, M. Codreanu, and A. Ephremides, "On the age of information in status update systems with packet management," *Transactions on Information Theory*, vol. 62, no. 4, pp. 1897–1910, 2016.
[5] N. Duffield, C. Lund, and M. Thorup, "Estimating Flow Distributions from Sampled Flow Statistics," *IEEE/ACM Transactions on Networking*, vol. 13, no. 5, pp. 933–946, 2005.
[6] R. Hark, N. Aerts, D. Hock *et al.*, "Reducing the Monitoring Footprint on Controllers in Software-Defined Networks," *IEEE TNSM*, vol. 15, no. 4, pp. 1264–1276, 2018.
[7] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications," in *Proc. of ACM SIGCOMM Workshop HotSDN*, 2012, pp. 19–24.
[8] A. Krishnamurthy, S. P. Chandrabose, and A. Gember-Jacobson, "Pratyaastha: An Efficient Elastic Distributed SDN Control Plane," in *Proc. of ACM SIGCOMM Workshop HotSDN*, 2014, pp. 133–138.
[9] D. Levin, A. Wundsam, B. Heller *et al.*, "Logically Centralized?: State Distribution Trade-offs in Software Defined Networks," in *Proc. of ACM SIGCOMM Workshop HotSDN*, 2012, pp. 1–6.
[10] D. A. Popescu, G. Antichi, and A. W. Moore, "Enabling Fast Hierarchical Heavy Hitter Detection Using Programmable Data Planes," in *Proc. of SOSR*, 2017, pp. 191–192.
[11] J. L. M. Saboia, "Autoregressive Integrated Moving Average (ARIMA) Models for Birth Forecasting," *Journal of the American Statistical Association*, vol. 72, no. 358, pp. 264–270, 1977.
[12] A. Tootoonchian and Y. Ganjali, "HyperFlow: A Distributed Control Plane for OpenFlow," in *Proc. of USENIX INM/WREN*, 2010, pp. 1–6.
[13] Y. Zhang, "An Adaptive Flow Counting Method for Anomaly Detection in SDN," in *Proc. of ACM CoNEXT*, 2013, pp. 25–30.