

Silent Consensus: Probabilistic Packet Sampling for Lightweight Network Monitoring

Marcel Wallschläger(✉), Alexander Acker(✉), and Odej Kao(✉)

Complex and Distributed Systems, Technische Universität Berlin,
Ernst-Reuter-Platz 7, 10587 Berlin, Germany
{marcel.wallschlaeger,alexander.acker,odej.kao}@tu-berlin.de

Abstract. Artificial intelligence based methods for operations of IT-systems (AIOps) support the process of maintaining and operating large IT infrastructures on different levels, e.g. anomaly detection, root cause analysis, or initiation of self-stabilizing activities. The foundation for the deployment of such methods are extensive and reliable metric data on the current state of the overall system. In particular, network information expressing the core parameters latency, throughput, and bandwidth have crucial impact on modern IoT and edge computing environments. Collecting the data is a challenging problem, as the communication is limited to existent network protocols, and adding new features requires a major infrastructure adaptation. The usage of additional monitoring protocols increases the CPU/network overhead and should be avoided as well. Therefore, we propose a two step approach for measuring latency between adjacent hops without manipulating or generating any network traffic. Inspired by audio and image compression algorithms, we developed a probabilistic method named *silent consensus*, where we keep the precision within a desired interval while reducing the overhead significantly. This method identifies the same packets on a sequence of network hops solely by observing the regular traffic. A linear regression helps to predict packets that are likely to appear after a fixed temporal offset based on a constrained set of historic observations. A correction of the predicted entity increases the probability for consensus between the involved hops. An extensive experimental evaluation proves that the approach delivers the expected foundation for further analysis of the network streams and the overall system.

Keywords: Network monitoring · Packet sampling · Measurement · Traffic engineering

1 Introduction

The cloud and virtualization trend, the increasing number of IoT applications with dynamically linked devices, and the embedding in real-world (smart) environments drive the creation of large multi-layered systems. These are typically

© Springer Nature Switzerland AG 2019
S. Misra et al. (Eds.): ICCSA 2019, LNCS 11619, pp. 241–256, 2019.
https://doi.org/10.1007/978-3-030-24289-3_19

characterized by a high number of attached devices with varying processing power, by a heterogeneous network traffic ranging from several bytes generated by IoT devices up to 4K video streams, and finally by increasingly important QoS requirements. In particular, the latency matters in case of many modern applications such as industry automation or self-driving cars. A significant support for this optimization across highly interconnected systems is provided by the software defined networking trend, as the traffic priorities, sending rates, and other parameters of the involved devices can be influenced dynamically and during the run-time. The adaptation requires however a precise picture of the current topology and traffic situation. Therefore, network monitoring solutions gained a significant importance during the last decades and enter currently a new phase of development driven by the increasing network complexity, heterogeneous nature of IoT networks, and the rising traffic volume. In this paper we focus on a novel approach for lightweight network monitoring, which is targeting video streaming applications as one of the major driver for the high-volume network traffic.

The sum of all forms of IP video including streaming, video on demand (VoD), file sharing, gaming, and video conferencing are in the range of 80% to 90% of the internet traffic today and expected to keep this share until 2022 [6]. However, this type of applications is not uniform, as they differ regarding the QoS requirements. While streaming applications demand a high bandwidth and can use buffering to mask delays, gaming applications are designed for synchronous communication between many actors and thus rely on low latency. The next development steps such as virtual reality even increase the latency requirements.

The network providers are aware of the QoS challenges and already reacted by introducing additional intelligence into maintenance, for examples as NREs (network reliability engineers) or as artificial intelligence based methods for IT operations (AIOps). They rely on precise measurements to recognize, localize, and remediate any upcoming anomalies threatening the guaranteed QoS demands. Traditional active latency measurement strategies such as ping or traceroute are not suitable for this type of maintenance, as they can only detect a limited number of latency anomalies since they are not treated the same as the user traffic. Further approaches implement active measurement, where additional traffic is generated to observe the network quality. For example, an in-band-telemetry provides latency information on top of the user traffic by appending arrival and departure times at each network hop onto a traversing user packet. Such an additive information requires space in a packet. Encapsulation protocols like VXLAN or Geneve solve this problem, but also expose the risk of packet growth beyond the original MTU size, resulting in fragmentation and additional network packets [13,16,22].

This paper aims at an alternative approach by introducing a lightweight solution for passive monitoring to observe routes, inter hop latencies, and throughput for packet streams. It utilizes a probabilistic packet sampling in both TCP directions on each network device in order to follow the same packet along the network path. A time stamp ordered sequence of the packets along with the sequence

number and host:port pairs shows the route and allows a latency approximation between each hop pair along the route. The sequence number comparison of successive packets contributes to the throughput estimation, which is then up-scaled to an entire switch and used to detect bottlenecks. As related to other probabilistic packet sampling methods (e.g. [25,27]), our approach does not introduce any traffic overhead besides the monitoring data itself. The networks hops require neither additional coordination messages nor a manipulation of existent packets is needed to synchronize their probing policies. The quality of the developed solution is proven by experimental measurements in a simulated environment.

The remainder of the paper is organized as follows. Section 2 provides an introduction into the related work on monitoring, probabilistic packet sampling, and P4 network monitoring. The silent consensus approach is then explained in Sect. 3. Section 4 presents an evaluation based on a video-streaming use-case. Finally, Sect. 5 concludes the paper with an outlook of open questions.

2 Related Work

Software defined networking (SDN) and the rising trend of programmable data planes make networks more dynamically at cost of increased complexity. The idea of self-optimizing and self-driving networks require trustworthy monitoring information for their operations [10].

The traditional network traffic monitoring falls into two categories: packet level sampling and flow level sampling. Packet level sampling is based on *libpcap* or on running a dedicated network traffic capturing system. Packet level sampling deploys filtering methods to capture solely a relevant subset of all packets. However, a packet sampling cannot be applied to the entire network traffic due to its sheer volume. A flow level sampling collects information for each individual connection, where advanced techniques are designed on top of programmable data-planes [9,29] or on dynamic query based approaches [14,15,18,28]. These query interfaces are integrated into a data analysis framework like Apache Flink [1] designed for streaming analytic. Further references explicitly covered the resource overhead of packet sampling [7,26]. In opposite to these query-based approaches, the approach of probabilistic packet sampling aims at reducing the overhead by minimizing the number of packets to be examined and thus resulting into fast control loops. OpenSample for example exploits the TCP sequence numbers and time stamps to produce such high-speed control loops [25]. Compared to the approach of the presented paper, OpenSample analyzes sampled packets per network node in a high frequency to obtain the throughput information. However, it does not provide latency and path information for ongoing TCP-connections.

The dynamic nature of today's networks with frequently changing routes exposes additional challenges to the monitoring tools. SDN-enabled flexibility allows the integration of new network devices automatically as well as a fast reaction to hardware faults or overload situations. However, this flexibility is limited by the necessity to support existing, often vendor-specific protocols over

long period of time. Therefore, add-on solution such as encapsulation protocols are developed to work around this problem. A prominent example is VXLAN, which was implemented in software-switches for cloud applications [21]. Nevertheless, the adoption takes time as well, as the vendors must design new products and bring these to the market.

The idea of programmable networking is followed by several vendors. Thereby, programmable processor are build into network devices, which provides the capability to customize them for specific requirements. Barefoot networks promotes customizable network devices [2], which allows a protocol implementation in software. To provide standardization among different vendors, P4 (Programming Protocol-Independent Packet Processors) is widely considered a de facto standard programming language for network devices [3,5]. Building uppon that, Shahbaz et al. published *Pisces* a software switch based on Open vSwitch (OVS) that is P4-programmable. The performance evaluation showed that P4 programmable software switches can process packets as fast as the original Open vSwitch [24]. The currently promoted hardware switches provide dynamically programmable packet handling without any additional overhead [4]. Since the P4 release in 2014, several publications showed its importance in network research, namely p4guard (firewall implemented in P4) or the network monitoring technique in-band-telemetry (INT) [8,17,19].

3 Silent Consensus

In this section we present an approach for a latency measurement without altering existing or introducing additional network traffic, solely utilizing TCP sequence numbers. The concept of the incremental calculation of those numbers based on the packet payload size is depicted in Fig. 1. The TCP sequence number is a 32-bit integer field in the header, separately maintained for both directions of each connection. The number of bytes in the packet payload is used to calculate its growth.

In order to enable a fine-grained diagnosis during anomaly situations, e.g. deterioration of QoS for clients, an identification of all possibly affected network hops is required. Information about the packet delay between each hop allows the pair-wise identification of the network devices that are responsible for an increased latency. Furthermore, recent information about the throughput of each hop represent a valuable metric for determining the faulty network devices. Utilizing the sequence number, we propose a hop-to-hop latency measurement by determining the arrival time of the same packet at each network component. Assuming the ability to determine the arrival times $t^{(h1)}$ and $t^{(h2)}$ of a packet p_y with the same sequence number y at two successive network hops $h1$ and $h2$, the hop-to-hop delay is defined as

$$\Delta t^{(h1,h2)}(p_y) = t^{(h2)}(p_y) - t^{(h1)}(p_y). \tag{1}$$

The main challenge is to identify the same packet in a successive chain of network hops without altering the existing or adding new packets. In case of successful

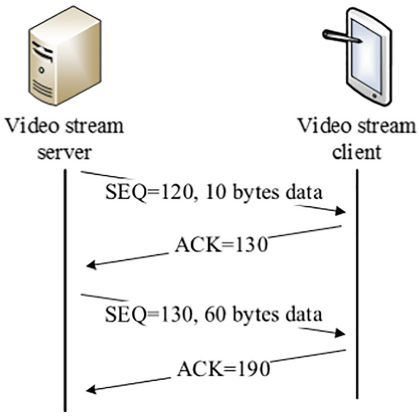

Fig. 1. Concept of the incremental TCP sequence number update during a TCP connection between a server and client. The TCP sequence number increases in dependence of the size of the TCP packet payload.

identification, we can compute many attributes, e.g. latency and throughput. For a successful identification, we rely on a probabilistic packet sampling and aim at maximizing the probability of sampling the same packet at every network device.

A straightforward solution for identifying the same packet on every network hop is to use predefined sequence numbers or packet counts, i.e. every nth packet. However, such static solutions have major drawbacks in case of a volatile throughput or packet losses. Throughput variations may range between Bytes/s and GBytes/s which result in alternating monitoring intervals. An initial constant guess on n combined with an increasing throughput might end up as an unnecessarily high monitoring traffic overhead due to a large number of sampled packets. Therefore, a fixed packet sampling frequency independent from the network traffic volume is a key requirement. This would necessitate an adjustment of n depending on the current throughput. Furthermore, as different TCP connections within a network have different purposes, they differ in terms of throughput, which demands the synchronization to be done for each ongoing connection. However, the additional overhead of synchronizing n between hops violates the requirement of not adding additional traffic.

To overcome the described problems, we introduce a hop-specific adjustment method to sample packets from an ongoing directed TCP connection. A network administrator configures a fixed sampling frequency f, at which monitoring metrics from every network hop should be reported. Considering t_i with $i = 1, 2, ...$ as a set of monitoring time stamps, the interval between two consecutive observations is

$$\Delta t_S = t_{i+1} - t_i = \frac{1}{f}. \tag{2}$$

Due to the fixed monitoring frequency, we need to determine a packet that will likely appear around each given sampling time stamp t_i at every network hop h_m, where $1 \leq m \leq M$ with M as the number of hops between source and destination. However, it exists a delay between two consecutive network hops, means that $t^{(h1)}(p_y) > t^{(h2)}(p_y) > \cdots > t^{(hM)}(p_y)$. Based on this, if every network device reports the currently observed packet p_{y_j}, where $j = 1, \ldots, M$ at a time t_i, the result is a set of M distinct packets, i.e. $\{y_j | 1 < j < M\}_{\neq}$. However, as stated in Eq. 2, the same packet is required to be observed across every hop, i.e. $\{y_j | 1 < j < M\}_{=}$, in order to calculate the latencies. We resolve this issue with a two step predictive approach:

1. Each network node forecasts the packet p_{y_+} for the next observation timestamp t_{i+1}, where $y_+ > y$ applies for the packet sequence numbers.
2. A correction of the prediction p_{y_+} is calculated in order to increase the probability of $\{y_j | 1 < j < M\}_{=}$, i.e. each network device predicts a packet with the same sequence number.

3.1 Next Packet Prediction

The packet arrival time depends on the throughput, so a reliable TCP throughput forecasting is necessary. Related approaches apply forecasting based on machine learning methods [11,12,20,23]. However, all approaches have the common goal of predicting the long-term throughput for a collection of different protocols and thus, focus on training models with data on hourly or daily basis. As shown in Fig. 2(a), the raw throughput value usually underlies strong variations, which poses a challenge for accurate predictions and demands complex models. On the other hand, the TCP sequence number, as an additive representation of the throughput, shows an approximately linear progress (see Fig. 2(b)). We state that the approximately linear progress of TCP sequence numbers together with the connection-based prediction allows a utilization of linear regressive forecasting models trained on data points from the recent past. Having a window of

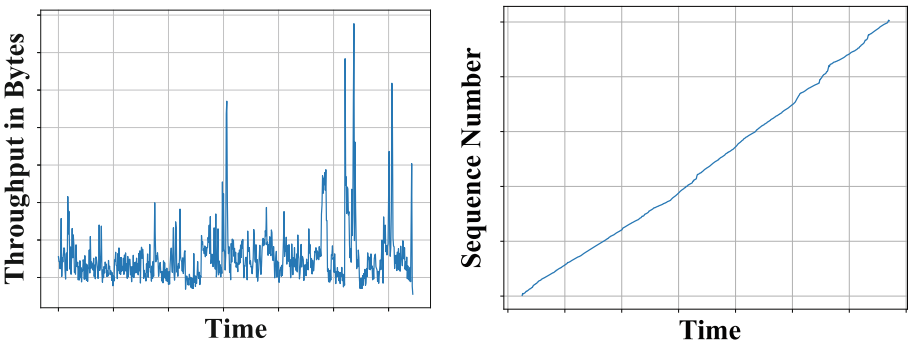

Fig. 2. (a) Throughput of a 480p video stream (b) TCP sequence number of a 480p video stream.

recently observed TCP sequence numbers $t_{i-w}(p_{y_1}), t_{i-(w-1)}(p_{y_2}), \ldots, t_i(p_{y_w})$, with w as window size, the goal is to predict a future packet p_{y_+} at time t_{i+1}. As we use the TCP sequence number to identify packets, these are defined as the dependent variable that needs to be predicted based on the time of its observation, described as

$$y_+ = \omega_1 \cdot t_{i+1} + \omega_0. \tag{3}$$

Utilizing the recent history of TCP sequence numbers together with the observation time stamps, we determine the parameters ω_1 and ω_0 by solving Eq. 3 for the least L-2 norm

$$\underset{\omega_1,\omega_0}{\operatorname{argmin}} \|y - (\omega_1 \cdot t + \omega_0)\|_2^2. \tag{4}$$

Using the recent history window allows the adjustment of the model to a temporally changing throughput but requires its retraining at each monitoring interval t_i. The retraining rate depends on the user defined sampling frequency f. Given the model fitting duration Δt_{fit}, the time to predict the next sequence number $\Delta t_{predict}$ and the monitoring interval Δt_S defined in Eq. 2, it must be assured that $\Delta t_S - (\Delta t_{fit} + \Delta t_{predict}) > 0$. The quantity of both Δt_{fit} and $\Delta t_{predict}$ depends on the number of model parameters in Eq. 3 while Δt_{fit} additionally depends on the historic window size w. Thus, we restrict our model to two parameters ω_0 and ω_1 while advising to keep the window size w as low as possible in order to support high frequent monitoring rates. Another limitation rises from the natural delay between the first and last network hop and entails the definition $\Delta t^{(h1,hM)} > \Delta t_S$, i.e. the defined packet sampling interval Δt_S cannot exceed the delay between the first and last network hop. For our approach, a packet that is transmitted through the network must have a realistic chance to be observed by each network device before a monitoring time interval Δt_S expires.

3.2 Prediction Synchronization

To determine the delay between every adjacent pair of network nodes, each node has to predict an identical sequence number. However, there exist two inherent sources of uncertainty. First, we must guess the future TCP connection throughput based on recent history in order to predict the next sequence number. As network traffic usually exposes non-deterministic behavior combined with the intended simplicity of our forecasting model, the predicted value will usually deviate from the actual observed value. Furthermore, due to a natural delay between each network hop, they observe distinct sequence numbers at every monitoring timestamp t_i. As our method is applied on packets from directed TCP connections, this aspect can be rendered more precisely as network nodes located closer to the source are observing a more recent extract of the packet stream than nodes located closer to the destination. Therefore, different packet sequence numbers are used to train the models at each network device, which results in heterogeneous model parameter values. Thus, the predicted sequence numbers will differ for every network hop. This delay-induced model parameter divergence is the second source of uncertainty.

Based on this, we propose to adjust the predicted TCP sequence number in a way that increases the probability of being identical at each network hop. The aspect of determining identical TCP sequence numbers across several network devices without exchanging any messages except the regular TCP packets is referred to as *silent consensus*. For this, the TCP number metric space is divided into bins of fixed size k. After that, the predicted future value y_+ is adjusted by applying

$$y_+^{'} = \lfloor \frac{y_+}{k} \rfloor \cdot k. \tag{5}$$

This relaxes the requirement for the linear regressive forecasting of predicting the same TCP sequence number across each network device towards the requirement of predicting a sequence number that lies in the same bin. The defined adjustment from Eq. 5 is subject to two problematic cases (note that a mixture of both is also possible). First, due to the floor operation, the corrected sequence number $y_+^{'}$ can end up being smaller than the last observed value y. This case can be detected individually on each network device. As the method operates on directed TCP sessions, it is impossible to observe packets from the past. Thus, this case is trivially corrected by selecting the next bin, i.e. $y_+^{'} + k$. Second, the predictions are not guaranteed to lie in identical bins. This results in a subset of nodes or no nodes at all to reach silent consensus. To further increase the probability of making identical predictions, we propose to use a number l of ascending bins $y_+^{'}, y_+^{'} + k, y_+^{'} + 2k, \ldots, y_+^{'} + lk$. This poses a trade-off between increasing the chance of having identical predictions across the nodes and the amount of monitoring data that has to be transmitted to the monitoring data sink.

4 Evaluation

The target system for the silent consensus algorithm evaluation consists of two VMs which are connected via a network path containing three switches. The scenario is visualized in Fig. 3. The mean latency between the two VMs is 3 ms, whereby the latency between switch1 and switch2 is higher than the latency between switch2 and switch3.

The traffic between the VMs is generated by an Nginx-based video streaming server running on one of the VMs. The other VMs hosts an `ffmpeg`-client which requests video streams through the RTMP protocol. An exemplary video can be requested in three different resolutions: 4k (530 MB), high definition (290 MB) and 480p (50 MB). During the video transmission, the traffic on the three switches is recorded with the `tcpdump` tool. Therefore, the conducted experiments result in nine `PCAP` files, i.e. one for each combination of switch and video resolution. These files form the evaluation basis for our prototypical implementation of the proposed silent consensus algorithm[1].

The following Sect. 4.1 shows the evaluation results of using linear regression for forecasting future sequence numbers. After that, Sect. 4.2 describes the evaluation of the silent consensus approach based on the forecasting results.

[1] https://github.com/mwallschlaeger/silent_consensus_packet_sampling.

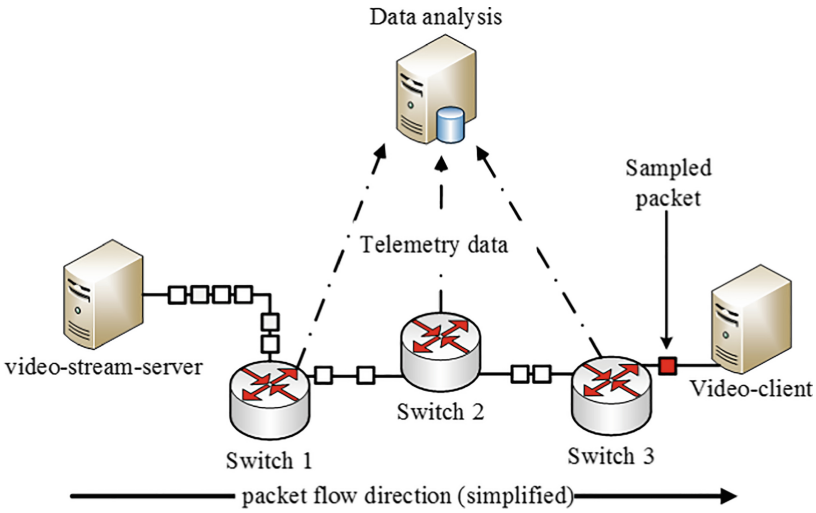

Fig. 3. Evaluation setup consisting of two virtual machines (video stream server and video client) connected by three network hops (switches). The server VMs are able to stream videos at different resolutions. Network traffic can be monitored with `tcpdump` individually on each switch.

4.1 Forecasting TCP Sequence Numbers

We define three major run time parameters for the TCP forecasting. Parameter $\Delta t_S = \frac{1}{f}$ is the interval, in which our system will predict a future sequence number, i.e. the time difference between the moment of prediction and the sequence number to be predicted. The other two parameters are the amount of historic TCP sequence number values w considered for the prediction and si as the number of history values considered within each interval Δt_S. We evaluate the linear regression forecasting with interval values $\Delta t_S = \{1, 2, 5, 10\}\, s$ in combination with $w = \{2, 3, 5\}$. Further we run the evaluation with two different values of $si = \{1, 2\}$.

Table 1 shows the mean absolute percentage error ($MAPE$) for each combination of the above defined system parameters. The results of each switch are averaged for every configuration.

The $MAPE$ results show a value below 2% for the lowest resolution 480p. The best configuration predicts the sequence number one second into the future using the two historical values resulting into $MAPE$ of 0.06%. For the HD resolution, where the average throughput is more than twice as much as for the 480p resolution, the $MAPE$ is below 5%. Considering smaller than five second forecast, the $MAPE$ is below 3%. Analogously, the best configuration here is a short-term forecasting using $w = 2$ historical sequence numbers from the last second to forecast the sequence number of the next second. The forecasting for

Table 1. Mean absolute percentage error (MAPE) for different configurations.

Δt_S in [s]	w	si	MAPE 480p	MAPE HD	MAPE 4K
1	2	2	0,0006	0,0015	0,0620
1	2	1	0,0014	0,0033	0,1343
1	3	1	0,0012	0,0033	0,1268
1	3	2	0,0007	0,0017	0,0658
1	5	1	0,0015	0,0043	0,1511
1	5	2	0,0008	0,0020	0,2723
1	*	*	0.0011	0.0027	0.1354
2	2	2	0,0049	0,0110	0,3754
2	2	1	0,0022	0,0063	0,2355
2	3	1	0,0023	0,0072	0,2423
2	3	2	0,0021	0,0050	0,1928
2	5	1	0,0031	0,0095	0,2947
2	5	2	0,0027	0,0060	0,1971
2	*	*	0.0028	0.0076	0.2563
5	2	2	0,0197	0,0452	1,5018
5	2	1	0,0060	0,0181	0,5845
5	3	1	0,0062	0,0199	0,5610
5	3	2	0,0076	0,0191	0,5684
5	5	1	0,0081	0,0284	0,7304
5	5	2	0,0070	0,0200	0,5402
5	*	*	0.0092	0.0251	0.7477
10	2	2	0,0439	0,1009	3,0760
10	2	1	0,0104	0,0356	0,9640
10	3	1	0,0124	0,0441	1,0869
10	3	2	0,0165	0,0437	1,1829
10	5	1	0,0149	0,0473	1,2771
10	5	2	0,0159	0,0463	1,1205
10	*	*	0.0190	0.0547	1.4512

the 4k video stream has a minimum *MAPE* of 6.2%. However for some configurations, the *MAPE* of the 4k stream goes up to 150% or even 307%. This high error results from using only one second of historical data to forecast 5 or 10 s into the future.

4.2 Silent Consensus

The evaluation of the silent consensus function is based on the forecasting results of the previous section. The results are summarized in Table 2.

To enable the observation of a packet p_y on each network hop (switch) $\{h1, h2, h3\}$, we conduct a number of sequence number predictions *pred*. The total number of predictions is defined by $\lfloor \frac{T}{\Delta t_S} \rfloor - 1$, where T is the total experiment duration. The subtraction by one results from the fact that the first Δt_S must be used as offset to aggregate initial training data. A prediction *pred* can be either correct (denoted as *true*) or incorrect (denoted as *false*).

Table 2. Silent consensus accuracy for different configurations and video stream resolutions.

Δt_S in [s]	w	si	ACC 480p	ACC HD	ACC 4K
1	2	2	0.8065	0.7291	0.8266
1	2	1	0.7941	0.7167	0.8188
1	3	1	0.7639	0.6878	0.7888
1	3	2	0.7872	0.7096	0.8012
1	5	1	0.7031	0.6296	0.7234
1	5	2	0.7562	0.6812	0.7671
1	*	*	0.7685	0.6924	0.7876
2	2	2	0.8037	0.7414	0.8193
2	2	1	0.7663	0.6915	0.8006
2	3	1	0.7053	0.6332	0.7398
2	3	2	0.7648	0.6959	0.7742
2	5	1	0.5841	0.5174	0.6158
2	5	2	0.7015	0.6380	0.7111
2	*	*	0.7210	0.6529	0.7435
5	2	2	0.8174	0.7380	0.8253
5	2	1	0.6666	0.6031	0.7063
5	3	1	0.5161	0.4516	0.5645
5	3	2	0.6854	0.5887	0.6935
5	5	1	0.225	0.15	0.2916
5	5	2	0.5416	0.45	0.55
5	*	*	0.5754	0.4969	0.6052
10	2	2	0.8360	0.7377	0.8196
10	2	1	0.6721	0.4918	0.7049
10	3	1	0.5084	0.2711	0.5593
10	3	2	0.6949	0.5423	0.6440
10	5	1	0.3272	0.1272	0.3818
10	5	2	0.5636	0.2545	0.4909
10	*	*	0.6004	0.4041	0.6001

$$pred_i = \begin{cases} true, & \text{if } p_y^{h1} = p_y^{h2} = p_y^{h3} \\ false, & \text{otherwise} \end{cases}. \tag{6}$$

Equation 6 states that a prediction result is regarded as *true* if the same packet, identified by its TCP sequence number, was observed on each network hop. Otherwise the prediction result is set to *false*. Based on this, the accuracy is defined as the ratio of *true* to the total number of predictions, formally defined as follows.

$$ACC = \frac{|pred^{true}|}{|pred|}. \tag{7}$$

For this evaluation k was set to 10. We used $l = 2$ ascending grids, thus having y'_+ and $y'_+ + k$.

The parameter value $\Delta t_S = 1\,\text{s}$ shows the best results. Further, it is revealed that an increasing value of w, results in an accuracy decrease of the silent consensus (Table 2).

Furthermore, considering the results of the forecasting evaluation, the consensus function shows acceptable accuracy values even for parameter configurations where the forecasting $MAPE$ values were high, e.g. for $\Delta t_S = 5\,\text{s}$, $w = 2$, $si = 2$ or $\Delta t_S = 10\,\text{s}$, $w = 2$ and $si = 2$. This contra-intuitive aspect is explained by the fact that the consensus function requires a consistent forecasting on each network hop, and not on an accurate one. As long all hops achieve similar forecasting results, i.e. the forecast sequence number is similar for each hop, the silent consensus has a high probability of selecting the same packet.

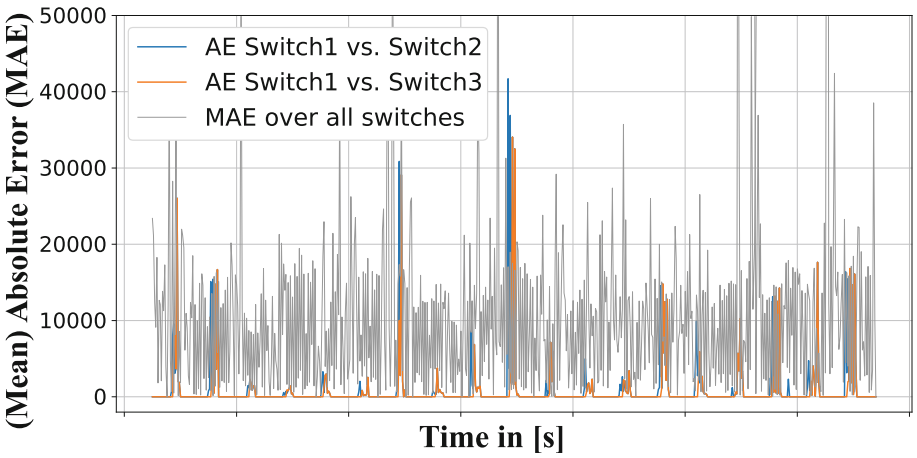

Fig. 4. Comparison of absolute error (AE) over time between the forecasting values of switch1 and switch2 and the forecasting values switch1 and switch3 in comparison to the MAE over all individual forecasting results. Based on the configuration $\Delta t_S = 5\,\text{s}$, $w = 2$, $si = 1$ of the 480p video stream. (Color figure online)

Figure 4 shows the absolute error (AE) between the sequence number forecast of switch1 and switch2 (blue line), and switch1 and switch3 (orange line). The grey line presents the mean absolute error (MAE) for each interval step Δt_S, whereby the mean of all three error values is calculated. The AE is 0 when the switches predict the same sequence number, independent of the correctness of the predictions. We observe that the error between switches spikes several times during the experiment. These spikes could be related to network congestion or application specific behavior like buffering in the video client, or processing on the server side. Further, this would interfere with the communication and invalidate the recent history entries w, resulting in a lower forecasting accuracy. Compared to the error between the switches, the forecasting precision behaves independent.

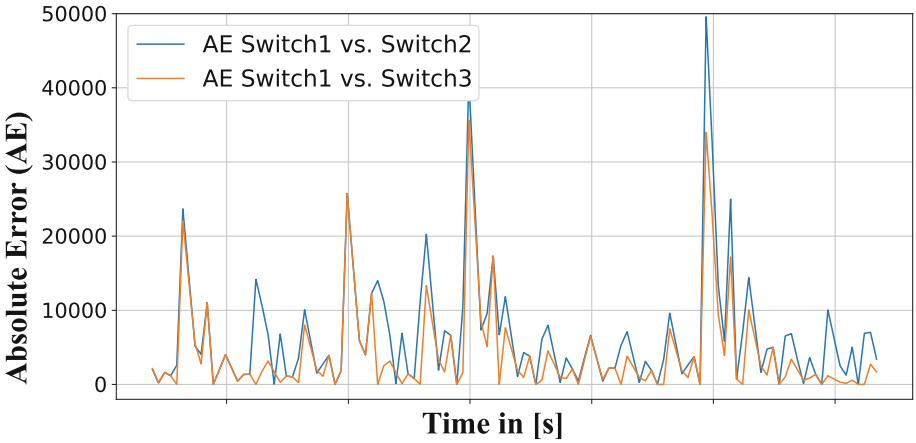

Fig. 5. Comparison of AE over time between the forecasting values of switch1 and switch2 and the forecasting values switch1 and switch3 in comparison to the MAE over all individual forecasting results. Based on the configuration $\Delta t_S = 5\,\text{s}$, $w = 2$, $si = 2$ of the high definition video stream.

As shown, the accuracy of the silent consensus depends on the similarity of the predicted sequence numbers among the network hops. Therefore, the predictions results are better for reliable hop-to-hop connections. Variations in latency or frequent network congestion between network node pairs will reduce the prediction accuracy. Also, once such fluctuations occurred, the time span where they are considered for the training of the prediction model should be as low as possible. This explains the better accuracy results for lower value of parameter w. As an example for comparison, Fig. 5 shows the sequence number forecast errors for the evaluation configuration $\Delta t_S = 5\,\text{s}$, $w = 2$, $si = 2$ of the high definition video stream. Compared to Fig. 4, which shows several spikes, the configuration for the HD stream shows a high number of small errors, which

are caused by a large difference between the forecasted sequence numbers of the different switches. That leads to a small number of identical predicted sequence numbers between the switches.

5 Conclusion

The presented approach shows a novel mechanism for network packet sampling to obtain latency, path and throughput information for each communication flow as foundation for anomaly detection and remediation. The approach analyses a minimal number of packets within a limited time window (1 to 10 s) and tracks the path of those packets from one network hop to the next. Based on the recorded timestamps between the hops, a number of the relevant traffic attributes can be computed and used as input into subsequent analysis algorithms. The advantage of the approach is the low overhead in computation and communication resulting from the decentralized decision making, as each network device individually determines which packet is to be sampled using a silent consensus function. The experimental evaluation examined the performance of the two major steps. The measured accuracy showed a probability of over 80% to sample the same packet when using an interval of one second.

The future work includes the development of a method for dynamic adjustment of the core parameter k, i.e. the width of the considered time window, which will simplify the deployment significantly. Moreover, we extend the experimental evaluation to IoT and edge computing scenarios.

References

1. Alexandrov, A., et al.: The stratosphere platform for big data analytics. VLDB J. Int. J. Very Large Data Bases **23**(6), 939–964 (2014)
2. Arista Networks, Inc.: arista.comwhite paperarista 7170 multi-function programmable networking. Technical report, Arista (2018). https://www.arista.com/assets/data/pdf/Whitepapers/7170_White_Paper.pdf
3. Bosshart, P., et al.: P4: Programming protocol-independent packet processors. ACM SIGCOMM Comput. Commun. Rev. **44**(3), 87–95 (2014)
4. Bosshart, P., et al.: Forwarding metamorphosis: fast programmable match-action processing in hardware for sdn. In: ACM SIGCOMM Computer Communication Review, vol. 43, pp. 99–110. ACM (2013)
5. Brebner, G.: P4 for an FPGA target. In: P4 Workshop (2015)
6. Cisco: Cisco visual networking index: forecast and trends, 2017–2022. Technical report, Cisco (2018). https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.pdf
7. Curtis, A.R., Mogul, J.C., Tourrilhes, J., Yalagandula, P., Sharma, P., Banerjee, S.: DevoFlow: scaling flow management for high-performance networks. In: ACM SIGCOMM Computer Communication Review, vol. 41, pp. 254–265. ACM (2011)
8. Datta, R., Choi, S., Chowdhary, A., Park, Y.: P4Guard: designing P4 based firewall. In: MILCOM 2018–2018 IEEE Military Communications Conference (MILCOM), pp. 1–6. IEEE (2018)

9. Duffield, N.G., Grossglauser, M.: Trajectory sampling for direct traffic observation. IEEE/ACM Trans. Netw. (ToN) **9**(3), 280–292 (2001)
10. Feamster, N., Rexford, J.: Why (and how) networks should run themselves. CoRR abs/1710.11583 (2017). http://arxiv.org/abs/1710.11583
11. Feng, Y., Wu, X., Hu, Y.: Forecasting research on the wireless mesh network throughput based on the support vector machine. Wirel. Pers. Commun. **99**(1), 581–593 (2018)
12. Goyal, M., Guerin, R., Rajan, R.: Predicting TCP throughput from non-invasive network sampling. In: INFOCOM 2002. Proceedings of the IEEE Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 1, pp. 180–189. IEEE (2002)
13. Gulenko, A., Wallschläger, M., Kao, O.: A practical implementation of in-band network telemetry in Open vSwitch. In: 2018 IEEE 7th International Conference on Cloud Networking (CloudNet), pp. 1–4. IEEE (2018)
14. Gupta, A., Birkner, R., Canini, M., Feamster, N., Mac-Stoker, C., Willinger, W.: Network monitoring as a streaming analytics problem. In: Proceedings of the 15th ACM Workshop on Hot Topics in Networks, pp. 106–112. ACM (2016)
15. Gupta, A., et al.: Sonata: query-driven network telemetry. arXiv preprint arXiv:1705.01049 (2017)
16. Kim, C., et al.: In-band network telemetry (INT). Technical report, The P4 Language Consortium (2016). https://p4.org/assets/INT-current-spec.pdf. Accessed 01 June 2018
17. Kim, C., Sivaraman, A., Katta, N., Bas, A., Dixit, A., Wobker, L.J.: In-band network telemetry via programmable dataplanes. In: ACM SIGCOMM Symposium on SDN Research (SOSR) (2015)
18. Liu, Z., Vorsanger, G., Braverman, V., Sekar, V.: Enabling a RISC approach for software-defined monitoring using universal streaming. In: Proceedings of the 14th ACM Workshop on Hot Topics in Networks, p. 21. ACM (2015)
19. Liu, Z., Bi, J., Zhou, Y., Wang, Y., Lin, Y.: NetVision: towards network telemetry as a service. In: 2018 IEEE 26th International Conference on Network Protocols (ICNP), pp. 247–248. IEEE (2018)
20. Lu, D., Qiao, Y., Dinda, P.A., Bustamante, F.E.: Characterizing and predicting TCP throughput on the wide area network. In: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems, ICDCS 2005, pp. 414–424. IEEE (2005)
21. Mahalingam, M., Dutt, D.G., Duda, K., Agarwal, P.: Virtual eXtensible Local Area Network (VXLAN): a framework for overlaying virtualized layer 2 networks over layer 3 networks. RFC 7348, RFC Editor, August 2014. https://www.rfc-editor.org/rfc/rfc7348.txt
22. Mestre, P.: In-band OAM for IPv6. Technical report, Cisco, June 2016. https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipv6_nman/configuration/15-mt/ip6n-15-mt-book/ioam-ipv6.pdf
23. Mirza, M., Sommers, J., Barford, P., Zhu, X.: A machine learning approach to TCP throughput prediction. SIGMETRICS Perform. Eval. Rev. **35**(1), 97–108 (2007). https://doi.org/10.1145/1269899.1254894
24. Shahbaz, M., et al.: PISCES: a programmable, protocol-independent software switch. In: SIGCOMM Conference, pp. 525–538. ACM (2016)
25. Suh, J., Kwon, T.T., Dixon, C., Felter, W., Carter, J.: OpenSample: a low-latency, sampling-based measurement platform for commodity SDN. In: 2014 IEEE 34th International Conference on Distributed Computing Systems (ICDCS), pp. 228–237. IEEE (2014)

26. Tammana, P., Agarwal, R., Lee, M.: Simplifying datacenter network debugging with pathdump. In: OSDI, pp. 233–248 (2016)
27. Yoon, S., Ha, T., Kim, S., Lim, H.: Scalable traffic sampling using centrality measure on software-defined networks. IEEE Commun. Mag. **55**(7), 43–49 (2017)
28. Yu, M., Jose, L., Miao, R.: Software defined traffic measurement with opensketch. In: NSDI, vol. 13, pp. 29–42 (2013)
29. Zhu, Y., et al.: Packet-level telemetry in large datacenter networks. In: ACM SIGCOMM Computer Communication Review, vol. 45, pp. 479–491. ACM (2015)