Low-Overhead Near-Real-Time Flow Statistics Collection in SDN

Kokouvi Benoit Nougnanke, Marc Bruyère, Yann Labit

Low-Overhead Near-Real-Time Flow Statistics Collection in SDN

Kokouvi Benoit Nougnanke*, Marc Bruyere†, Yann Labit*
* *LAAS-CNRS, Université de Toulouse, CNRS, UPS, F-31400 Toulouse, France*
† *IIJ Innovation Institute, University of Tokyo, Tokyo, Japan*
nougnanke@laas.fr, marc@iij.ad.jp, ylabit@laas.fr

***Abstract*—In Software-Defined Networking, near-real-time collection of flow-level statistics provided by OpenFlow (e.g. byte count) is needed for control and management applications like traffic engineering, heavy hitters detection, attack detection, etc. The practical way to do this near-real-time collection is a periodic collection at high frequency. However, periodic polling may generate a lot of overheads expressed by the number of OpenFlow request and reply messages on the control network. To handle these overheads, adaptive techniques based on the pull model were proposed. But we can do better by detaching from the classical OpenFlow request-reply model for the particular case of periodic statistics collection. In light of this, we propose a push and prediction based adaptive collection to handle efficiently periodic OpenFlow statistics collection while maintaining good accuracy. We utilize the Ryu Controller and Mininet to implement our solution and then we carry out intensive experiments using real-world traces. The results show that our proposed approach can reduce the number of pushed messages up to 75% compared to a fixed periodic collection with a very good accuracy represented by a collection error of less than 0.5%.**

***Index Terms*—SDN, OpenFlow, Continuous Monitoring, Data Collection, Monitoring Overhead, Time Series**

I. Introduction

Software-Defined Networking (SDN), by separating the control plane from the data plane, provides good abstractions to the network control plane, that becomes programmable, flexible and more innovation in networking is eased. The OpenFlow API is used to control the forwarding plane.

Apart from being used to control the forwarding plane through flow rules installed by the control plane, OpenFlow exposes flow statistics (packets and bytes count, flow duration, etc.) and other relevant flow information (flow entry expiration and packet-In). These statistics and event-based information need to be gathered to enriched the global view of the network and be used by many control and management functions as traffic engineering [1], routing [2], anomaly detection [3], congestion detection, identification of architectural bottlenecks, etc. This flow-level measurement is done either actively, by querying the statistics such as traffic counters, flow duration, etc. on a pull basis with Read-State messages, or passively after an event as flow entry expiration and packet-In.

Moreover, for efficient and near-optimal control and management, the network global view needs to be up-to-date which means near-real-time network state observation. One practical way to do this is by collecting non-event-based data (traffic counters especially) periodically at high frequency (e.g. reports interval of few seconds). But OpenFlow's pull-based statistics retrieving may impose high overheads [1], expressed by the number of request and reply messages on the data plane.

To handle these overheads, i.e. provide timely accurate statistics to the control plane but with low overheads, pull-based adaptive approaches consisting of using adaptive polling rates, instead of a fixed one were proposed. The collection granularity adjustment is done either according to the stability of the statistic being collected, by comparing the difference between two consecutive data-points to one or several thresholds [4], or by using a prediction-based approach which adjusts the reporting intervals according to the ability to provide good estimations of future data-points based on the history composed of the previously collected data-points [3] [5].

The aforementioned approaches provide good accuracy-overhead trade-offs, but we can do better by detaching from the classical OpenFlow request-reply model for the particular case of periodic statistics collection. We then propose a push and prediction based adaptive periodic OpenFlow statistics collection with low overheads and almost no accuracy degradation. This will be possible since the switch, source of the statistic to be collected is involved in the adaptive process. Then it automatically knows the adaptive collection points in time, and on its own will just push to the controller the statistic value without the need for request messages. Doing this way, the reduction of overhead will be more consequent. Hence, we propose a push and prediction based adaptive for OpenFlow statistics near-real-time collection, which ensures low overheads with almost no accuracy degradation.

The main contributions of this paper are summarized below:

- We propose an adaptive collection mechanism that decreases considerably the overhead in terms of the number of messages used to collect flow bytes count in near-real-time (Section III).
- We implement the proposed algorithm using the Ryu Controller and Mininet, and after that, we carry out intensive experiments using real-world backbone and university data-center traces. The evaluation results are presented and analyzed (section IV).
- We also present a direct use-case of our low-cost near-real-time bytes count collection, bandwidth monitoring for traffic engineering and then we discuss our work regarding P4 and INT (Section V).

II. Related Works

For continuous statistics collection in SDN, especially flow bytes count, traditional network monitoring flow sampling techniques like sFlow, NetFlow/IPFIX could be used. But we don't need this extra instrumentation on OpenFlow Switches since they provide directly flow statistics, by maintaining records on packets matching installed flow rules. We just need to collect these statistics efficiently. However, OpenFlow will face some issues to provide the same level of flow-level measurements compared to the sampling techniques mentioned above, especially the limitation of the number of flow rules that a switch can support. In light of this, [6] emulates NetFlow/IPFIX operation in SDN, providing OpenFlow compatible flow sampling methods. For near-real-time statistics collection with good accuracy-efficiency trade-off, [7] proposes per-flow sampling with adaptive polling frequency, polling frequency being adjusted depending on whether the sampled traffic is stable or busy. In the same context to provide timely data-plane state information with low overheads, [4] proposes a threshold-based adaptive scheduling algorithm for flow statistics retrieving by polling.

III. Periodic Collection Overhead Handling

A. Problem Statement

A statistic S to be collected in near-real-time, here an OpenFlow bytes count maintained by the data-plane switch, is modeled by:

$S : \mathbb{R}^+ \longrightarrow \mathbb{N}$

$\forall u, t \in \mathbb{R}^+ : u \leq t \implies S(u) \leq S(t).$

The bytes count S(t) matching a specific flow, or a port bytes count in the interval [0, t), is a cumulative, non-negative and non-decreasing function. It must be part of the global and up-to-date view exposed by the control plane. Practically it will be collected periodically and control and management functions' requirements may define the period T_0 to be used.

Then the statistic S may be seen as $S_{T_0} = \{s_i\}_{i=0}^{n}$, n $\to \infty$, a large sequence of data-points s_i at regular time interval T_0 with $s_i \leq s_{i+k}, \forall k \in \mathbb{N}$, representing an increasing trend time series. We denote the collected version at the controller level as $\hat{S}$. If collected at a fixed T_0, $\hat{S_{T_0}}$ will be considered equal to S_{T_0} (with the hypothesis that there is no packet loss and that data plane and controller communication delay is zero, ideally with an out-of-band control deployment).

For fine-grained near real-time control and management, T_0 needs to be as small as possible but this may generate a lot of overhead. A question that could raise at this point is: "Isn't any magic T_0 to be used that will give a good trade-off between the collection accuracy and the overhead ?"

Our approach consists of not to have to look for this magic T_0. Instead, we will provide the samples of S(t) with the application chosen T_0, but not all the samples will be really collected. We will effectively collect at adaptive frequency (T_0, $2T_0$, $3T_0$, ...) and the non-collected data-points will be predicted based on the already existing ones.

B. Pushed and Prediction based Adaptive Collection

Our proposition is an easy to deploy, transparent and lightweight solution that comes from our early previous work [8], where we propose, a generic adaptive algorithm based on a confidence index for collecting any type of information (CPU usage, switch information, etc). Here we leverage this adaptive algorithm that is enhanced and optimized for statistics presenting a particular pattern, an increasing trend (counters).

Our adaptive approach is built upon two mainstays: **the adaptive push model** and **the power of time series forecasting**. A push model because the classical OpenFlow pull model (request/reply) is inefficient since with a periodic collection or even with our adaptive collection mechanism, the collection points in time are known by the source node. This latter has to push to the controller at the collection points on its own without the need for a request message.

The adaptive frequencies or periods are represented by $T_\alpha = (1+\alpha)T_0$, where α is a confidence index expressing the degree of confidence between the collector and the node, source of the statistic, on the ability of the prediction method to provide good estimations of uncollected data-points. The confidence index α is constructed incrementally, it means that if at a given α the predictions are good, we pass to a new confidence level $\alpha + 1$. Conversely, the confidence index will degrade when we do not have good estimations. Hence our push and prediction based adaptive collection solution is called COCO for **CO**nfidence based adaptive **CO**llection.

Our collection overhead reducing mechanism operates on two entities: the collector and the data plane agent as shown in Fig. 1. Their algorithms are presented in the next section.

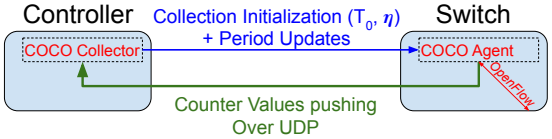

Fig. 1. COCO Collection Architecture

C. Algorithms

The COCO Agent executes the adaptive push procedure (see Algorithm 1) as a software-based monitoring solution on the node (switch), of course in collaboration with the SDN controller. In fact, for a statistic that is required to be up-to-date at T_0 granularity at the controller level, we attach the confidence index α, and $T_\alpha = (1 + \alpha)T_0$ is the period with which this information (the samples $\hat{s_i} = s_i$) will be pushed. The reporting interval begins with a low period $T_\alpha = (1 + \alpha)T_0 = T_0$ (α = 0) and α will be increased each β data-points pushed if a deviation alarm is not raised by the controller. Otherwise, α will be decreased according to the deviation degree γ. α evolves according to an AIMD (Additive-Increase/Multiplicative-Decrease) pattern.

Algorithm 2 describes the controller side process, where the collector processes collected data-points ($\hat{s_i}$), forecasts uncollected ones ($\hat{s_j}$) and raises deviation alarm if needed.

Algorithm 1: COCO Agent: Adaptive Push Procedure

Push β_0 data-points $\hat{s_i}$ each T_0, ($\alpha = 0$) ;
$\alpha \longleftarrow \alpha$ + 1 and $T_\alpha \longleftarrow (1 + \alpha)T_0$;
$i \longleftarrow i + 1$;
while *Forever* **do**
  Push $\hat{s_i}$ at T_α ;
  if *($\hat{s_i}$ is the β ith data-point for α)* **then**
    if $\neg$ *Deviation occurred* **then**
      $\alpha \longleftarrow$ min(α + 1, α_{max}) ;
    else
      $\alpha \longleftarrow \alpha$ / (2 * γ) ;
    end
    $T_\alpha \longleftarrow (1 + \alpha)T_0$;
  end
  $i \longleftarrow i + 1$;
end

Algorithm 2: COCO Collector: Forecasting and Deviation Checking

Collect the first β_0 data-points $\hat{s_i}$;
$\alpha \longleftarrow \alpha$ + 1 and *steps* $\longleftarrow \alpha + 1$;
while *Forever* **do**
  Collect data-point $\hat{s_i}$;
  Add $\hat{s_i}$ to last_collect ;
  if *($\hat{s_i}$ is the β ith data-point for α)* **then**
    if *error_indicator(last_collect, last_forecast)* $< \eta$ **then**
      No Deviation: $\alpha \longleftarrow$ min(α+1, α_{max}) ;
    else
      Deviation γ: $\alpha \longleftarrow \alpha$ / (2 * γ) ;
      Send Asynchronously γ to the switch;
    end
    steps $\longleftarrow \alpha + 1$;
    last_collect $\longleftarrow \emptyset$;
    last_forecast $\longleftarrow \emptyset$;
  end
  Forecast *steps* future data-points ($\hat{s_j}$) using previous $\hat{s_i}$;
  Add the *steps* ith forecast to last_forecast ;
end

The controller receives samples $\hat{s_i}$ from the node each $T_\alpha = (1 + \alpha)T_0$. Firstly it forecasts *step* equals to $\alpha + 1$ next data-points. The first α forecasts will be used as an estimation of $\{\hat{s_j}\}_{j=1}^{\alpha}$ uncollected data-points between $\hat{s_i}$ and $\hat{s_{i+1}}$ on a T_0 basis. Secondly, The last forecast the $\alpha + 1$ ith one will be compared to $\hat{s_{i+1}}$ to estimate the prediction algorithm accuracy. For a given α, after collecting β data-points $\hat{s_i}$, an accuracy deviation checking is done using these data-points $\{\hat{s_i}\}_{i=1}^{\beta}$ and their corresponding forecasts (the $\alpha + 1$ ith) to compute an indicator of the prediction error. This indicator is compared to the threshold η, when violated, a deviation alarm of degree γ (default γ = 1) is raised. γ is sent to the node for updating (decreasing) α then T_α.

To compute the prediction error indicator we use **Mean Absolute Scaled Error (MASE)** [9], a time series forecasting accuracy metric. It has many desirable properties compared to existing error metrics (e.g. MAPE). MASE is computed using β last forecasts, β last collected data-points, according to the previous data-points used for prediction (See Eq. 1).

$$MASE = \frac{1}{\beta}\sum_{k=1}^{\beta}\left(\frac{|\hat{s_k} - pred(\hat{s_k})|}{1/(\beta-1)\sum_{k=2}^{\beta}|\hat{s_k} - \hat{s_{k-1}}|}\right) \quad (1)$$

It compares the actual forecasting method to the one-step naive forecast computed in-sample with a ratio of their respective Mean Absolute Errors (MAE). Then when MASE < 1, the actual forecasting performs well than the naive one and vice versa. We transform this threshold of 1 to η.

Concerning uncollected data-points forecasting, the two following statistical techniques are investigated:

1) The popular Box-Jenkins ARIMA family of methods shortened as **ARIMA (AutoRegressive Integrated Moving Average)**: It exploits information embedded in the autocorrelation pattern of the data. Estimations are done based on the maximum likelihood. The univariate statistical time-series ARIMA model parameters (p, d, q) are computed automatically using Auto ARIMA [10].
2) **Holt's Trend Corrected Exponential Smoothing**: Or Double Exponential Smoothing is used to do forecasting when the time series presents a particular trend (increasing or decreasing) without a seasonal pattern. For this model, we may retain two main hyper-parameters: α^* the smoothing factor for the level (mean) and β^* the smoothing factor for the trend or slope smoothing.

For convenient forecasting, we need to satisfy a minimum training set size which generally is 50. Then, we define a specific β, β_0 equals to 50 (default) for α = 0 at the beginning. Moreover, we emphasize recent data-points supposing older ones less relevant, and also for the fact that we are doing an on-line prediction, we set a sliding window of size W = 1.5*β_0 for previous data-points to be used for the prediction.

The parameters used in our overhead reduction proposition are summarized in TABLE I.

TABLE I
PARAMETERS

Parameters	Description
α	Confidence index on the ability to provide good estimations of uncollected data-points, $\in$ [0,..,9]
T_0	The required time granularity for statistic updates
$T_\alpha = (1+\alpha)T_0$	Effective pushing period for the confidence index α
β_0	The minimal number of samples to be collected at the beginning on T_0 basis
β	Number of samples for a given α after which a deviation checking is done allowing α update
η	A threshold on the prediction error
γ	The degree of a deviation alarm

IV. EVALUATION

A. Setup and Evaluation Metrics

We evaluate our proposition using the topology in Fig. 2 on a virtual machine of 32 GB of RAM and 20 vCPUs.

With tcpreplay, we replay real network traffics from the two following public anonymized datasets. **UNIV2 [11]**: A university data-center trace collected on 22 January 2010. We

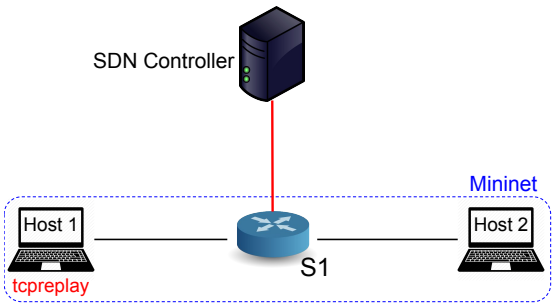

Fig. 2. Experimental Setup Topology

extract 8 samples of 450s from 19:02:15 to 20:10:00. And **MAWI [12]**: A backbone trace from the WIDE MAWI archive collected in September 2019 at the transit link (1Gps) of WIDE to the upstream ISP. We extract 7 samples of 450s from the archives from 19 to 22 September.

Host1 replays the traces on the OpenFlow switch S1, executing the COCO Agent, that forwards all the traffic to the sink node Host2. Host1, Host2 and the switch S1 are emulated with Mininet. The SDN controller (Ryu) executes the COCO Collector and collects the flow statistics from S1.

Two main metrics are used in this evaluation. We compute these metrics in comparison to a fixed push mechanism at T_0 (which will be the case when we use for example sFlow).

Accuracy: To evaluate the collection error induced by our adaptive push mechanism compared to a fixed T_0 one, we use the Mean Absolute Percentage Error (MAPE). It's expressed in percentage, the lower this value, the better it is.

Overhead: We express the overhead reduction with the percentage of the number of messages with a fixed push collection compared to the effective number of messages with COCO. We want this value to be as high as possible.

B. Overhead Reduction and Collection Accuracy

Fig. 3 and Fig. 4 show the percentage of messages reduced (on the left) and the collection error with MAPE (on the right) according to the threshold η respectively for the traces UNIV2 and MAWI, for 2 combinations of (β_0, β): (50, 10), (100, 10).

For all $\eta > 0$, meaning a fraction of inaccuracy is authorized on the collection, we always have a reduction of the overhead compared to the fixed periodic push collection. This reduction increases when η increases and may stabilize. We achieve up to 75% of reduction. And we have this with a very low error (less than 0.5 %) on the whole collection compared to the fixed T_0 collection. This very low error for both MAWI and UNIV2 increases with η and the reduction percentage.

C. Computation Time and Synthesis

When a data-point $\hat{s_i}$ is collected, before collecting the next one ($\hat{s_{i+1}}$) some computations are done, like computing estimations of missing data-points, and sometimes deviation checking. This inter-data-points processing time needs to be bounded for the good functioning of our algorithm and also this time may impact the possible values of T_0. In Fig. 5, we

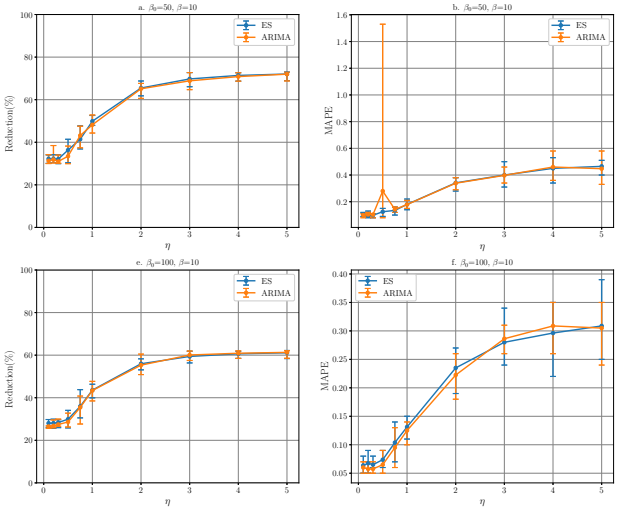

Fig. 3. UNIV2: Overhead Reduction and Collection Error (MAPE)

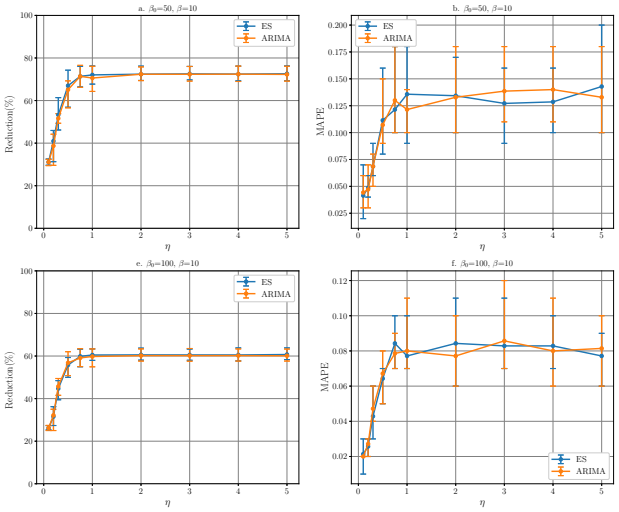

Fig. 4. MAWI: Overhead Reduction and Collection Error (MAPE)

show the range of this computation time for both ARIMA and Exponential Smoothing forecasting. Exponential Smoothing gives better results (around 0.025 seconds). ARIMA gives computation time around 0.125 seconds.

From Fig. 3 and Fig. 4, ARIMA and ES are providing the same quality of forecasting even with the big difference in terms of computation time. Hence for our particular case where the time series presents an increasing trend without seasonality pattern, the better choice for uncollected data-points is Double Exponential Smoothing.

V. Use Case and Discussion

A. Use Case: Network Utilization Monitoring

With our lightweight near-real-time bytes count collection, we contribute a lot to Traffic Engineering in SDN. As part of TE, we can mention bandwidth management that consists of measuring and controlling traffic on network links. Fig. 6

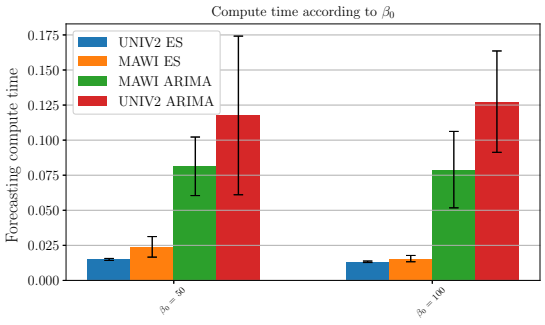

Fig. 5. Compute time

shows bytes count evolution collected with our adaptive push mechanism compared to a fixed push collection and the bandwidth estimated from this statistic. The bandwidth is expressed in bytes per second (Bps).

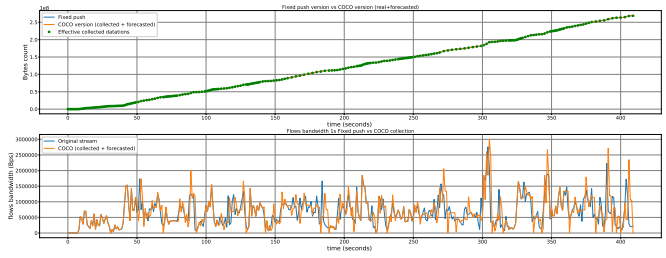

Fig. 6. Bandwidth Estimated from the bytes count collected

B. Discussion

The objective here is to position our work with the recent directions in SDN intending to take complete control over the data-plane with P4, INT, and data-plane programmability.

P4 (Programming Protocol Independent Packet Processor) [13] is a high-level language to express how a switch is to be configured and it has to process packets. It's worth pointing out that P4's goal is not to replace OpenFlow, but OpenFlow may be seen as part of P4 and is one of many possible programs (e.g P4Runtime) to describe the forwarding behavior. Since there, our OpenFlow-based implementation can be considered P4-compatible. Moreover, instead of running our solution as a software agent on data-plane elements, it can be directly integrated into programmable switches through P4.

Generally implemented with P4, INT (In-band Network Telemetry) [14] is a new abstraction that allows data packets to collect switch internal state (queues size, queues latency, byte counts, etc), enabling monitoring of the network state by the data-plane. Even with this approach sometimes we need to collect from a selected node statistics to a central server. For example in [15] a low-overhead network-wide heavy hitter detection is done where large flows are detected locally in the data-plane with a per-key threshold that triggers reports to a central coordinator. A similar approach is used in [16]. At the triggering moment, our adaptive solution will be useful in reducing the overall monitoring task cost.

VI. Conclusion

This paper proposes an efficient adaptive push algorithm for near-real-time flow statistics collection done as a periodic collection. It is based on time series prediction that guides the adjustment of the adaptive periods. Our proposition has the advantage of being lightweight and easy to deploy. We evaluate it using real-world backbone and university data-center traces. It reduces the collection overhead up to 75% compared to a fixed periodic push collection mechanism at the price of almost no accuracy degradation with less than 0.5% of collection error.

As future works, we will investigate the use of machine learning forecasting methods as RNN (Recurrent Neural Networks). And also we will implement our solution with P4 for data-plane elements instead of using it as a software agent.

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 *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 254–265.

[2] E. Akin and T. Korkmaz, "Comparison of routing algorithms with static and dynamic link cost in software defined networking (sdn)," *IEEE Access*, vol. 7, pp. 148 629–148 644, 2019.

[3] Y. Zhang, "An adaptive flow counting method for anomaly detection in sdn," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM, 2013, pp. 25–30.

[4] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "Payless: A low cost network monitoring framework for software defined networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–9.

[5] G. Tangari, D. Tuncer, M. Charalambides, Y. Qi, and G. Pavlou, "Self-adaptive decentralized monitoring in software-defined networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1277–1291, 2018.

[6] J. Suárez-Varela and P. Barlet-Ros, "Reinventing netflow for openflow software-defined networks," *arXiv preprint arXiv:1702.06803*, 2017.

[7] G. Cheng and J. Yu, "Adaptive sampling for openflow network measurement methods," in *Proceedings of the 12th International Conference on Future Internet Technologies*. ACM, 2017, p. 4.

[8] K. B. Nougnanke and Y. Labit, "Novel adaptive data collection based on a confidence index in SDN," in *2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC) (CCNC 2020)*, Las Vegas, USA, Jan. 2020.

[9] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International journal of forecasting*, vol. 22, no. 4, pp. 679–688, 2006.

[10] http://www.alkaline-ml.com/pmdarima/develop/index.html.

[11] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 267–280.

[12] C. Sony and K. Cho, "Traffic data repository at the wide project," in *Proceedings of USENIX 2000 Annual Technical Conference: FREENIX Track*, 2000, pp. 263–270.

[13] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[14] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," in *ACM SIGCOMM*, 2015.

[15] R. Harrison, Q. Cai, A. Gupta, and J. Rexford, "Network-wide heavy hitter detection with commodity switches," in *Proceedings of the Symposium on SDN Research*. ACM, 2018, p. 8.

[16] L. Castanheira, R. Parizotto, and A. E. Schaeffer-Filho, "Flowstalker: Comprehensive traffic flow monitoring on the data plane using p4," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.