Novel Adaptive Data Collection based on a Confidence Index in SDN

Kokouvi Benoit Nougnanke, Yann Labit

Novel Adaptive Data Collection based on a Confidence Index in SDN

Kokouvi Benoit Nougnanke
CNRS, LAAS, F-31400 Toulouse, France
Univ de Toulouse, UPS, LAAS, F-31400 Toulouse, France
Email: nougnanke@laas.fr

Yann Labit
CNRS, LAAS, F-31400 Toulouse, France
Univ de Toulouse, UPS, LAAS, F-31400 Toulouse, France
Email: ylabit@laas.fr

***Abstract*—SDN makes networks programmable by bringing flexibility in their control and management. An SDN controller decides how packets should be forwarded and installs flow rules on the data-plane devices for this end. For efficient control and management, the SDN controller needs to monitor the network state continuously in order to have an accurate and up-to-date view of the underlying data-plane. The need for this high-visibility on the network brings specific types of monitoring as streaming telemetry where data is streamed continuously from network devices (wired switches, wireless energy constrained nodes, etc.) as bulk time series data. But this may generate a lot of overhead. Adaptive monitoring techniques provide ways to reduce this overhead, but generally, they require complex user parameter tuning and also they effectively handle data dissemination overhead with counterpart certain energy-consuming treatment on the source nodes. In light of this, we propose novel adaptive sampling technique based on a confidence index that considerably reduces the number of exchanged messages about 55-70 % while maintaining an accurately collected data, represented by the explained variance score that is about 0.7 and 0.8. And more, our proposition achieves these results while being a lightweight solution for source nodes.**

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

I. Introduction

Software Defined Networking (SDN) [1] is the physical separation of the control plane from the forwarding plane. It eases control and introduces flexibility in network management. Innovation and optimization in networks are also eased. SDN paradigm is used for both wired and wireless networks, but it was traditionally developed for wired environments where it is more mature. Its application to the wireless context has a lot of advantages even if it needs readjustments and adaptations.

For an SDN architecture to work well, and especially in the case of dynamic networks, we need to frequently or continuously monitor the network and collect its information. This monitoring consists essentially of network state information (performance metrics, physical topology, hop latency, queue occupancy, remaining energy, etc.) and traffic/flow information collected from data-plane devices such as wired servers, switches, routers, wireless nodes, base stations, etc. This collected information may help in the control of packet forwarding allowing the controller to adapt flow rules optimally. It is also used by management or monitoring services or applications such as DDoS Attack Detection [2].

Inspired by self-driving cars, self-driving networks constitute a new promising research topic [3] [4]. Their implementation will leverage SDN, machine learning, and automation, just to name a few. And here data collection is of great interest, and more real-time telemetry is needed.

Continuous monitoring is then ineluctable in SDN and this monitoring must not be restricted to only traffic or flow information but all relevant ones that can be useful in any decision making process at the controller level. The only downside is that it may generate a lot of overhead for an SDN architecture. To handle this, adaptive data collection techniques are developed. Based on [5], an overview of network monitoring in SDN, [6] an adaptive approach for monitoring in SDN and [7], presenting a state-of-the-art review of monitoring in edge computing and its requirement, the literature adaptive techniques handle efficiently overhead reduction problem, but they still present these 3 aspects:

- they are tailored for a polled-based monitoring model and generally for traffic and flow information collection. Other information may be useful, and for continuous monitoring a push model is more adequate as in streaming telemetry. So, adaptive techniques have to be compatible with this latter model or more have to be generic;
- they require complex parameters tuning, which limits their adoption;
- the lightness of the solutions for source nodes, that could be energy constrained devices, is questionable.

Considering the aspects discussed above, we propose a new adaptive sampling approach, COCO (information COnfidence index COllection) with the following contributions:

- We propose, for continuous monitoring overhead reduction, an adaptive monitoring framework using a confidence index α assigned to any information to be collected continuously. Its algorithms are presented (section III).
- Then, we carry out experiments on our proposed framework, its evaluation results are discussed (section IV).
- And finally, we present an overview of adaptive monitoring techniques from the literature (section V).

II. Background

A. SDN Controller

SDN controller is the heart, not to say the brain of an SDN architecture [8]. It exercises management-control over

the data-plane devices to satisfy the needs of applications by translating their requirement (high-level intent) into flow rules (instruction) that will be executed by devices [9]. When satisfying these requirements, the controller has to achieve certain goals: QoS provisioning, optimal network resources utilization, security aspects, etc.

The SDN controller ensures the control part essentially with the OpenFlow protocol [10]. For management, protocols used are OF-CONFIG, NETCONF, SNMP, etc.

By controlling and managing the underlying data-plane, the SDN controller plays the role of an active control element in a feedback loop, responding to network events, recovering from failure, reallocating resources [11], etc. Because the data-plane is a dynamic environment that changes over time, the controller by mediating between applications and networks resources [12], continually adapts the state of resources of its data-plane to maintain a desired (optimal) state.

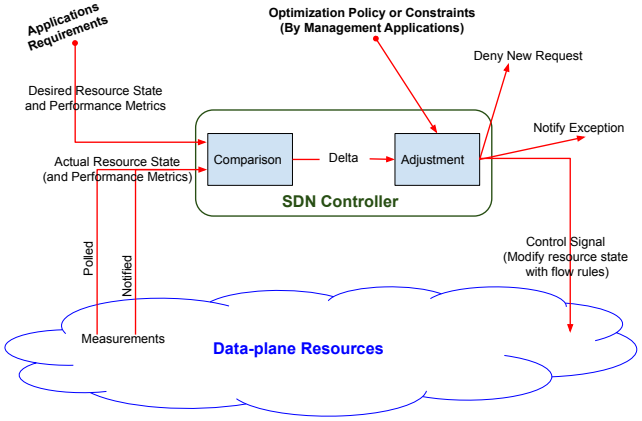

Fig. 1. SDN Controller in a Control feedback loop

From Fig. 1, adapted from [8] [12], we can see control as the process of establishing and maintaining a desired state on the data-plane, and feedback is of utmost importance in this process. The feedback process constitutes a management task (events handling, statistics collection, monitoring, etc.).

B. Data collection in SDN

As shown above, with an SDN architecture we are in the presence of closed-loop of control. For this control loop to be efficient (i.e. allows applications to dynamically control the network while improving its utilization, QoS provisioning, etc.), the most active element of this loop, the SDN controller, has to provide to the applications network information pertinent to their needs and objectives [9]. We can distinguish two main categories of information:

- **Traffic Forwarding information**: flow statistics such as switch interfaces (ports) information like flow volume, data throughput, packet loss, port down or up, etc.
- **Network Resource State information**: related more to the management part e.g. CPU usage, disk usage, etc.

The information may be directly informative (port is down or up) or can be used as performance metrics and so on. As mentioned by ONF in SDN Scope and Requirements this information that can be static (eg. datapath ID) or dynamic (eg. CPU usage) may be collected in several manners: **provisioned**, **discovered**, **queried**, **notified** and **streamed**.

Using these techniques, SDN controller would gather relevant information and provide global view and full knowledge of the data-plane under its control. The data collection importance is not to be proved but is increasing with the advent of machine learning techniques in networking in general [13] and especially in the control loop of SDN [14], machine learning who breathes "only" by data. Monitoring will also have a great role in the implementation of self-driving networks, inspired by autonomous vehicles, which leverages SDN and Machine Learning [3] [4]. Self-driving networks need continuous resource monitoring and streaming telemetry fits well with this requirement [15]. Nevertheless, care should be taken to the abstract level and granularity of collected information and to the cost of this task.

Indeed, Streaming Telemetry [16], powered by the OpenConfig working group, is a new approach for network monitoring in which data is streamed from devices continuously, thus bulk time series data is collected (e.g. statistics are uploaded every 15 seconds). An event-driven update and requested for ad-hoc data are also available. Unlike SNMP, Streaming Telemetry uses a publish-subscribe pattern and is model-driven telemetry enabling vendor-agnostic monitoring. It is based on open-source initiatives like YANG model, OpenConfig data model, gRPC (Google Remote Procedure Call) etc.

C. Problem with periodic monitoring

We have seen in section II-B the stress of network state information collection in SDN. This collection is usually done periodically, and this for several reasons. For example, we want network state information to be available when SDN services or applications request them, proactive approach for the network management. This periodic data collection is not without cost, but more it presents a dilemma: at which frequency (sampling period), data may be collected?

- If we use high frequency, we will have real-time and accurate information but with significant overhead.
- Low frequency resolves the problem of overhead but we will not have accurate network state information.

Continuous monitoring, not to say periodic monitoring at high frequency, is needed to ensure high-visibility and deep or real-time insights on the data-plane. Streaming telemetry (discussed above) has arisen and is compatible with these requirements by streaming data each X seconds to an interested receiver, on a push basis, generating bulk time series data. Concerning overhead, the use of a push model offers a better efficiency compared to a collection in a polling fashion. Furthermore, to add more efficiency and tackle this overhead problem, we propose to stream the data, not with high frequency, but with a **"virtual high frequency"**, or in other words, to use adaptive sampling in the data streaming process. The adaptive sampling procedure is based on a confidence

index α representing the trust the receiver, here the controller, has on the source node for adjusting the reporting interval.

Our adaptive monitoring algorithm consists of adjusting collection period for a push-based monitoring model according to α. Indeed network devices stream to the SDN controller each T_α (equals to $T_{\alpha min} = T_0$, initially) unit of time and this reporting interval increases automatically $[T_{\alpha min}, T_{\alpha max}]$ over time, and it will be adjusted (decreased) only if the receiver side is unable to capture the runtime evolution of the information being collected and thus can not guarantee a certain accuracy, initially defined. Then, it raises a deviation alarm and the reporting interval is decreased. A detailed description of our proposition is given in section III.

III. Our Solution: An Information Confidence Based Adaptive Data Collection

We represent, like [17], the metric M to be monitored as a sequence, at regular time intervals (e.g. T_0 = 1s), of values m_k, $M = \{m_k\}_{k=0}^{n}$, n $\to \infty$. These n + 1 observations $m_0, ..., m_n$ form a time series and m_k may be seen as a tuple (id, $\mathbf{t_k}$, **value**, α_k). With our adaptive sampling algorithm, only some of these samples m_k will be pushed to the SDN controller by the network node, source of the monitoring stream M.

The data collection process between the controller and the node according to our proposition COCO (information COnfidence COllection) is described in Fig. 2.

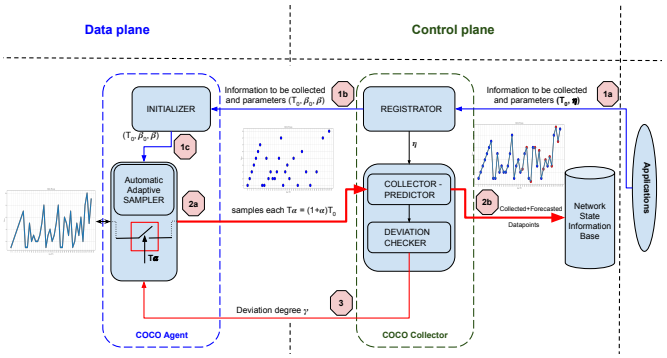

Fig. 2. COCO Framework

Initially, a certain application, control or management service expresses its interest in certain information (1a). It specifies the metric M to be collected, at which frequency it may be available with T_0, and η the collection (or forecast) imprecision tolerance indicator. The registrator then initialize the collection process on the node (1b) and (1c).

Information to be collected periodically is given a confidence index α, and T_α is the period with which this information will be pushed i.e. at $t_k = t_{k-1} + T_{\alpha_k}$ (2a). T_α will evolve in a similar manner as TCP's congestion window evolution: slow-start and congestion avoidance. Indeed, we will begin with a low period $T_\alpha(k) = (1 + \alpha(k))T_0 = T_0$ (α = 0) and α will be increased each β samples uploaded if a deviation alarm is not raised by the controller. Otherwise, α will be decreased according to the deviation degree γ (3). This is presented with our Adaptive Sampling Procedure on Algorithm 1.

Algorithm 1: COCO Adaptive Sampling Procedure

Upload β_0+1 metric samples m_k each T_0, $k := 0$ to β_0 ;
$\alpha_{k+1} \longleftarrow \alpha_k$ + 1 and $T_{\alpha_{k+1}} \longleftarrow (1 + \alpha_{k+1})T_0$;
$k \longleftarrow k + 1$ and $j \longleftarrow 0$;
while *Forever* **do**
  Upload m_k at $t_k = t_{k-1} + T_{\alpha_k}$;
  $j \longleftarrow j + 1$;
  if $j == \beta$ **then**
    if $\neg$ *Deviation occurred* **then**
      $\alpha_{k+1} \longleftarrow \min(\alpha_k + 1, \alpha_{max})$;
    else
      $\alpha_{k+1} \longleftarrow \alpha_k$ / (2 * γ) ;
    end
    $T_{\alpha_{k+1}} \longleftarrow (1 + \alpha_{k+1})T_0$;
    $j \longleftarrow 0$;
  end
  $k \longleftarrow k + 1$;
end

Algorithm 2: COCO Forecasting and Deviation Checking

Retrieve the first β_0 + 1 metric samples m_k ;
steps $\longleftarrow \alpha_k + 1$;
while *Forever* **do**
  Retrieve metric sample m_k ;
  if *(m_k is the β th sample for α_k)* **then**
    if *MASE(last_collected, last_forecasted) < MASE_th* **then**
      No Deviation: $\alpha \longleftarrow \min(m_k.\alpha_k+1, \alpha_{max})$;
      Send Asynchronously γ to the source;
    else
      Deviation γ: $\alpha_{k+1} \longleftarrow m_k.\alpha_k$ / (2 * γ) ;
    end
    steps $\longleftarrow \alpha_k + 1$;
    last_collected $\longleftarrow \emptyset$;
    last_forecasted $\longleftarrow \emptyset$;
  end
  Add m_k to last_collected ;
  forecasted $\longleftarrow$ Auto_ARIMA_forecasting ;
  Add the steps th forecast to last_forecasted ;
end

A mutual trust environment exists between the node and the controller. Indeed, the controller allows the nodes to upload samples with confidence α which means each (1+α)T_0 and the node trusts the controller to be able to have good forecasts and then an accurate representation of the metric evolution based on past data-points.

The COCO Agent is executed by the node (switch) as a software-based monitoring solution, of course in collaboration with the SDN controller which processes collected samples, forecasts uncollected ones and raises deviation alarm if needed. The complete process on the controller is described with Algorithm 2 and its details are given below.

The controller receives information samples at $t_k = t_{k-1} + T_{\alpha_k}$ from the node. Firstly, it stores it and then it forecasts step equals to $\alpha + 1$ next data-points (2b). The first α th data-points may be used in place of uncollected ones and the latter will be used in deviation checking process. The forecast

is done, on previously collected samples and forecasts, using the univariate statistical time-series ARIMA (AutoRegressive Integrated Moving Average) model. ARIMA parameters (p, d, q) are computed automatically using Auto ARIMA [18] which chooses the best model order, selecting the combination that minimizes models quality estimators such as the Akaike Information Criterion (AIC) or the Bayesian information criterion (BIC). For convenient time series forecasting, we need to satisfy a minimum training set size which generally is 50. But this is for general cases with seasonality. For this study, we focus on non-seasonal time series and 30 gives good results. for this reason, we define a specific β, β_0 equals to 30 (default) for α equals to 0 in the beginning.

When the received sample is the β th data-point for the current α, deviation checking is done using Mean Absolute Scaled Error (MASE) on β last forecasts, β last really collected data-points, according to the training set (previous data-points) used by the forecasting model. MASE, a measure of the accuracy of forecasts proposed in [19], has many desirable properties compared to existing forecasting errors measurements, and it fits well with our deviation checking task. It is computed using Eq. 1, with y_k representing a real observation, which is the value contained in the metric sample m_k, and $\hat{y_k}$ its corresponding forecast.

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

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 MASE_th = 1 + η which will be used in the deviation alarm raising decision. When the threshold is violated, the controller raises a deviation alarm of degree γ (default γ = 1). γ is sent to the node for updating (decreasing) α then T_α.

An illustration of T_α's evolution is represented in Fig. 3 as the evolution of α since T_α = (1 + α)T_0, where the α_{max} reached is 6 then $T_{\alpha_{max}}$ is 7T_0 and where we observe 5 raised alarms at the following time: (170, 240, 260, 470, 540)T_0.

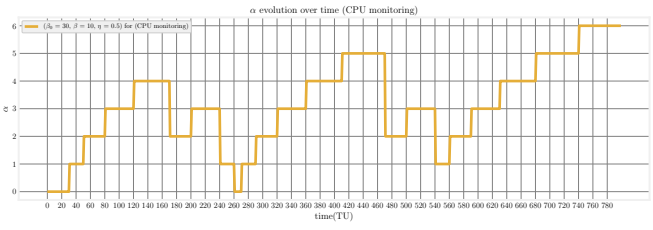

Fig. 3. α evolution over time for a real trace (CPU monitoring data)

Our framework follows these 2 objectives: have a rather accurate evolution (samples) of the target information and with a minimal number of messages exchanged between the controller and the node. This aims to reduce overhead, mainly on the data-plane. Indeed our framework proposes a lightweight adaptive solution for source nodes compared to related works where the adjustments of reporting intervals are done after computing, predictions on the nodes. In our case, apart from the case where a deviation alarm is raised, the adjustment is automatic and most of the computation is done on the controller side.

Also our approach requires minimal external parameters (T_0 and η) all the other parameters are internal to the framework.

IV. COCO EVALUATION

We evaluate our proposition COCO with three real traces and a synthetic one. N represents the number of data-points:

- Battery levels, discharge of a laptop with N = 211
- ARMA generate, synthetic data with N = 300
- CPU monitoring, CPU usage data with N = 800 [20]
- Switch Traffic (in bits per second) with N = 288

Fig. 4 shows the traces (as on the source node) and their corresponding collection at the controller level with COCO.

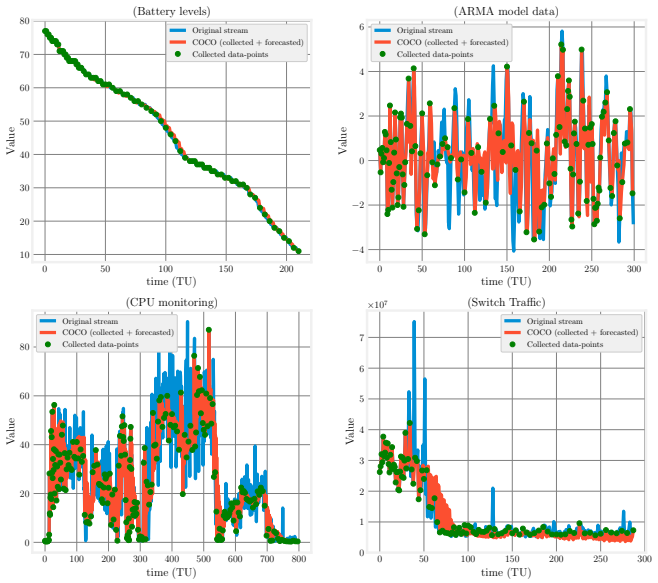

Fig. 4. Original traces and their corresponding collection with COCO

We implement our proposed approach in Python and the simulation experiments were carried out on a machine Intel Core i7-7500U CPU 2.70 GHz x 4 with 16 GB of RAM running Ubuntu 16.04 LTS. For the evaluation, we use the following two metrics:

- **Percentage of the number of samples collected compared to fixed T_0 collection mechanism** for the efficiency of COCO. And we want this value to be as small as possible. It includes also messages from the controller to the node when a deviation alarm is raised.
- **Explained Variation Score (EVS)** for the quality of COCO (accuracy). It expresses the proportion to which COCO accounts for the variation (dispersion) of a data

set. It is computed with Eq. 2, where y represents the original trace and $\hat{y}$ the collected version of y :

$$EVS(y, \hat{y}) = 1 - \frac{Variance(y - \hat{y})}{Variance(y)} \quad (2)$$

The best score possible is 1.0.

Firstly, we evaluate the impact of η, for β_0 = 30 and β = 10, on COCO's efficiency and its quality. η is used in the decision process as an indicator of the tolerable imprecision. Negative values ($>$ -1) means MASE threshold $<$ 1 imposes us to compulsory do better than the naive one-step forecast. And with positive values, we have more flexibility. Fig. 5 shows the importance of η on COCO efficiency and its collection quality. Indeed increasing its value improves the percentage from 70% to around 35% while maintaining a quite stable EVS 0.75, 0.8. An interesting result is for the Battery levels trace (with a quite simple evolution), where we have an EVS equals to 1.0 when η increases. More interesting this is followed by a decreasing of the percentage of messages exchanged. It reaches its lowest value of 36.49% from η equals to 1.75, and this with the same EVS 1.0.

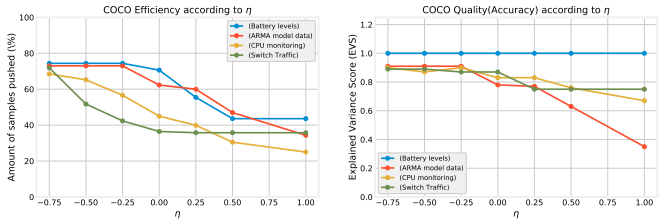

Fig. 5. The impact of η on the collection efficiency and quality

These interesting results presented may be seen in an intern point of view of COCO with α evolution according to η, represented in Fig. 6 by α_{max} reached and the number of deviation alarms raised. They evolve in opposite manner when η increases.

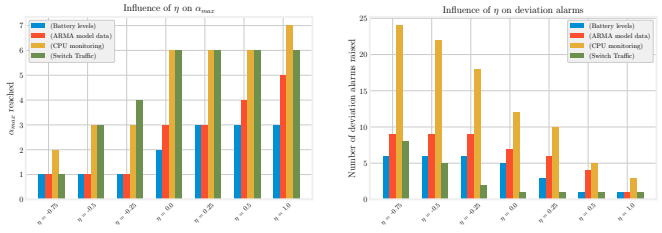

Fig. 6. α_{max} reached and Number of deviation alarms raised

Then, in Fig. 7 (Left), for η equals to 0.5, we present the percentage of the number of samples pushed to the controller with COCO compared to a fixed period T_0 mechanism for different values of β. This percentage at β equals to 10 is around 30% and 45% which represents a reduction of 55%, 70% of the number of messages exchanged to have at the controller level an acceptable accurate evolution of the metric on the target node. The accuracy is presented with EVS on the right graph. We can see also that COCO's efficiency is more important in the long term, where represented with the size of the traces. So we have the lowest percentage with CPU monitoring trace with 800 data-points. It depends also on the values of β. Obviously when increasing β, this percentage increases also, and this with relatively the same EVS. Slower values of β are then recommended.

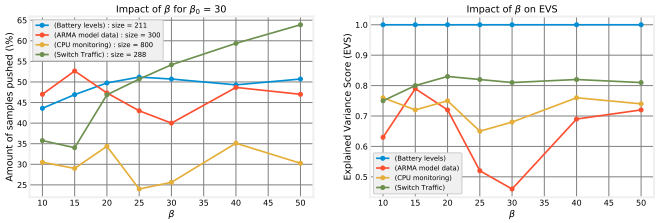

Fig. 7. Efficiency (Number of samples collected compared to Fixed T_0 collection mechanism)

To sum up this evaluation, we see that our proposed adaptive approach reduces considerably the number of messages exchanged in continuous monitoring, with good accuracy, and with the great advantage to being lightweight for source nodes, that mainly have just to push the samples with automatic sampling rate.

V. Related Works

Lots of adaptive algorithms are provided in the literature from wireless sensor networks data collection [21] and Internet of Things (IoT) [17] to monitoring in networking and especially in SDN for monitoring overhead optimization. As pointed out in [6], we can distinguish two main techniques:

- **Threshold-based adaptive monitoring**, where the monitoring period is adjusted by comparing the two last measurements variations to an upper and a lower thresholds. Payless [22] uses this approach. Its authors focus on the trade-off between monitoring accuracy and network overhead by proposing a frequency adaptive statistics collection scheduling algorithm applied on selected switches. Payless's adaptive algorithm adjusts dynamically polling frequency based on collected data granularity.
- **Prediction-based adaptive monitoring**, where the next measurement is predicted with last ones. the deviation between the forecast and the real observation guides the period adjustment.
  This approach is used in [23] which uses a linear prediction based dynamic adjustment scheme to provide dynamic zooming into the flow space (temporal and spatial dimensions).

Authors in [6] propose SAM (Self-tuning Adaptive Monitoring) which uses a prediction-based approach to dynamically adjust polling rate but with minimal parameter tuning effort, not like [23] and [22].

Other relevant works on the topic include [24] and [25]. OpenNetMon [24], for traffic monitoring (flow-related information collection with OpenFlow), uses an adaptive polling rate that increases and decreases respectively when the measurement values differ between samples and when the values

stabilize. TENNISON [25] is a distributed security framework with effective and proportionate monitoring. It offers multi-level monitoring using appropriate tools (sFlow, IPFIX, DPI) from layer 1 to layer 2. Its sampling and polling rate are dynamically adjustable based on three thresholds.

Although the adaptive monitoring works presented above are very interesting, they are mainly tailored for traffic monitoring using a polling approach on the first hand. And on the other hand, most of them require complex parameters tuning. For fine-grained management at the SDN controller level and with the advent of self-driving networks with its real-time telemetry requirement, we do not need only flow-related information collection with a poll-based approach but all relevant information that may contribute to the decision making process.

Taking into consideration all this, we have proposed COCO, an adaptive data collection framework for a push-based monitoring model and with low parameter tuning. It uses the two main techniques presented above: a **prediction technique** to forecast uncollected data-points and a **threshold decision making process** for deviation alarm raising, that expresses if T_α should be increased or decreased (section III). It has also the advantage of being lightweight for source nodes.

VI. Conclusion

In this work, we presented an adaptive data collection framework in an SDN environment. It is based on a confidence index and with the evaluation results, it allows reducing periodic monitoring overhead about 55-70 % with a good accuracy represented by the explained variance score that is about 0.7 and 0.8. Compared to the state-of-the-art solutions it has the advantage to be a lightweight solution for data-plane nodes. This is very interesting, especially in Software Defined Mobile Ad-hoc NETworks (SD-MANETs) where nodes are constrained in energy but we need a continuous collection of the data-plane state for the good functioning of the architecture. Then SDN advantages could be brought efficiently to other environments like MANETs without its shortcomings, mainly the overhead generated by communications between the separated control and data planes. And this is a good start to stem the pessimism around the application of the SDN paradigm to the Wireless Multi-hop networks because of the high cost of data plane and controller interactions.

In our future works, we will implement our proposition in a real platform, continue the evaluations by comparing it to other related techniques.

References

[1] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, "Software-defined networking (sdn): Layers and architecture terminology," Tech. Rep., 2015.

[2] X. Yang, B. Han, Z. Sun, and J. Huang, "Sdn-based ddos attack detection with cross-plane collaboration and lightweight flow monitoring," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–6.

[3] N. Feamster and J. Rexford, "Why (and how) networks should run themselves," *arXiv preprint arXiv:1710.11583*, 2017.

[4] P. Kalmbach, J. Zerwas, P. Babarczi, A. Blenk, W. Kellerer, and S. Schmid, "Empowering self-driving networks," in *Proceedings of the Afternoon Workshop on Self-Driving Networks*. ACM, 2018, pp. 8–14.

[5] P.-W. Tsai, C.-W. Tsai, C.-W. Hsu, and C.-S. Yang, "Network monitoring in software-defined networking: A review," *IEEE Systems Journal*, 2018.

[6] 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.

[7] S. Taherizadeh, A. C. Jones, I. Taylor, Z. Zhao, and V. Stankovski, "Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review," *Journal of Systems and Software*, vol. 136, pp. 19–38, 2018.

[8] ONF TR-521, "Sdn architecture, issue 1.1, 2016."

[9] ONF TR-516, "Framework for sdn: Scope and requirements, version 1.0, june 2015."

[10] O. S. S. Version, "1.5. 1 (protocol version 0x06), december, 2014."

[11] ONF TR-502, "Sdn architecture, issue 1, june 2014."

[12] S. Schaller and D. Hood, "Software defined networking architecture standardization," *Computer Standards & Interfaces*, vol. 54, pp. 197–202, 2017.

[13] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 1, p. 16, 2018.

[14] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, and Y. Liu, "A survey of machine learning techniques applied to software defined networking (sdn): Research issues and challenges," *IEEE Communications Surveys & Tutorials*, 2018.

[15] D. Rafique and L. Velasco, "Machine learning for network automation: Overview, architecture, and applications [invited tutorial]," *Journal of Optical Communications and Networking*, vol. 10, no. 10, pp. D126–D143, 2018.

[16] http://www.openconfig.net/.

[17] D. Trihinas, G. Pallis, and M. Dikaiakos, "Low-cost adaptive monitoring techniques for the internet of things," *IEEE Transactions on Services Computing*, 2018.

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

[19] 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.

[20] I. Shafer, K. Ren, V. N. Boddeti, Y. Abe, G. R. Ganger, and C. Faloutsos, "Rainmon: an integrated approach to mining bursty timeseries monitoring data," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2012, pp. 1158–1166.

[21] D. Laiymani and A. Makhoul, "Adaptive data collection approach for periodic sensor networks," in *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE, 2013, pp. 1448–1453.

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

[23] 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.

[24] N. L. Van Adrichem, C. Doerr, and F. A. Kuipers, "Opennetmon: Network monitoring in openflow software-defined networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–8.

[25] L. Fawcett, S. Scott-Hayward, M. Broadbent, A. Wright, and N. Race, "Tennison: A distributed sdn framework for scalable network security," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 12, pp. 2805–2818, 2018.