A Low-overhead Network Monitoring for SDN-Based Edge Computing

Hou-Yeh Tao, Chih-Kai Huang, Shan-Hsiang Shen

A Low-overhead Network Monitoring for SDN-Based Edge Computing

Hou-Yeh Tao
National Taiwan University of Science and Technology
Taipei City, Taiwan
m10515075@mail.ntust.edu.tw

Chih-Kai Huang
Univ Rennes, Inria, CNRS, IRISA
Rennes, France
chih-kai.huang@irisa.fr

Shan-Hsiang Shen
National Taiwan University of Science and Technology
Taipei City, Taiwan
sshen@csie.ntust.edu.tw

***Abstract*—Using Software-Defined Networking (SDN) in edge computing environments allows for more flexible flow monitoring than traditional networking methods. In SDN, the controller collects statistics from all switches and can communicate with switches to dynamically manage the entire network. However, monitoring per-flow or per-switch mechanisms to obtain the flow statistics from all of the switches may significantly increase bandwidth costs between switches and the control plane. In this paper, we propose a Bandwidth Cost First (BCF) algorithm to reduce the number of monitored switches and therefore lower the monitoring cost. The experiment results show that our algorithm outperforms the existing technique by reducing the number of monitored switches by 56%, leading to a reduction in bandwidth overhead of 41% and switch processing delay by 25%.**

***Index Terms*—Monitoring, Software-defined networking, Multi-access edge computing environment, Reroute network flows.**

I. Introduction

The advent of distributed computing concepts, such as fog or edge computing, has opened up new opportunities to process data from sources outside traditional data centers and support low-latency, high-bandwidth consuming applications. Software-Defined Networking (SDN) splits the network into a control plane and a data plane; then it relies on the central controller to control the whole network, which suits to support multi-access edge computing environments [1]. Monitoring the network in such environments is essential to detect and suspend malicious traffic efficiently, especially since the volume of global Distributed Denial-of-Service attacks (DDoS) rapidly grows. Software-defined networking provides flexible network resource management and programmable traffic controls. The detection and defense of network attacks in SDN can be easier with the global view of the network provided by the SDN controller.

However, monitoring the network is a high-resource consuming activity. It is critical to monitor the network accurately and promptly for effective network management and to detect malicious traffic. Moreover, it is particularly challenging in edge computing environments where resources may be limited and network traffic more diverse and dynamic. The deployment of edge computing is geo-distributed that may cover a vast region or even an entire country. This high network traffic across such a large area will lead to a waste of network bandwidth. As a result, network monitoring encounters a trade-off between accuracy and resource consumption.

In SDN, the controller polls switches to collect statistics on the active flows and there are two main approaches for flow statistics collection, per-flow collection [2], [3] and per-switch collection [4], [5]. However, both schemes may cause overhead on the control channel bandwidth and lengthy switch processing delays. The per-flow collection involves the controller sending a request to a switch to collect traffic statistics for each individual flow. However, if there are many flows on the network, each collection request and reply will result in high overhead on bandwidth and switch CPU usage. On the other hand, the per-switch collection is when the controller sends a request to gather the traffic statistics of all flow entries on the switch's flow table from a switch. However, this method may collect redundant flow information from other switches. When flows pass through multiple switches, the same flow statistics are collected redundantly in the flow table. This causes the wasteful expenditure of resources in collecting duplicate flow information. Excessive resource consumption overloads the control channel and may eventually saturate the existing control channel so that switches cannot connect to the controller for management.

To address this problem and minimize monitoring overhead, we propose a solution that utilizes OpenFlow multipart messages to monitor flows on switches and introduce the Bandwidth Cost First (BCF) algorithm, which focuses on reducing the number of monitored switches to lower the overall monitoring costs. To further reduce the number of monitored switches, BCF is considering rerouting certain network flows, directing them from lower-usage switches to higher-usage switches. This approach would enable a controller to monitor only the higher-usage switches, which can collect statistics for multiple flows in a single control message.

Our evaluation shows that BCF can significantly reduce the number of monitored switches by selecting crucial switches and rerouting certain flows to gather statistical information. Compared to the state-of-art, our algorithm can reduce the number of monitored switches by 56%, which leads to a decrease in both bandwidth overhead of 41% and switch processing delay of 25%.

The rest of this paper is organized as follows. Section II

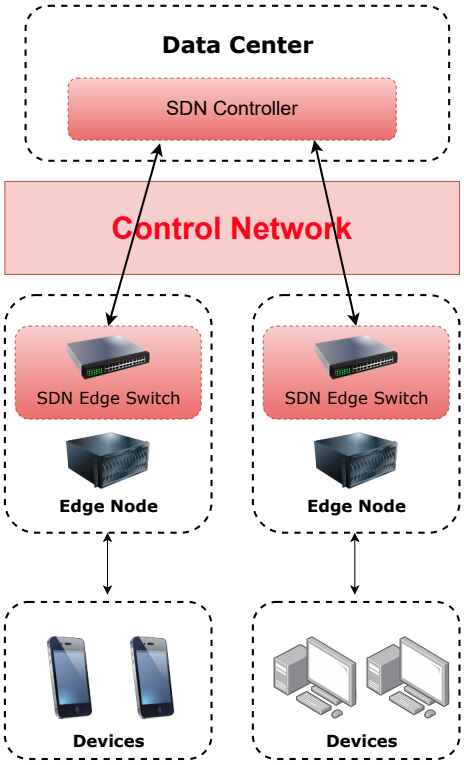


Fig. 1. High-level schematic of SDN-based edge computing network

reviews related work on MEC, SDN, and network monitoring. Section III shows the monitoring cost function and explains the detail of BCF. Section IV evaluates the overhead of network monitoring and presents the simulation results. We conclude this paper in Section V.

II. Related Work

Multi-access Edge Computing (MEC) or fog computing expands on cloud computing by incorporating computing resources closer to end-users. This concept has received much attention from academia. Many prior works have presented different facets of this field, including placement of jobs and services [6], seamless application migration [7], and monitoring computing resources [8]. However, these works are based on the traditional network that does not have enough flexibility to control the network. To overcome this issue, we are witnessing research works applying Software-Defined Networking (SDN) as their network solution for MEC [9]–[14].

To monitor the network, the most famous monitoring solutions in traditional networks are NetFlow [15] and sFlow [16]. NetFlow is the most prevalent monitoring tool that attaches switches to complete or sample traffic statistics. sFlow provides a network traffic sampling mechanism to collect traffic information by the sFlow agent. However, these solutions with the specific hardware and software for monitoring have a high overload on bandwidth and CPU resources [2], [17], [18]. In order to mitigate the monitoring overhead caused by specific hardware and software, employing SDN to monitor the network is a promising solution.

Within a software-defined networking environment, two types of traffic monitoring solutions exist: passive monitoring [19]–[21] and active monitoring [2], [3], [22]–[28]. Passive monitoring collects information about the network without injecting any additional traffic. This approach can be less intrusive and consume fewer resources than active monitoring. Active monitoring involves injecting additional traffic in the form of probe packets, which are sent to the network to collect information about the status of the network. This approach can be more accurate in measuring performance metrics but can also introduce additional traffic and consume network resources.

In the passive monitoring solutions: OpenNetMon [2] polls the flow information on the ingress and the egress switches. MicroTE [20] is a passive monitoring mechanism that adapts to traffic variations by leveraging the short-term and partial predict network traffic. MicroTE considers that 80% flows from the data center is not live longer than 10 seconds, and below 0.1% flows live longer than 200 seconds if any elephant flow occurs. OpenSample [21] is a low-latency sampling-based network considering 99% flows in the data center is TCP protocol. It captures TCP sequence numbers from header fields and uses the same packet samples to estimate port utilization to reconstruct flow statistics. OpenSketch [29] separates the measurement data plane from the control plane and provides a three-stage pipeline (hashing, filtering, and counting) in the data plane. Using a three-stage pipeline approach, we can use hashing to reduce the bandwidth and filter the coarse-grand and fine-grand flows.

In the active monitoring solutions: OpenTM [3] estimates the network traffic matrix by directly measuring the traffic. It distributes the measurement tasks to multiple switches in the network to reduce the overhead. However, this approach may lead to high bandwidth and delay costs, especially in large networks with high traffic volume. Another similar method is PayLess [22], which offers an adaptive scheduling algorithm for the polling from a controller to switches. PayLess focuses on the polling rate of the switches by considering monitoring accuracy and the reduction of control channel communication overheads. Planck [24] is a network measurement architecture that uses port mirroring to gather network information, improving the accuracy of flow collection. This approach overcomes the limitations of traditional methods constrained by the sampling rate bottleneck, which can negatively impact monitoring accuracy. Lonely-Flow-First (LFF) [26] considers that polling unnecessary switches makes redundant flow statistics so that LFF reduces monitored switches to lower monitoring costs. Low-Cost Monitoring (LCM) [27] algorithm is an active monitoring method focusing on the monitoring cost by reducing the monitored switches. The LCM algorithm considers minimizing the monitoring bandwidth consumption and reporting delay. However, both LFF and LCM did not make good use of the flexible mechanisms of SDN. The SDN architecture provides flexibility, allowing flows to be easily

rerouted, and can reduce the need for monitored switches and associated costs.

To minimize the monitored switches and bandwidth cost in SDN-based edge computing environments, we introduce Bandwidth Cost First (BCF) algorithm to benefit from fewer monitored switches that bring the knock-on effect of monitoring overhead. Our approach will also reroute some parts of the network flow from low-usage switches to high-usage switches to reduce the number of monitored switches further.

III. Algorithm Design

The objective of this paper is to reduce monitoring costs by decreasing the number of monitored switches and minimizing bandwidth consumption in the control channel. In this section, we discuss the design of the Bandwidth Cost First (BCF) algorithm and its time complexity.

A. Idea of BCF Algorithm

In this paper, we design an algorithm that operates on top of an OpenFlow controller to minimize monitoring costs. The algorithm aims to reduce the number of monitored switches corresponding to the number of control messages. By using OpenFlow, we also avoid installing unnecessary monitoring hardware or software, such as sFlow on SDN, which requires numerous ports to receive flows and establish flow rules on each switch based on the number of monitored switches.

Switches rely on flow tables to convey flow information, such as packet count, byte count, and duration time. However, multiple switches along a flow's routing path may collect the same flow information. The controller can collect flow statistics from a single switch to avoid redundancy instead of gathering redundant information from all switches along the path. Collecting flow statistics from fewer switches reduces the flow information sent to the controller, minimizing control channel bandwidth costs. Therefore, selecting a subset of switches to collect flow statistics while ensuring that all flow information is still monitored can be a more efficient approach.

To further reduce the monitoring cost and leverage the flexibility of SDN, BCF will consider rerouting some parts of the network flow to minimize the number of monitored switches. In SDN, the rerouting of traffic can be achieved through SDN policies, which can be easily updated and modified without requiring significant changes to the underlying network infrastructure. By rerouting traffic from switches with lower flows to switches with higher flows, we can minimize the number of monitored switches while achieving comprehensive flow monitoring. Monitoring fewer switches with more flows can lead to better cost-performance ratios regarding bandwidth and delay cost. This approach is more efficient than traditional methods and does not require network hardware or software configuration changes.

B. Bandwidth Cost First (BCF) Algorithm

This subsection provides the problem formulation for BCF to clarify the issue and proposes an algorithm to address the issue.

TABLE I
Notations Table

Notation	Description
V	A set of switches $\{v_1, v_2, v_3, ...\}$
V_c	A set of switches that already covered
F	A set of flows $\{f_1, f_2, f_3, ...\}$
$\|F\|$	The number of flows in flow set
F_{v_k}	A set of flows that pass through switch v_k
$F^c_{v_k}$	A set of flows that covered in switch v_k
α	Threshold for limit reroute range
L_{req}	Length of request packet
L_{rpyh}	Length of reply packet header
L_{bf}	Length of one flow statistic in reply packet
d_{v_k}	Distance between controller and switch v_k
L_P	Latency of switch process of one request packet
L_{df}	Latency of switch process of one flow statistic
$sp(f_n)$	A set of switches which are the shortest path of f_n pass through
$rsp(f_n, v_k)$	A set of switches which are the shortest path of f_n reroute to v_k

1) Problem formulation: Given a network topology $G(V, E)$ and a set of flows F, where V is a set of switches and E is a set of links between two switches. The goal of BCF is to reduce the number of monitored switches. Reducing the number of monitored switches can improve the overall latency of monitoring task processing in all switches. To this end, the object of the problem is

$$Min. \sum_{v \in V} x_v \tag{1}$$

Where $x = 1$ indicates if switch v is selected as a monitored switch.

$$\sum_{v \in N_s} \varepsilon_{f,s,v} - \sum_{v \in N_s} \varepsilon_{f,v,s} = 1 \ \ \forall f \in F \ \ and \ \ s \in S^f \tag{2}$$

$$\sum_{u \in N_v} \varepsilon_{f,v,u} - \sum_{u \in N_v} \varepsilon_{f,u,v} = 0 \ \ \forall f \in F \ \ and \ \ v \in V \tag{3}$$

$$\sum_{v \in N_d} \varepsilon_{f,v,d} - \sum_{v \in N_d} \varepsilon_{f,d,v} = 1 \ \ \forall f \in F \ \ and \ \ d \in D^f \tag{4}$$

Next, we formulate some constraints that must be met, including traffic routing, because BCF takes flow rerouting into consideration. Equation 2 shows the number of the output flows from source s should be one more than the number of input flows, and the number of input flows to destination d should be exactly one more than the number of output flows as Equation 4. Moreover, the number of output flows from any other switches should equal the number of input flows shown in Equation 3.

$$\sum_{v \in V} \gamma_{v,f} \geq 1 \ \ \forall f \in F \tag{5}$$

$$\sum_{u \in N_v} \varepsilon_{f,v,u} \geq \gamma_{v,f} \quad \forall f \in F \ \ and \ \ v \in V \tag{6}$$

$$\gamma_{v,f} \leq x_v \quad \forall f \in F \ \ and \ \ v \in V \tag{7}$$

BCF should satisfy all monitoring tasks by monitoring all target flows, which means the selected switches should cover all target flows. If a flow f goes through the switch v ($\varepsilon_{f,v,u} = 1$) that is selected as a monitored switch (x_v), the flow is covered. Each flow should be covered by at least one monitored switch; thus, the summation of $\varepsilon_{f,v,u} x_v$ should be equal to or larger than one as shown in Equation 5.

$$\sum_{v \in V, u \in N_v} \varepsilon_{f,v,u} \leq \alpha SP_f \quad \forall f \in F \tag{8}$$

Finally, to limit data plane latency, the length of rerouted paths should be restricted by Equation 8, where α is a parameter to determine the rerouting limitation. This equation shows that the length of rerouted paths cannot be longer than α times their shortest paths.

2) Algorithm: The BCF algorithm reduces the number of monitored switches to minimize the bandwidth cost. The algorithms comprise two phases: (A) selecting the monitored switches and (B) rerouting the network flow. In Phase (A), the algorithm selects the minimum number of switches to monitor the traffic. First, the algorithm scans all switches in the network and counts the number of flows passing through each switch. Then, the algorithm sorts the switches according to the number of flows passing through and the bandwidth consumption by Equation 10 in descending order. Considering the bandwidth consumption, selecting the switch closest to the controller and with the highest number of flows is prioritized. To determine monitored switches, the algorithm iteratively selects switches as monitored switches until all flows in the network are covered. In each iteration, the algorithm selects the switch covering the highest number of uncovered flows, which can cover more flows. After each iteration, the network flows that have already been covered are removed from the switches that have not been selected. The monitored switches selection phase provides a monitored switch set V_c that the controller only needs to monitor these switches.

To further reduce the number of monitored switches, the algorithm runs phase (B): rerouting network flows. The basic idea is to reroute the flows from lower to higher usage switches. After rerouting, the controller can monitor the lower number of switches. In the first step of phase (B), the algorithm uses the ascending order to sort monitored switch set V_c by the number of covered flows. If the number of monitored flows in switch ($v_m \in V$) ($|f_{v_m}|$) is lower than a monitored switch ($v_n \in V$), the algorithm iteratively tries to combine monitored switch ($v_m \in V_c$) to another monitored switch ($v_n \in V$). After rerouting f_{v_m} to switch ($v_n \in V$), the controller polls the flow information from single switch v_n instead of both v_n and v_m. With the rerouting limit α, if the rerouting costs for all rerouted flows are lower than α, then the v_m is combined with v_n. Otherwise, the combination is rejected to avoid higher extra end-to-end latency caused by a longer path. The flow rerouting phase keeps processing the combination of monitored switches until no monitored switch can be combined. Eventually, fewer monitored switches are required in a network to collect traffic information for security and consume fewer control plane resources.

3) Time complexity: BCF must assign each flow to a monitored switch, which requires scanning all flows and has a time complexity of $|F|$. For each flow, in the worst-case scenario, BCF checks all switches to determine which one can cover the most flows, resulting in a time complexity of $|V|$. After selecting a monitored switch, BCF appends the flows covered by the monitored switch to the set of covered flows that takes $|F|$ time. BCF takes $|V| + |F|$ time to assign flows to a monitored switch. Therefore, the time complexity in the monitored switches selection phase is $|F|(|V| + |F|)$. In the rerouting phase, BCF sorts the monitoring switches in $|V_c|$ and takes $|V|log|V|$ using a quick sort algorithm. Next, BCF checks pairs of monitored switches for combination with a worst-case time complexity of $|V|^2$. Each monitored switch combination takes $|F|$ to reroute flows. In the rerouting phase, the time complexity is $|V|log|V| + |V|^2|F|$. Because $|V| > log|V|$, the time complexity can be shorten to $|V|^2|F|$. Thus, the overall time complexity of BCF is $|F|(|V|+|F|)+|V|^2|F|$.

4) Delay reduction: BCF aims to reduce the number of monitored switches, which can lead to lower overall delay. According to Equation 9, it costs L_{df} to collect statistics for a single flow in switches. With $|F|$ flows in a network, the total latency required is $L_{df} \times |F|$. In addition, each monitored switch incurs an additional latency of L_p to process a control message. Therefore, if there are $|V_c|$ monitored switches, the total cost is $|V_c| \times L_p$. Overall delay cost for the monitoring is

$$|V_c| \times L_p + L_{df} \times |F| \tag{9}$$

where $L_{df} \times |F|$ is fixed cost that BCF cannot reduce. The delay reduction provided by BCF is mainly coming from $|V_c| \times L_p$ and fewer monitored switches result in lower $|V_c| \times L_p$. For example, compared with traditional per-switch collection, BCF reduces the delay cost by $(|V| - |V_c|) \times L_p$.

5) Bandwidth reduction: Regarding monitoring schema, a controller sends a "$ofp\_multipart\_request$" message to a switch. The switch processes the request to aggregate the specific flow information, then replies "$ofp\_multipart\_reply$" back to the controller. Based on [30] and [26] ,the bandwidth cost function in switch v_k is

$$(L_{req} + L_{rpyh} + L_{bf} \times |F^c_{v_k}|) \times d_{v_k} \tag{10}$$

Bandwidth cost function involves $ofp\_multipart\_request$ and $ofp\_multipart\_reply$ with the distance d_{v_k} between a controller and a switch v_k. The parameters of the bandwidth cost function are set as follows: L_{req} as the length of $ofp\_multipart\_request$, L_{rpyh} as the length of the reply packet header, L_{bf} as the length of single flow statistic in reply packet body, and $|F^c_{v_k}|$ as the number of monitored flows in v_k.

IV. Performance Evaluation

In this section, we evaluate the performance of BCF from different perspectives, including the number of monitored switches, delay cost, and bandwidth cost. To compare BCF with related algorithms, we choose the Lonely-Flow-First (LFF) algorithm [26], which prioritizes the collection of statistics from lonely flow switches to reduce polling costs. A lonely flow is defined as a flow that passes through only one switch, making it necessary for the controller to collect flow statistics from that switch. LFF assigns higher priority to lonely flow switches when collecting flow statistics and employs an algorithm to find monitored switches with better cost performance. By monitoring fewer switches, the bandwidth and delay costs are reduced.

A. Simulation Setup

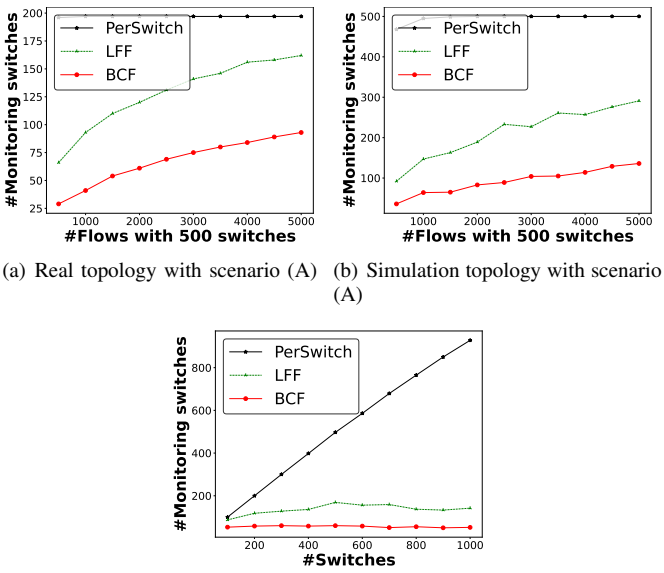
1) Simulation scenarios: This experiment is conducted in two scenarios, which are listed as follows.

Scenario (A): Fixed number of switches and dynamic number of flows in simulation.

Scenario (B): Dynamic number of switches and fixed number of flows in simulation.

2) Network topologies and flows: In our simulation, we randomly generate two topologies. The first topology includes 500 switches and 500 to 5000 flows for scenario (A). The second topology includes 100 to 1000 switches with 1000 flows for scenario (B). Additionally, we run experiments in real-world network topology, which is $Cogentco\text{-}Topology$, to verify the performance of BCF in a real network environment. The routing path for each flow is generated using the shortest path algorithm.

3) Algorithm parameters setup: Based on experiment results obtained from [31], we set the parameters L_P = 1.21 ms and L_{df} = 0.198 ms into delay cost function Equation 9. Bandwidth cost parameters is according to [30] and set the parameters L_{req}= 122 bits, L_{rpyh} = 78 bits and L_{bf} = 96 bits into Equation 10. We vary the threshold α from 1.0 to 2.0, where α = 2.0 implies that we can reroute the origin flow path to a twice-as-long path. On the other hand, if the rerouting threshold α is 1.0, it means there is no rerouted flow path. In our experiments, we set the rerouting threshold α to 1.5 because we believe it is a reasonable trade-off between bandwidth cost on the control plane and high end-to-end latency on the data plane. To avoid obtaining extreme results from the simulation, we run the experiment 100 times with different flow sets on the same topology and then average the results from each experiment.



(a) Real topology with scenario (A) (b) Simulation topology with scenario (A)

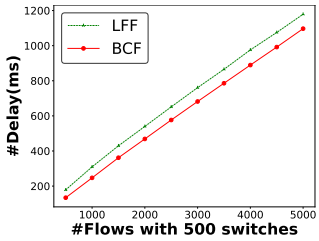
(c) Simulation topology with scenario (B)

Fig. 2. Number of monitored switches

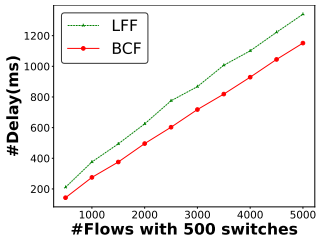
B. Simulation Results

1) The performance results for the number of monitored switches: Figures 2 are the performance results for the number of monitored switches. Figure 2(a) and Figure 2(b) show the results of monitored switches on average in real network topology and the simulated topology, respectively. The flow statistics collection method in LFF combines per-switch and then per-flow polling. Compared with the LFF, BCF collects flow statistics by multipart message, which can collect the specific flows in a single request. Also, LFF does not consider rerouting the flow path. As a result, LFF selects the switches with better cost performance for monitoring, leading to performance improvements compared to the per-switch method. At the same time, our algorithm can significantly reduce the number of monitored switches even more, especially when compared to the per-switch method. The per-switch method is a traditional SDN monitoring approach that collects all flows' statistics on all switches along the paths, resulting in high resource consumption. In contrast, our algorithms offer a more efficient polling mechanism that reduces the number of monitored switches. Figures 2 demonstrate that BCF can significantly reduce the number of monitored switches compared to the per-switch and LFF methods. Given its prioritization of monitored switches, BCF exhibits superior performance. By decreasing the number of monitored switches, the network bandwidth of the control channel can be conserved. Furthermore, in the actual topology, the results indicate an improvement of over 56% compared to the LFF method and 85% compared to the per-switch method.

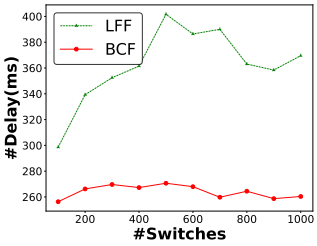
2) The performance results for delay cost: Figures 3 display the performance results of the delay cost metric. Delay cost is defined as the average time it takes for a switch to



(a) Real topology with scenario (A)

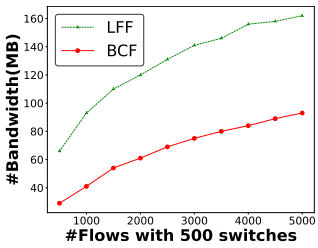


(b) Simulation topology with scenario (A)

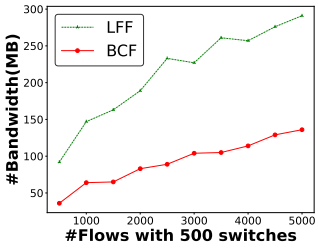


(c) Simulation topology with scenario (B)

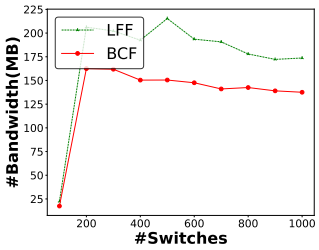
Fig. 3. Delay cost



(a) Real topology with scenario (A)



(b) Simulation topology with scenario (A)



(c) Simulation topology with scenario (B)

Fig. 4. Bandwidth cost

process the monitoring task. Figures 3 demonstrates that the number of monitored switches for BCF is lowest than other methods. At the same time, and as shown in Figures 2, this reduction in the number of monitored switches decreases delay cost.

In the delay cost function of an SDN switch, a monitoring request includes both the request processing time and the flow statistics processing time. However, the request processing time requires significantly more resource time than processing flow statistics. Gathering flow statistics on fewer switches can be an effective approach to reducing global delay costs.

The LFF algorithm's performance improvement is weak due to its need for more monitored switches and tasks to gather flow statistics. In scenario (B), which involves a fixed number of flows and dynamic switches, the delay cost is based on the number of monitored switches, meaning that a higher number of switches will not significantly impact the delay cost. However, LFF exhibits a significant performance gap in scenario (B) because it does not reroute the flows, resulting in more switches that LFF needs to monitor compared to our algorithm. As shown in Figure 3(a), our algorithm achieves a 25% better delay cost improvement than LFF.

3) The performance results for bandwidth cost: The performance of bandwidth cost is presented in Figures 4. For the sake of clarity, we only show the results of LFF and BCF. Bandwidth cost considers monitoring packet length and the distance between switches and the controller. In Figures 4, BCF significantly reduces the bandwidth cost up to 97% compared to the per-switch method. LFF has a weak performance in reducing bandwidth overhead because LFF did not consider rerouting the network flows to aggregate the flow statistics to reduce the bandwidth cost. Moreover, LFF's flow statistics collection method is per-switch and then per-flow polling. Instead, BCF collects flow statistics by multipart message, which can collect the specific flows in a single request. Our simulation shows that BCF outperforms the LFF, which reduces bandwidth cost by up to 41% in the case of real topology with scenario (A).

V. Conclusion

In this paper, we design a flow monitoring algorithm called Bandwidth Cost First (BCF) to reduce resource consumption by choosing crucial switches and rerouting the network flow. Compared to an existing Lonely Flow First algorithm and exam in two topologies with two scenarios, we show that BCF can save more unnecessary resource consumption while monitoring the whole network. Based on the experimental results, it is evident that our algorithm surpasses the current technique by decreasing the number of monitored switches by 56%, which in turn results in a reduction of bandwidth overhead by 41% and switch processing delay by 25%.

VI. Future Work

In future work, we plan to propose an advanced network monitoring algorithm for edge computing architecture, which is more efficient in the resource-limited environment and considers network congestion and QoS. Apart from the network congestion issue, we have observed in the [32] that machine learning were used to reduce network latency in SDN. This is worth considering and referencing for our algorithm.

Acknowledgment

This work is supported by MOST 109-2221-E-011-105-MY3 and MOST 111-2218-E-002 -017 -MBK.

REFERENCES

[1] A. C. Baktir, A. Ozgovde, and C. Ersoy, "How can edge computing benefit from software-defined networking: A survey, use cases, and future directions," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, 2017.

[2] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, "OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks," in *Proc. NOMS*, 2014.

[3] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "Opentm: Traffic matrix estimator for openflow networks," in *Proc. PAM*, 2010.

[4] Z. Su, T. Wang, Y. Xia, and M. Hamdi, "Flowcover: Low-cost flow monitoring scheme in software defined networks," in *Proc. GLOBECOM*, 2014.

[5] Z. Su, T. Wang, Y. Xia, and M. Hamdi, "Cemon: A cost-effective flow monitoring system in software defined networks," *Computer Networks*, vol. 92, 2015.

[6] A. J. Fahs, G. Pierre, and E. Elmroth, "Voilà: Tail-latency-aware fog application replicas autoscaler," in *Proc. MASCOTS*, 2020.

[7] P. Souza Jr, D. Miorandi, and G. Pierre, "Good shepherds care for their cattle: Seamless pod migration in geo-distributed Kubernetes," in *Proc. IEEE ICFEC*, 2022.

[8] C.-K. Huang and G. Pierre, "Acala: Aggregate Monitoring for Geo-Distributed Cluster Federations," in *Proc. SAC*, 2023.

[9] J. Guo, C. Li, and Y. Luo, "Resource management and switch migration in SDN-based multi-access edge computing environments," *The Journal of Supercomputing*, vol. 78, no. 13, 2022.

[10] K. Li, X. Wang, Q. Ni, and M. Huang, "Entropy-based reinforcement learning for computation offloading service in software-defined multi-access edge computing," *Future Generation Computer Systems*, vol. 136, 2022.

[11] C.-K. Huang, S.-H. Shen, C.-Y. Huang, T.-L. Chin, and C.-A. Shen, "S-cache: Toward an low latency service caching for edge clouds," in *Proc. PERSIST-IoT*, 2019.

[12] E. Ollora Zaballa, D. Franco, M. Aguado, and M. S. Berger, "Next-generation SDN and fog computing: a new paradigm for SDN-based edge computing," in *Proc. Fog-IoT*, 2020.

[13] C.-K. Huang and S.-H. Shen, "Enabling service cache in edge clouds," *ACM Transactions on Internet of Things*, vol. 2, no. 3, 2021.

[14] M. Murtadha and B. Mushgil, "Flexible handover solution for vehicular ad-hoc networks based on software defined networking and fog computing," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 13, 2023.

[15] Cisco, "NetFlow," 2023. [Online]. Available: https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html

[16] sFlow.org, "sFlow," 2023. [Online]. Available: https://sflow.org/

[17] S. Rowshanrad, S. Namvarasl, and M. Keshtgari, "A queue monitoring system in openflow software defined networks," *Journal of Telecommunications and Information Technology*, no. 1, 2017.

[18] K. Phemius and M. Bouet, "Monitoring latency with openflow," in *Proc. CNSM*, 2013.

[19] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. Madhyastha, "Flowsense: Monitoring network utilization with zero measurement cost," in *Proc. PAM*, 2013.

[20] T. Benson, A. Anand, A. Akella, and M. Zhang, "Microte: Fine grained traffic engineering for data centers," in *Proc. CoNEXT*, 2011.

[21] J. Suh, T. T. Kwon, C. Dixon, W. Felter, and J. Carter, "Opensample: A low-latency, sampling-based measurement platform for commodity SDN," in *Proc. ICDCS*, 2014.

[22] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "Payless: A low cost network monitoring framework for software defined networks," in *Proc. NOMS*, 2014.

[23] C. Yu, C. Lumezanu, A. B. Sharma, Q. Xu, G. Jiang, and H. V. Madhyastha, "Software-defined latency monitoring in data center networks," in *PAM*, 2015.

[24] J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, and R. Fonseca, "Planck: Millisecond-scale Monitoring and Control for Commodity Networks," in *Proc. SIGCOMM*, 2014.

[25] Q. Huang, X. Jin, P. P. C. Lee, R. Li, L. Tang, Y.-C. Chen, and G. Zhang, "SketchVisor: Robust Network Measurement for Software Packet Processing," in *Proc. SIGCOMM*, 2017.

[26] Z. Yang and K. L. Yeung, "An efficient flow monitoring scheme for SDN networks," in *Proc. CCECE*, 2017.

[27] H. Yahyaoui, M. F. Zhani, O. Bouachir, and M. Aloqaily, "On minimizing flow monitoring costs in large-scale software-defined network networks," *International Journal of Network Management*, vol. 33, 01 2023.

[28] Z. Yang and K. L. Yeung, "Flow monitoring scheme design in SDN," *Computer Networks*, vol. 167, 2020.

[29] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *Proc. NSDI*, 2013.

[30] The Open Networking Foundation, "OpenFlow Switch Specification v1.5.1," 2015. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

[31] H. Xu, Z. Yu, C. Qian, X. Li, and Z. Liu, "Minimizing flow statistics collection cost of SDN using wildcard requests," in *Proc. INFOCOM*, 2017.

[32] S. Iqbal, H. Maryam, K. N. Qureshi, I. T. Javed, and N. Crespi, "Automised flow rule formation by using machine learning in software defined networks based edge computing," *Egyptian Informatics Journal*, vol. 23, no. 1, 2022.