**Plan/feedback - ITC reviews**

**Comments of Reviewer 1:**

**Paper summary (What is the paper about?)**

This paper investigates the challenges of performing network-wide monitoring in SDN networks. Specifically, the authors tackle the problem of distributing monitoring rules within the forwarding elements and, consequently, removing duplicated monitoring rules.

**Strengths (Reasons to accept the paper)**

- the scalability of the monitoring infrastructure is a relevant and important problem that entails understanding the delicate balance between information that is locally monitored and information that is gathered and collected in a global manner.

**Weaknesses (Reasons to reject the paper)**

- the authors do not tackle the problem in a meaningful manner. Namely, the specific aspects of the problem tackled by the authors (ie, distributing monitoring rules) do not present deep research challenges. In fact, the proposed techniques are trivial heuristics of a well-known algorithmic problem called Minimum Multiprocessor Scheduling studied for over 40 years. - the evaluation is performed in a small emulated environment. - the writing quality is poor with many grammar errors.

=> The paper tackled the right problem but the way writing about problem/proposal in paper was not so well. The important and main focus of the proposal should be about the whole monitoring architecture including monitoring agent at switch and global monitoring (distributed or multiple switches monitoring) at controller to improve the adaptation for fine-grained monitoring with large number of flows, not just about the mechanism of switch selection for task balancing. Distributed monitoring mechanism is to support for better adaptation for the architecture, since we believe that in case there are multiple switches that could handle monitoring tasks in a network topology, monitoring load should be shared among switches instead of letting a single one handle all monitoring tasks. It benefits in decreasing monitoring overhead in each switch.

=> Should reorganize paper for a better way of expressing the problem and proposed solution (i.e., explain about the whole monitoring architecture, including the benefits of our approach for monitoring in both cases including single and multiple switches).

**Detailed comments to authors (Remarks that will help the TPC assess the paper and provide feedback to the authors for improving the paper)**

The authors should make an effort to improve the presentation quality of the paper, which contains many grammar errors (I reported a few below). The only algorithmic metric addressed by the authors is the load on the switches. Several comments on this:

- this algorithmic challenge is a specific instantiation of the well-known Minimum Multiprocessor Scheduling (MMS) problem (https://www.nada.kth.se/~viggo/wwwcompendium/node181.html), where switches represent processors and a monitoring rule is a task to be assigned to a monitor. The load of the switches corresponds to the length of the processing load on the processors. Inapproximability and performance guarantees achievable in the MMS problem should be discussed in the paper as well as the proposed dynamic greedy algorithm with respect to dynamic and static algorithms proposed in the literature of MMS.

- OpenFlow 1.3 allows operators to disable counters to reduce CPU usage at switches so it is not clear why a local monitoring agent is needed. Can't the global controller enable and disable counters on the switches based on the forwarding state to be monitored? Why do we need SDN-Mon? Can you explain the role of the Bloom Filters in the architectures?

=> The algorithm in the paper is basically an illustration of processing steps of the proposal mechanism to reject m-entries duplication and keep m-entries assigned in switches in a balancing fashion. It has basic difference (context, processing steps, and functionality) from MMS algorithm as reviewer mentioned. -> paper should consider to revise the current writing/description about proposed mechanism and the system as a whole.

=> Simply enabling or disabling counters on switches does not make any sense as a part of traffic would not be monitored (missed) in the disabled switches. If so, the global monitoring data may not be usable as it only monitors a part of traffic and miss monitoring data of a big number of flows. Moreover, monitor traffic by installing flow entries cause delay/overhead in switch and controller, which result in low performance/throughput especially for monitoring in a fine-grained context for a large number of flows (as showed in our previous performance experiment on Lagopus vs. SDN-Mon).

=> SDN-Mon brings a flexible approach/solution for monitoring with (much) better performance for SDN, especially for fine-grained monitoring (as showed by our performance evaluation/experiment SDN-Mon vs. original SDN).

Bloom filter is for the case of monitoring with sampling rate less than 1 (sampling is used only when traffic is busy that switch could not handle monitoring all flows in traffic). In this case, Bloom filter is used to mark flows/m-entries that were already checked, avoiding it to be repeatedly checked again in m-table in follow-up times when follow-up packets of the same flows coming to switch (Bloom filter is chosen in proposal because it is a light-weight data structure, which is faster for checking existence of elements (flows/m-entries) than checking in m-table).

There are several aspects of the monitoring scalability that would be interesting to study: how to compute the monitoring rules for the given use cases, instead of monitoring the forwarding rules in a one-to-one manner. Understanding the impact of monitoring on the network infrastructure and the different needs for monitoring at the single switch level and global (possibly distributed) controller level.

=> Should revise paper on these good suggestions.

Detailed comments: - Abstract. "However, the current SDN has little support for network-wide monitoring. This results in the duplication of monitoring rules assigned to multiple switches in the network." -> SDN is a paradigm for separating the control and data plane. It does not dictate where and how monitoring should be performed so it is not a limitation of SDN.

=> About this point, paper has unsuitable way for expressing the problem. It would be fixed in revised version of paper. The main point is the scalability/adaptation when fine-grained monitoring requires installing large number of flow entries in switch(es). Current monitoring mechanism of SDN has limited adaptation for that issue. Only a controller program or controlling mechanism (as related papers about switch selection mechanisms) would not solve this limitation/bottleneck, because switch is the one who records traffic monitoring information at first.

- pg 1 col 2. remove "one or some certain" - pg 1 col 2. "unstability" -> "instability" - "These approaches basically rely on unmodified flow tables for monitoring that is inflexible and little scalable for fine-grained monitoring." -> I do not understand what is the limitation. - pg 3. "moinitoring" -> monitoring - pg 3 "supports to prevent " -> prevents - pg 4 "and distributing of the receiving monitoring data to enhance its adaptability and stability for large-scale SDN monitoring" -> does not parse well - pg 4. "set by the maximum" -> set to the maximum - pg 4 "a SDN-Mon" -> an SDN-Mon - pg 4 "flow that traverse through multiple " - > flow that traverses multiple

=> Typing/grammatical errors, would be corrected in the revised version of paper.

- pg 4. "Similarly, S3 is also chosen for monitoring e2, and S1 is chosen for monitoring e3. ". Why? Then the load is not balanced.

=> The author has inexact understanding about this point. As this illustration (Fig. 3), e2 was created in S1, S2, S3 for monitoring f2. Since S3 has lowest memory usage at the current status (31%, compared to 42% of S1 and 54% of S2) and should not have any non-trivial change after a previously adding e1, S3 should be chosen for monitoring e2.

Similarly, e3 was created in S1 and S3 when f3 travel through them. Since S1 has lower memory usage than S2 (42% compared to 54%), S1 should be chosen for monitoring e3. This is a correct logic to assign monitoring entries to switches in a balanced fashion.

- pg 4 "data origanizing" -> data organizing" - pg 4 "can accumulatively becomes" -> can accumulatively become - pg 4 "require a large" -> requires a large - pg 6 buffering tables has not been introduced and defined. - pg 7 " on ruducing " -> on reducing

=> Typing/grammatical errors, would be corrected in the revised version of paper.

**Comments of Reviewer 2:**

**Paper summary (What is the paper about?)**

This paper proposes an extension to improve the scalability of SDN-Mon by detecting when duplicate monitoring entries are generated and removing such entries from all but the least-loaded switch. To quickly compare whether a new SDN-Mon flow report duplicates an existing entry elsewhere, the controller compares the hash of the flow report (calculated by the SDN-Mon switch-side client) to a table of such hashes---equivalent hashes mean duplicated rules, which are then scheduled for removal.

**Strengths (Reasons to accept the paper)**

1 - Addresses a real problem: The controller in an SDN can be a bottleneck, and monitoring resources on switches are limited. 2 - The solution is a nice realization that takes advantage of the "lazy" installation of monitoring rules by a remote monitoring agent. 3 - This is not broadly applicable to OpenFlow, as it requires an SDN-Mon agent to run locally on the switch. However, other emerging measurement platforms, like OpenSketch and FlowRadar, may be able to adopt a similar approach.

**Weaknesses (Reasons to reject the paper)**

The evaluation is quite limited, especially for a paper whose core contribution is about scalability. Specifically, the experiment topology only contained three switches, and the controller was located one hop away, which makes it hard to gauge: - How many switches (i.e. incoming/outgoing monitoring messages) could the controller support? - What is the maximum rate messages can be received by the controller before (a) causing delay in responding, or (b) dropping messages? Would this be a limitation of the algorithm or the infrastructure?

* Evaluate in a more complicated environment (e.g. 6 switches or more).
* Evaluate maximum rate of messages proposed system can support (before causing delay in responding/dropping messages). Thereafter evaluate/inference the number of switches that controller could support.

- Presumably message arrival will be bursty, with one message being sent per switch per flow? It would also be interesting to know what impact the added burden of installing and then immediately removing monitoring rules has on switch throughput.

=> In proposed architecture, not one message being sent per switch per flow. Instead, a reply message is sent each time receiving a (m-entries) request from controller. For each request, switch collects only new m-entries and updated m-entries. Old entries which has no update since previous query/request would not be sent to controller (no need to sent these old entries to controller because of no update in them).

**Detailed comments to authors (Remarks that will help the TPC assess the paper and provide feedback to the authors for improving the paper)**

--------- Strengths ---------

The authors offer a nice insight: When monitoring exists locally on switches in a software-defined network, they lose the global view of the network provided by the SDN controller---which can be recovered through coordination between the switch-local monitoring agents and the centralized controller. In this case, the controller can eliminate redundant flow monitoring rules across multiple switches in a flow's path. This is a simple insight made possible by switch-local monitoring agents. Rather than carefully computing where to place monitoring rules for a particular forwarding configuration, as tools like OpenWatch and FlowCover do, instead simply wait for the local monitors to contact the controller when a flow arrives, and then de-duplicate it. (Indeed, careful rule placement is perhaps not possible when rules are reactively installed by the local monitoring agent without first involving the controller.) As there have been several other switch-local monitoring systems proposed in recent years, including OpenSketch and FlowRadar, this solution may be applicable beyond SDN-Mon.

* Reviewer has realised the strength of SDN-Mon, different from existing works. Indeed, careful rule placement as existing works (e.g., OpenWatch, FlowCover,...) is perhaps not possible when rules are reactively installed by the local monitoring agent without first involving the controller. And requesting controller for every new flows/monitoring rules (as existing works) creates much delay/overhead (to wait for reply from controller to know where to place a monitoring rule), and perhaps not possible, especially for traffic with high rate of new flows).

---------- Weaknesses ----------

This paper is really a scalability paper, with a mechanism for improving the performance of SDN-Mon. Unfortunately, the evaluation is too limited to fully support the claim of improved scalability. There are several issues: 1. Scale. While it's unrealistic to expect large measurements on real networks, I would hope for a simulation at the scale of a small enterprise network at the least, or ideally at datacenter scales, especially because you highlight the problem of controller scalability for handling monitoring data for "large-scale networks and applications that require storing and processing large amount of monitoring data" in your introduction.

* Clarify the targeted points and scope of paper.
* Evaluate proposed mechanism in a more complicated network (as a previous comment)

2. Fair comparison. It shouldn't be surprising that removing duplicate rules from two of three switches results in a roughly 66% reduction in average number of rules installed---presumably the 3% discrepancy has to do with the short period of time when a new (duplicate) rule has been installed and the controller has not yet removed it. It would be nice to see a comparison against other tools like OpenWatch or FlowCover: Is your "lazy evaluation" approach of waiting for flows to be installed and reported to the controller comparable to their "eager" rule installation to avoid duplication?

=> Evaluate further to check about this point

=> Comparison with other tools like OpenWatch or FlowCover.

3. Corner cases. This goes hand in hand with larger experiements: Are there tricky corner cases that degrade performance of the controller or the switches? For example: - What effect is there on switch throughput, now that flows may be set up and then torn down shortly thereafter?

=> SDN-Mon should be considered to support ‘rule expiration’ (e.g. expire m-entry when there is no update for a certain time period).

- Presumably the controller will receive one message per flow per switch, for each switch involved in that flow. What would this look like at scale?

=> Reviewer may misunderstand this point. In proposed architecture, controller receives a reply message when it requests switch, and frequency of request is one per query-time-interval, not one message per flow (as existing works). Switch will send only new and updated m-entries since the previous query. This approach would scale better than the approach of interrupting/requesting controller for every new flow as existing works.

Where does your algorithm "top out" and become unable to support more switches/flows? - What happens if many flows are short-lived and arrive rapidly? Can the controller balance load effectively? Furthermore, the authors should take care to avoid claims of decoupling the monitoring functionality with forwarding; this is a contribution of the existing monitoring framework. It would be nice to see the discussion of related work highlight where other solutions \*are not possible\* in your setting, eg. because the local monitoring agent can take actions independently of the controller.

=> Paper should be considered to evaluate these points additionally.

As an example, the last paragraph of Section II states, "These approaches basically rely on unmodified flow tables for monitoring that is inflexible and little scalable for fine-grained monitoring. Moreover, the lack of support for efficient organizing and managing of global monitoring data at the controller causes these proposals to be little adaptable with large-scale monitoring that demands processing a large number of monitoring entries." First, the fact that these approaches rely on unmodified flow tables is orthogonal. The real question is, do they scale as well as SDN-Mon equipped with the new technique you propose here? (Or does SDN-Mon have capabilities that other tools cannot emulate, which makes the scalability problem harder/unique?) Second, you claim that these other tools lack support for handling global monitoring data. But don't the strategies that OpenWatch and FlowCover take actually reduce the number of messages sent to the controller by preemptively reducing the duplication of monitoring rules?

=> Revise paper about these points (focus on the points that clearly showed/evaluated in paper, avoid claiming these above mentioned points or others that may be beyond the scope of paper). Mainly the strong point of paper is the approach that allows switch actively monitor flows (without interrupting controller for every new flow like existing approaches, which make much delays in switch and overhead in controller), and sharing the monitoring load/tasks over switches in a balancing fashion.

**Comments of Reviewer 3:**

**Paper summary (What is the paper about?)**

The paper proposes a method to provide network-wide monitoring in the software defined network environments. The goal is to reduce the redundant data transmission and monitoring information in switches. The simulation results show some benefits.

**Strengths (Reasons to accept the paper)**

- The problem in the paper is a very important and hot problem. - The paper explains well about the problem and the approach. - The experimental results show some benefits.

**Weaknesses (Reasons to reject the paper)**

- The proposed method requires lots of changes in switches and controller, and the paper does not well articulate the benefits. - How this can scale is not well explained. - The paper does not compare with the state-of-the-arts.

**Detailed comments to authors (Remarks that will help the TPC assess the paper and provide feedback to the authors for improving the paper)**

Major Comments: - The main problem of the proposed method is that it is confined to one SDN controller rather than covering distributed SDN controllers. Other state-of-the-arts such as snet, netflow have a central location where the logs are collected and analyzed. The paper may want to articulate why the proposed approach is better than other methods. - The proposed method requires some changes in both switches and controllers, which may hard to be adopted unless it provides clear benefits.

* The paper belongs to a branch of approaches that leverage switch’s idle resource for monitoring (e.g., for fast response in necessary cases such as anomaly mitigation), basically different from another branch of approaches that requires additional/external servers to collect monitoring data (e.g., netflow, sflow,...).
* At least, SDN-Mon approach benefits in improving scalability of monitoring for SDN (our recent experiments, which SDN-Mon achieves better performance than original SDN, has demonstrated that). The previous paper (‘SDN-Mon: Fine-grained....’) did not show such evaluation and that might be the source of confusion to readers. -> Should include such evaluation to paper (?)

The paper may want to articulate well about what benefits can be obtained by changing switches and controllers. - The paper does not scale to distributed SDN controllers so that it may be hard to obtain what it wants to achieve. Especially, the monitoring system needs to have a global view (end-to-end) to diagnose problems. The local view in a single SDN controller may require collaborative data exchanges unless there is a central collector and analyzer.

=> SDN-Mon supports REST APIs that allows exporting monitoring data to external computers/servers. This could be leverage for distributing monitoring data in case controller could not handle all monitoring data (a similar approach as using multiple controllers). Note that currently SDN/OpenFlow supports multiple controllers but only a single primary controller being in charge of managing network. And since a controller could manage a lot of switches (enough for managing various kinds of networks, as claimed by SDN/OpenFlow), deploying multiple controllers in a network may not be so necessary/beneficial (except for backup/recovery purpose).

- The main motivation of this paper is to remove the redundancy in the network, the paper may want to provide some motivation data points that can show how much overheads the current system imposes. - What it is required in recent SDN monitoring is that an intelligent detection of flow types such as ant or elephant flows, and this could be done in switches rather than in SDN controllers. The paper may want to discuss about what additional information can be obtained through the proposed method.

=> This point could be considered to discuss in revised paper.

- The experiment is limited in that the number of switches is only three and also the number of hosts is just three. There should be scalable test via emulation or simulation.

=> Evaluate with more complicated network/scenario (e.g. 6 switches or more).

Minor Comments: - There are typos and grammar errors.

=> Typos/grammar errors will be corrected in revised paper.

**Comments of Reviewer 4:**

**Paper summary (What is the paper about?)**

The paper proposes an SDN monitoring system to eliminate duplicate monitoring rules and balance monitoring long across network switches. The system relies on SDN-Mon, an existing SDN monitoring framework proposed by the authors which requires custom-built switches that contain an SDN-Mon module.

**Strengths (Reasons to accept the paper)**

The authors correctly recognize that monitoring is tightly tied with forwarding in SDN and propose a scheme to make monitoring lightweight. They propose an algorithm to compress switch monitoring tables by eliminating redundant entries from switches that carry a heavier load.

**Weaknesses (Reasons to reject the paper)**

The motivation of the paper is unclear: why can't operators install monitoring entries in an efficient way from the beginning, thereby eliminating the need for optimizing them later.

=> Installing monitoring entries/rules in a chosen/selective way from the beginning requires switch to ask/request controller (for making decision) for every new flow. This approach (as used in existing works) creates a lot of delay and overhead to switch and controller (as mentioned in above part).

The evaluation is not very strong and uses a small toy topology that is not indicative of a real world network.

=> Evaluate with more complicated network/scenarios.

**Detailed comments to authors (Remarks that will help the TPC assess the paper and provide feedback to the authors for improving the paper)**

The authors propose a method to optimize the placement of monitoring rules by eliminating the redundant ones and placing the remaining ones according to switch load. It is unclear what the scenario for this problem/solution is. Why can't operators optimize the placement when installing the rules in the first place?

* As the above answer.

Can rules be installed by more than one operators? If so, there should be some discussion about rules that are overlapping and how to deal with them. I suggest that the authors clearly describe the assumptions and setting of the problem: how are rules installed? who can install/delete them?

=> Currently rules (m-entries) are installed by a controller, and supporting multi-users/operators is beyond the scope of paper at the moment. This aspect could be considered in future work.

The evaluation of the algorithm is carried over a small topology not representative of a real network. Since the authors are doing a simulation, I suggest at least increasing the size of the topology. This would not solve the real-world network issue but would offer more indications about the scalability of the approach.

=> Same suggest as other reviewers -> should evaluate with more complicated network/scenario.

Some of the figures are unclear. In particular, figure 9 presents the standard deviation of the number of entries. Why is the standard deviation a good metric without the mean around which it is computed. I suggest to add the parameters of the distribution for a better comparison, otherwise it's unclear what the figure shows.

=> Consider to evaluate other parameters, e.g., add parameter of distribution.

**Comments of Reviewer 5:**

**Paper summary (What is the paper about?)**

With the observation that network wide measurements are valuable and, with SDN-controller perspective could capitalise on global optimizations, the authors lay claim to an adaptive distributed mechanism for such network-wide monitoring.

**Strengths (Reasons to accept the paper)**

Monitoring is the Achilles heel in SDN; one more reason preventing operators from considering/adopting it seriously. Despite this, the authors recognise the strength of a control-plane with network wide knowledge and motivate their efforts thus.

=> Reviewer realizes monitoring as an Achilles heel of SDN. The problem that paper is targeting to solve is an important problem in SDN.

**Weaknesses (Reasons to reject the paper)**

Focus on optimal monitoring appears to turn into a more classical workload distribution problem. The constraints about the (need to) optimize rules is just not clear to a reader. Some of the subsystems are simply unjustified (even if this review can see the point but recognizes the authors don't make it clear at all). Eg what are the roles SDN-Mon is actually performing?

* At least, SDN-Mon bring a monitoring solution for SDN that supports it monitor traffic with more flexibility, and (much) better performance (especially for fine-grained monitoring). Moreover, it supports sharing monitoring tasks over switches in network, so that each switch monitor only a part of the whole traffic (decrease/simplify monitoring task in every switch). This feature offers better scalability/adaptability for network monitoring for SDN. (The SDN-Mon journal paper didn’t show our recently performance evaluation where SDN-Mon perform better than original SDN switch, this might be a reason that make reviewers question about benefit of SDN-Mon, as the above question) -> Anyway, paper should consider explaining better about this point in revised version.

**Detailed comments to authors (Remarks that will help the TPC assess the paper and provide feedback to the authors for improving the paper)**

Annoyingly (maybe its me projecting) this really good opportunity for work turns into a really odd paper as some of the most interesting aspects seem ignored or not understood. The evaluations are unconvincing not because of their bad or goodness but because the case for the solutions the paper proposes is not specifically motivated. The authors would do well to describe the underlying constraint motivating this approach(es) - for example, what is the rate of change? is that the reason the approach needs the dynamism and if so, do you meet the need? what causes the blip at 80000 flows? Is this a lot of flows? the authors have not given me enough context to understand if the attempt to fix scaling in measurement systems provides such.

* The approach for describing/expressing and writing about the work/problem/proposal of the current version of paper seems not well. Should consider reorganizing it and use another approach to explain the problem and proposal in the paper (with corresponding to evaluation).
* Distributing monitoring load/tasks to switches should be dynamic. Since load at each switch could change dynamically, monitoring load/tasks assigned to switch should be changed according to its available resource.
* Depending on the traffic, numbers of new/updated flows (m-entries) at switches for a query-time-interval could be different. Thus, the variance of #m-entries at switches could be different in different query-time-interval. This could result in the blip at 80000 flows as reviewer observed. => Anyway, try further evaluation with a more complicated network and with greater number of flows, in reference with #flows required to be handled in some specific kinds of networks/scenarios.