ReMon: A Resilient Flow Monitoring Framework

Fangye Tang
Faculty of Computer Science
Dalhousie University
Halifax, Canada
fangye.tang@dal.ca

Israat Haque
Faculty of Computer Science
Dalhousie University
Halifax, Canada
israat@dal.ca

***Abstract*—Network measurement and monitoring are essential for learning the current network state and act accordingly. Recent trends in network programmability and the Software-Defined Networking (SDN) architecture make measurement and monitoring more flexible and dynamic. However, monitoring tools may suffer from reduced accuracy when collecting measurement data in the presence of link failures. Therefore, in this paper, we propose a resilient monitoring framework, called ReMon, that can efficiently recover from link failures. ReMon combines active measurement along with state aggregation at the data plane switches to furthermore improve the measurement cost. We evaluate its performance in Mininet considering two real-world network topologies. The results confirm that ReMon can provide timely and accurate measurements even in the presence of link failures. It furthermore reduces 50% link failure recovery time compared to its counterparts.**

***Index Terms*—SDN, traffic measurement and monitoring, measurement cost, Fast Failover Group.**

I. Introduction

Network measurement and monitoring are essential for efficient traffic engineering and Quality of Service (QoS). Network measurement must obtain an accurate network state in a timely fashion without hindering the data traffic. Traffic measurement schemes can be *active* or *passive* [1]. The active approach injects probe packets in the network to learn the necessary network state, whereas in the passive scheme statistics directly come from a monitoring device. The passive measurement does not insert any probe packets in the network but requires full access and control on the devices. On the other hand, active measurement enables demand-based statistics gathering, where the accuracy depends on the probing frequency. The higher the rate, the better the accuracy at the price of measurement cost.

In software-defined networking (SDN) [2], [3] design, network control task is decoupled from forwarding devices and placed in a logically centralized controller. This separation brings hardware abstraction, network programmability, and global network view to realizing efficient network monitoring, management, and operations. SDN furthermore enables both active and passive measurements without incurring extra deployment cost. However, It is essential to decide on which switches to be monitored at what interval. For instance, OpenTM [4] generates a query for each source-destination pair to get per-route statistics, which may not scale with the increasing number of switches. FlowSense [5] exploits OpenFlow [6] control messages to learn the network state to reduce the measurement cost at the price of accuracy. FlowCover [7] also relies on the OpenFlow control messages and proposes a flow statistics aggregation heuristic to minimize the measurement cost. None of these solutions considered link failures that can occur in any networks and impact measurement performance. Accurate, high-performance and high-coverage monitoring systems are essential for preventing and recovering from failures in large scale networks. For example, extensive investigations on Google's data center and wide-area networks revealed that 80% of their failures span between 10 to 100 mins, a majority of which lead to a significant packet loss [8].

In this paper, we propose a flow monitoring framework, called *ReMon* that exploits both the packet probing and statistics aggregation to balance between the measurement accuracy and cost. In particular, we first design a new algorithm, called *Weight Assisted Selecting (WAS)*, that uses OpenFlow control messages to select a subset of switches to be monitored to reduce the monitoring cost. Then, we deploy sFlow [9] to poll flow statistics from those chosen switches that aggregate flows from the entire network. We consider to use sFlow to offer fine-granular measurement ability as OpenFlow control messages (e.g., *Packet_In* and *Flow_Removed*) do not have such granularity.

Furthermore, we consider the fact that link failure impacts the measurement time and accuracy while gathering statistics from a chosen set of switches. However, no measurement and monitoring scheme addresses this issue, i.e., recovering from link failures and updating the measurement. There are two types of failure recovery approaches: *protection* and *restoration* [10] while using SDN and OpenFlow. In restoration scheme, a switch contacts the controller upon detecting a link failure to get an alternative route setup. In the protection scheme, backup routes are configured before a failure occurs, where switches can locally detect a failure and redirect the affected traffic to an alternative route without communicating the controller to reduce failure recovery time.

We first integrate Fast Failure Group (FFG) [11] of the OpenFlow protocol in the ReMon framework, so that the data plane elements can locally recover from link failure. FFG requires an alternative route from the affected switch to a destination. In the absence of such an alternative route, we can use Crankback approach [12]. In Crankback approach,

978-3-903176-17-1 / © 2019 IFIP

affected packets backtrack on a primary route up to a switch that offers an alternative route. Our initial investigation reveals that Crankback increases the failure recovery delay. Thus, we propose and integrate a couple of algorithms, *Anchor Assisted Recovery (AAR)*, and *Weight Assisted Recovery (WAR)*, in the ReMon framework to offer measurement resiliency in the presence of link failure. Finally, we update the measurement algorithm, WAS, to accommodate the change in the routes of flows due to the failure recovery.

We use real network topologies USNET [13] and Darkstrand [14] to evaluate ReMon in Mininet [15] emulator. In USNET, each source has one or more alternative routes to a destination; thus, FFG can be used at the data plane. On the other hand, Darkstrand does not have such alternative path properties; thus, we need to use Crankback. In Mininet, we use Ryu controller and Open Virtual Switch (OVS) [16]. The evaluation results on the above two topologies reveal that ReMon offers accurate measurement while reducing around 30% measurement cost compared to a baseline measurement approach, where we need to measure all switches. It furthermore improves the link failure recovery time 50% compared to its counterparts.

The remaining of the paper is organized as follows. Section II presents the necessary background to understand the proposed work. In Section III, we compare and contrast work related to ReMon. We present the design of ReMon and proposed algorithms in Section IV. The experimental setup is explained in Section V. Section VI presents and discusses the evaluation results following a concluding remark in Section VII.

II. Background

In this section, we present the necessary background to understand the design and operation of ReMon. A naive monitoring approach in SDN measure each flow from every switch, which generates a large number of control packets and increases the overhead with the increasing number of flows and switches. The number of control packet will reach $\mathcal{O}(nm)$, where n is the number of switches and m is the number of flow entries at a switch. Thus, we propose to aggregate flow statistics at the data plane elements to reduce the control overhead without degrading the measurement accuracy.

Once we decide on the set of switches, we use sFlow [9] to monitor the chosen switches in fine-granularity. sFlow consists of sFlow-controller and sFlow-agents to offer both the active and passive measurements. The sFlow-controller actively polls the selected set of switches at a regular interval, whereas sFlow agents passively get back to the sFlow-controller after observing a certain configurable number of packets. We argue that these days switches come with sFlow support, where using an appropriate fine-granular sampling and polling rate can control the overall control traffic.

The main objectives of ReMon are to offer monitoring resiliency, e.g., in the presence of link failure. In SDN architecture, the OpenFlow protocol supports Fast-Failover Group (FFG) [6] to implement the failure recovery at the data plane. In particular, in addition to flow-tables, a switch maintains a group table with active buckets. Each bucket is associated with a port from a route, and only a single bucket is active at a time. The incoming packet flows through a port from an active bucket. In the case of a link failure, the next active port and bucket are chosen to redirect the affected traffic.

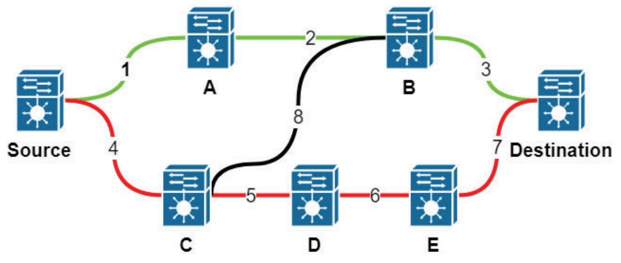


Fig. 1. Example Topology.

Nonetheless, if there is no such backup route, we need alternative redirecting approach. For example, in Figure 1, if the link between switch *Source* and *A* fails, *Source* can locally redirect the affected traffic to link four. However, if the link between switch *E* and *Destination* fails, *E* cannot use FFG due to the unavailable backup routes, but rather needs to use Crankback approach.

Crankback is a restoration approach that reacts to failure after it occurs. The affected packet traverses backward towards the source until finds a switch having a backup route towards a destination. The packet carries a unique tag to inform switches along with the backward route about the failure. All the subsequent packets for the same source-destination pair follow the alternative route. Thus, the affected packet from switch *E* in the previous example will backtrack to switch *C* and follow link eight towards the destination. However, Crankback introduces additional delay and communication cost due to packet-by-packet backtracking. Thus, we propose two algorithms Anchor Assisted Recovery (AAR) and Weight Assisted Recovery (WAR) that are used in ReMon to reduce the delay.

III. Related Work

The network measurement work in SDN can be classified as active and passive schemes. In this section, we compare and contrast the existing solutions that are related to ReMon. OpenTM [4] polls OpenFlow switches at a regular interval to measure traffic matrix from each source-destination pair. The author proposed different polling algorithms to reduce monitoring overhead. OpenTM is an offline approach; thus, OpenNetMon [17] proposed an online flow monitoring approach for throughput, packet loss, and delay. However, both of these solutions monitor all flows over all source-destination pairs, which may not scale with the increasing number of flows and switches.

RFlow [18] is an active measurement scheme for WLAN, which deploys a set of collectors managed by the controller to gather the flow statistics. The hierarchical structure of RFlow may reduce the bottleneck at the controller, but the solution needs to access every switch. Payless [19] reduces the measurement cost by dynamically polling switches while a flow expires within a collection timeout. In contrast, ReMon can dynamically change monitoring switches according to the network state. This subset of switches can adaptively be polled as in Payless. SDN-Mon [20] is a monitoring framework similar to sFlow, which monitors a single flow from a switch. Thus, SDN-Mon needs to check every new flow entry to decide on the monitoring flow, which is not the case in ReMon.

FlowCover [7] reduces measurement traffic by accessing only a subset of switches. It assigns a weight to each switch proportional to the number of flows passing through it. The switches are then monitored according to their weight until all flow across the network is covered. CeMon [21] is a multi-controller variant of FlowCover, i.e., the measurement task is distributed among a set of controllers to improve the measurement cost and scalability. Partial Flow Statistics Collection (PFSC) [22] also collects flow statistics from a subset of switches such that the flow recall ratio on every switch reaches a predefined value while minimizing the number of queried switches.

None of the above measurement schemes considers link failures while performing the measurement. ReMon performs extensive experiments in real network topologies to evaluate the measurement performance in the presence of link failures. Also, ReMon proposes a new flow aggregation scheme and two link failure recovery algorithms to offer resilient measurement and monitoring.

IV. Design

In this section, we first provide an architectural overview of ReMon, which is presented in Figure 2. The software enabled switches are configured to initiate OpenFlow *ofp_packet_in* and *ofp_flow_removed* messages. In the ReMon controller, we have a statistics gathering module to collect these control packets to learn the current network state. We use *Weight Assisted Selecting (WAS)* algorithm to determine a set of switches to be queried to retrieve the aggregated flow statistics. Then, we deploy sFlow to query those chosen switches. Note that sFlow has two components; namely, sFlow-controller and sFlow-agent. The former one is deployed in the ReMon controller, and the latter one is in the switches. A switch after receiving a sFlow query generates corresponding statistics for the sFlow controller. We will first describe our flow aggregation algorithm *WAS*, which is implemented in the controller.

Weight Assisted Selecting (WAS): Suppose we have a network $G = (V, E)$, where $V = \{v_1, v_2, ..., v_n\}$ is the set of switches and E represents the set of links. Therefore, $n = |V|$ is the total number of switches in this network each of which carries m flows. We set a dictionary S which contains the shortest path for each source-destination pair. This process can be configured during the network setup period. At the same time controller can maintain this route information. The WAS algorithm is presented in Algorithm 1.

In WAS, we first sort all flows according to their end-points. We start with the largest group of flows sharing the same source-destination pair. Usually, it is enough to monitor a single switch along the path (shortest) of a flow. Thus, it is possible to consider only the destination to check all flows that share that end-point. Then, we move to the next group of flows to pick a monitoring switch. We continue until all flows are covered. If the corresponding monitoring switch is already chosen for a group of flows, we move forward to the next group. For example, in Figure 1, suppose we have six flows installed in the order: $Source - A - B - Destination$, $Source - C - D - E - Destination$, $A - B$, $C - D$, $Source - C - D - E$, and $Source - A - B - C$.

In FlowCover, the sorted list of switches is $\{Source, C, A, B, D, E, Destination\}$ according to the number of flows passing through a switch. FlowCover first picks switch $Source$ with the highest number of flows and keeps choosing the sorted switches until all flows are covered. As a result, the final set of switches in FlowCover includes $\{Source, C, A\}$. On the other hand, WAS does not assign any weight to switches, but rather to the incoming flows. It picks the destination switch if a newly installed flow is

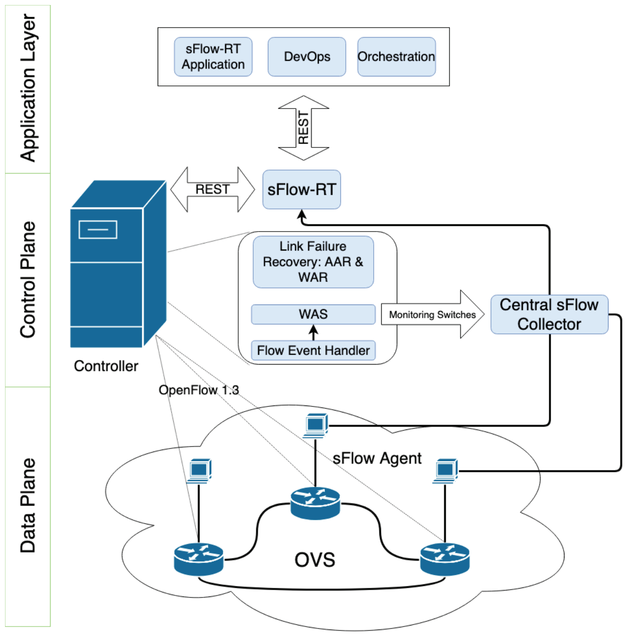


Fig. 2. ReMon Architecture.

Algorithm 1 Weight Assisted Selecting (WAS)

```
Input:
    Aggregated Flows: $S = \{(src, dst) : (src, ..., dst), ...\}$
Output:
    Monitored Switches: $P$
1: $P = []$
2: for each $(pair, path) \in S$ do
3:     $(src, dst) = pair$
4:     if $path \cap P = null$ then
5:         $P$.append($dst$)
6:     end if
7: end for
```

not monitored by any other switches. Thus, WAS first picks $Destination$ for the flow $Source - A - B - Destination$, which also covers the flow $Source{-}C{-}D{-}E{-}Destination$. Then, it selects B and D for the next two flows, which furthermore cover the last two flows. The set of monitored switches for WAS is $\{Destination, B, D\}$.

The main difference between WAS and FlowCover is that the former considers the number of flows through a source-destination pair as a weight to choose a switch; whereas the latter starts with a switch carrying the highest number of flows. Thus, the computation complexity of WAS and FlowCover is $\mathcal{O}(nm)$ and $\mathcal{O}(n^2m)$, respectively.

Next, we present two algorithms AAR and WAR, whose purpose is two-fold; 1) recovering from link failures in the cases where a switch does not have an alternative route towards a destination and 2) updating WAS, i.e., the list of monitoring switches based on the updated routes of the affected flows.

Anchor Assisted Recovery (AAR): In AAR (Algorithm 2), an *anchor node* is a switch that has at least one alternative routes towards the destination for a given route. For example, in Figure 1, the anchor node is C along the route $Source - C - D - E - Destination$. The anchor nodes can be configured during the FFG creation. At the same time controller can maintain a list of anchor nodes from current topology.

Algorithm 2 Anchor Assisted Recovery (AAR)

```
Input:
    Failure path: $P = (src, ..., s1, s2, ..., dst)$
    Anchor list of $(src, dst)$: $anchor\_list$
Output:
    Alternative path: $P$
    Affected switch: $switch$
1: for each $switch \in P[src : s1]$ do
2:     if $switch \in anchor\_list$ then
3:         if There is a new path $path_{new}$ to $dst$ then
4:             $path_{old} \leftarrow P[src : switch]$
5:             $P \leftarrow path_{old} + path_{new}$
6:             return $P, switch$
7:         end if
8:     end if
9: end for
```

In AAR algorithm, a switch will inform the controller if it cannot recover from a failure using FFG upon detecting link failure. In that case, the controller will recompute all alternative routes in advance irrespective of current active one that is using the failed link. For example, if link seven between *E* and *Destination* fails, the affected packets will take an alternative route from the anchor node C. Thus, the new route will be $Source - C - B - Destination$ instead of $Source{-}C{-}D{-}E{-}D{-}C{-}B{-}Destination$ in the case of Crankback. Thus, in contrast to Crankback, AAR reduces the recovery time without recomputing alternative routes packet-by-packet. Furthermore, using AAR a controller can insert new flow entries right after receiving a link failure event instead of waiting for the affected packets to be backtracked to an anchor node. The ongoing packets between the anchor and failed link can use Crankback.

The computation complexity of AAR is $\mathcal{O}(n)$ for one route. Since a link failure may cause more than one path failure, the total complexity is $\mathcal{O}(n^3)$ because in the worst case, there can be $\mathcal{O}(n^2)$ source-destination pairs. The computation complexity of AAR, WAR, and Crankback is the same, except that the former two reduces the recovery time by exploiting the global network view and proactive alternative path computation.

Algorithm 3 Weight Assisted Recovery (WAR)

```
Input:
    Failure path: $P = (src, ..., s1, s2, ..., dst)$
    Weight list: $Weight = (w_1, w_2, ..., w_n)$
Output:
    Alternative path: $P$
    Affected switch:$switch$
1: Update $Weight$
2: Label all $switch \in P$ as $unchecked$
3: while $\exists$ $unchecked$ $switch \in P[src : s1]$ do
4:     find a $switch$ such that $Weight[switch]$ is maximum
       and $unchecked$
5:     if There is a new path $path_{new}$ to $dst$ then
6:         $path_{old} \leftarrow P[src : switch]$
7:         $P \leftarrow path_{old} + path_{new}$
8:         return $P, switch$
9:     end if
10:    Label $switch$ as $checked$
11: end while
```

Weight Assisted Recovery (WAR): In WAR (Algorithm 3), we assign a weight to a switch proportional to its degree (number of neighbors). For example, in Figure 1, switch C and B have weight three, whereas others have weight two. Thus, in the case of a link failure, say link seven again, the affected packet will take the alternative route from C. Thus, AAR and WAR differ in terms of their switch selection approach.

The controller can assign weight during the network setup phase. Thus, after detecting a link failure, the controller can choose a set of switches with the highest weight to redirect the affected traffic through these chosen switches. In AAR and WAR, the idea is to take the help of the controller to find alternative routes for all possible flows affected by a single link failure instead of recovering from a failure in packet-by-packet fashion.

Updating WAS over link failure: We compute new alternative routes using AAR or WAR upon detecting link failure. This route change may need a new switch to monitor. Some switches, on the other hand, may no longer need to be monitored because of removing the failed route. As a consequence, a link failure impacts the monitoring lists of WAS. Thus, we define Algorithm 4 to update WAS after detecting link failure.

Algorithm 4 Updating WAS

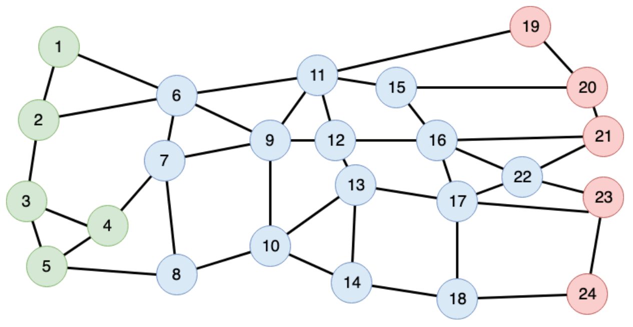
```
Input:
    Failure link: $Link_{fail} = (fw_1, fw_2)$
    Monitoring List: $M = (s_1, s_2, ..., s_n)$
Output:
    Updating Monitoring List: $M$
1:  while $\exists$ $path_{fail}$ contains $Link_{fail}$ do
2:      $path_{new}$ = AAR/WAR($path_{fail}$)
3:      if $\exists$ $switch \in path_{new}$ is monitored then
4:          pass
5:      else
6:          $M$.append($path_{new}[destination]$)
7:      end if
8:  end while
9:  if $\exists$ $flow \in fw_1$ or $fw_2$ and monitored only by $fw_1$ or
    $fw_2$ then
10:     pass
11: else
12:     remove $fw_1$ or $fw_2$ from $M$
13: end if
```

Let us consider Figure 1 and the six flows we used to illustrate the operation of WAS. Suppose link five between C and D fails, which will impact three flows: $Source - C - D - E - Destination$, $Source - C - D - E$, and $C - D$. We first compute new routes for the affected flows using AAR or WAR. The new routes will be $Source - A - B - Destination$, $Source - C - B - Destination - E$, and $C - B - Destination - D$. Thus, we need to update the list of switches to be monitored. Note that all three flows are covered by the switch $Destination$. Therefore, the current monitoring list, $\{Destination, B, D\}$, remains unchanged. However, we furthermore need to check the endpoints of the failed link. If an endpoint is in the current monitoring list, we need to check whether we need to update it. For instance, D is in the list, which is not used by any flows; thus, can be removed from the existing monitoring list.

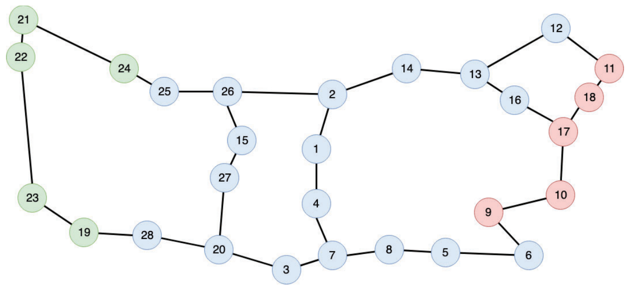
V. Experimental Setup

We first evaluate the performance of WAS and compare the outcome with FlowCover in real topologies. Next, we evaluate the performance of ReMon in the presence of data plane link failures. In particular, we first test the performance of ReMon with and without link failures in two real topologies to learn their impact. Then, we show how the proposed algorithms AAR and WAR can improve the recovery time and delay. Recovery time is the time between a link failure detection to rerouting the affected packets to an alternative route. We furthermore compare the performance of AAR and WAR with Crankback and the restoration approaches.

We deploy the Mininet 2.2.2 emulator in an Oracle VirtualBox (VM) having an Intel Core i5 2.90 GHz (4 cores) CPU processor and 4GB RAM. The server runs Ubuntu (64-bit) operating system. We use a Ryu 4.30 controller to manage a set of Open vSwitch 2.11.90. The data and control planes use OpenFlow 1.3 protocol to communicate.



(a) USNET



(b) Darkstrand

Fig. 3. The USNET and Darkstrand Topologies.

We use two real topologies; namely, USNET and Darkstrand. The USNET topology consists of 24 switches and 42 links as shown in Figure 3(a) [13]. The second topology (Darkstrand) is shown in Figure 3(b) [14], which consists of 28 switches and 31 links. Each switch from both topologies supports a connected host. If we consider that each source-destination pair has at least one flow, then USNET and Darkstrand will have 276 and 325 possible pairs, respectively. USNET offers at least a pair of routes between any source-destination pairs, which is not the case in Darkstrand. Thus, we can evaluate the performance of ReMon in topologies with different properties.

We first evaluate the performance of the proposed switch selection algorithm, WAS, in terms of monitoring accuracy. We randomly choose a set of five source-destination pairs, where sources use iperf to generate UDP traffic at a rate of 1 Mbps. We double this rate at every ten seconds and monitor the link utilization. We set the sFlow sampling rate at switches as one every 200 packets and switch polling interval at the controller as 20 seconds.

Then, we compare the performance of ReMon with FlowCover in terms of the measurement cost and computation cost. The measurement cost for monitoring a single switch is expected to be consistent as the sampling and polling rates are fixed. Thus, the number of monitoring switches can represent the measurement cost. We consider the USNET topology in the case of accuracy and cost evaluations. We measure the computation time as the computation cost.

Next, we evaluate the resiliency of ReMon both in USNET and Darkstrand in the presence of data plane link failure. We use iperf to generate UDP traffic at a rate of 1Mbps between ten source-destination pairs. In Fig. 3, the green and red nodes are sources and destinations, respectively. We randomly choose one from each set as a source-destination pair and repeat the process ten times. We randomly fail five links from both the primary and backup routes between each chosen source-destination pair to get the average recovery time and end-to-end delay, where we use timestamps and ”ping”. In the final set of evaluation, we implement our proposed algorithms AAR and WAR, which are then compared against Crankback and restoration based link failure recovery schemes. In addition, we measure the number of computation that Crankback, AAR, and WAR need to reconstruct new routes after facing a link failure.

VI. Discussion on the Evaluation Results

In this section, we present the evaluation results of ReMon and related works. At first, we compare and contrast ReMon with FlowCover and baseline (monitor all switches) measurement approaches in real topologies. Then, we show the performance of ReMon in the presence of link failures. Finally, we show that the proposed algorithms AAR and WAR can significantly improve the failure recovery time and end-to-end delay.

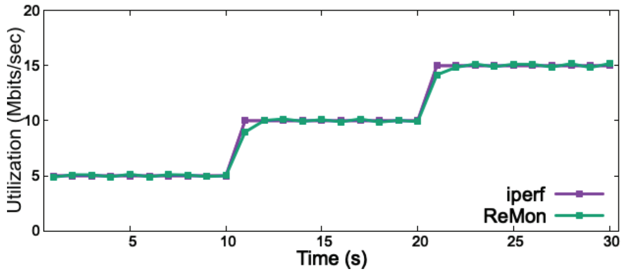


Fig. 4. The measurement accuracy of ReMon.

The measurement performance of ReMon: Figure 4 shows the network utilization over time in USNET topology. We generate a total of five Mbps UDP traffic in the first ten seconds, then double the traffic volume. The results show that ReMon can measure link utilization accurately. sFlow collects some additional control traffic including ARP and LLDP packets, which we record at the controller at every second and subtract it from actual measurement traffic. This computation will not create any overhead on our measurement framework. However, we observe slight discrepancy of sFlow at measurements points where utilization changes.

In Figure 5 and 6, we present the measurement cost of ReMon in terms of the number of monitoring switches and the computation cost in time, respectively. We consider both the USNET and Darkstrand topologies. ReMon needs to monitor one additional switch compared to FlowCover, where both the approaches reduce around 30% communication cost compared to the baseline approach. However, the computation cost of FlowCover is significantly higher than that of ReMon

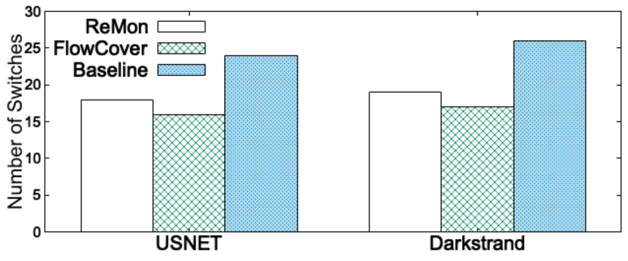


Fig. 5. The measurement cost of different schemes.

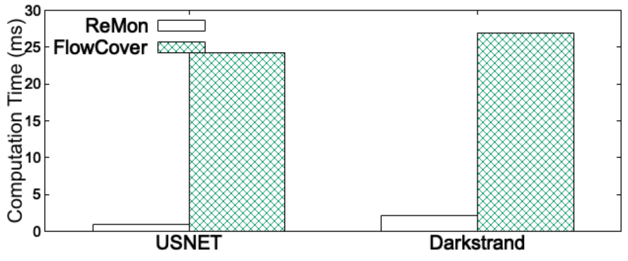


Fig. 6. The average computation cost of ReMon and FlowCover.

(Figure 6). It reaches over 25 ms in both topologies while ReMon only spends 1 to 2 ms. However, 25 ms computation time is still acceptable. Thus, we evaluated the computation time in another topology called, DFN, with 58 switches and 87 links, where ReMon takes 37 ms while FlowCover takes over 1100 ms. Thus, ReMon reduces the similar amount of measurement cost with significantly less computation cost compared to FlowCover. Both ReMon and FlowCover perform better compared to the baseline approach, where we need to monitor all switches from the data plane.

The impact of link failure:

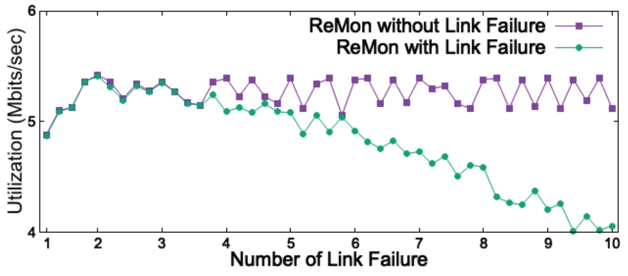


Fig. 7. The link failure impact on measurement accuracy in USNET.

First, we compare the measurement accuracy of ReMon with and without link failures, which is shown in Figure 7. The results indicate that with a small number of link failures, we can still have good accuracy as the traffic may always find alternative routes to reach a destination within a reasonable time. However, the accuracy degrades with the increasing number of link failures. Thus, we integrate FFG and Crankback in ReMon to recover from link failures and update the monitoring list.

The number of monitoring switches changes with link failures and corresponding pattern depends on the structure

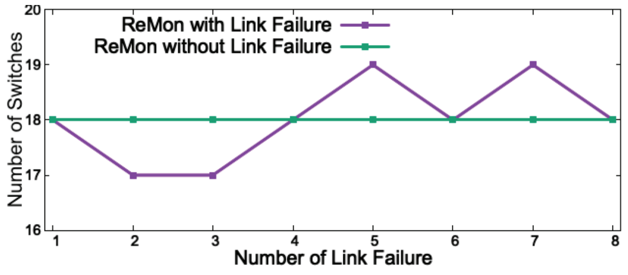


Fig. 8. The link failure impact on measurement cost in USNET.

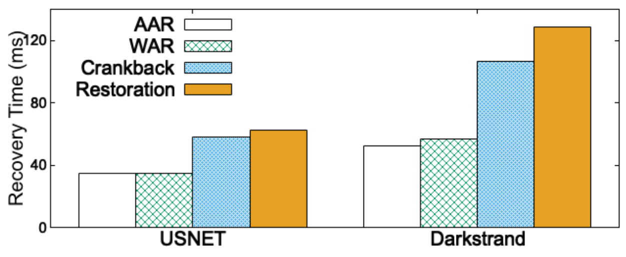


Fig. 10. The total link failure recovery time in USNET and Darkstrand.

of network topology. Figure 8 shows the change in the total number of monitoring switches in USNET in the presence of link failures while using ReMon. We first randomly fail a set of links from around the perimeter of the USNET topology. Thus, the affected flows are likely redirected towards the center, and multiple flows shared common routes. As a consequence, we see a decrease in the number of monitoring switches as the affected traffic finds an alternative route through a smaller number of switches that aggregates the measurement traffic. Then, we start failing link also from the center of the topology, which leads to an increase in the number of monitoring switches as flows are spreading out.

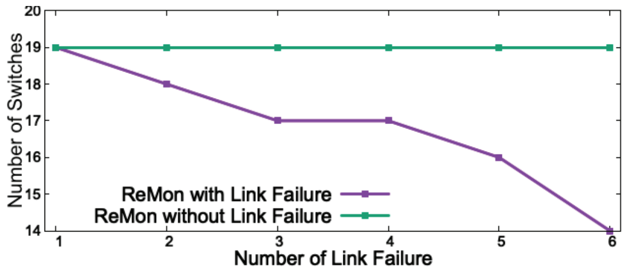


Fig. 9. The link failure impact on measurement cost in Darkstrand.

Next, we evaluate the performance of ReMon in Darkstrand, which is shown in Figure 9. We fail less number of links in Darkstrand compared to the USNET topology to maintain network connectivity while performing the measurement. The number of monitoring switches decreases with the increasing number of link failures. The reason is, again, the structure of the Darkstrand topology, which has a less number of links compared to the number of switches. Thus, the flows are more likely to pass through a small number of switches, especially with an increasing number of link failures.

Recall that in USNET topology switches have alternative routes to any destination; thus, we can use FFG in the case of a link failure. However, in Darkstrand due to the lack of alternative routes packets follow Crankback approach, which may lead to using a small set of measurement switches, especially with increasing link failures. Therefore, we conclude that the number of switches to be monitored will depend on the structure of a topology.

The performance analysis of AAR and WAR: We first focus on measuring the total link failure recovery time of four methods: AAR, WAR, restoration, and Crankback both in USNET and Darkstrand, which is shown in Figure 10. A single link failure may impact multiple routes; thus, we define the total link failure recovery time as the time to update all affected routes. In USNET, the recovery time of AAR and WAR is around 34 milliseconds. The restoration approach takes about 62 milliseconds, and the Crankback takes the longest time. We observe similar performance trend in Darkstrand topology. Thus, AAR and WAR reduce more than 50% and 40% of the recovery time compared to the restoration and Crankback approaches, respectively.

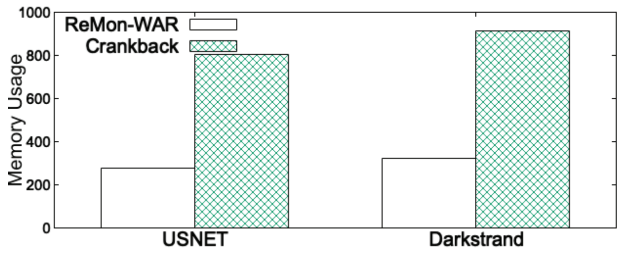


Fig. 11. The total number of flow entries used in USNET and Darkstrand.

We present the memory usage of ReMon and Crankback approach in Figure 11. We measure the memory usage as the total number of flow entries used at the switches in a given topology. Crankback needs to install flow entries for primary and backup routes along with the backtracking rule, which is not the case in ReMon. Thus, the memory usage of Crankback is significantly higher than that of ReMon.

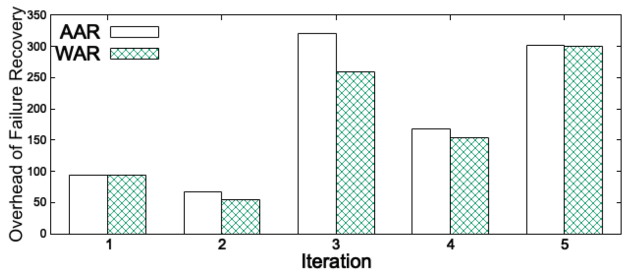
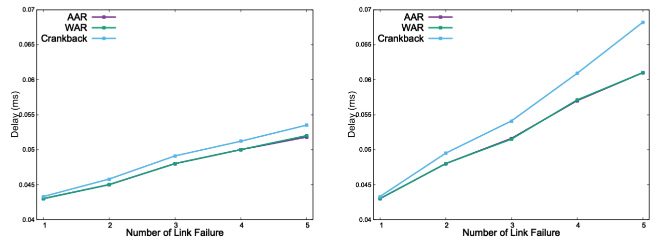


Fig. 12. The performance comparison of AAR and WAR in Darkstrand.

Next, we compare the performance of AAR and WAR. We use the number of computation as the failure recovery overhead, which is the total number of switches checked in each algorithm. The result is shown in Figure 12, where we show the results for Darkstrand topology as the performance is similar in USNET. The number of computation for WAR is significantly lower than AAR in Darkstrand. The reason is the structure of the topology. WAR only checks switches with higher weights, i.e., those switches more likely have alternative routes towards a destination. However, an anchor node in AAR is a switch having more than two neighbors. Thus, AAR will need more time to find the set of anchor nodes. Therefore, we conclude that in the case of a moderate size topology with a large number of links AAR and WAR will perform similarly. However, if the topology is a less-links and more-switches one, like Darkstrand, WAR will be useful.

The network performance enhancement using AAR and WAR: AAR and WAR not only improve the resiliency of ReMon but also improves the average network throughput and delay.



(a) USNET (b) Darkstrand

Fig. 13. The average delay of different failure recovery schemes.

Figure 13 presents the delay that ReMon experiences with AAR, WAR, and Crankback in USNET and Darkstrand topologies. Crankback has the highest delay in both topologies because it needs to backtrack to a switch along the primary route with an alternative route to a destination. All the affected switches will use this packet-by-packet backtracking, which is not the case in AAR and WAR. The latter approaches rather proactively reconstruct all possible affected routes for a given link failure. We observe similar results in the case of throughput, which is not included due to space constraints.

VII. Conclusions

In this paper, we have proposed a resilient monitoring framework, called ReMon. It reduced the measurement cost while offered high accuracy compared to its counterparts. We have evaluated the measurement performance of ReMon in real network topologies, where it reduced 30% measurement cost compared to the baseline approach, and significant computation cost compared to FlowCover. We furthermore extended ReMon to operate in the presence of link failure. The performance evaluation under link failures revealed that the monitoring cost is correlated to the structure of the underlying topology. Thus, we proposed two new link failure recovery algorithms for topologies that cannot use FFG due to the lack of available alternative routes. The evaluation results conceded that ReMon reduced 50% of link failure recovery time compared to its counterparts. It furthermore improves the overall network throughput and delay. As part of our future work, we plan to investigate the impact of other types of failures like node, sFlow agent, and controller.

References

[1] P.-W. Tsai, C.-W. Tsai, C.-W. Hsu, and C.-S. Yang, "Network monitoring in software-defined networking: A review," *IEEE Systems Journal*, vol. 12, no. 4, pp. 3958–3969, Dec 2018.

[2] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 1–12, Aug. 2007.

[3] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Gude, N. McKeown, and S. Shenker, "Rethinking enterprise network control," *IEEE/ACM Transactions on Networking (TON)*, vol. 17, no. 4, pp. 1270–1283, August 2009.

[4] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: traffic matrix estimator for openflow networks," in *the 11th international conference on Passive and active measurement*, April 2010, pp. 201–210.

[5] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "FlowSense: Monitoring network utilization with zero measurement cost," in *the 14th international conference on Passive and Active Measurement*, March 2013, pp. 31–41.

[6] Open Networking Foundation, "Openflow switch specification," September 2012.

[7] Z. Su, T. Wang, Y. Xia, and M. Hamdi, "FlowCover: Low-cost flow monitoring scheme in software defined networks," in *2014 IEEE Global Communications Conference*. IEEE, December 2014.

[8] R. Govindan, I. Minei, M. Kallahalla, B. Koley, and A. Vahdat, "Evolve or Die: High-availability design principles drawn from googles network infrastructure," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16, 2016, pp. 58–72.

[9] "sFlow-RT," https://sflow-rt.com/index.php.

[10] P. C. da Rocha Fonseca and E. S. Mota, "A survey on fault management in software-defined networks," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2284–2321, June 2017.

[11] N. L. van Adrichem, B. J. van Asten, and F. A. Kuipers, "Fast recovery in software-defined networks," in *2014 Third European Workshop on Software Defined Networks*. IEEE, September 2014.

[12] N. L. M. van Adrichem, B. J. van Asten, and F. A. Kuipers, "Fast recovery in software-defined networks," in *EWSDN '14 Proceedings of the 2014 Third European Workshop on Software Defined Networks*, September 2014.

[13] "USNET topology," https://www.researchgate.net/figure/Network-topologies-aNSFNET-bUSNET_fig3_321952636.

[14] "Darkstrand Topology," http://www.topology-zoo.org/maps/Darkstrand.jpg.

[15] "Mininet," http://mininet.org/.

[16] "Open vSwitch," https://www.openvswitch.org/.

[17] N. L. M. 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, May 2014.

[18] R. Jang, D. Cho, Y. Noh, and D. Nyang, "RFlow: An sdn-based wlan monitoring and management framework," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*. IEEE, May 2017.

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

[20] X. T. Phan and K. Fukuda, "Toward a flexible and scalable monitoring framework in software-defined networks," in *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. IEEE, March 2017.

[21] Z. Su, T. Wang, Y. Xia, and M. Hamdi, "CeMon: A cost-effective flow monitoring system in software defined networks," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 92, no. 1, pp. 101–115, December 2015.

[22] H. Xu, X.-Y. Li, L. Huang, Y. Du, and Z. Liu, "Partial flow statistics collection for load-balanced routing in software defined networks," *Computer Networks*, vol. 122, pp. 43–55, July 2017.