Reliable Network Performance for Edge Networks with QoS-Aware Adaptive Routing

Osama Abu Hamdan
Department of Computer Science and Engineering
University of Texas at Arlington
oma8085@mavs.uta.edu

Engin Arslan
Department of Computer Science and Engineering
University of Texas at Arlington
engin.arslan@uta.edu

***Abstract*—Edge networks are commonly used to collect data from remote areas with limited accessibility such as deserts and mountaintops. They rely on wireless communication to stream the collected data to data centers, which makes them susceptible to bandwidth fluctuations due to extreme weather conditions such as snowstorms. Although these networks are designed to have alternate routes to recover from link failures, existing rigid network management solutions (i.e., OSPF) make it difficult to utilize alternate routes when the bandwidth of wireless links degrades. This paper proposes an adaptive routing algorithm, *RENET*, to quickly detect bandwidth fluctuations and re-adjust flow paths to increase the utilization of available links. *RENET* introduces a novel delay-based measurement technique to quickly detect bandwidth fluctuations in wireless links. Experimental results collected in real-world and emulated networks show that *RENET* offers a robust solution to bandwidth fluctuations and outperforms the state-of-the-art solutions by more than 30% in terms of QoS satisfaction rate. Moreover, *RENET* increases the quality of adaptive video streams by up to 54%.**

***Index Terms*—Adaptive routing, wireless link bandwidth changes, Software Defined Networking, traffic engineering.**

I. Introduction

The adoption of the Internet of Things (IoT) phenomenon across many science and engineering fields has led to the wide-area deployment of sensor devices such as public safety cameras, LiDAR sensors, and advanced atmospheric monitoring devices. In these networks, edge devices communicate (via wireless or wired ethernet) with a set of middle aggregation points, which are in turn connected by fiber or high-speed microwave backhaul links to form an interconnected infrastructure leading to private data centers or cloud services as illustrated in Fig. 1. Since edge networks heavily rely on long-distance wireless links both at the edge and aggregation sites, the resulting end-to-end network topology is highly variable in terms of availability and capacity over space and time due to conditions outside engineering control such as weather, radio interference, or physical disturbance.

Current network management practices for edge networks rely on static routing technologies, such as Open Shortest Path First (OSPF) and Multiprotocol Label Switching (MPLS), that are manually configured by experienced network administrators across the physical and virtual WANs. Although traditional distributed routing algorithms work well for uncongested networks (30% or less utilization), they require highly skilled network managers to frequently tune link weights

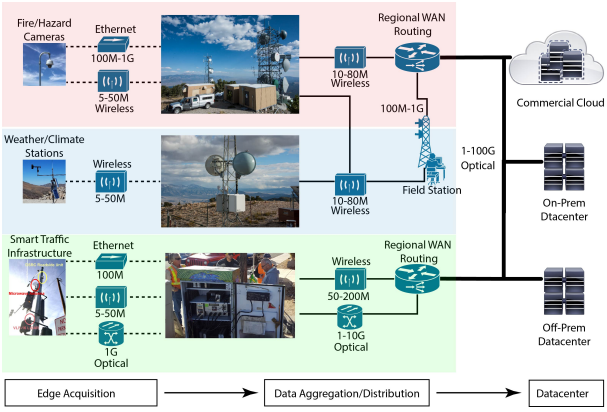


Fig. 1: An edge network infrastructure used by the University of Nevada, Reno researchers to stream data from remote sensors to data centers.

when link capacity, network demand, and QoS metrics are exceedingly dynamic. While researchers proposed Software Defined Network (SDN) based traffic engineering solutions to optimize network routing to mitigate congestion in dynamic environments [1]–[8], they assume the bandwidth of links to be static. However, edge networks frequently face bandwidth fluctuations in wireless links due to environmental conditions such as icing on the antenna, necessitating an effective way of measuring link capacities in real time to adapt to changing conditions.

In this paper, we propose an SDN-based adaptive routing solution for edge networks, *RENET*, to minimize the impact of link capacity changes on the performance of applications. *RENET* introduces a lightweight probing approach to detect bandwidth fluctuations. The probing method conducts delay measurements for each link to identify congested links. It then reroutes flows to alternate paths to better utilize available network resources and minimize congestion. Moreover, *RENET* allows (does not mandate) applications to specify their bandwidth requirements such that it will assign them to paths that can meet their demand. Experimental results collected in both real-world and emulated networks show that *RENET* enhances the quality of service for bandwidth-sensitive flows by more than 30% compared to the state-of-the-art solutions. Moreover, it improves the quality of adaptive video streams by up to 54%. In summary, this work makes the

following contributions:

- We propose a delay-based congestion detection method to identify link capacity drops in wireless channels under extreme weather conditions. This lightweight probing solution can quickly detect congested links without requiring access to port statistics from wireless devices.
- We introduce a heuristic model to find optimal routes for new and existing flows. The model allows *RENET* to reserve sufficient capacity for future flows while satisfying the Quality of Service (QoS) metrics of current flows with specific bandwidth demands.
- We conduct experiments using both physical and virtual (i.e., Mininet) networks and show that *RENET* improves application QoS by more than 30% compared to the state-of-the-art solutions. Moreover, *RENET* improves the quality of adaptive video streams by up to 54%.

II. Related Work

Previous studies showed that SDN improves Quality of Service [1]–[4] and facilitates network monitoring [9]–[11].

Network Monitoring and Measurement: Researchers developed many network-monitoring techniques for SDN using OpenFlow. For example, OpenTM [12] proposes a heuristic solution to choose a set of switches to monitor in an SDN network to accurately capture the traffic matrix. PayLess [13] takes advantage of OpenTM to identify and monitor the most important switches and adapt the probing frequency. FlowSense [14] presents a passive push-based monitoring technique to estimate per-flow link consumption. Although FlowSense's communication overhead is minimal, it only performs well when the network has many short flows.

OpenNetMon [15] uses an adjustable polling rate to poll data from switches, which minimizes its resource overhead. LLDP-looping [16] uses agents installed on switches and controller modules to monitor the latency of links. The agents initially require each probe packet to circle a connection three times before calculating their RTT to track delay. IPro [8] uses reinforcement learning to minimize probing intervals, leading to precise monitoring with low control-channel overhead.

Quality of Service: Several well-known methods have been suggested to enhance QoS in legacy networks, including integrated services (IntServ) [17], differentiated services (DiffServ) [18], and multi-protocol label switching (MPLS) [19]. However, these decentralized solutions cannot completely capture the network conditions; hence, they fail to offer optimal solutions in all scenarios.

On the other hand, several SDN-based QoS enhancement mechanisms have been proposed in the literature. Egilmez et al. proposed a multimedia controller to offer dynamic end-to-end routing [5]. The controller designates a path for multimedia streaming data flow to enhance the delivered video quality. In another work, HiQoS divides user traffic into three categories video stream, interactive audio/video, and best-effort [6]. Then it uses a modified version of the Dijkstra algorithm to choose the routes with a maximum available bandwidth for new flows. HiQoS preemptively addresses link failures by computing multiple backup paths for each flow. The routing engine in [7] calculates the least-loaded shortest paths between end host pairs using statistics it collects from OpenFlow switches. Upon congestion, the engine re-routes large flows to alternate paths, ensuring no congestion on those paths.

DICES [20] is a dynamic, adaptive Search-Based Software Engineering (SBSE) approach for IoT environments. It monitors the network in real time to detect congestion which is then alleviated by changing the route of some flows. It adopts multi-objective search algorithm Non-dominated Sorting Genetic Algorithm II (NSGA-II) to find routes that optimizes network utilization, minimizes transmission delay, and require minimal reconfiguration.

BACAR [21] aims to dynamically adjust network routes based on real-time bandwidth and congestion measurements. It utilizes lightweight active delay measurements to detect link congestion and real-time bandwidth fluctuations. Upon detecting potential congestion due to high traffic rates or a bandwidth drop, BACAR uses delay measurements to identify links that experience capacity drop. If bandwidth drops leads to congestion, BACAR moves large flows to alternate paths to mitigate the congestion.

Different from existing work, *RENET* is designed to capture transient bandwidth drops that can affect wireless connections due to extreme weather conditions. *RENET* uses a novel delay-based mechanism to identify and detect the capacity of wireless links experiencing bandwidth fluctuations. It then recalculates optimal routes for flows on congested links to divert them to alternate paths, thereby maximizing network utilization and increasing application QoS.

III. Motivation

The available network capacity of radio channels can be affected by environmental conditions. In particular, winter conditions such as heavy snowfall can lower the effective transmission rate of long-distance directional radio links as they can increase the signal-to-noise ratio by disturbing antenna orientation, damaging the antenna dish, and causing moisture in the coaxial cable. In addition to extreme weather conditions, interference is also a common problem when using unlicensed frequencies, which can adversely affect the transmission rate of wireless links. Consequently, the bandwidth of wireless links exhibits a highly fluctuating behavior in production wide-area edge networks.

As an example, Fig. 2 shows the capacity of one of the wireless links used in The Nevada Climate-ecohydrological Assessment Network (NevCan) and Alert-Wildfire projects [22], [23] over six weeks. Each data point marks the average link capacity for 30 minutes. The figure shows that path capacity exhibits drastic changes as snowstorms cause icing and disorientation on the antenna. While the capacity is around 40 Mbps during the first week, it reaches 80 Mbps in the second week. However, capacity fluctuations continue to happen consistently in the following weeks. While the capacity remains mostly between $60 - 80$ Mbps, it falls to around 40

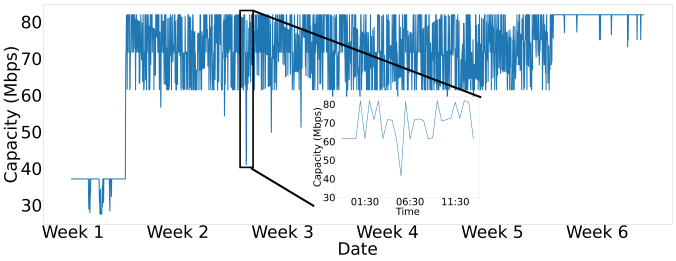


Fig. 2: Example of bandwidth fluctuations in a wireless link used in The Nevada Climate-ecohydrological Assessment Network (NevCAN) project [22]. Extreme weather conditions lead to a more than 50% reduction in transmission rate.

Mbps occasionally, likely as a result of weather conditions. While it may be possible to capture the signal-to-noise ratio on wireless interfaces to detect such bandwidth degradation, it is not always easily accessible to network administrators and, therefore, requires a manual process of examining system behavior to detect such changes. Although there are backup links that can be used in the event of primary link failures, bandwidth degradation events typically do not trigger route changes as primary links are still functional. This in turn can cause applications to experience congestion while alternate links go unused.

SDN offers a great opportunity to take advantage of available links, as it allows us to centrally monitor and control all switches in the network. However, existing solutions in this domain assume that link bandwidth does not change over time, so they solely focus on detecting congestion and mitigating it by choosing custom routes to improve network utilization. As a result, they fail to resolve network congestion caused by link performance degradation events in wireless links. We, therefore, introduce *RENET* to detect bandwidth fluctuations in wireless links and adapt to them by rerouting some of the flow to underutilized links.

IV. System Design

We implemented *RENET* by extending the ONOS controller. *RENET* uses two data stores and three services to keep track of network status and make routing decisions as illustrated in Fig. 3. We parse *OpenFlow* stats using *Stats Parser* service and save flow and link-level information in respective data stores. We call the *Path Selection* algorithm when a new flow arrives, an existing flow terminates, or congestion is detected in any of the links to find alternative routes. Finally, *Bandwidth Tracking* service plays a key role in detecting bandwidth changes due to environmental conditions. It supports two approaches to detecting the current capacity of a link. The first and preferred approach uses SNMP metrics sent from wireless bridges which include current throughput as well as capacity. However, if SNMP data is not available, either due to access limitations or lack of support by the wireless device, we utilize the *Bandwidth Tracking* service with the information stored in the *Link Store* to detect link capacity changes. We next explain each of these services and data stores in detail.

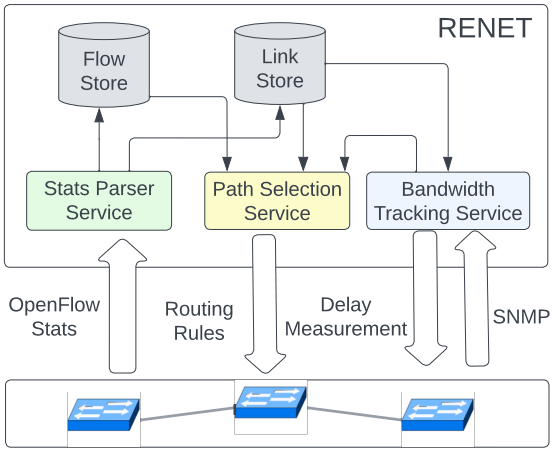


Fig. 3: *RENET* utilizes two stores to keep track of link and flow statistics. It also implements three services to monitor the network, detect bandwidth fluctuations, and make routing decisions.

A. Stats Parser Service

This service is responsible for capturing and processing *OpenFlow* stats the network devices send periodically. This allows *RENET* to make informed routing decisions, thereby maximizing network utilization and improving the quality of service for the active flows. *OpenFlow* stats include both port statistics and flow statistics. Port statistics are utilized in measuring link congestion, and flow statistics in the calculation of individual flow throughput. Estimating an individual flow throughput benefits in satisfying the demands of bandwidth-sensitive applications, by measuring the distance between the current throughput and the demanded bandwidth of that application.

B. Flow Store

Since *RENET* aims to improve the QoS for applications, it keeps track of a few statistics for active flows as shown in Table I. *Current path* is determined by the *Path Selection* service. *Current rate* is an estimation of the real-time throughput of the flow. The *Current rate* is calculated using the *Stats Parser* service from the *OpenFlow* stats from the edge switch, which are collected every 5 seconds by default. The *Desired rate* is specified just before the data flow initiates for bandwidth-sensitive applications. This value is set to -1 for the other applications, indicating a preference for maximum available bandwidth without strict adherence. The *Update time* records the timestamp of a flow's route change, ensuring that a flow will not be rerouted more than once within two consecutive *OpenFlow* stats intervals. This promotes stability and allows all active flows the opportunity to secure improved routes. Finally, the *Stats Parser* service sets a flow as *Active* when it starts, and switches it to *False* if the edge switch does not report new statistics of the flow in two consecutive intervals. This helps *RENET* ignoring paused or completed flows when making scheduling decisions for other flows.

TABLE I: Statistics stored in *Flow Store* and *Link Store* for active-flows and active-links characteristics

Flow Store Metrics	
Src-Dst	Source-Destination IP and port information
Current path	The current route of the flow
Current rate	The current throughput for the flow
Desired rate	The desired throughput for the flow
Update time	Timestamp of the last routing decision
Active	Status of the flow (active or not)
Link Store Metrics	
Current throughput	Instantaneous traffic rate on the link
Base delay	Base delay of the link
Current delay	Currently measured delay of the link
Base BW	The base capacity of the link
Current BW	Current measured capacity of the link
Last updated	Timestamp of the last update to the link

C. Link Store

Link Store records basic statistics for each link in the network. It tracks five metrics as in Table I. One of this work's contributions is to estimate a link's effective capacity and transmission delay, and detect the temporary changes (delay increase or bandwidth decrease) due to extreme weather conditions, as shown in Fig. 2. We assume that system administrators have prior knowledge about the link's capacity and the transmission delay under ideal, non-congested conditions. *Bandwidth Tracking* service either polls SNMP metrics periodically or executes lightweight probing delay measurements to identify congested links and update their capacity.

D. Bandwidth Tracking Service

Running active throughput measurements using tools such as Netperf [24] and iPerf [25] can help estimate the available throughput between two nodes. However, this approach produces overhead on the target link, as measurements must be performed periodically. While some researchers have proposed lightweight methods [26], [27], these are mainly designed to estimate end-to-end bandwidth between end hosts, which doesn't align with our needs as it necessitates link-level information.

Another approach to determining if a link is fully utilized involves capturing packet loss information. When a link reaches full utilization, the receiving switch may encounter a buffer overflow, leading to packet loss. We calculate packet loss on a link by comparing the number of packets received by the receiving switch to those transmitted by the transmitting switch at each end of the link. Conversely, if a packet loss occurs in a link, it's an indication that the link is fully utilized.

We examined this method by simulating data flow between two nodes: $N1$ acting as the sender and $N2$ as the receiver. The flow passed through an Allied-Telesis SDN switch, denoted as S. The bandwidth of the link between $N1$ and S is set at 1 Gbps, while the link between S and $N2$ operates at 100 Mbps. We captured the difference between the packets received by S from $N1$ (RX) and the packets sent by S to $N2$ (TX) in two scenarios: normal operation and congestion. The graph in Fig. 4 illustrates that $RX - TX$ values show

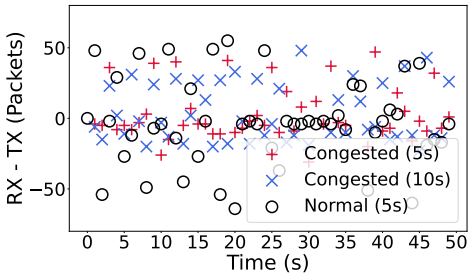


Fig. 4: The difference between received (RX) and sent (TX) packet counts for a switch port as reported by OpenFlow stats. $(RX)-(TX)$ cannot be used to infer congestion since both normal and congestion scenarios produce similar values.

significant fluctuations in both normal and congested scenarios. Although further analysis is necessary to understand why $RX - TX$ yields nonzero values when the link is not fully utilized, it is apparent that this method is not feasible for detecting congestion-related packet loss events.

Alternatively, we employ two approaches to keep track of the estimated utilization of links. The first and preferred approach relies on SNMP statistics reported by wireless devices. Different from wired switches, the SNMP statistics of wireless bridges include capacity information for wireless connections, allowing us to quickly detect performance degradation. However, SNMP data may not always be accessible due to device support limitations or administrative control issues. Therefore, we've also developed a lightweight delay probing technique to identify link congestion while not fully utilizing their initial bandwidth. This helps us detect potential bandwidth drops.

Previous work showed that network delay can be used to infer congestion as queuing delay on bottleneck links increases significantly [28]–[30]. Hence, we adopted the one-way-delay (OWD) measurement technique proposed in [31] to detect congested links. The technique works as follows: To measure the one-way delay for the link between Switch-1 (S1) and Switch-2 (S2) in Fig. 5, we first measure the delay between the switches and the controller, t_{s1} and t_{s2}, using *Echo* messages sent from the controller. When a switch receives an *Echo Request* message, it responds to it with an *Echo Reply* message; thus time duration between the departure of *Echo Request* message and the arrival of the *Echo Reply* is expected to be equal to the round trip time between the switch and the controller. Next, we send a packet to loop between S1, S2, and the controller whose delay is denoted by t_{s1-s2}. Finally, we use the Equation 1 to estimate the one-way delay from S1 to S2:

$$d_{s1-s2} = t_{s1-s2} - (\frac{t_{s1}}{2} + \frac{t_{s2}}{2}) \quad (1)$$

It is expected that the OWD, d_{s1-s2}, will increase significantly when the link $S1 - S2$ is congested due to queue buildup at S1. To verify this assumption, we conducted delay measurements using software and hardware switches. We used the same topology in Fig. 5 and measured the OWD for the link between S1 and S2. The bandwidth of links between the source and S1, and S2 and the destination is set to 1 Gbps,

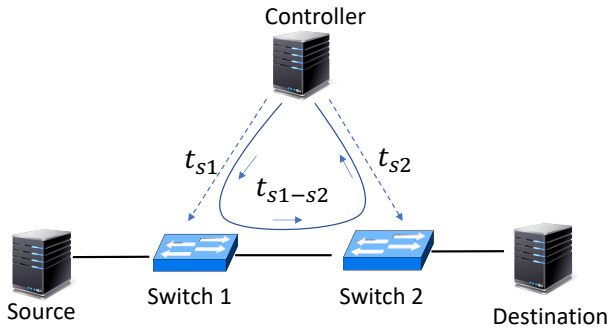


Fig. 5: One way delay measurement technique proposed in [31]

whereas the bandwidth between S1 and S2 is set to 100 Mbps. The base delay (i.e., delay without congestion) is measured as $1ms$ for the $S1 - S2$ link and $0.1ms$ for all other links.

In Fig. 6a, we observe the impact of link utilization on the OWD with virtual switches using Mininet. We set the buffer size on the egress port of S1 to 100 packets and use TCP Cubic as the congestion control algorithm. We use Iperf to create TCP and UDP transfers and set traffic rates to gather statistics under various utilization conditions. We measure the delay a hundred times and report the average and standard deviation values. The OWD remains close to the base delay until the link utilization reaches 95%, at which point it increases considerably. When the link utilization hits 100%, the delay becomes more than $10x$ of the base delay with TCP and UDP flows.

Since delay measurement is highly dependent on the queue size on S1, we examine the impact of queue size on observed delay as in Fig. 6b. We varied the queue size on S1 between 10 and 200 packets with Mininet. We ran TCP traffic with 90 Mbps, 95 Mbps, and 100 Mbps rates to represent 90%, 95%, and 100% link utilization scenarios. As expected, the increasing queue size increases the measured delay as probing packets wait longer in the queue. However, the rate of increase depends on the link utilization rate. Specifically, when link utilization is 90% or 95%, the measured delay ranges between $1 - 7x$ of the base delay. However, when the link utilization reaches 100%, the measured delay ranges between $6 - 70x$ of the base delay.

On the other hand, we notice that Echo Request/Response messages (Control Layer Probing) do not work as expected with physical switches (Allied Telesis IE340-20GP) due to significant fluctuations in response time to Echo Request messages by the switches as shown in Fig. 6c. We suppose this is because Echo Request messages are sent over the control link between the switch and the controller and handled by the control port of the switch. As a result, they require the involvement of a switch CPU, which causes instability in response time.

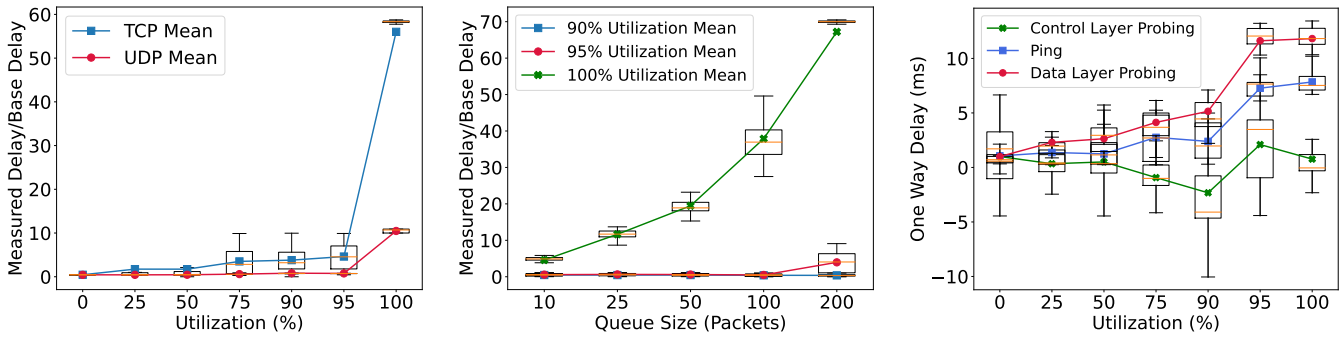
To overcome this issue, we developed a delay measurement that works with the data layer. In this method (Data Layer Probing), we set the links between the controller and the switches as "hybrid" links which can carry both control messages as well as data messages. Then, we send custom probing packets (crafted with Scapy) from the controller to switches to be forwarded as data packets. Switches process these probing packets based on OpenFlow match-action rules, and the switch CPU is not involved. Now, all three probing packets (t_{s1} and t_{s2}, and t_{s1-s2}) in Equation 1 are processed in the data layer. To avoid the delay that can be caused by installing OpenFlow rules to the switches, we install the routing rules to S1 and S2 before sending the probing packets.

We compared the proposed data-layer delay measurement approach against [31] and *ping* methods. We run *ping* between source and destination nodes and report half of the measured round trip time. Since the delays from Source to S1 and from S2 to Destination are negligible compared to the delay between S1 and S2, the observed values in ping are expected to be a close estimation of the link delay. We repeat the measurements 100 times and report the average and standard deviation values in Fig. 6c. There is no strong correlation between utilization and measured delay when using Echo Request/Response messages in the control layer. On the other hand, both our approach and ping can capture the change in the delay when utilization hits 100%. Specifically, we can still observe that the link delay increases by nearly $10x$ compared to the base delay.

Based on these experiments, *RENET* will detect capacity changes if there is an increase in the OWD even when the link is still underutilized compared to the *Base BW* value in the link store. When this occurs, we adjust the *Current BW* of the link to match the *Current throughput*. This approach is preferable to conducting active throughput measurements, as underutilized links are not expected to significantly impact performance, even if their current bandwidth is less than the maximum.

Our approach to OWD measurement involves injecting only three small packets (each less than 100 bytes) per link, resulting in almost negligible overhead. Additionally, we avoid executing delay measurements if the estimated capacity of a link exceeds 90% of its *Base BW*. *RENET* resets the *Current BW* of a link to its *Base BW* after a timeout unless a new delay measurement still indicates low capacity. This timeout mechanism accounts for the transient nature (lasting from a few hours to days) of extreme environmental conditions. However, if the estimated *Current BW* shows an increase (possibly due to improved weather conditions) at any point, *RENET* immediately adjusts the bandwidth to match the link utilization to reflect these changes

As we'll see in the next section, setting a new *Current BW* involves invoking the resource-intensive *Path Selection* service. Therefore, we only consider changes exceeding 30% of the *Current BW* value. Additionally, whenever a change in link bandwidth is detected via delay measurements or SNMP reports, we update the *Last updated* timestamp in the *Link Store*. We then ignore any further bandwidth updates in the next two time intervals, which guarantees stability in the application



(a) Impact of link utilization on One-way delay (OWD) - Mininet

(b) Impact of queue size on OWD - Mininet

(c) OWD measurement - Hardware Switch

Fig. 6: Increased link utilization leads to higher delays for both TCP and UDP traffic as packets spend more time in queues (a). Specifically, delay increases by more than $10x$ when link utilization reaches above 95% as long as queue size is larger than 10 packets (b). However, the use of control-layer messages (i.e., Echo Request/Reply) for delay measurements leads to inconclusive results, thus we introduce a data-layer-based delay probing method to accurately measure the link delay. Figure (c) shows that data-layer probing returns similar results as *ping* for increased link utilization.

E. Path Selection Service

This service finds a new path for existing or new flows to meet their demand and mitigate network congestion. Algorithm 1 shows the logic of the path selection algorithm. We utilize `getKShortestPath` API from ONOS to find the top K shortest paths between any two endpoints. Although we used $K = 100$ in the experiments, Section V-B shows that smaller K values in around $5 - 10$ are sufficient to discover good alternate routes while minimizing the execution time of the Algorithm 1.

For each considered alternate path, we find available capacity, $pathThr$, by going through individual links and estimating achievable throughput, $availableBW$ (lines 10-23). If a link is not fully utilized, the target flow may claim the free capacity. However, the flow's fair share can be more than the available bandwidth (especially if the link is already fully utilized); thus, we also estimate its fair share by dividing the link's bandwidth by the number of active flows, $\frac{link.estBW}{link.flowCount+1}$. We take the maximum of available bandwidth and the flow's fair share in a link as an estimated throughput for the flow on a link. The throughput of a path is calculated as the throughput of its link with the lowest estimated capacity (line 18). Next, we sort the paths in ascending order based on the estimated throughput and check if the application has a specific bandwidth request, $desiredRate$. If it does, we select a path with the lowest estimated free capacity that meets the flow's bandwidth requirement (line 28). Otherwise, if either the flow has no predefined bandwidth requirement or none of the paths has sufficiently high free bandwidth to meet the bandwidth requirement, we assign it to the path with the highest estimated throughput. The motivation behind selecting the lowest bandwidth route in the first case is to leave the paths with a higher capacity to future flows which may request higher bandwidth. If multiple paths have the same free capacity, we choose the one with the lowest hop count.

The path selection service is triggered by three events (i) the start of a new flow, (ii) the termination of an existing flow, and (iii) capacity change in wireless links due to environmental conditions. In the first case, *RENET* uses the Algorithm 1 to find a route that fulfills the bandwidth requirements of the new flow if its bandwidth demand is specified. The second case is for active flows to benefit from this change, especially if their demand is not satisfied. Therefore, when the *Flow Store* detects that a flow is not active anymore (through missing flow reports), the path selection service iterates over active flows (using *Flows Store*) to find the one(s) that can be rerouted and runs Algorithm 1 to check if they can be assigned to a different path for improved performance. Finally, capacity change events also trigger path selection service to consider rescheduling flows on the affected link.

Since it will be costly to go through all active flows every time a flow terminates, the service only evaluates (i) flows with 1 Mbps throughput or more (ii) whose throughput demands are not satisfied (iii) that have not been rerouted recently (within the past two monitoring intervals) and (iv) whose *Current rate* is below 75% of their *Desired rate* if applicable. We next sort the remaining flows in ascending order based on their satisfaction ratio to prioritize the flows with lower fulfillment rates. Then, we iterate over the flows and call Algorithm 1 to check if they can be routed through different paths for better performance. To implement stability in the network, we apply route changes only if the new route is expected to offer 25% or higher throughput compared to the current route.

Managing applications without specific bandwidth requirements can be challenging because their flow's desired rate is undefined. Initially, we neglect flows where the bottleneck link in their current path is utilized at 80% or less. This implies they don't need additional bandwidth. Then, we invoke Algorithm 1 to analyze other possible paths that could offer these flows a greater bandwidth. To give a fair chance for all flows in the network, we exclude a recently rerouted flow from rerouting analyses in the subsequent rounds.

Algorithm 1: The heuristic algorithm in Path Selection Service to find routes for new and existing flows.

Result: $path$ for a flow between src and dst pair
getKShortestPath(n1, n2, K) = Returns K shortest paths between nodes n1 and n2
getLowestBWLink (path) = Returns the link with the lowest free capacity
sort(paths) = Sort paths in ascending order based on their free bandwidth
$LinkStore$ = List of links
$FlowStore$ = List of active flows
$flow$ = Flow to be routed
$desiredRate$ = $flow$.rate
$paths$ = getKShortestPath($flow$.src, $flow$.dst, K)
$pathList$ = {}
foreach $path \in paths$ **do**
    $pathThr = INF$
    foreach $link \in path.links$ **do**
        $flowCount = link.flowCount$
        $link.usage = \sum_{n=1}^{flowCount} link.flows[n].thr$
        $availBW$ = $link.estBW - link.usage$
        $fairShare$ = $\frac{link.estBW}{flowCount+1}$
        $expectedThr$ = max($availBW$, $fairShare$)
        if $expectedThr < pathThr$ **then**
            $pathThr$ = $expectedThr$
        end
    end
    $pathList[path]$ = $pathThr$
end
sort($pathList$)
if $desiredRate \neq Null$ **then**
    foreach $path \in pathList$ **do**
        $pathThr$ = $pathList[path]$
        if $pathThr > desiredRate$ **then**
            return $path$
        end
    end
end
else
    return $pathList[-1]$
end

V. Experimental Results

We evaluate the performance of *RENET* by testing it in real-world and emulated networks using multiple network topologies and traffic patterns. We compare *RENET* against BACAR [21], DICES [20], and the reactive forwarding (RFWD) solutions. The RFWD shortest-path algorithm is the built-in flow-forwarding solution in the ONOS controller. DICES uses NSGAII to find the optimal routes for new flows as well as to detect and mitigate network congestion. Unlike *RENET*, DICES and RFWD assume that the network bandwidth is static, so their routing logic is oblivious to bandwidth fluctuations. Finally, BACAR is the closest work to *RENET* as it conducts delay measurements to detect bandwidth variations in wireless links. Its routing algorithm, however, does not consider application requirements. As a result, it adopts a best-effort approach and chooses paths with the highest available capacity. We present the performance of *RENET* using both delay (*RENET-D*) and SNMP (*RENET-S*) based capacity detection approaches as described in Section IV-D.

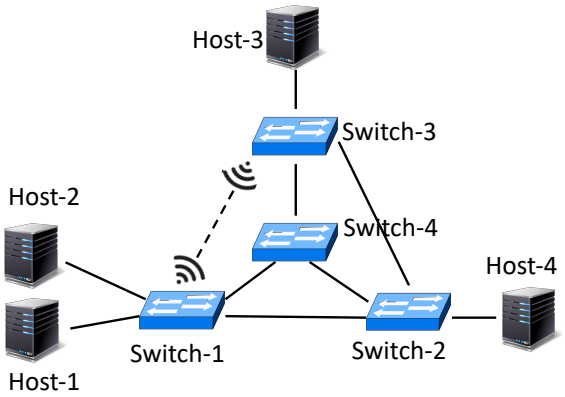


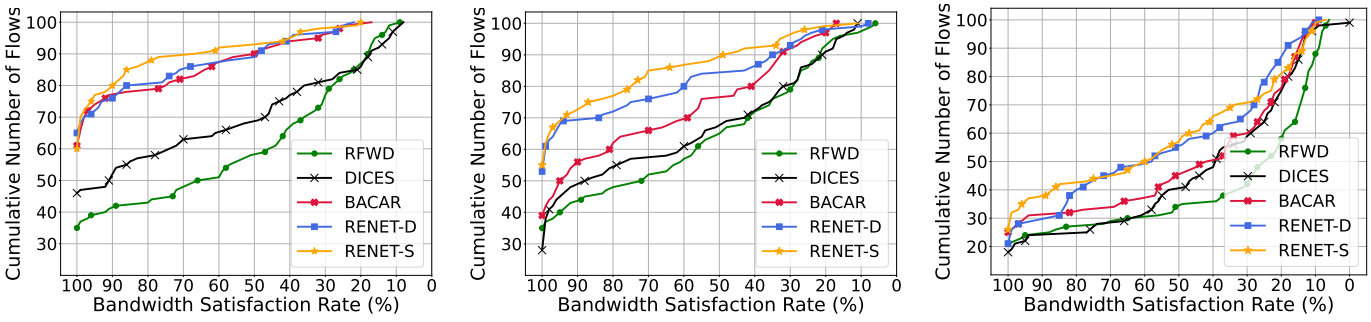
Fig. 7: The network topology used in the evaluations.

A. Real-World Experiments

Fig. 7 displays the in-lab topology we used to emulate the edge network used by the University of Nevada, Reno researchers for weather sensing and wildfire monitoring [22], [23]. We used Allied Telesis (AT) IE340-20GP OpenFlow 1.3 compatible low-power, weather-resistant switches. We used four AT switches connected with 50 Mbps wired links except for the link between Switch-1 and Switch-3, which uses UISP airFiber 2.4 GHz wireless bridge, which provides around 50 Mbps throughput.

1) Bandwidth-Sensitive Applications: We first evaluate the performance of *RENET* when flows specify their bandwidth demand before they start. We use *Poisson Distribution* with three different mean values (3, 5, and 10 seconds) to create randomized workloads that lead to varying degrees of network congestion. Flow rates and sizes are in the $5-25$ Mbps and $50-250$ MiB ranges respectively. We selected flow sizes to emulate video streaming with 144p to 4k resolution for $10-50$ seconds under optimal network conditions. Finally, we select source and destination hosts randomly to simulate data flow on both ways of the link. Hence, flows last anywhere between $16-400$ seconds depending on size and rate.

Due to our limited wireless device availability, and to replicate the fluctuations in link performance experienced in wireless environments, we temporarily decrease the capacity of two wired links, namely Switch-1 to Switch-2 and Switch-2 to Switch-3, by $40\%-80\%$ for around 250 seconds. After this period, we restore the links to their original capacity. Since switch interfaces only support 10/100/1000 Mbps rates, we limited the capacity of the links by utilizing the Linux traffic controller (`tc`). We placed an obstacle between the transmitter and the receiver for the wireless link (between Switch-1 and



(a) Light Congestion (b) Medium Congestion (c) Heavy Congestion

Fig. 8: Evaluation of fixed-rate flows in terms of bandwidth satisfaction ratio under different congestion scenarios. BACAR attains competitive results when the network is lightly congested but falls behind *RENET-S* and *RENET-D* by around $10-30\%$ as network congestion intensifies.

Switch-3) to reproduce the bandwidth degradation. We noticed that its capacity drops from 50 Mbps to around $30-40$ Mbps as signal strength drops significantly.

Fig. 8 presents the bandwidth satisfaction rate for all flows. Since the flow throughput changes during its execution, we calculated the average rate of the flow during the execution time. Then, we calculate the "Satisfaction Rate" by comparing the average rate rate to the *Desired rate*. We observe that *RENET-D* and BACAR perform similarly when network congestion is light (Fig. 8a). This can be attributed to the demand-agnostic congestion avoidance approach, as implemented by BACAR, proving to be a sufficient solution when the network is not congested.

RENET-S attains a slightly better performance. The reason is that SNMP-based capacity tracking can learn about bandwidth changes as soon as they happen, so the *Path Selection* service will assign only as much as the link's current capacity, thereby proactively avoiding congestion. This contrasts with delay-based capacity tracking in *RENET-D* and BACAR, which detects bandwidth changes only after identifying a congested link. Fig. 9 compares the estimated bandwidth using SNMP and delay-based approach for the wireless link as in the topology in Fig. 7. Although the link capacity is dropped after around 70 seconds, the delay-based approach detects it around 15 seconds later.

On the other hand, both DICES and RFWD underperform BACAR, *RENET-S*, and *RENET-D* significantly. Specifically, while *RENET-S* achieves 80% satisfaction ratio for 88 of flows, DICES, and RFWD attain 80% satisfaction for less than 60 and 45 flows respectively. Despite using a robust model to find optimal routes, DICES fall short of noticing bandwidth fluctuations. Even worse, it oversubscribes degraded links as it assumes that they are underutilized. For example, if the bandwidth of the link drops from 50 Mbps to 20 Mbps while it is being fully utilized, DICES will assume that it is 40% utilized and assign more flows to it, exacerbating the congestion.

As network congestion intensifies as in Fig. 8b, *RENET* starts to outperform BACAR by a considerable margin. For instance, *RENET-S* is able to offer at least half of the *Desired rate* to more than 90 flows, while this value drops to

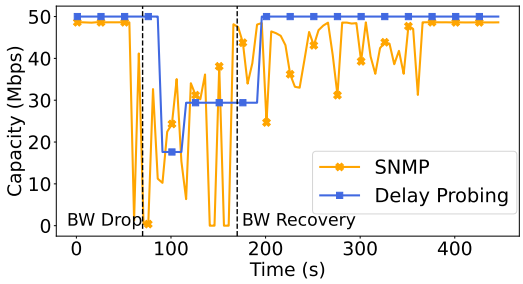


Fig. 9: The impact of antenna misalignment on the capacity of a wireless link. Although SNMP polling helps to detect bandwidth changes immediately and accurately, the delay-based approach offers a reasonable alternative when SNMP metrics are not available

around 85 flows for *RENET-D*, around 75 flows for BACAR, and less than 70 flows for DICES and RFWD. Although both *RENET* and BACAR can detect bandwidth changes, taking application demand into account helps *RENET* to perform better by routing bandwidth-insensitive flows to less utilized paths and minimizing environmental-conditions impact on bandwidth-sensitive flows.

As expected, bandwidth satisfaction worsens as the network congestion amplifies (Fig. 8c). In this scenario, only 60 flows got more than 50% of their *Desired rate* with *RENET-S*. Yet, less than 45 flows obtained 50% of their *Desired rate* with the other solutions.

Fig. 10 demonstrates an in-depth view of the network utilization for the medium congestion scenario. The utilization is calculated by summing up the throughput on egress ports of the edge switches. In particular, RFWD and DICES suffer significantly when the capacity of a link drops. DICES is severely impacted because it cannot detect drops in bandwidth and incorrectly considers these links as underutilized. After the first bandwidth drop, *RENET-S* and BACAR swiftly detect the drop and reroute the flows to the other underutilized links, while avoiding the congested links.

Although *RENET-S* and BACAR are adversely affected by the second capacity drop event, they can quickly recover from that and avoid operating in the affected links. During bandwidth recovery scenarios, *RENET-S* quickly detects recovery events by capturing SNMP updates. In contrast,

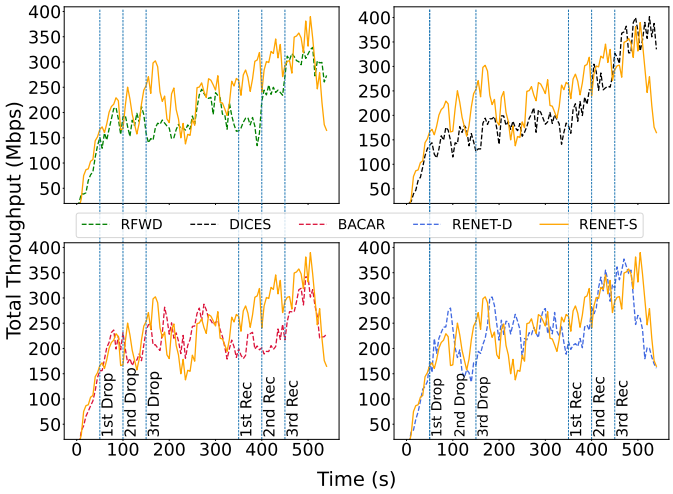


Fig. 10: Overall network utilization comparison under medium network congestion scenario. *RENET-S* significantly outperforms RFWD and DICES before and during link-bandwidth drops. *RENET-S* yields 10% higher utilization compared to BACAR by means of detecting link-bandwidth degradations and proactively rerouting flows to underutilized links.

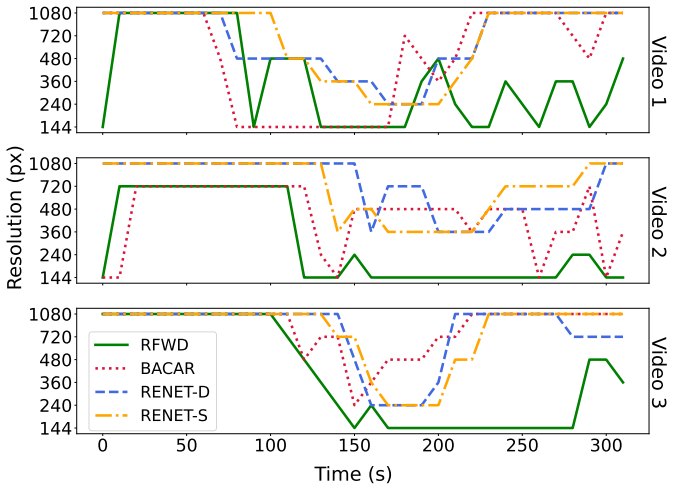


Fig. 11: Performance comparison of *RENET*, BACAR, and RFWD algorithms using adaptive video streaming applications. *RENET-S* achieves 21% and 54% higher average quality for Video-1 and Video-3 compared to BACAR while attaining 0.8% lower quality for Video-2.

BACAR relies on a timeout mechanism to reset the link's bandwidth, which is set to 50 seconds in this experiment. Overall, we observe that BACAR attained 207 Mbps average network utilization whereas *RENET-S* achieved 227 Mbps, almost a 10% improvement.

2) Adaptive Video Streaming: Next, we evaluate the impact of learning the *Desired rate* of the adaptive video streaming applications, whose quality depends on the available network bandwidth. This is particularly relevant to the research network used by the University of Nevada, Reno researchers. Their network carries traffic from multiple wildfire monitoring cameras using an adaptive video streaming application. Avoiding congested links when routing video flows will allow first responders to access high-quality video streams from incident locations.

We set up an experiment to stream a 300-second long video from two servers to three clients using the network topology shown in Fig. 7. Host 2 streamed a video to Host 4 and Host 3 and Host 3 streamed to Host 1. We processed the video file to generate five different resolutions 144p, 240p, 360p, 480p, 720p, and 1080p. Specifically, the 1080p version is served when transfer throughput is more than 10 Mbps while the 144p version is served when throughput is below 4 Mbps. We use the HTTP-Live-Streaming (HLS) protocol to stream video files with an NGINX server. To congest the network, we executed 50 TCP flows in the background in a similar pattern as discussed in Section V-A1.

Fig. 11 shows the quality of all three video streams when using *RENET-S*, *RENET-D*, BACAR, and RFWD. We again emulate bandwidth fluctuations by either reducing the link capacity manually (using `tc`) or putting an obstacle between the wireless antennas. We initiate the video streams under light network load conditions thus they attain a good resolution at the beginning. However, as soon as the first link bandwidth drops, the RFWD algorithm struggles to manage the situation, leading to a fall in video quality across all streams.

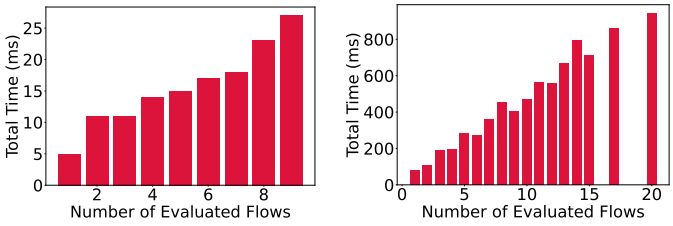
Although BACAR handles the issue better than RFWD, faces challenges in meeting the demand for certain video streams, primarily Video-1. A lack of information about each video's *Desired rate* leads to a lower quality Video-1 with only 140p for almost 100 seconds. On the other hand, *RENET-S* and *RENET-D* achieve the best overall performance by offering at least 240p quality for all video streams.

Although *RENET-S* yields 0.8% lower quality for Video2 compared to BACAR, it obtains 21% and 54% higher quality for Video1 and Video3, respectively. It is important to note that *RENET-S* leads to lower fluctuations compared to *RENET* and BACAR since it learns about capacity changes immediately and proactively avoids congesting the affected link as much as possible.

B. Scalability Analysis

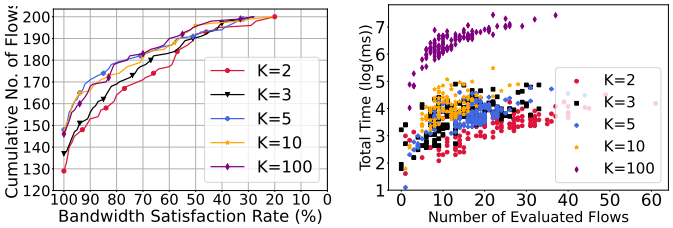
In this section, we examine the scalability of *RENET* in terms of handling large-scale networks. Since *RENET* creates multiple cache objects (i.e., Link and Flow store), runs delay measurements to detect bandwidth changes (if SNMP is not available), and introduces a new route selection algorithm, it is important to test it with more complex networks to ensure it does not lead to performance issues. We created a network topology with 20 switches, 50 hosts, and 100 inter-switch links using Mininet. We adopted the same traffic generation pattern as in the real-world experiments.

Fig. 12 illustrates *RENET*'s ability to assign a single flow to the best available route within 5ms, utilizing the hardware-switches topology in Fig. 7. We observe a linear relationship between the execution time and the number of flows considered, which is a good sign for scalability. Specifically, it takes less than $25ms$ to evaluate 9 flows. Shifting to a broader scale



(a) Allied Telesis IE340-20GP (b) OVS - Mininet

Fig. 12: Execution time of *RENET*'s path selection algorithm using real-world hardware (a) and software (OVS) (b) switches. It takes less than $5ms$ to find a path for one flow with real hardware and less than $40ms$ with a software switch. The total execution time increases almost linearly for increasing number of flows that are considered for rerouting.



(a) Bandwidth Satisfaction Rate (b) Path Selection Time

Fig. 13: Evaluation of different K values in the *K-shortest-path* algorithm regarding path selection time and bandwidth satisfaction rate. K values of 5 and 10 attain a near-optimal bandwidth satisfaction rate for flows while lowering the execution time.

with the Mininet simulated topology, evaluating the optimal path for a single flow among thousands of paths available requires approximately 40ms while evaluating 20 flows takes roughly a second.

In comparison, we noticed that it takes nearly $2-5$ seconds for DICES to find a solution for each flow when evaluating 20 flows. This is because DICES uses a linear programming-based solution whose execution time increases drastically as the number of flows and available paths increases. On the other hand, *RENET* uses a heuristic approach to process each flow independently and keep the execution time low. As we consider at most 100 flows for rerouting in each interval, the execution time of the *Path Selection* service is guaranteed to stay under 5 seconds, the polling interval of OpenFlow stats.

The *Bandwidth Tracking* service relies on the data stored in *Link Store* to detect congestion and bandwidth change events. Therefore, it's essential to keep the execution time of Algorithm 1 at or below the reporting frequency of the OpenFlow stats. This enables the *Link Store* to update its records before the next execution of *Bandwidth Tracking* service, which expects to retrieve the new updates.

We next evaluate the impact of the number of routes that the *Path Selection* service considers when rerouting flows (K in Algorithm 1 line 8). A higher K value means that the *K-Shortest-Path* algorithm will return more paths to evaluate, thereby increasing the execution time of *RENET*. To assess its impact on large-scale networks, we created a 5-pod fat-tree network topology in Mininet using 10 nodes and generated similar traffic as described in Section V-A.

We observe that restricting the number of alternate routes to less than 5 (i.e., $K = 2, 3$) adversely affects the bandwidth satisfaction rate as shown in Fig. 13a. On the other hand, large K values result in long execution times as shown in Fig. 13b. Specifically, $K = 100$ leads to $1.7s$ execution time when handling 40 flows. While this duration is not detrimental to the performance of *RENET*, it can be a limiting factor when handling larger-scale networks with hundreds of active flows. Thus, setting K values of 5 or 10 appears to be sufficient to strike a good balance between reducing execution time and increasing application quality of service (i.e., satisfaction rate).

C. Limitations

While offering significant QoS improvements for bandwidth-sensitive flows and adaptive video streaming applications, the proposed solution makes a few assumptions about the underlying network architecture. First, it requires that all switches are OpenFlow-enabled and under the administration of a single entity to take advantage of network-wide traffic engineering. If this is not the case, the proposed solution can still be applied in a small network segment where link bandwidth fluctuations can occur. Moreover, the delay-based link capacity detection method necessitates either a dedicated out-of-band control channel between the controller and switches or utilizing priority queues on switches to avoid probing packets affected by congestion on links other than the target link. As separating control and data paths is a common practice in production networks, we believe this is a fair assumption.

VI. Conclusion

Science projects heavily rely on edge networks to collect data from remote locations such as mountaintops. However, extreme environmental conditions such as freezing temperatures and snowstorms can deteriorate the capacity of wireless links used in these networks, adversely affecting the performance of delay-sensitive flows. Although edge networks are designed to have backup routes to cope with link failures, existing rigid routing solutions impede the use of these alternate routes in the event of bandwidth fluctuations.

In this paper, we introduce *RENET* to detect bandwidth changes in wireless links and automatically reroute flows to underutilized routes. Doing so increases overall network utilization and enhances the quality of service for applications. Experimental results collected in both emulated and real-world networks show that *RENET* improves application QoS satisfaction rate by more than 30% compared to state-of-the-art solutions. We further demonstrate that *RENET* can improve the quality of adaptive video streams by more than 50%. Finally, we show that *RENET* can find a solution for a flow in less than $5ms$ by adopting a heuristic path selection algorithm.

References

[1] S. Khan, F. K. Hussain, and O. K. Hussain, "Guaranteeing end-to-end qos provisioning in soa based sdn architecture: A survey and open

issues," *Future Generation Computer Systems*, vol. 119, pp. 176–187, 2021.

[2] M. Karakus and A. Durresi, "Quality of service (qos) in software defined networking (sdn): A survey," *Journal of Network and Computer Applications*, vol. 80, pp. 200–218, 2017.

[3] S. K. Keshari, V. Kansal, and S. Kumar, "A systematic review of quality of services (qos) in software defined networking (sdn)," *Wireless Personal Communications*, vol. 116, no. 3, pp. 2593–2614, 2021.

[4] S. Mehraban and R. Yadav, "Quality of services in hybrid sdn (hsdn): A review," in *2022 7th International Conference on Communication and Electronics Systems (ICCES)*. IEEE, 2022, pp. 652–658.

[5] H. E. Egilmez, B. Gorkemli, A. M. Tekalp, and S. Civanlar, "Scalable video streaming over openflow networks: An optimization framework for qos routing," in *2011 18th IEEE international conference on image processing*. IEEE, 2011, pp. 2241–2244.

[6] T.-F. Yu, K. Wang, and Y.-H. Hsu, "Adaptive routing for video streaming with qos support over sdn networks," in *2015 International Conference on Information Networking (ICOIN)*. IEEE, 2015, pp. 318–323.

[7] R. Kanagevlu and K. M. M. Aung, "Sdn controlled local re-routing to reduce congestion in cloud data center," in *Proceedings of the 2015 International Conference on Cloud Computing Research and Innovation (ICCCRI)*, ser. ICCCRI '15. USA: IEEE Computer Society, 2015, p. 80–88. [Online]. Available: https://doi.org/10.1109/ICCCRI.2015.27

[8] E. F. Castillo, O. M. C. Rendon, A. Ordonez, and L. Zambenedetti Granville, "Ipro: An approach for intelligent sdn monitoring," *Computer Networks*, vol. 170, p. 107108, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128619304608

[9] A. A. Abushagur, T. S. Chin, R. Kaspin, N. Omar, and A. T. Samsudin, "Hybrid software-defined network monitoring," in *International Conference on Internet and Distributed Computing Systems*. Springer, 2019, pp. 234–247.

[10] 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, 2018.

[11] A. Yassine, H. Rahimi, and S. Shirmohammadi, "Software defined network traffic measurement: Current trends and challenges," *IEEE Instrumentation & Measurement Magazine*, vol. 18, no. 2, pp. 42–50, 2015.

[12] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "Opentm: Traffic matrix estimator for openflow networks," in *Passive and Active Measurement*, A. Krishnamurthy and B. Plattner, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 201–210.

[13] 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)*, 2014, pp. 1–9.

[14] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "Flowsense: Monitoring network utilization with zero measurement cost," in *Passive and Active Measurement*, M. Roughan and R. Chang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 31–41.

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

[16] L. Liao, V. C. M. Leung, and M. Chen, "An efficient and accurate link latency monitoring method for low-latency software-defined networks," *IEEE Transactions on Instrumentation and Measurement*, vol. 68, no. 2, pp. 377–391, 2019.

[17] R. Braden, D. Clark, and S. Shenker, "Integrated services in the internet architecture: an overview," 1994.

[18] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," Tech. Rep., 1998.

[19] E. Rosen and Y. Rekhter, "Rfc2547: Bgp/mpls vpns," 1999.

[20] S. Y. Shin, S. Nejati, M. Sabetzadeh, L. C. Briand, C. Arora, and F. Zimmer, "Dynamic adaptation of software-defined networks for iot systems: A search-based approach," in *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2020, pp. 137–148.

[21] O. A. Hamdan, S. Strachan, and E. Arslan, "Bandwidth and congestion aware routing for wide-area hybrid networks," in *2022 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. IEEE, 2022, pp. 1–6.

[22] "NevCAN: Nevada Climate-ecohydrological Assessment Network," https://nevcan.dri.edu/, 2023.

[23] "ALERTWildfire Project," https://www.alertwildfire.org/, 2023.

[24] "Netperf - a network performance benchmark," https://linux.die.net/man/1/netperf, 2023.

[25] "iPerf," https://iperf.fr/, 2023.

[26] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell, "pathchirp: Efficient available bandwidth estimation for network paths," in *Passive and active measurement workshop*, 2003.

[27] K. Harfoush, A. Bestavros, and J. Byers, "Measuring capacity bandwidth of targeted path segments," *IEEE/ACM Transactions on Networking*, vol. 17, no. 1, pp. 80–92, 2008.

[28] K. Nichols and V. Jacobson, "Controlling queue delay," *Communications of the ACM*, vol. 55, no. 7, pp. 42–50, 2012.

[29] L. Le, K. Jeffay, and F. D. Smith, "A loss and queuing-delay controller for router buffer management," in *26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*. IEEE, 2006, pp. 4–4.

[30] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "Timely: Rtt-based congestion control for the datacenter," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 537–550, 2015.

[31] K. Phemius and M. Bouet, "Monitoring latency with openflow," in *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, 2013, pp. 122–125.