Bandwidth and Congestion Aware Routing for Wide-Area Hybrid Networks

Osama Abu Hamdan, Scotty Strachan, and Engin Arslan
Department of Computer Science and Engineering
University of Nevada, Reno, USA
oabuhamdan@nevada.unr.edu, sstrachan@nshe.nevada.edu, earslan@unr.edu

***Abstract*—An increasing number of science projects rely on sensors to gather data from remote areas. For example, a wildfire detection project deploys video cameras to various high-risk areas to identify wildfires as quickly as possible. These projects typically depend on hybrid networks that are composed of a combination of wireless and wired links to stream data from sensors to datacenters. The management of such hybrid networks is a burdensome task as both link capacities and traffic rates of flows are dynamic and unpredictable.**

In this paper, we introduce a BAndwidth- and Congestion Aware Routing (*BACAR*) for hybrid networks using Software Defined Networks. We use active delay measurements to identify congested links and update their capacity to measure traffic rate to detect any link capacity degradation events. We test *BACAR* in Mininet and show that it offers a robust solution to detect bandwidth fluctuations that can happen in long-distance wireless links and find optimal routes for rate-sensitive flows to improve overall network utilization and enhance flow performance.

***Index Terms*—Software Defined Networks, Hybrid Networks, Congestion-aware routing**

I. Introduction

Individual edge devices and locations 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 datacenters or cloud services as illustrated in Figure 1. The resulting end-to-end network topology is highly variable in terms of availability and capacity over both space and time, often due to conditions outside of engineering control such as weather, radio interference, or physical disturbance.

Current network management practices for hybrid 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 fairly well for homogeneous networks with rare link failures, it requires highly-skilled network managers to frequently tune link weights in large-scale hybrid networks where link capacity, network demand, and QoS metrics are exceedingly dynamic. While Software Defined Network (SDN) concept is proposed to facilitate network management and improve Quality of Service (QoS), existing SDN-based solutions assume the bandwidth of links to be static and decide on flow routes based on

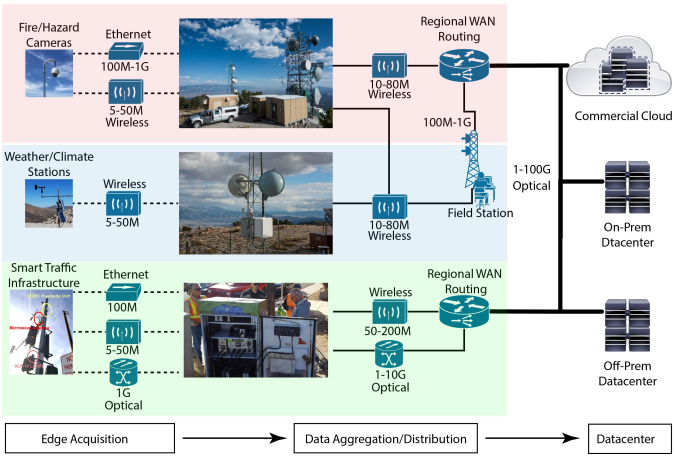

Fig. 1: A Sample hybrid network infrastructure used to stream data from sensors to central clusters.

link utilization values. However, hybrid sensor networks frequently face bandwidth fluctuations in wireless links due to environmental conditions, thus requiring an effective way of measuring link capacities to adapt to changing conditions.

In this paper, we propose bandwidth and congestion-aware routing for wide-area hybrid networks. To capture link capacity degradation, we monitor port statistics and schedule lightweight delay measurement probes if the traffic rate on any link changes significantly. Since a bottleneck link will affect the traffic rate on most links, we conduct lightweight active delay measurements to localize the link with degraded capacity. Finally, flows are rerouted to alternate routes based on the utilization of primary and alternate paths.

The rest of the paper is structured as the following: Section II provides the literature study and related works. Our implementation details are provided in Section IV. Section V demonstrates our experimentation, results, and analysis. Finally, conclusions are drawn in Section VI.

II. Related Work

Congestion control techniques in SDN can be classified into two in-network solutions [1] [2] [3] [4] and hybrid congestion methods [5] [6] [7] [8] and [9] which utilize both in-network and end hosts to control traffic rates. In-network approaches mainly modify receive buffer field in TCP packets when congestion is detected to force senders to throttle their sending rate. Hybrid solutions, on the other hand, send signals to the

traffic sender (or virtual machine hypervisor) when congestion is detected such that the sender can lower its rate to avoid overwhelming network resources.

Jamali et al. [10] targeted the load balancing problem and proposed a routing algorithm to find routes with high throughput and shortest path. They also aimed at keeping the number of changes at a minimum to minimize the impact of rerouting on application performance. Sminesh et al. [11] identify a link as a bottleneck if the link is over-utilized by periodically collecting the port statistics. Once a bottleneck is detected, the SDN controller initiates alternate route computation. If an alternate path exists, the flow admission module models a Bayesian network to decide whether the alternate path can handle the new flow load without leading to congestion propagation.

Kanagavelu et al. [12] proposed a local re-routing mechanism in SDN, where the OpenFlow controller collects the port, table, and flow statistics from all OpenFlow switches at fixed intervals. The routing engine computes the least-loaded, shortest candidate paths between any pair of end hosts, based on these statistics. It checks for congestion periodically across all the links and if any link load exceeds a threshold value, the controller re-routes one or more large flows across the link to an alternate path one by one, ensuring that the large flow will not overload the newly chosen alternate path. Kao et al. [13] proposed an effective proactive traffic re-routing mechanism for congestion avoidance using an SDN controller to manage actions and forwarding rules. The controller observes the current traffic of switches and updates the topology according to link weights that are assigned based on measured bandwidth usage. Then, the traffic on the congested link is instantly transferred to other links, if possible. Park et al. [14] proposed a routing architecture called Automatic Re-routing with Loss Detection (ARLD). In ALRD, the SDN controller treats a link as a bottleneck if a packet drop is observed, then it invokes a rerouting algorithm to find alternate paths for flows passing through the congested link.

Attarha et al. [15] proposed a congestion avoidance algorithm in which the SDN controller monitors the network status periodically and routes the newly arrived flows through a path that can forward the flow without resulting in congestion. Whenever the utilization exceeds 70% of the link capacity, the controller computes the amount of traffic and the shortest backup path through which the flows are to be re-routed to avoid congestion. This in turn reduces the load on the congested ports of an over-utilized switch.

In the area of available bandwidth measurement, most previous works adopted an active probing approach. They conduct transfers between hosts periodically to discover available capacity in different routes [16]–[18]. However, these approaches can have an adverse impact on production traffic, and thus cannot be executed frequently. Therefore, we chose to execute delay measurement to infer congestion, then use port statistics to calculate current link capacity.

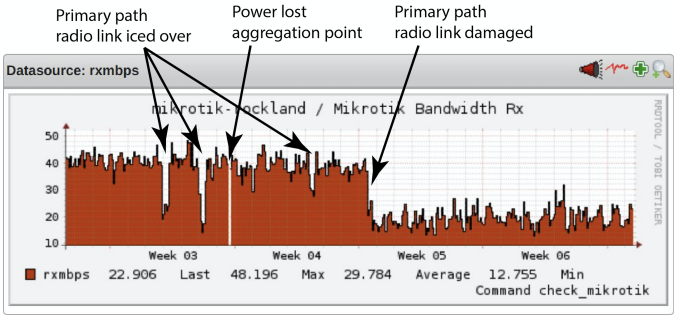

Fig. 2: Example of bandwidth fluctuations in a wireless link used in The Nevada Climate-ecohydrological Assessment Network (NevCAN) project. Extreme weather conditions lead to a nearly 50% reduction in transmission rate.

III. Background and 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 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 hybrid networks.

As an example Figure 2 demonstrates the available bandwidth of one of the wireless links used in The Nevada Climate-ecohydrological Assessment Network (NevCan) project. It can be seen from the figure that path capacity exhibits drastic changes as snowstorms cause icing and disorientation on the antenna. While the average available capacity is nearly 40 Mbps during the first and second week in monthly data, it falls below 20 Mbps in the third week as a result of weather conditions damaging individual wireless link capacities on the network. 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 still appear to be functional. This in turn can lead to applications experiencing congestion while alternate links go unused.

SDN offers a great opportunity to take advantage of available links as it can automatically divert some flows to backup routes when the primary route is congested. However, existing solutions in this domain assume that link bandwidth does not change over time, so they solely focus on detecting congestion and mitigating them by means of 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 BAndwidth and Congestion-Aware Routing (*BACAR*) to find optimal paths

Src-Dst Pair	Mac addresses for source and destination hosts
Current Path	The current route this flow is passing through
Alt Paths	Alternative paths connecting the same src-dst
Current Rate	The current recorded rate of the flow
Highest Recorded Rate	The highest recorded rate for the flow
Start Time	Start time of the flow

TABLE I: Data saved in the *Active Flows database*.

by taking both bandwidth and congestion into account.

IV. Implementation Details

BACAR consists of several data stores and services that are working in parallel to keep track of performance statistics and detect and mitigate network congestion events.

A. Link Utilization Store

This data store keeps track of link utilization statistics for every link in a network which plays an important role in detecting congestion events. It keeps track of four metrics as *latest* rate, *current capacity* rate, *bandwidth*, and *propagation delay*. *Latest* is a list of reported traffic rates in last m reporting intervals whereas *current capacity* is maximum traffic rate observed in last n reporting intervals where $n >> m$. The default value of n and m are set to 15 and 5, respectively. Unlike latest and capacity rates that depend on observed traffic rates and network congestion, *bandwidth* is a fixed value specifying the maximum traffic rate a link can carry in ideal conditions. The *current capacity* and *bandwidth* are expected to be the same for wired links all the time whereas it can be different for wireless links due to environmental conditions. As an example, if a wireless link is exposed to ice accumulation on the antenna, then its capacity for that time interval can be lower than its "true" bandwidth. To estimate the latest and current capacity rates, *BACAR* relies on port metrics included in OpenFlow statistics.

Delay is the propagation delay of a link. While it is reasonable to assume that propagation delay can be calculated ahead of time, we keep track of it in case it is not available or the initial estimation is incorrect. Since propagation delay does not change over time, we assign it to the minimum observed value among all measurements. Section IV-E details how to link delay is measured.

B. Active Flows Store

This data store keeps track of active flows in the network. When a new flow joins, we record five metrics as shown in Table I. In addition to some static values such as source-destination MAC address pair, alternate routes, and start time, we keep track of its current rate, maximum reported rate, and current path. Note that alternative paths are lazy-loaded, which means that it is not calculated on initialization.

C. Congestion Detection

Network congestion in hybrid networks can happen due to three main reasons (1) start of a new flow, (2) an increase in sending rate of existing flows, and (3) capacity decrease of wireless links due to environmental conditions. When a new flow joins, *BACAR* executes *Path Selection Algorithm* to find a route with the highest available bandwidth. To detect congestion caused by increasing sending rate or decreased link capacity, we rely on *Link Utilization Store* which parses OpenFlow statistics as they arrive from switches. After saving the reported values, it initiates the rerouting algorithm if the last reported rate is more than 30% different than the average of values in *latest* reported link rates for that link. *Link Utilization Store* can also be updated when *Link Capacity Tracker* service if the current measured link capacity appears to be different than the previous value. As a result, the start/end of a new flow, the increase/decrease in sending rate of existing flows considerable, and the increase/decrease of link capacity can all trigger the rerouting algorithm.

When the rerouting algorithm is called, it first identifies the flows passing through the target link, then tries to find alternate routes to divert some of the active flows. We first sort the flows on the link in ascending order based on traffic rate, then pull one flow from the list. We add it to *retain* list and subtract its sending rate from the target link's *current maximum*. If the flow did not demand full link capacity, we then pick the next flow from the list and add it to *retain* list then subtract its sending rate from the remaining capacity of the link. We keep adding flows until the total of their sending rates is smaller than the link capacity. As soon as the link capacity is reached, we then try to divert the remaining flows to alternate routes using *Path Selection Algorithm*. If any flow is rescheduled to an alternate route, then we run a global search for all flows to make sure that the changes we make for one link do not lead to suboptimal performance for other flows in the network.

D. Path Selection Algorithm

When path selection is called for a flow, we first calculate all available paths using Dijkstra's algorithm and sort them in ascending order based on hop count. We then iterate over available paths and calculate the available capacity of each path. To calculate the throughput of a path, we estimate the available bandwidth for each link on the path using below following formula and take the minimum capacity of all links in the path.

$$Max(capacity - max(latest), \frac{capacity}{\#offlows + 1}) \quad (1)$$

where capacity is the *current capacity* of a link and the latest is the reported traffic rates in the last few (by default 5 intervals) consecutive reporting intervals. If a link is not fully utilized, the candidate flow can claim the available capacity. However, if the link is already fully utilized by one or more flows, then the candidate flow can obtain its fair share, which can be calculated by dividing the bandwidth, *current maximum*, by the number of active flows plus 1. Note that we categorize flows with less than 1 Mbps sending rate as mice flow and do not consider them for rerouting or available link capacity estimation.

E. Link Capacity Tracker

As link capacity plays an important role in determining flow paths, it is important to detect any changes in a timely manner. Thus, *BACAR* conducts active probing to discover the current

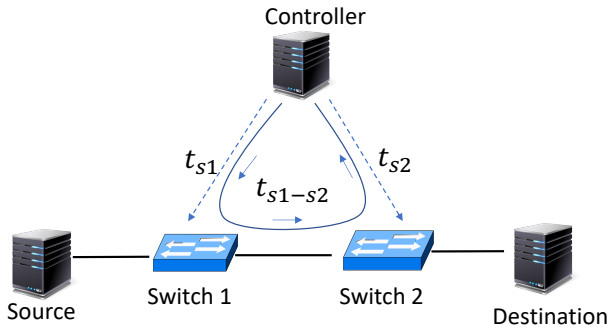

Fig. 3: When the traffic rate on a link changes more than 30%, we schedule delay measurement to determine if the link is congested by means of comparing the observed delay to propagation delay.

capacity of links when the traffic rate on a link changes more than 30%. However, as opposed to transferring a large volume of data to measure the available capacity of a path, we conduct delay measurements to detect if a link experinces a congestion. In other words, if a link with actual bandwidth B currently transfers at B' rate ($B' < B$) and it is showing the symptoms of congestion, then we can infer that the link capacity is no longer equal to B and set its new capacity, *current capacity*, to B'. Since OpenFlow statistics do not capture packet drops caused by queue overflow, we are unable to sense congestion using OpenFlow statistics. We therefore conduct delay probes to notice queue build-up on any link, which can be used to deduce congestion events.

Figure 3 illustrates the delay measurement method we adopted to detect congested links, which is originally proposed in [19] to discover the propagation delay of each link in the network. To measure the delay between *Switch 1* and *Switch 2* in Figure 3, we first send an *Echo Request* messages to both switches, wait for the *Echo Reply* messages, and measure the response time. We next create a custom packet at Controller and send it to *Switch 1* which forwards it to *Switch 2*. The packet is then forwarded to Controller by *Switch 2*, where its travel time is calculated. After all three messages are received and their duration is calculated, we use the following equation to estimate one-way delay between *Switch 1* and *Switch 2*:

$$d_{s1-s2} = t_{s1-s2} - (\frac{t_1}{2} + \frac{t_2}{2}) \quad (2)$$

where t_1 and t_2 round trip time between the Controller to switches. t_{s1-s2} is the duration of the custom packet to travel from the controller to Switch 1 to Switch 2 and back to the Controller. It is expected that d_{s1-s2} will be equal to the propagation delay of the link between Switch 1 and Switch 2 when the egress queue in Switch 1 is empty and will increase in proportion to the queue build up. Figure 4 shows the value of d_{s1-s2} relative to propagation delay for the middle link in Figure 3. Links between source and Switch 1 and Switch 2 and destination are set to 1 Gbps and middle link has 100 Mbps bandwidth and $2ms$ propagation delay. We use Iperf to create TCP and UDP transfers between source and destination nodes with fixed rates. As an example, we set the UDP traffic rate between source and destination nodes to 50 Mbps to measure link delay when the middle link is 50%

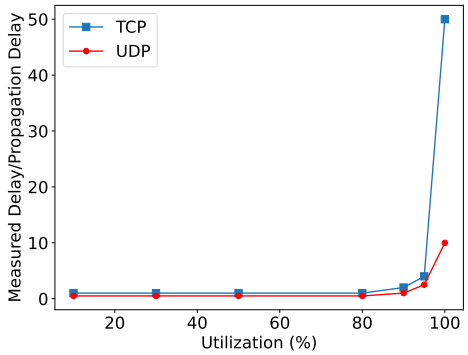

Fig. 4: Delay measurement returns high values when link utilization is near 100% utilization.

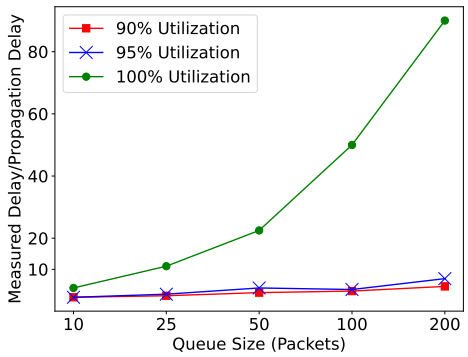

Fig. 5: Queue size has a significant impact on observed delay. Yet, when a link is fully utilized, the observed delay is still $4-80x$ higher than the propagation delay.

utilized by a UDP flow. We set the buffer size on the egress port of the Switch 1 to 100 packets. The congestion control algorithm for the TCP traffic is set Cubic. We start the Iperf transfers between source and destination before we initiate our delay measurement method for the link between Switch and Switch 2. We measured the delay ten times and reported the average value. It is evident that measured delay remains close to propagation delay until link utilization reaches 95% at which point it increases considerably. When the link utilization is at 100%, the measured delay becomes more than $10x$ of the propagation delay in the presence of both TCP and UDP flows between source and destination.

Since our delay measurement is highly dependent on queue size on the first link (Switch 1), we next examine the impact of queue size on observed delay in Figure 5. We used the same topology as shown in Figure 3, but varied queue size on Switch 1 between 10 and 200 packets. 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 leads to an increase in measured delay as probing packets now share the queue with the traffic between source and destination nodes. However, the rate of increase depends on the link utilization rate. Specifically, when link utilization is 90% or 95%, measured delay ranges between $1-7x$ of propagation delay. On the other hand, when the link utilization is 100%, the measured delay ranges between $4-90x$ of propagation delay. Therefore, we can exploit the

variation in the measured delay to determine whether or not a link is fully utilized. For example, if the measured delay is more than $10x$ of propagation delay, then we can say that the link must be fully (i.e., 100%) utilized. Please note measured delay for 100% utilization case with a queue size of 10 overlaps with the measured delay of 95% utilization with a 200 queue size, so using a low (e.g., $5x$) threshold to distinguish 100% utilization from lower utilization values may result in misclassification if the queue size is not known. We, therefore, expect the measured delay to be at least $10x$ of propagation delay to determine whether or not a link is 100% utilized, which requires queue size to be more than 25 packets. We believe this is a reasonable assumption for wide area networks and leave further optimizations for lower queue sizes as future work.

When a link delay measurement indicates 100% utilization, we update its *current maximum* to be the last reported sending rate, which could be smaller than its bandwidth if a link experienced an increase in signal to noise ratio due to environmental conditions. Please note that it is possible that the utilization of a link drops more than 30% even if its rate does not change. This can happen either when a flow terminates, reduces its sending rate, or when another link experiences capacity degradation which reduced the sending rate of a flow that passes multiple links including the degraded link. In this case, the delay measurement will return a value that is less than $10x$ since the link does not experience congestion. Thus, we will not update its *current maximum* and leave it as is. Moreover, links with degraded bandwidth may recover to full performance when system conditions improve (e.g., melting of ice on antenna due to increased temperature) we implemented a timer for *current maximum* rate to forget bandwidth limitations after being degraded such that we can reuse these links again and discover their new rate. While the timer can be set to values in the order of minutes or hours, we set it to 15 seconds in our experiments to demonstrate the functionality on a smaller time scale.

V. Experimental Results

We compare the performance of *BACAR* against two approaches shortest path and congestion-aware algorithms. The shortest path algorithm shows the performance of traditional decentralized route selection techniques such as OSPF. The congestion-aware solution, on the other hand, represents the performance of state-of-the-art solutions for SDN-based traffic engineering as existing work in this domain mainly focused on mitigating congestion in wired networks with static link capacity. Upon the detection of congestion (i.e., when a link utilization reaches above a certain threshold), the congestion-aware routing employs a similar route selection algorithm as *BACAR* , but assumes link capacity is fixed.

Figure 6 shows the experimentation topology created in Mininet and inspired by UNR wide-area hybrid network as illustrated in Figure 1. We created 16 TCP flows between the *Hosts* and the *Servers* with randomly chosen rates in $400-700$ Mbps range, and randomly chosen duration between $30-60$ seconds. All links have 1 Gbps bandwidth, but we reduce the capacity of some links while transfers are running to represent wireless link performance degradation changes due to environmental conditions.

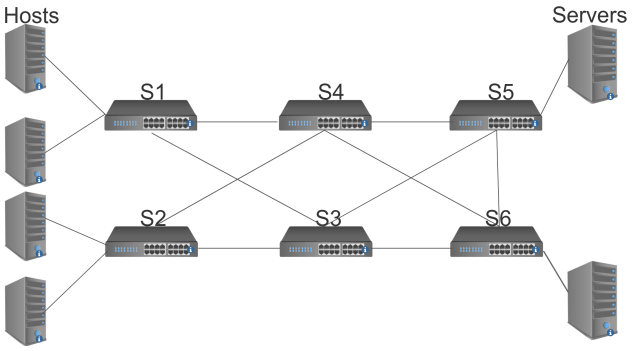

Fig. 6: The network topology used in the evaluations.

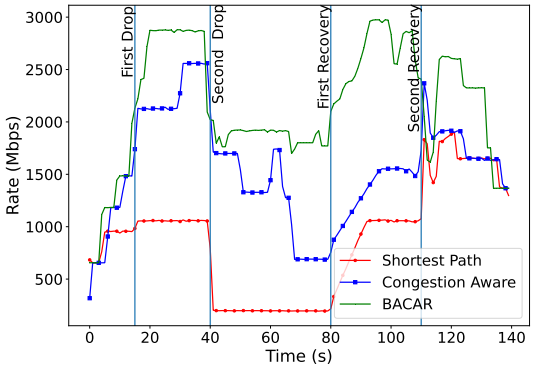

Fig. 7: Performance evaluation of *BACAR* in comparison to the shortest path and congestion-aware routing algorithms. While the shortest path solution falls short to detect and avoid congestion, the congestion-aware routing fails to notice bandwidth fluctuations. Thus, *BACAR* outperforms both approaches and increases network utilization significantly.

Figure 7 shows overall network utilization using different routing approaches under link capacity changes. At the beginning of the experiment, all three approaches yield similar performance. However, as more transfers join, the shortest path algorithm leads to congestion as multiple flows share the same bottleneck link. Congestion-Aware routing and *BACAR* , on the other hand, can assign new flows to underutilized links to increase the bandwidth of flows as well as to maximize network utilization. At $t = 15s$, we reduce the bandwidth of the link between $S4$ and $S6$ from 1 Gbps to 100 Mbps, which causes Congestion-Aware to suffer as it still assumes that link bandwidth is fixed and causes congestion on that link. On the other hand, *BACAR* detects link bandwidth change immediately and schedules delay measurement probes to calculate the delay for each link. It then notices that the link between $S4$ and $S6$ is fully utilized whereas its adjacent links (sharing the same path) are not. Thus, it assigns the current traffic rate on link $S4$ and $S6$ to be its current bandwidth, *current capacity* and saves it into *Link Utilization Store*. Finally, it reschedules some of the existing flows to alternate routes to alleviate the

congestion on the $S4 - S6$ link and lead to an increase in the overall network utilization.

When the capacity of the link between $S3$ and $S6$ drops from 1 Gbps to 100 Mbps at $t = 40s$, both Congestion-Aware and *BACAR* are affected negatively as there is no backup link with higher capacity to move traffic. Yet, the impact of bandwidth drop is more severe for Congestion-Aware as it still assigns more flows to these low bandwidth links than they can handle. When the first link recovers to its full speed at $t = 80$, the traffic rate on it increases more than 30%, triggering the *Path Selection Algorithm* to reconsider this link for congested links. The effect of this recovery is obvious at $t = 80s$ as the network utilization enhanced significantly for *BACAR* , while other algorithms attain slightly better performance. Finally, at $t = 110s$ and when the second link recovers to its original bandwidth, all algorithms experience improvements in overall utilization, but the largest gain is only observed by *BACAR* since its *Path Selection Algorithm* is triggered again to make use of the spare capacity on the second link.

VI. Conclusion

Wide-area hybrid networks are increasingly used to deploy sensors to remote areas to collect data. Harsh environmental conditions in these hard-to-access locations cause significant fluctuations in the capacity of wireless links, adversely affecting the performance of flows. While there have been alternate routes to take, existing rigid routing solutions impede the use of these alternate routes in the event of bandwidth fluctuations. While Software Defined Network is proposed to facilitate network management and increase the quality of service for applications, existing solutions to utilize SDN mainly focused on scenarios in which link capacity is fixed, thus they are unable to provide a solution to bandwidth fluctuation issues faced in wide-area hybrid networks. In this paper, we present Bandwidth and Congestion Aware Routing (*BACAR*) to detect bandwidth changes and incorporate them into the route selection algorithm along with congestion information. Experimental results show that *BACAR* improves the network utilization significantly in the presence of bandwidth fluctuation in comparison to state-of-the-art congestion-aware solutions. It does this by detecting link bandwidth changes and rerouting some flows from degraded links to others.

Acknowledgement

The work in this study was supported in part by the NSF grant #2019164.

References

[1] S. Song, J. Lee, K. Son, H. Jung, and J. Lee, "A congestion avoidance algorithm in sdn environment," in *2016 International Conference on Information Networking (ICOIN)*, 2016, pp. 420–423.

[2] 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

[3] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," ser. NSDI'10. USA: USENIX Association, 2010, p. 19.

[4] Y. Lu, Z. Ling, S. Zhu, and L. Tang, "Sdtcp: Towards datacenter tcp congestion control with sdn for iot applications," *Sensors*, vol. 17, no. 1, 2017. [Online]. Available: https://www.mdpi.com/1424-8220/17/1/109

[5] M. Ghobadi, S. H. Yeganeh, and Y. Ganjali, "Rethinking end-to-end congestion control in software-defined networks," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XI. New York, NY, USA: Association for Computing Machinery, 2012, p. 61–66. [Online]. Available: https://doi.org/10.1145/2390231.2390242

[6] J. Gruen, M. Karl, and T. Herfet, "Network supported congestion avoidance in software-defined networks," in *2013 19th IEEE International Conference on Networks (ICON)*, 2013, pp. 1–6.

[7] H. A. Pirzada, M. R. Mahboob, and I. A. Qazi, "Esdn: Rethinking datacenter transports using end-host sdn controllers," ser. SIGCOMM '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 605–606. [Online]. Available: https://doi.org/10.1145/2785956.2790022

[8] A. M. Abdelmoniem and B. Bensaou, "Sdn-based incast congestion control framework for data centers: Implementation and evaluation," *CSE Dept, HKUST, Tech. Rep. HKUST-CS16-01*, 2016.

[9] H. Haile, K.-J. Grinnemo, S. Ferlin, P. Hurtig, and A. Brunstrom, "End-to-end congestion control approaches for high throughput and low delay in 4g/5g cellular networks," *Computer Networks*, vol. 186, p. 107692, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128620312974

[10] S. Jamali, A. Badirzadeh, and M. S. Siapoush, "On the use of the genetic programming for balanced load distribution in software-defined networks," *Digital Communications and Networks*, vol. 5, no. 4, pp. 288–296, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S235286481830261X

[11] C. N. Sminesh, E. G. M. Kanaga, and K. Ranjitha, "A proactive flow admission and re-routing scheme for load balancing and mitigation of congestion propagation in sdn data plane," *ArXiv*, vol. abs/1812.02474, 2018.

[12] R. Kanagevlu and K. M. M. Aung, "Sdn controlled local re-routing to reduce congestion in cloud data center," in *2015 International Conference on Cloud Computing Research and Innovation (ICCCRI)*. IEEE, 2015, pp. 80–88.

[13] M.-T. Kao, B.-X. Huang, S.-J. Kao, and H.-W. Tseng, "An effective routing mechanism for link congestion avoidance in software-defined networking," *2016 International Computer Symposium (ICS)*, pp. 154–158, 2016.

[14] S. M. Park, S. Ju, and J. Lee, "Efficient routing for traffic offloading in software-defined network," *Procedia Computer Science*, vol. 34, pp. 674–679, 2014, the 9th International Conference on Future Networks and Communications (FNC'14)/The 11th International Conference on Mobile Systems and Pervasive Computing (MobiSPC'14)/Affiliated Workshops. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S187705091400951X

[15] S. Attarha, K. Haji Hosseiny, G. Mirjalily, and K. Mizanian, "A load balanced congestion aware routing mechanism for software defined networks," in *2017 Iranian Conference on Electrical Engineering (ICEE)*, 2017, pp. 2206–2210.

[16] 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.

[17] F. Pakzad, M. Portmann, and J. Hayward, "Link capacity estimation in wireless software defined networks," in *2015 International Telecommunication Networks and Applications Conference (ITNAC)*. IEEE, 2015, pp. 208–213.

[18] A. Al-Najjar, F. Pakzad, S. Layeghy, and M. Portmann, "Link capacity estimation in sdn-based end-hosts," in *2016 10th International Conference on Signal Processing and Communication Systems (ICSPCS)*. IEEE, 2016, pp. 1–8.

[19] 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.