

TCP Congestion Avoidance in Data Centres using Reinforcement Learning

Ali Hassan

Faculty of Business and Information Technology
University of Ontario Institute of Technology
 Oshawa, Canada
 ali.karwani@ontariotechu.net

Shahram Shah Heydari

Faculty of Business and Information Technology
University of Ontario Institute of Technology
 Oshawa, Canada
 Shahram.Heydari@ontariotechu.ca

Abstract—TCP, a transport layer protocol which ensures the reliable delivery of information on the network, is the basis of Internet connectivity, with 85% of the worlds Internet traffic being TCP based. TCP however, is slow to adapt to changes in the network, drastically reducing the throughput at the first sign of possible congestion, thereby preventing rapid restoration of the throughput. Mitigating this problem has been a very active area of research, as, until recently, the idea of using Artificial Intelligence (AI) in this space was relatively limited. Recently, Reinforcement Learning (RL), a form of AI, has been explored in the networking space, and in enhancing the performance of TCP, this paper aims to expand the use of RL for TCP (TCP-RL) in a software-defined data centre for the purpose of network congestion avoidance based on host-based TCP metrics. We demonstrate that our proposed approach is able to significantly reduce the impact of congestion on the end-to-end network throughput within the data centre.

Index Terms—Reinforcement Learning, OpenAI Gym, Software-Defined Networking (SDN), TCP Congestion Avoidance

I. INTRODUCTION

Datacenter network traffic has recently become a topic of interest due to its special characteristics and requirements, in particular with regard to the traffic moving within the datacenter (east/west flow) as opposed to inbound/outbound or north-south flow. As mentioned in [1], Datacenter traffic is quite different from Wide-Area Network (WAN) and the same methods of mitigating congestion do not apply because data-center networks are high bandwidth, low latency environments, while WAN traffic is typically low bandwidth, high latency. In the datacenter, TCP operates at a much higher throughput as the situation allows for it, meaning that once congestion is detected, throughput is decreased at a higher rate than in WAN environments.

TCP is natively built to handle congestion; however, this is done through controlling the traffic it sends while in a congested state, as network conditions that attribute to congestion over the network are not controllable by the Transport layer. Some mechanisms to deal with congestion include slow-start, where the network is slowly probed with data so to not overwhelm the network, however in doing so the full capacity of the network is not used. This method is used when data transmission is initially started and when congestion (e.g.

packet loss) is detected. Some causes for congestion include that packets arrive faster than the router can forward it, either due to hardware restraints of the device, or due to a congested link that has to be used to forward traffic through. Routing algorithms such as OSPF do not consider available bandwidth on the link, so traffic will continually be routed over a link either because that is the only path available, or because the specific flow has been hashed to take a particular link. But changes made on the network layer can directly impact TCP performance, which will affect user experience. To add on, if a queue on the router overflows because it is still waiting to send traffic over the link, the packets will be dropped or severely delayed resulting in a negative user experience.

In a software-defined network, network functions can be controlled from one or more central location(s) that are not on the data path. This would allow deployment of cross-layer software-based modules with access to global network information to detect a network scenario (e.g. congestion) and allow re-routing to be done, based on the TCP parameters of a particular flow. Such modules can employ a variety of techniques, including artificial intelligence (AI).

AI is one of the fastest growing areas of research, and its applications have been tried and tested in areas such as stock markets, real estate, big data, and more. However, adoption in the area of networking has been slow due to the ever-changing nature of network traffic, making it very hard to predict and distinguish between normal and abnormal traffic conditions. Also, due to the characteristics of network traffic, it is not possible to have realistic and accurate traffic datasets which can be used for training in every scenario, as the AI will learn the intricacies of the training data, but the training models will not translate well to other situations, especially into production networks. The main reason is that as traffic patterns are unpredictable, network traffic from one day to another day can vary greatly which may inaccurately classify traffic as problematic when that may not be the case.

Among AI techniques, Reinforcement Learning (RL), due to its operational characteristics and its continuous interactions with its environment to learn via trial-and-error, can be a viable option in network control and management, especially due to its ability to respond to a changing environment [2]

Comparing this to Supervised and Unsupervised Learning,

the algorithm must be fully trained and then applied to a dataset for testing. If the algorithm is not able to respond as predicted, it would require further re-training of the algorithm, which is not scalable for networking due to the continuous learning required. [3] Such techniques can also be used for translating intents into network policies, which can be done in conjunction with AI models to monitor performance and detect network problems [4]

In this paper we will discuss our contribution to the area of TCP congestion using RL (TCP-RL), by proposing a routing-based mitigation method which preserves throughput by avoiding congestion paths rather than merely detecting congestion as TCP-RL does. We call our approach TCP Congestion Avoidance using RL (TCP-CA/RL) to distinguish it from the former. We will consider the performance of TCP-CA/RL as satisfactory if we reach similar or better transfer times of data as other approaches.

This paper is broken down into three further sections Section II which discusses work in this area that closely related to TCP congestion control, TCP-CA/RL in Section III which will describe the framework of our algorithm, our results and analysis in Section IV, and the conclusion and further areas of research in Section V.

II. RELATED WORK

Deep TCP-RL for congestion control has been explored in [5], employing the Aurora algorithm against TCP to increase the throughput by trying to distinguish between packet loss and congestion, utilizing Neural Networks. This was done by developing a framework utilizing Open AI Gym Environment. The two most important parameters which were considered was history length - how much in the past the agent would go based on data retention, and discount factor - which showed that with a higher discount factors, good results are achieved quickly. Simulation results using Mininet showed that Aurora was comparable to or outperformed many state of the art congestion control algorithms such as BBR, PCC-Vivace, RemyCC, and Copa. The reward function utilized was a linear function that rewarded throughput and penalized loss and latency with coefficients used to force the model to balance throughput and latency. In the analysis, it was found that TCP Cubic achieves higher throughput at the expense of higher latency, however it is plagued with bad performance in many different scenarios [6]. In [7], the best Initial Window value and Congestion Control scheme is determined to be used by RL, based on network characteristics. In [8] both Supervised Learning and RL is used, with received ACKs, state (e.g. RTT) as the RL feature input.

In [9], a framework was also presented utilizing Long-Short Term Memory (LSTM) in a Deep RL (DRL) framework for MPTCP, in a physical network, which provides benefit to user traffic by utilizing multiple network connectivity (e.g. Ethernet, Wi-Fi, and LTE) to load balance a single TCP flow. The results of this research shows that the use of this type of framework outperforms traditional MPTCP congestion control algorithms. While this work does not focus on data centres, it

shows the promise of TCP-RL and further confirms the use of Q-Learning as the datacenter environment is more predictable than utilizing multiple network connectivity points, but still very critical to increasing throughput.

Other works have been done to optimize TCP for data center networking [10]. In this paper, D-TCP estimates the variable bandwidth, and based on end-host information, quickly adapts to changing environments where traditional TCP is not sufficient for the modern era of network traffic demands.

Work in Omniscient TCP [11] further demonstrates the inadequacy of TCP to operate in low-latency, high throughput environments. OTCP in particular achieves a higher level of performance than standard TCP by configuring TCP congestion parameters on a route by route basis based on network characteristics.

Lastly in [12], which utilizes the TCP-RL algorithm, it was proven that QTCP operated better than TCP NewReno, further giving momentum to our research.

The idea that is being proposed in this paper, has some common aspects to the aforementioned paper, however we will be addressing a slightly different problem. In the data center, although there are equal-cost multipath (ECMP) routing enabled which allow for multiple paths to be used to send traffic simultaneously, there is still an inefficiency there as the ECMP path selected is based on a hash of source/destination IP/port. This means although there are multiple paths, traffic may continuously take a congested path if that is the path which is selected by ECMP routing. In our research, we are utilizing OpenAI Gym for RL, which will take input from TCP statistics in a Software-Defined Network to make decisions on routing over ECMP paths.

III. TCP-CA/RL

Reinforcement Learning has recently gained a lot of momentum due to its applicability in learning from a continuously changing environment by continuously modifying the state of the environment, which returns a positive or negative reward.

Once we know the current state, the algorithm must choose an action that may lead to a win based on a plan (or in the case of TCP-RL – a policy).

This action/move will result in a new scenario/state and a reward will be given based on the quality of the move. This reward will be fed into an equation, which will be stored in a Q-table acting as a cheat-sheet of all possible options and the greatest Q-value for each state. [2]

The main Q-learning formula for the RL algorithm is given in Equation 1. The η parameter is used to dictate how much of the old value we retain. A η of 0 will mean nothing new will be learnt. A η of 1 will mean the old value will be completely discarded. γ is the discount factor and quantifies how much importance we give for future rewards. If γ is closer to zero, the agent will tend to consider only immediate rewards. If γ is closer to one, the agent will consider future rewards with greater weight, willing to delay the reward.

$$\text{New}Q(\text{state}, \text{action}) = (1 - \eta) * Q(s, a) + \eta * \text{reward} + \gamma * \max_{a'} Q'(s', a') \quad (1)$$

In terms of the TCP-CA/RL algorithm itself, there is an element of randomness. This is further explained in Algorithm 1.

Algorithm 1 TCP-CA/RL

```

E = 1,  $\alpha$  = 15,  $E_{decay}$  = 0.3,  $r$  = iteration number
 $E\_THRESHOLD$  = 20000
Start priority flow by randomly selecting a path
Start background flow by randomly selecting a path
while  $\text{reward} \neq \text{TERMINATE}$  do
  for  $k \leftarrow 1$  to  $\alpha$  (this is considered one step) do
     $\text{random\_decimal} = \text{RAND } 0,1$ 
    if  $\text{random\_decimal} > E$  then
      RL chooses the next path
    else if  $\text{random\_decimal} < E$  then
      choose the next path randomly
    end if
    Change the path for the priority flow
     $\text{tcp\_bytes\_acked} = \text{getTCPBytes}()$ , if no value returned, then return TERMINATE
    if  $\text{tcp\_bytes\_acked} == \text{TERMINATE}$  then
      Break
    else if  $\text{tcp\_bytes\_acked}(r, r-1) > \text{tcp\_bytes\_acked}(r-1, r-2)$  or if  $\delta(r, r-1) < E\_THRESHOLD$  then
      Return positive reward
    else
      Return negative reward
    end if
    Calculate Q_Table value based on Equation 1
  end for
   $E = E_{decay}$ 
end while

```

Once the reward is calculated, it is fed into Equation (1) which is used to calculate how 'good' of a decision was made by the RL algorithm. The value that is calculated is known as the Q-Table value for that particular state/action transition (e.g. going from path x to path y) which is stored in a matrix, with the best decision having the highest value. After the for loop completes iteration as defined by α , then E is decreased by the variable $E_THRESHOLD$ so in the next iteration, there is a greater chance that TCP-CA/RL will select the path, eventually having the algorithm adopt a greedy model, selecting the Q-value at each state transition that has yielded the highest reward based on past iterations.

To relate back to TCP, by calculating the δ of the tcp_bytes_acked and received between two steps, we are able to make a judgment on whether TCP is working as it should be, if the δ of the amount of data acknowledged is increasing, or if there could be possible congestion on the network as

the tcp_bytes_acked are decreasing. With this information provided by TCP, we can pinpoint the location of congestion, and re-route traffic thereby mitigating congestion.

IV. RESULTS/EVALUATION

We used Mininet [13] to simulate an SDN environment, and the framework defined in OpenAI Gym [14] for RL. The topology for our experiment is shown in Figure 1. In this environment, three hosts (H1, H2, H3), connect to three leaf switches respectively. The leaf switches upstream, connect to 4 spine switches respectively. The leaf switches upstream, connect to 4 spine switches upstream, using the spine and leaf topology which is the latest trend inside the data-center allowing for east-to-west traffic to move efficiently. From an expansion point of view, new leaf switches are connected to the spine as the path length from any host is always just three hops so this environment can grow without disruption to existing traffic, making it ideal for the core data center.

Background traffic was generated on one of the four uplinks to simulate congestion. As the links connected in Mininet are set to unlimited bandwidth by default, the bottleneck, or congestion which will be artificially induced will be on the uplinks.

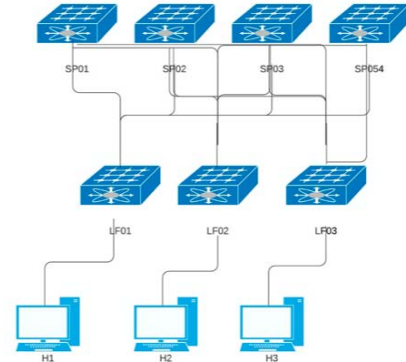


Fig. 1. Network Topology

H1 runs three different streams – priority traffic, background traffic, and a server function on port 123 (which is used to send TCP statistics to H2, which runs the RL-TCP algorithm, taking input of the TCP statistics from H1).

To conduct testing, the TCP-CA/RL algorithm will continue operating until 100 Megabytes of data have been transferred via iperf, at which point the TCP-CA/RL algorithm will terminate. To simulate congestion, all uplinks between LF01 to the upstream spine switches were limited to 1 Mbps.

Through manual analysis, it was seen that when the background traffic and the priority traffic are taking the same link, the throughput as per the information for iperf, was significantly lower which was expected. The algorithm provides a 20,000 byte buffer for the δ as this amount of data can be considered negligible from a data transfer point of view, and as a result a δ less than this should not be counted as a negative reward. This is further described in

Table 1, outlining background flow, b , and priority flow, p . Furthermore, in Table 1 we outline a baseline of the environment and the time taken to transfer 100 Megabytes of data. Another key data point to note is when random paths were used to send traffic for the priority traffic, every five seconds, we calculated it took an average 17 and a half minutes, this will be compared to the time taken by TCP-CA/RL below.

Table 1 - Bandwidth and Time Baseline

	b & p = same path	b & p = different path
BW of priority flow	250 Kbps	980 Kbps
Transfer Time	59:00	14:15

We conducted tests with all variable combinations for α , E_{decay} , γ , and η , the breakdown of the variable values is illustrated in Table 2.

Table 2 - Variable Combination

γ	E_{decay}	α	η
0.25, 0.5, 0.75	0.1, 0.3, 0.5	20, 30, 40, 50, 60, 70	0.25, 0.5, 0.75

After conducting the tests, it was found that the least amount of time taken to transfer 100 Megabytes of data was 14 minutes and 15 seconds, and the greatest amount of time was 18 minutes and 50 seconds.

In the results in Table 3, we can see that on average our algorithm takes the same amount of time as if there was no congestion by effectively not using the path that was congested.

Table 3 - Top Ten Result Analysis - 4 Paths with 1 Congested Path

α	γ	η	E_{decay}	Transfer Time
40	0.25	0.5	0.3	14:13
30	0.75	0.75	0.5	14:23
70	0.25	0.5	0.1	14:41
20	0.25	0.25	0.5	14:41
30	0.5	0.25	0.3	14:50
20	0.75	0.25	0.5	14:50
40	0.75	0.75	0.5	14:57
30	0.5	0.25	0.5	14:59
20	0.25	0.5	0.5	15:03
20	0.5	0.25	0.5	15:07

The results in table 1 indicate that γ and η do not play a major role in determining the transfer time as when these values are modified, while the α and $E_{THRESHOLD}$ values are kept similar, the results are relatively the same. To

further confirm this theory, and to establish a pattern, the top ten tests were repeated with varying values for γ and η , with the hypothesis in mind that the transfer time directly relates to the α and the $E_{THRESHOLD}$. The results were very close to the top ten results shown earlier, with an average difference of between 45-75 seconds in the transfer time, which is expected considering how even when a file is being downloaded in production network, or even from a user's PC at home, the time taken to transfer the same amount of data will vary.

In our case each test was repeated 40 times with the same values, as well as varying values. As the results were the same in all cases, we can say with confidence that although the η and γ is important in TCP-CA/RL when it comes to gaming, the same cannot be said about networking. In networking, what matters most is the different number of paths tried in a short amount of time (α) and $E_{THRESHOLD}$ so we eventually converge into a greedy algorithm, specifically making decisions that will yield the greatest reward.

In this environment, there are 16 possible state changes (e.g. going from taking path 1, to path 2, or from path 1 to path 3) and in the top result we can see why the results which closely resembled the data show in Table 3, in terms of transfer time, was with lower amounts of α as using this value, TCP-CA/RL will learn about the network a limited amount of time. After this point, E will be decayed and then RL would start choosing some paths, while still exploring others, eventually having E decremented to 0, and any value chosen between 0 and 1 would be greater than ϵ , thereby always choosing a value with the highest reward yield.

To compare, the highest transfer time witnessed was with for $\alpha = 70$ and $E_{THRESHOLD} = 0.1$. in this scenario, it would take TCP-CA/RL 700 path changes to converge into a purely greedy model where TCP-CA/RL would make all the decisions, which is both inefficient not needed as the data transfer would have completed well before 700 path changes were completed.

Lastly, the comparison for the I/O data transfer can be seen in Figure 2, 3, and 4 where we are visualizing a single TCP stream of 100 Megabytes of data transfer via iperf using RL, Random path selection every five seconds, and using standard TCP congestion control when both priority and background flows are on the same path, (which is provided to provide a baseline), respectively. Comparing specifically Figure 2 and Figure 3, we can see that the time taken to transfer 100 Megabytes of data varies by approximately 60 seconds. Furthermore, on average the number of packets being transferred is hovering between 80 and 100 packets, except for the dip down to 20 when both priority and congested flow paths are on the same path, which is significantly greater in Figure 3 which is purely random. The drop in packet transfer significantly lessens with time as well, which shows that the RL algorithm is able to learn which path is congested and actively avoid that, further proving our algorithm. In this case, the RL algorithm is able to correctly choose the path 750 seconds into the data transfer, which is completed by around

1000 seconds, meaning if the data transfer was for a longer period of time, we would have seen far superior transfer times compared to random path selection.

Fig. 2. Reinforcement Learning - TCP-CA/RL

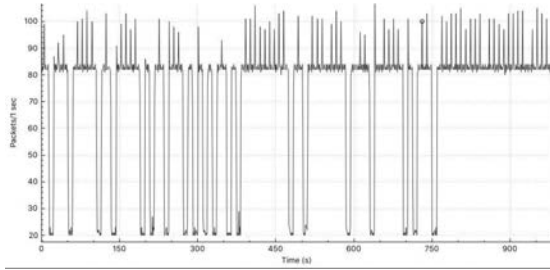


Fig. 3. Random Path Changes

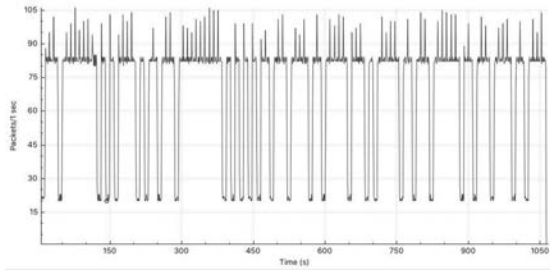
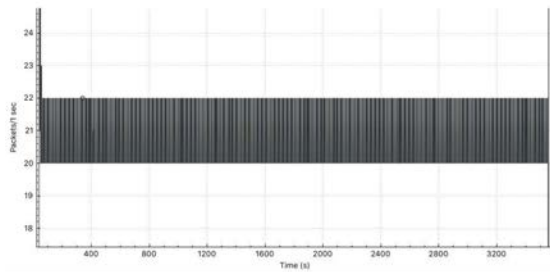


Fig. 4. TCP Congestion Control



To further analyze the algorithm, we conducted two additional tests, while keeping the topology the same. In the first test, congestion was generated on three of the four uplinks to test how well the algorithm would be able to choose the un-congested path. In this case, just like the testing conducted earlier, the TCP-CA/RL algorithm reset itself after transferring 100 MB of data, so any information learnt from the previous topology was not carried over. The top ten results in this test are shown in Table 5. From the transfer times, we can see that it takes on average a minute and a half more to transfer data when compared to Table 3, which is expected given that prior there was a 25% chance of taking a congested path, but in this scenario there was a 75% change of selecting a path which was already congested so TCP-CA/RL needed a little more time to learn the environment.

Table 5 - Top Ten Result Analysis - 4 Paths with 3 Congested Path

α	γ	η	E_{decay}	Transfer Time
70	0.75	0.75	0.3	15:02
40	0.25	0.5	0.3	15:08
40	0.25	0.25	0.5	15:30
60	0.5	0.5	0.1	16:11
60	0.25	0.75	0.5	16:29
60	0.75	0.5	0.5	16:29
60	0.5	0.25	0.5	16:38
60	0.25	0.25	0.3	16:47
60	0.25	0.5	0.5	16:47
60	0.75	0.25	0.3	16:56

In the second test, the scenario was kept the same, however the data learnt by TCP-CA/RL was not discarded after each data transfer. Instead, it was carried over. The top ten results of this data transfer is shown in Table 6. Although not shown in the results, the data transfer time for all tests conducted were very close to each other, as opposed to the first test where there were some instances where the data transfer time exceeded 25+ minutes. Furthermore, in this test, the values chosen to define the TCP-CA/RL do not make a major difference for the transfer time, as when the Algorithm starts making decisions, it uses the knowledge of all past iterations.

Table 6 - Top Ten Result Analysis - 4 Paths with 3 Congested Path - One TCP-CA/RL

α	γ	η	E_{decay}	Transfer Time
20	0.75	0.25	0.5	16:53
20	0.25	0.5	0.5	17:09
20	0.5	0.5	0.5	17:19
20	0.5	0.25	0.5	17:20
20	0.25	0.75	0.5	17:29
20	0.75	0.25	0.5	17:41
20	0.25	0.25	0.5	17:56
20	0.75	0.5	0.5	18:07
30	0.75	0.5	0.5	18:07
30	0.75	0.25	0.5	18:15

V. CONCLUSION AND FUTURE WORK

In this paper, we presented a TCP-CA/RL based solution of *mitigating* congestion caused by saturation at the network layer, by using statistics from end hosts to approximate congestion using RL in a Software-Defined environment and we were able to show that in our tests, we were completely able to minimize congestion caused by re-routing traffic in real-time. We also noticed the results were reproducible when run multiple times, with some variation which was expected due to the nature of our testing. Furthermore, we also confirmed that

the longer we have TCP-CA/RL learning about the topology and if the knowledge carries over from iteration to iteration, data transfer time is kept consistently low even in situations where 3 out of the 4 uplinks are congested. The next steps would be to test the algorithm in larger environments, with a greater amount of data exchange to analyze the response of the algorithm when run for a longer period of time. Theoretically, the longer the algorithm is run, the better understanding it will have of the environment so we should see an optimal transfer time as if there was no congestion to begin with.

For future work, it would be more practical to have a complex reward function returning multiple values as opposed to -1 or +1. In doing so, we will be able to theoretically determine the *rate* of congestion, and which links have the worst congestion based on rewards as currently congestion is binary. Furthermore, this environment specifically focuses on re-routing one TCP stream, which at the same time can be scaled which should be explored further. As this algorithm was engineered with the mindset of the core data center, this algorithm can run one TCP-CA/RL algorithm *per* leaf switch, and look at TCP statistics from all flows which are considered high priority that are leaving the switch. In doing so, re-routing will be done locally, whilst allowing for easy scaling.

REFERENCES

- [1] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of datacenter traffic: Measurements analysis," *Proceedings of ACM SIGCOMM IMC'09*, pp. 202–208, 01 2009.
- [2] R. S. Sutton and A. G. Barto, 1998.
- [3] A. Graves and M. A. Ranzato. Unsupervised deep learning - google deepmind facebook artificial intelligence neurips 2018. Youtube - The Artificial Intelligence Channel. [Online]. Available: <https://www.youtube.com/watch?v=rjZCjosEFpI&feature=youtu.be&t=137>
- [4] Cisco. (2020, Jan) Cisco intent-based networking. [Online]. Available: <https://www.cisco.com/c/en/us/solutions/intent-based-networking.html>
- [5] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, "A deep reinforcement learning perspective on internet congestion control," pp. 3050–3059, 2019.
- [6] M. Dong, Q. Li, D. Zarchy, B. Godfrey, and M. Schapira, "Pcc: Re-architecting congestion control for consistent high performance," 09 2014.
- [7] X. Nie, Y. Zhao, Z. Li, G. Chen, K. Sui, J. Zhang, Z. Ye, and D. Pei, "Dynamic tcp initial windows and congestion control schemes through reinforcement learning," *IEEE Journal on Selected Areas in Communications*, vol. PP, pp. 1–1, 03 2019.
- [8] Y. Kong, H. Zang, and X. Ma, "Improving tcp congestion control with machine intelligence," pp. 60–66, 08 2018.
- [9] C. Y. Y. W. Xu, J. Tang and G. Xue, "Experience-driven congestion control: When multi-path tcp meets deep reinforcement learning," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1325–1336, 2003.
- [10] M. Kanagarathinam, S. Singh, S. Irlanki, A. Roy, and N. Saxena, "D-tcp: Dynamic tcp congestion control algorithm for next generation mobile networks," 01 2018.
- [11] S. Jouet, C. Perkins, and D. Pezaros, "Otcp: Sdn-managed congestion control for data center networks," pp. 171–179, 04 2016.
- [12] W. Li, F. Zhou, K. R. Chowdhury, and W. Meleis, "Qtcp: Adaptive congestion control with reinforcement learning," *IEEE Transactions on Network Science and Engineering*, vol. 6, no. 3, pp. 445–458, 2019.
- [13] A. Chaurasia, S. Mishra, and S. Chinara, "Performance evaluation of software-defined wireless networks in it-sdn and mininet-wifi," pp. 315–319, 07 2019.
- [14] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 06 2016.

Ali Hassan Karwani is currently studying toward his Masters degree in Computer Science at the University of Ontario Institute of Technology. He received his Bachelor's degree with honours in Information Technology from the same university in 2018. His area of research is the applications of Artificial Intelligence (AI) in network management.

Shahram Shah Heydari is an Associate Professor at University of Ontario Institute of Technology, Ontario, Canada. Prior to joining the university in 2007, he was a System Engineer and Member of Scientific Staff at Nortel Networks where he worked on element management in ultra high speed IP/MPLS routers, performance modeling of automatically switched optical networks (ASON), and proprietary voice-over-IP transport control protocols. He received his B.Sc. and M.Sc. degrees in Electronic Engineering from Sharif University of Technology (Iran), M.A.Sc. degree from Concordia University, Montreal, and Ph.D. degree from University of Ottawa, Canada in 2007. His main research interests include network design and planning, software-defined networking, critical infrastructure resilience and security, and quality of service.