This is a peer-reviewed, author's accepted manuscript of the following research article:
as, H. (2023). Intelligent subflow steering in MPTCP-based hybrid Wi-Fi and LiFi networks using model-augmented DRL. In GLOBECOM 2022 - 2022 IEEE Global Communi
(GLOBECOM 2022 - 2022 IEEE Global Communications Conference). IEEE.
https://doi.org/10.1109/globecom48099.2022.10001388

Intelligent Subflow Steering in MPTCP-based Hybrid Wi-Fi and LiFi Networks Using Model-Augmented DRL

Ardimas Andi Purwita[1], Anil Yesilkaya[2], and Harald Haas[3]
[1]Computer Science Department, Faculty of Computing and Media, Bina Nusantara University
[2]MathWorks Glasgow, Scotland
[3]LiFi R&D Centre, Department of Electronic & Electrical Engineering, The University of Strathclyde
E-mail: ardimas.purwita@binus.edu, ayesilka@mathworks.com, harald.haas@strath.ac.uk

***Abstract*—A hybrid Wi-Fi and light fidelity (LiFi) network combines the best of two worlds with the ubiquitous coverage of Wi-Fi and the high peak data rate of LiFi as the radio spectrum does not interfere with the light spectrum. This hybrid network might be realized by using multipath TCP (MPTCP), where Wi-Fi and LiFi paths can be simultaneously employed to potentially boost the total throughput of Wi-Fi while increasing the resilience towards network failure of LiFi due to, for example, blockage. However, naively implementing MPTCP in a hybrid Wi-Fi and LiFi network can yield an unexpected result, such as a lower throughput compared to the single-path TCP due to a Head-of-Line delay during the slow start phase of the TCP congestion control. Even though this problem can be avoided by improving the existing flow c ontrol o r c ongestion control of TCP, these solutions still lack intelligent decision making that can improve the adaptability of MPTCP. Therefore, in this paper, we propose a model-augmented deep reinforcement learning (DRL) approach to intelligently steer MPTCP subflows (i.e., TCP connections) by using a close-to-reality scenario emulated by considering random orientation, random blockage, and random mobility of Wi-Fi-and-LiFi-enabled mobile devices. As a result, we will show later that a performance gain can be achieved compared to the state-of-the-art while maintaining ease implementation to existing MPTCP implementations.**

I. Introduction

According to [1], Cogalan and Haas predict that the entire radio frequency (RF) spectrum will be fully utilized by 2035. Due to the fact that the light spectrum is unlicensed and does not interfere with the RF spectrum, optical wireless communications (OWC), e.g., light fidelity (LiFi), are a good candidate to complement and offload t raffic fr om RF communications [2]. LiFi supports high-speed, bidirectional, and multiuser communications [3]. Even though LiFi can support very high peak data rates, LiFi suffers from high variations in channel quality caused by random orientations and random mobility of LiFi-enabled devices [4]. Therefore, it would be ideal if both RF communications and LiFi are combined in order to support combinations of use cases of future communications, such as reliable, mobile broadband communications [5].

In this paper, we focus on the integration of Wi-Fi and LiFi due to the following reasons. According to [6], more than half of global mobile traffic and internet protocol (IP) traffic are offloaded and carried by Wi-Fi. In addition, based on the ongoing LiFi standardization, i.e., IEEE 802.11bb[1], LiFi can use the same implementations of the medium access control and physical layers from the Wi-Fi, except that the LiFi analog front-ends adhere to specifications defined in IEEE 802.11bb documents. Consequently, the integration of Wi-Fi and LiFi can be done in a high layer of the TCP/IP protocol stack, such as in the transport layer, by using multipath TCP (MPTCP) [7], where multiple paths can be utilized in parallel to increase the total throughput as well as improve resilience towards networks failures. The integration is straightforward as Wi-Fi and LiFi interfaces are recognized as different paths having different IP addresses from the perspective of a transport layer protocol.

In the MPTCP implementation in the Linux kernel [8], TCP connections are referred to as subflows. According to [9, Figure 3], applying existing MPTCP congestion controls, such as balanced linked adaptation algorithm (BALIA) [10], in a hybrid of a high speed, low coverage communication (mmWave) and a lower speed, wider coverage communication (LTE) can lead to lower throughput compared to that of a single-path TCP. The main cause of this phenomenon is explained in [11]. That is, the congestion window from the mmWave subflow is overshot during an outage and causes an acute packet reordering problem, which further results in a Head-of-Line blocking problem. It is suggested in [9] that a proper congestion control should be designed in order to harness the full potential of MPTCP. According to [12], the other problems (other than the out-of-order packet delivery) are the bufferbloat and bottleneck bandwidth. All these problems are eventually related to MPTCP congestion controls; therefore, many studies have been focussing on improving the congestion controls, see [12] and references therein.

A recent survey on the MPTCP congestion control as discussed in [12] categorizes two types of controls, i.e., traditional ones and machine learning (ML)-based ones. According to [12], [13], the traditional congestion control algorithms mostly

[1]https://www.ieee802.org/11/Reports/tgbb_update.htm

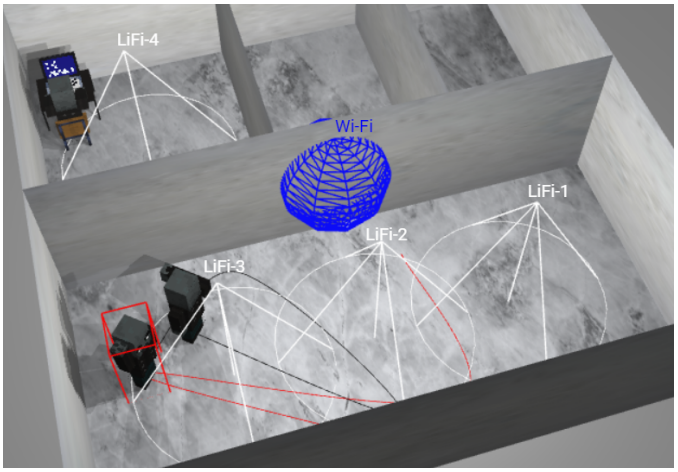

Fig. 1. A visualization of our emulator where human models are assumed to operate Wi-Fi and LiFi-enabled devices inside a room.

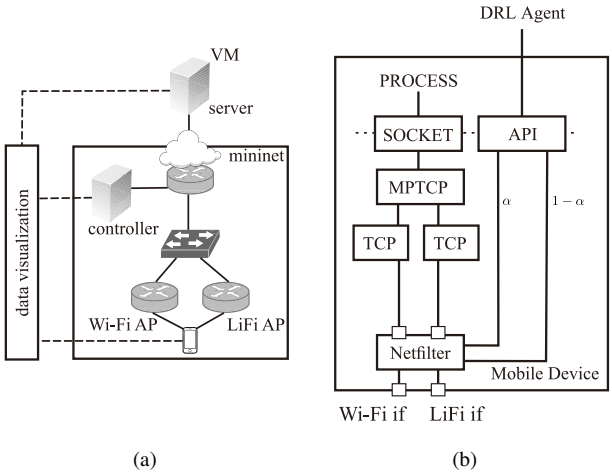

Fig. 2. (a) Our emulator diagram and (b) subflow-steering using Netfilter

lack adaptability and intelligent decision making as well as their sub-optimal performances. This is where machine learning works well as it can learn from past experience and balance exploration and exploitation to find an optimal solution. In this paper, we focus on a machine learning technique which takes a sequence of actions in order to optimize future rewards obtained from a dynamic environment. Specifically, we will emulate a scenario where there are Wi-Fi and LiFi-enabled mobile devices held by mobile users who move around inside a room that is covered by Wi-Fi and LiFi signals as depicted in Fig. 1. Our objective is to develop an intelligent system to maximize the total average throughput of all users. A well-known machine learning technique for this type of problem is reinforcement learning (RL). Specifically, we will use a combination of RL and deep learning, which is referred to as deep reinforcement learning (DRL).

In the context of the use of RL for a hybrid Wi-Fi and LiFi network, there are early works by Ahmad *et al.* that are presented in [14], [15]. They compare the performance of different non-ML algorithms with a RL algorithm in finding an optimal load balancing strategy to maximize a long-term system throughput while ensuring the required users fairness and satisfaction. It is shown that the RL algorithm can compete with an exhaustive search algorithm while having a significantly lower computational complexity. However, an on-policy-based RL algorithm, called the trust region policy optimization (TRPO), is still used in [14], [15], which is well known for its disadvantage in being trapped in local optima. Another study by Xu *et al.* in [16] shows an off-policy-based DRL (i.e., deep deterministic policy gradient (DDPG)), where an agent can learn a policy not only from the current agent's policy (observational policy), but also from behavioral policy, e.g., from past experience, for a hybrid network employing MPTCP. It is shown in [16] that the DRL-based MPTCP congestion control significantly outperforms the conventional MPTCP congestion controls.

Contributions: Compared to the previously mentioned studies, this paper introduces three new contributions. *First*, unlike [14], [15], we will develop an emulator for a hybrid Wi-Fi and LiFi network that is close to a real scenario, where random mobility, random orientation, random blockage, and the MPTCP are considered. *Second*, many proposals focus on directly modifying the MPTCP protocol, e.g., its congestion control by means of DRL as discussed in [16] and references in [12]. In this paper, we try a new approach by steering MPTCP subflows by means of intelligently adjusting load balancing coefficients in the Netfilter using a DRL. *Third*, compared to [16], we use a more state-of-the-art DRL algorithm called soft actor-critic (SAC) by [17], which performs significantly better compared to DDPG in terms of training efficiency. In addition, we augment the soft actor critic by using a model that can help predict a future state so that the MPTCP can act earlier, which is the main reason behind our performance advantage compared to [16].

The rest of this paper is organized as follows. In Section II, we will discuss our system model that includes a discussion on the emulator. In Section III, our proposed DRL architecture is presented. Our results and discussions are explained in Section IV. Lastly, we will conclude our paper in Section V.

II. System Model

We aim to implement a close-to-reality scenario where randomness factors, such as random oriented mobile devices, random blockage, and random mobility, are considered. In order to accommodate such a purpose, we use a reference model as described from the IEEE TGax in [18]. This reference model is used because the LiFi standard, i.e., IEEE 802.11bb, refers a lot to IEEE 802.11ax. Therefore, the reference model can be extended so that LiFi access points (APs) can be added. In this paper, we focus on using a 10 m × 10 m × 3 m indoor room with a Wi-Fi AP being deployed in the center of the room, and LiFi APs are uniformly placed inside the room.

In order to create a virtual network from our scenario, we use Mininet. Specifically, we use the Mininet-WiFi [19], which depends on Linux utilities such as *tc, wmediumd, hostapd,*

wpa_supplicant, to emulate a wireless medium and a Wi-Fi connectivity. For example, the utility *wmediumd* and a technique called link-to-system mapping [20] (where a packet error ratio can be estimated from the quality of the received signal, e.g., signal-to-interference-plus-noise ratio (SINR) or received signal strength (RSS)) enable us to emulate IEEE 802.11n. The main reason for choosing IEEE 802.11n in this paper is that even if we combine two wireless access technologies that have a contrast link capacity (or asymmetric links), we could still gain an advantage by using the proposed method. In addition to IEEE 802.11n, we emulate IEEE 802.11bb, i.e., the LiFi standard, by using the same technique, where we further refer it to as Mininet-WiFi-LiFi, which is described in [21]. Other detailed information regarding our random orientation, random blockage, and random mobility models are also described in [21, Chapter II]. In short, the random orientation model is based on our measurements reported in [22], and the random blockage and random mobility models are based on our LiFi channel simulator published in [23]. Our DRL agent later runs in a controller in our Mininet implementation that uses a Linux kernel implementation of MPTCP from [7]. Having implemented this emulator, we can mimic an MPTCP-enabled mobile device that can simultaneously connect to both a Wi-Fi AP and a LiFi AP. We also deploy a virtual machine that acts a server and MPTCP proxy. For demonstration purposes (which will be discussed in Section IV), we can monitor internal states of these devices and visualize them in real time over WebSocket connections. Fig. 2(a) illustrates our Mininet description.

Fig. 2(b) illustrates the proposed subflow steering diagram using the Netfilter [24]. A request coming from a running process is handled by a socket which further communicates to the MPTCP in the kernel space before being passed to multiple TCP connections. Then, the Netfilter, which can be used as a firewall or a network address translator, is used to filter segments from the TCP connections, which are also known as MPTCP subflows. Suppose α denotes a proportion of the total traffic that will pass the Wi-Fi interface, then the DRL agent aims to devise an optimal policy to dynamically adjust the values of α for all mobile devices. To summarize, Fig. 3 illustrates the system model from the perspective of the DRL agent. That is, the DRL agent interacts with an environment described by Fig. 2. Then, the environment returns rewards, e.g., throughput, and internal states of the environment, such as telemetry data. With the help of deep learning architectures, the DRL agent then returns back actions, such as the portion of traffic that should flow over LiFi connections. These interactions are performed over an application programming interface. In the next section, we will focus on the DRL agent and explain the proposed solution that uses our model-augmented DRL algorithm.

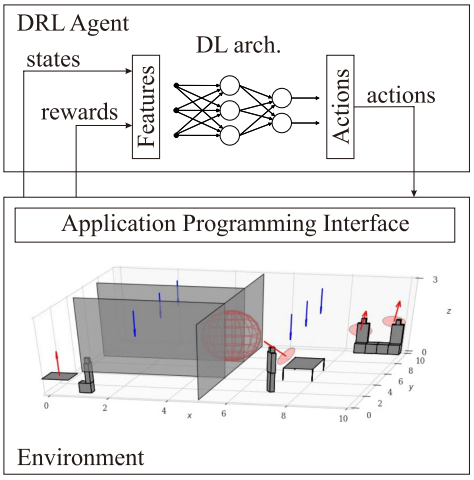

Fig. 3. DRL system model

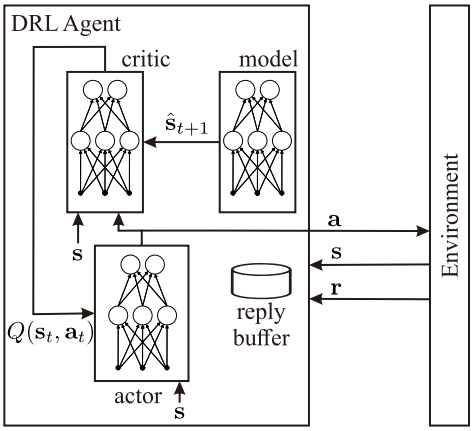

Fig. 4. Model-augmented soft actor-critic model

III. Proposed Solution

Fig. 4 depicts a high level view of our model-augmented SAC architecture. There are four important components in our architecture, i.e., a reply buffer and three parameterized networks, which are a model network, an actor network, and a critic network. The parameters for the networks, namely $\boldsymbol{\nu}$ (for the model network), $\boldsymbol{\theta}$ (for the actor network), and $\boldsymbol{\phi}$ (for the critic network), are subjects of our training later.

The reply buffer stores experiences as memories to enable an offline training for the networks, which further help the DRL agent to perform an exploration to find the optimal solution and avoid being trapped in a local optima. Our replay buffer is implemented as a fixed size buffer in the form of a circular buffer, where the oldest data is replaced with the newly obtained data. As for our sampling strategy, we follow a prioritized sampling that is proposed in [25]. We use the replay buffer to hold past information, such as actions that have already been taken (denoted by **a**), recorded internal states from the environment (denoted by **s**), and previously received rewards from the environment (denoted by **r**).

The state **s** is defined as a collection of congestion windows from all MPTCP subflows and the round trip times from all user interfaces. As for the rewards, we follow [16] to use the

sum of $\log g_{t,i}$, i.e.,:

$$r(t) = \sum_{i=1}^{K} \log g_{t,i}, \tag{1}$$

where $g_{t,i}$ is the goodput of the i^{th} MPTCP subflow at time t, and K is the total number of subflows. The DRL agent will later try to optimize the expected cumulative future rewards by taking actions $\mathbf{a}_t = [\alpha_{t,i}]_i^K$.

In this paper, we employ a long short-term memory (LSTM) model to use the historical data stored in the reply buffer to predict future values, which are denoted by $\hat{\mathbf{s}}_{t+1}$. The model network is trained to minimize the error between the predicted future value $\hat{\mathbf{s}}_{t+1}$ and the actual future value $\mathbf{s}_{t+1}$, which are obtained from the replay buffer. The predicted future value $\hat{\mathbf{s}}_{t+1}$ is then used as one of the features of the critic network. We employ regular feedforward neural networks for the critic and actor models. In a high level explanation, the critic network passes the quality value $Q(\mathbf{s}_t, \mathbf{a}_t)$ for given states and actions to the actor network, which then determines the future actions.

The critic and actor networks are trained based on the SAC training method that is specified in [17]. Unlike other RL algorithms that mostly maximize only the expected cumulative future rewards, SAC also optimizes an entropy regularization, i.e.,:

$$\pi^* = \arg\max_{\pi_{\boldsymbol{\theta}}} \sum_t \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t)} \left[r(t) + \beta \mathbb{E}_{\mathbf{a}_t} \left[-\log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t | \mathbf{s}_t) \right] \right], \tag{2}$$

where $\pi_{\boldsymbol{\theta}}$ denotes the actor network that outputs the standar deviations and the means of K Gaussian distributions. Then, the actions, which are the values α, are obtained by taking a random sample based on the standard deviations and the means. That is,:

$$\alpha_{t,i} = \tanh(n), \text{ where } n \sim \mathcal{N}(\mu_{t,i}, \sigma_{t,i}), \tag{3}$$

where $\mu_{t,i}$ and $\sigma_{t,i}$ are the mean and standard deviation at time t and the i^{th} subflow. The variable β is used as a temperature variable to weigh the regularization factor.

Based on [17], the critic network is trained to minimize the following objective function:

$$J_Q(\boldsymbol{\phi}) = \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})} \left[\left(Q(\mathbf{s}_t, \mathbf{a}_t) - (r(t) - \gamma V(\mathbf{s}_{t+1})) \right)^2 \right], \tag{4}$$

where:

$$V(\mathbf{s}_{t+1}) = \mathbb{E}_{\mathbf{a}} \left[Q(\mathbf{s}_{t+1}, \mathbf{a}_t) - \beta \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t | \mathbf{s}_t) \right]. \tag{5}$$

Note that in order to obtain the value of $Q(\mathbf{s}_{t+1}, \mathbf{a}_t)$, we perform the forward propagation to the critic network with the help of $\hat{\mathbf{s}}_{t+1}$ from the model network.

The actor network is trained to minimize the following:

$$J_\pi(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{s}} \left[\mathbb{E}_{\mathbf{a}} \left[\beta \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t | \mathbf{s}_t) - Q(\mathbf{s}_t, \mathbf{a}_t) \right] \right]. \tag{6}$$

Instead of using (7), we use the re-parameterization trick described in [26] as it has a lower variance. That is, the values of alpha are calculated based on:

$$\alpha_{t,i} = \tanh(\mu_{t,i} + \epsilon \sigma_{t,i}), \text{ where } \epsilon \sim \mathcal{N}(0, 1). \tag{7}$$

By using this trick, it is possible to perform the backward propagation with respect to the parameter $\boldsymbol{\theta}$. As a summary, the Algorithm 1 provides the methodology to train our model-augmented SAC model, where λ is a learning rate for the networks.

Algorithm 1 Pseudocode of Model-Augmented SAC

Input: $\boldsymbol{\nu}$, $\boldsymbol{\theta}$, and $\boldsymbol{\phi}$ ▷ Initial parameters
Initialize $\boldsymbol{\nu}$, $\boldsymbol{\theta}$, and $\boldsymbol{\phi}$ with random weights
Initialize the replay buffer
for each iteration **do**
  for each environment step **do**
    Sample future states $\hat{\mathbf{s}}_{t+1}$ from the model network
    Sample actions $\mathbf{a}_t$ from the actor network
    Sample $\mathbf{s}_t$ and $r(t)$ from the environment
    Store $\{\hat{\mathbf{s}}_{t+1}, \mathbf{a}_t, \mathbf{s}_t, r(t)\}$ to the replay buffer
  end for
end for
Train the model network w.r.t. $\boldsymbol{\nu}$
for each gradient step **do**
  $\boldsymbol{\phi} = \boldsymbol{\phi} - \lambda_Q \nabla_{\boldsymbol{\phi}} J_Q(\boldsymbol{\phi})$
  $\boldsymbol{\theta} = \boldsymbol{\theta} - \lambda_\pi \nabla_{\boldsymbol{\theta}} J_\pi(\boldsymbol{\theta})$
end for
Output: $\boldsymbol{\nu}$, $\boldsymbol{\theta}$, and $\boldsymbol{\phi}$ ▷ Optimized parameters

IV. Results and Discussions

In this section, our results will be presented and discussed. Firstly, the experimental setup will be described and then the results will be presented. Lastly, we also explain the source of our performance gain.

A. Experimental Setup

Both the DRL agent and the environment run on a workstation having the following details:

- MPTCP v0.92 from [7],
- Linux Kernel long-term support release v4.4,
- the Ubuntu 20.04 focal operating system,
- the AMD Ryzen Threadripper CPU,
- a Quadro RTX 6000 GPU, and
- 256 GB of RAM.

As previously mentioned, one of the advantages of our method is that we can still use the existing congestion controls without any modification. From [7], there are four existing MPTCP congestion controls, i.e., LIA, BALIA, OLIA, and wVegas. Later, we will use these congestion control algorithms, add a Netfilter implementation whose the α coefficient driven by our DRL agent, and compare them with the state-of-the-art from [16]. Moreover, tools or utilities such as *tcpdump*, *wireshark*, *tshark*, *captcp*, and *iperf*, are used to capture the values of round trip times, goodputs, and TCP congestion windows.

Regarding the number of users in the room, it follows a discrete uniform distribution with a support ranging from 2 to 6. The users behaves randomly, for example operating a laptop equipped with both Wi-Fi and LiFi, sitting while watching a streaming video over a mobile device, or walking while

phone calling. These activity models are added with random orientation and random mobility models in our emulator. During the training process, we uniformly pick these activities for each user in a random way.

B. Experimental Results

First, we refer the readers to watch our video demonstration showing a real-time comparison of our DRL implementation and a vanilla MPTCP implementation as shown on YouTube[2]. Other than the fact that the proposed algorithm generally outperforms the vanilla MPTCP, we can infer from the video that the Netfilter coefficient dynamically reflects the signal quality indicated by the RSS values. That is, when the RSS value of a LiFi channel is high, it shows that a high bandwidth is available in a LiFi channel, the Netfilter coefficient corresponding to the LiFi channel is also high. It means that our DRL implementation can adaptively adjust the Netfilter coefficients depending on the signal quality of the channels.

By running a massive number of episodes and ensuring that all measurements are taken fairly by using the same realizations from all random models, Fig. 5 depicts our performance comparison. We compare the vanilla MPTCP implementation based on [7] by using the wVegas congestion control algorithm, the state-of-the-art from [16] (which is referred to as DRL-CC for short), and our proposed method by using different, existing MPTCP congestion control algorithms. It is shown that the proposed approach generally outperforms the others. Other than the performance comparison, there are two insights that can be obtained from Fig. 5. First, even with the BALIA algorithm which suffers from a Head-of-Line delay as described in [9], we can still outperform the state-of-the-art. Another important insight to note is that thanks to the SAC, our implementation has a lower variance compared to the DDPG-based implementation in DRL-CC. Note that this result conforms with the observation from [17]. In order to show a more concrete result in this regard, Fig. 6 depicts the training efficiency of our approach. That is, our approach trains faster compared to the counterpart method. The higher fluctuation of the DRL-CC training curve also explains the high variance as well as the brittleness of the DRL-CC, which is mainly due to the DDPG. Following [16], we also investigate if our approach compromise the fairness. Fig. 7 shows the Jenkin's fairness index comparison. Since all values are close to 1, we can conclude that our approach does not significantly compromise the users' fairness.

For the sake of explainability, it is important to understand where the performance advantage comes from. One way to achieve it is by monitoring the TCP retransmission timeout (RTO). According to MPTCP, a cross subflow retransmission (i.e., a vertical handover) occurs after 2× the longest RTO of all subflows. Therefore, we can infer that the shorter the TCP RTO is, the earlier the DRL agent can predict if a vertical handover will occur in the near future. As depicted in Fig. 8, it can be said that due to the model network, the proposed algorithm can predict the vertical handover earlier compared to the DRL-CC. Another reason why the proposed algorithm can predict a vertical handover event earlier is that by placing a Netfilter in all devices, the Netfilter can emulate a congestion occured in the network; therefore, the MPTCP can adapt earlier.

[2]https://bit.ly/drl-mptcp-simple-demo

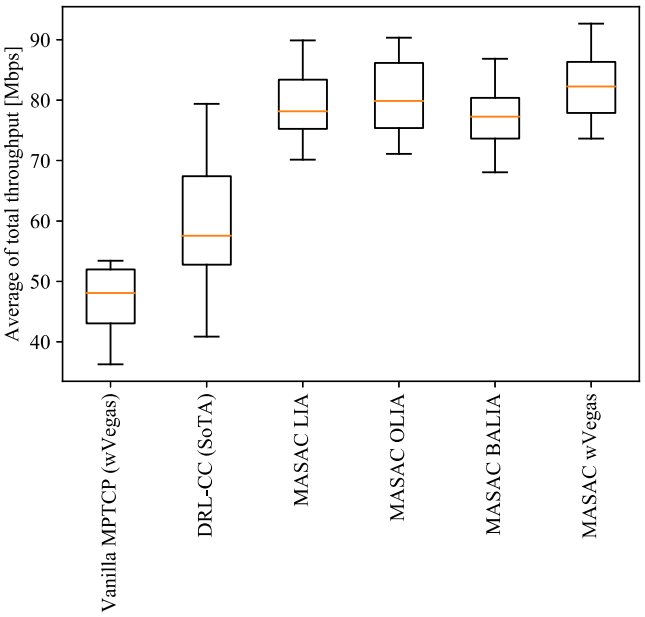

Fig. 5. Performance comparison between the vanilla MPTCP implementation based on [7], DRL-CC based on [16], and the proposed DRL approach (referred to as 'MASAC') for the scenario depicted in Fig. 1.

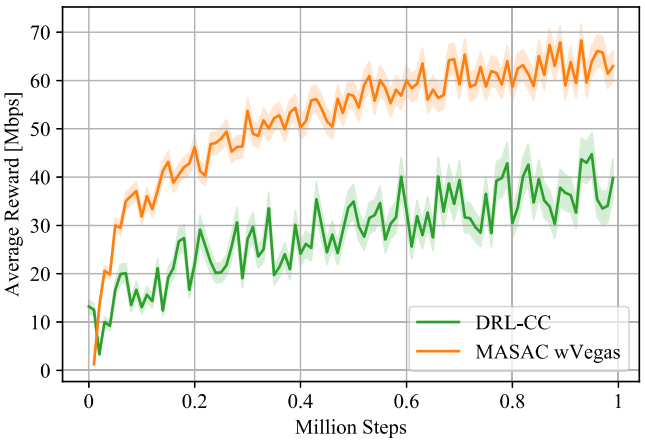

Fig. 6. Training curve comparison

V. Conclusions

In this paper, we investigated our proposed approach to intelligently steer the MPTCP subflows in a hybrid Wi-Fi and LiFi network. There are two main contributions made in this paper. First, instead of modifying an MPTCP implementation directly, which is challenging due to the fact that it must be done in the kernel space, we proposed to dynamically adjust the Netfilter coefficients. By having the Netfilter present in all devices and being able to configure it dynamically, an MPTCP implementation can act earlier. A further contribution

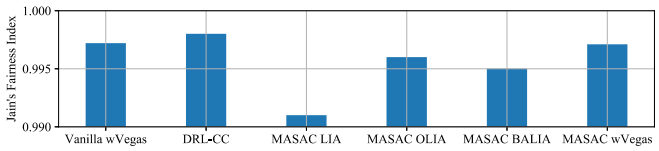

Fig. 7. Jain's fairness index

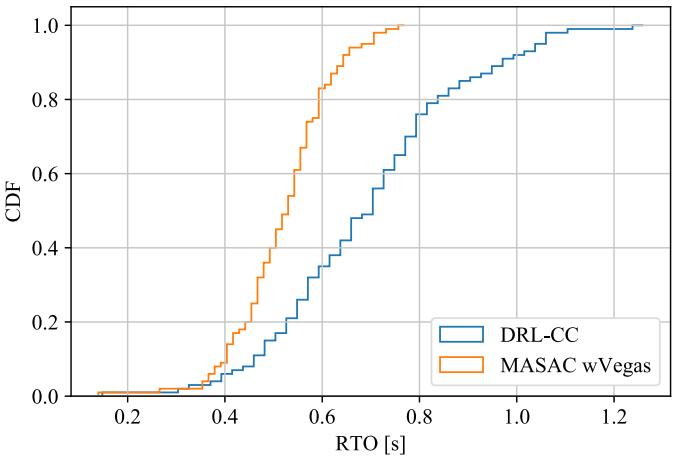

Fig. 8. TCP RTO comparison

is the use of a model-augmented SAC algorithm, where the model network can provide an estimate of future values of states to the critic network on top of the SAC algorithm. Compared to the vanilla MPTCP implementation (which can achieve an average total throughput of 48 Mbps) and the state-of-the-art (which can achieve an average total throughput of 57 Mbps), the proposed approach can achieve 82 Mbps. Note that this performance is obtained by running the DRL agent in an environment where we could emulate a close-to-reality scenario. That is, a random orientation model of mobile devices, random mobility and random blockage models of users are considered. In addition, by using the link-to-system mapping, we also emulated IEEE 802.11n for the Wi-Fi connection, and IEEE 802.11bb for the LiFi connection. Therefore, ensuring that the proposed algorithm can perform well in a close-to-reality scenario increases our confidence to implement it in a real world scenario, which will be reserved for our future work.

Acknowledgment

This research has been supported in part by EPSRC under Established Career Fellowship Grant EP/R007101/1, Wolfson Foundation and European Commission's Horizon 2020 research and innovation program under grant agreement 871428, 5G-CLARITY project.

References

[1] T. Cogalan and H. Haas, "Why would 5G need optical wireless communications?" in *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 2017, pp. 1–6.

[2] H. Haas, L. Yin, C. Chen *et al.*, "Introduction to indoor networking concepts and challenges in LiFi," *Journal of Optical Communications and Networking*, vol. 12, no. 2, pp. A190–A203, 2020.

[3] H. Haas, L. Yin, Y. Wang, and C. Chen, "What is LiFi?" *Journal of Lightwave Technology*, vol. 34, no. 6, pp. 1533–1544, 2016.

[4] A. A. Purwita, M. D. Soltani, M. Safari, and H. Haas, "Terminal Orientation in OFDM-Based LiFi Systems," *IEEE Transactions on Wireless Communications*, vol. 18, no. 8, pp. 4003–4016, 2019.

[5] Z. Zhang, Y. Xiao, Z. Ma *et al.*, "6G Wireless Networks: Vision, Requirements, Architecture, and Key Technologies," *IEEE Vehicular Technology Magazine*, vol. 14, no. 3, pp. 28–41, 2019.

[6] "Cisco Annual Internet Report (2018–2023)," White Paper, Cisco, March 2020.

[7] C. Paasch, S. Barre, *et al.*, "Multipath TCP in the Linux Kernel." [Online]. Available: https://www.multipath-tcp.org

[8] C. Paasch and O. Bonaventure, "Multipath TCP," *Communications of the ACM*, vol. 57, no. 4, pp. 51–57, 2014.

[9] M. Polese, R. Jana, and M. Zorzi, "TCP and MP-TCP in 5G mmWave Networks," *IEEE Internet Computing*, vol. 21, no. 5, pp. 12–19, 2017.

[10] Q. Peng, A. Walid, J. Hwang, and S. H. Low, "Multipath TCP: Analysis, Design, and Implementation," *IEEE/ACM Transactions on Networking*, vol. 24, no. 1, pp. 596–609, 2016.

[11] Y. Liu, X. Qin, T. Zhu *et al.*, "BESS: BDP Estimation Based Slow Start Algorithm for MPTCP in mmWave-LTE Networks," in *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, 2018, pp. 1–5.

[12] S. J. Siddiqi, F. Naeem, S. Khan *et al.*, "Towards AI-enabled traffic management in multipath TCP: A survey," *Computer Communications*, vol. 181, pp. 412–427, 2022.

[13] T. Zhang and S. Mao, "Machine Learning for End-to-End Congestion Control," *IEEE Communications Magazine*, vol. 58, no. 6, pp. 52–57, 2020.

[14] R. Ahmad, M. D. Soltani, M. Safari *et al.*, "Reinforcement learning based load balancing for hybrid LiFi WiFi networks," *IEEE Access*, vol. 8, pp. 132 273–132 284, 2020.

[15] R. Ahmad, M. D. Soltani, M. Safari, and A. Srivastava, "Reinforcement learning-based near-optimal load balancing for heterogeneous LiFi WiFi network," *IEEE Systems Journal*, 2021.

[16] Z. Xu, J. Tang, C. Yin *et al.*, "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, 2019.

[17] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[18] S. Merlin, G. Barriac, H. Sampath *et al.*, "TGax Simulation Scenarios," https://mentor.ieee.org/802.11/dcn/14/11-14-0980-16-00ax-simulation-scenarios.docx, 2015.

[19] R. Fontes, S. Afzal, S. Brito *et al.*, "Mininet-WiFi: emulating Software-Defined wireless networks," in *2nd International Workshop on Management of SDN and NFV Systems, 2015(ManSDN/NFV 2015)*, Barcelona, Spain, Nov. 2015.

[20] K. Brueninghaus, D. Astely, T. Salzer *et al.*, "Link performance models for system level simulations of broadband radio access systems," in *2005 IEEE 16th International Symposium on Personal, Indoor and Mobile Radio Communications*, vol. 4, Berlin, Germany, 2005, pp. 2306–2311 Vol. 4.

[21] A. Purwita, "Studies of Optical Wireless Communications: Random Orientation Model, Modulation, and Hybrid WiFi and LiFi Networks," Ph.D. dissertation, Edinburgh UK, 2021. [Online]. Available: https://era.ed.ac.uk/handle/1842/38221

[22] A. A. Purwita, M. D. Soltani, M. Safari, and H. Haas, "Terminal Orientation in OFDM-Based LiFi Systems," *IEEE Transactions on Wireless Communications*, vol. 18, no. 8, pp. 4003–4016, 2019.

[23] A. A. Purwita and N. N. Qomariyah, "Owcsimpy: A 3D simulator for indoor optical wireless channels," in *2021 IEEE Symposium On Future Telecommunication Technologies (SOFTT)*, 2021, pp. 6–11.

[24] The netfilter project, "netfilter: firewalling, NAT, and packet mangling for linux," https://www.netfilter.org/, 2022.

[25] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[26] J. Schulman, N. Heess, T. Weber, and P. Abbeel, "Gradient estimation using stochastic computation graphs," *Advances in Neural Information Processing Systems*, vol. 28, 2015.