# AI-driven Dynamic Millimeter-Wave Mesh Backhaul

Wenyuan Zhao
University of California San Diego
La Jolla, CA, USA
wez030@ucsd.edu

Zheng Liu
University of California San Diego
La Jolla, CA, USA
zhl139@ucsd.edu

Ziyan Cui
University of California San Diego
La Jolla, CA, USA
z5cui@ucsd.edu

## Abstract

Millimeter-wavelength (mmWave) mesh network can provide multi-Gbps transmission but with large path loss and heterogeneous objectives which is hard to solve by traditional rule-based models. Machine learning (ML) techniques, especially reinforcement learning (RL), have great potential in solving multi-objective, non-linear, and non-convex problems that often happen in mmWave mesh network configuration. On the other hand, network configuration policies learned from simulations cannot always help physical networks meet performance requirements due to sim2real gap. In this work, we develop a reinforcement learning (RL) model to train a policy for dynamic topology management and a self-supervised policy adaptation algorithm to bridge the domain gap. The experimental results shows that our RL agent can learn a policy to avoid blockage links and self-supervised learning model can help to eliminate domain gaps. The testbed we built can establish multiple routes and can be controlled effectively by a central controller. We successfully ran the simulation-trained RL policy and self-supervision agent on the real testbed.

***Keywords:*** mmWave mesh, topology management, reinforcement learning, self-supervised policy adaptation

**ACM Reference Format:**
Wenyuan Zhao, Zheng Liu, and Ziyan Cui. 2022. ECE257A Project Report: AI-driven Dynamic Millimeter-Wave Mesh Backhaul. In *Proceedings of Modern Communication Networks (UCSD ECE257A).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

## 1 Introduction

5G NR, released in 2019, has brought new standards and challenges to the industry. Millimeterwave (mmWave) technology enables multi-Gbps transmission. With 5G network infrastructures, technologies and applications are quickly evolving in recent years. For example, ultra-high definition video streaming, autonomous vehicle and remote medical care. These applications share the same requirement of ultra-low latency and ultra-high reliability [4]. To meet these performance requirements, it is important that network management settings get updated in time, so that they can adapt to dynamic network conditions. However, in mmWave mesh network, this remains to be an open problem. This is due to high directional mmWave beam and heterogeneous objectives in mmWave mesh networks, which makes traditional pre-programmed and rule-based models hard to accommodate. 5G network adopts virtually sliced paradigm [3]. 'Slice' refers to partitioning physical infrastructure into logically independent networks for more programmable network. This feature also makes configuration decisions highly intractable. Another challenge of mmWave mesh network configuration is network dynamics. Multitudes of parameters need to be reconfigured, which is beyond the scope of any standard specification.

This inspires increasing exploration in data-driven approaches to solve this networking problem. In machine learning (ML), algorithm can be derived without explicit preprogramming. ML-based method, especially reinforcement learning (RL), has the potential in solving multi-objective, non-linear and non-convex problems in mmWave mesh network management.

In this project, we consider the dynamic mmWave mesh network management problem shown in Fig. 1. The software-defined mesh network supports dual-interface network nodes: highly directional mmWave links for data plane and omnidirectional low-frequency links (WiFi links) for control plane. The packet sent from the source node usually needs several relays to reach the destination node, and the controller has to select appropriate links for high-rate data transmission.

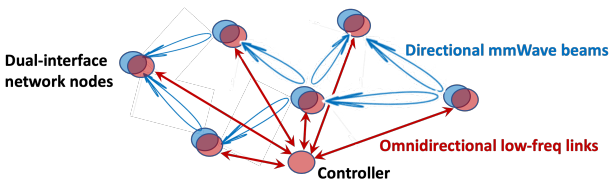


**Figure 1.** mmWave mesh network topology.


*UCSD ECE257A, 2022, La Jolla, CA*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

To solve dynamic mmWave mesh network configuration, we developed a reinforcement learning (RL) model, which focuses on selecting transmit uplinks which provide optimal quality of service (QoS). However, training a policy requires a large amount of data, and collecting data on real mesh is very expensive. So we intend to utilize simulations to generate data and train the RL policy in simulations first. But it corresponds to another challenge: directly deploying simulation-trained policy to real-world environment cannot achieve expected performance due to the domain gap. Therefore, we also aim to develop a policy adaptation model to bridge the domain gap between training environment and testing environment.

## 2 Related Works

As we mentioned in Section 1, this project mainly focuses on mesh network management. This problem has two categories based on the objective: network traffic delivery, and economical and environmental operation. The first one includes problem like routing management and data flow control, while the second one is related to power management. RL, as the best ML-model for real-world network management, has been adopted in several works. Q-learning based AP selection in IEEE 802.11 mesh network is proposed by Niyato and Hossain [1], which makes decisions by estimating collision probability and received signal strength (RSS). D. Oriol et al. adopts a Neural Network (NN) to predict link bandwidth and then uses it as a routing metric [2]. The input metrics are SNR, transmission time, MCS and re-transmission rate.

One common challenge of applying RL algorithm to physical communication networks is the sim2real gap. This occurs when the network configuration policy learned from simulations is deployed to physical networks. Using simulations has advantages of reducing experimental overhead, as well as improving flexibility and enriching configurable parameters. But these come at the expense of questionable credibility of the results. Shi et al. [5] employed a deep learning based domain adaptation method to close this gap. This work also leveraged a teacher-student neural network to transfer the network configuration knowledge learned from a simulated network to its corresponding physical network.

Among the existing mainstream ML models, RL excels at real-time decision making by exploring better policy where an optimal solution is not known as priori. The limitation of most RL algorithms, is that they are limited as "offline learning", which means that training model in simulation and operation in real networks are relatively separated. This also corresponds to sim2real gap. An online RL (OnRL) framework was proposed to make video bitrate decisions in real-time video telephony system [6]. This method involved multiple individual RL agents in the system and they evolved their models over time. These agents were then aggregated to form a high-level RL model, so that each agent can react to unseen network conditions.

## 3 Method

In this section, we describe our self-supervised reinforcement learning method for dynamic mmWave mesh network configuration. With appropriate task design, it can be implemented on top of different network management problems.

### 3.1 Reinforcement Learning

We design a reinforcement learning (RL) framework to dynamically configure mmWave mesh topology given the observations collected from the mmWave mesh network (Fig. 2). The RL agent keeps observing the mesh network states $s_t$ (e.g. signal strength, throughput, loss rate) and decides an action $a_t$ (e.g. the routing path) which gives the optimal QoS. Then this action $a_t$ is encoded as a command sent by the controller, and the next multi-hop data transmission follows the routing path given by $a_t$. After the traffic goes through the mesh network, it generates a new state $s_{t+1}$ and a reward value $r_t$, which are fed into the RL agent for a new round of policy update.

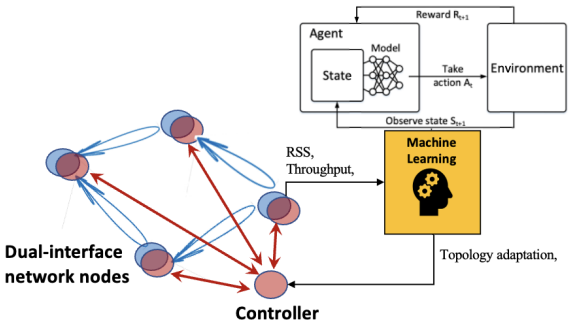


**Figure 2.** The framework of PPO-based RL for mesh configuration.

The decision of $a_t$ is made by RL policy $\pi_\theta : s_t \rightarrow a_t$. Intuitively, if the RL agent chooses a link with low quality, the quality of service (QoS) will be degraded, which is reflected in the reward value $r_t$ that the end-to-end throughput is lower. By observing a trajectory of $s_t$, $a_t$, $r_t$, $s_{t+1}$ ..., the RL agent knows how the chosen action will affect communication quality of the mmWave mesh network, and updates its policy to achieve better QoS.

**State.** The state is defined as $s_t = \left(\vec{l}_t, \vec{t}_t\right)$, which denotes the RSS of all links and the end-to-end throughput, respectively.

**Action.** At each step, the RL policy $\pi_\theta$ maps observed state $s_t$ to $a_t$ in an action space $\mathcal{A} : \{0,\ 1,\ 2\}$. Each number represents a specific routing path.

**Reward.** The RL policy is associated with a reward value at each step, which in our project is defined as the normalized throughput:

$$r_t = \frac{1}{N}\sum_{i=1}^{N} t_t^{\{i\}} \tag{1}$$

To update the RL policy, we utilize the state-of-art PPO-based RL algorithm. The objective of RL algorithm is to maximize a Q-value $Q^{\pi_\theta}(s_t, a_t) = E_{\pi_\theta}[\mathbb{R}_\tau | s_t, a_t]$, where $\mathbb{R}_\tau = \sum_{k=t}^{T-1} \gamma^{k-t} r_{k+1}(s_k, a_k)$ is the total cumulative reward. So the final objective can be derived as

$$\begin{aligned} &\max J(\theta) = \max E_{s_0 \sim p_0}\left[Q^{\pi_\theta}(s_0, a_0)\right] \\ &s.t.\ \nabla_\theta J(\theta) = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T-1} Q^{\pi_\theta}(s_t^i, a_t^i)\nabla \log \pi_\theta(a_t^i | s_t^i) \end{aligned} \tag{2}$$

PPO algorithm calibrates $J(\theta)$ as a clipped objective to improve data efficiency and convergence:

$$L(\theta) = \hat{\mathbb{E}}_t\left[\min\left(\gamma \hat{A}_t, \text{clip}\left(\gamma, 1-\varepsilon, 1+\varepsilon\right)\hat{A}_t\right)\right] \tag{3}$$

$\gamma = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the ratio of new policy to old policy. $\hat{A}_t$ is an estimator of the advantage function at timestep $t$. $\hat{\mathbb{E}}$ indicates the empirical average over a batch of samples. By clipping the probabilty ratio within the range of $[1-\epsilon, 1+\epsilon]$, the policy is updated more smoothly.

### 3.2 Self-supervised Policy Adaptation

As discussed in Section 2, RL policies generally require many training iterations and huge amount of data to converge. Due to high cost of collecting data in the real mesh network, a common solution is to train the policy in simulations and deploy it the real network. However, our policy is trained in offline simulations, so directly deploying the simulation-trained policy to the real network may not achieve an expected performance because of domain gap. We present a self-supervised learning (SSL) method that could adapt an offline policy to a new environment and bridge the domain gap.

Our self-supervised reinforcement learning framework is shown in Fig. 3. Instead of directly feeding the observed state $s_t$ in the PPO-based policy network, encoder $\pi_\phi$ first extracts some features of $s_t$ and policy network $\pi_\theta$ acts as a controller based on extracted features. The mapping $s_t \rightarrow a_t$ may change in a new environment due to the domain gap. Intuitively, we utilize the auxiliary self-supervision task to refine the encoder so that $\pi_\phi$ (and consequently $\pi_\theta$) can generalize. Our great concern is to design an appropriate self-supervision task to bridge the domain gap without introducing more complex model design or greater data collection cost. In this project, we design the self-supervision task as the inverse dynamic prediction.

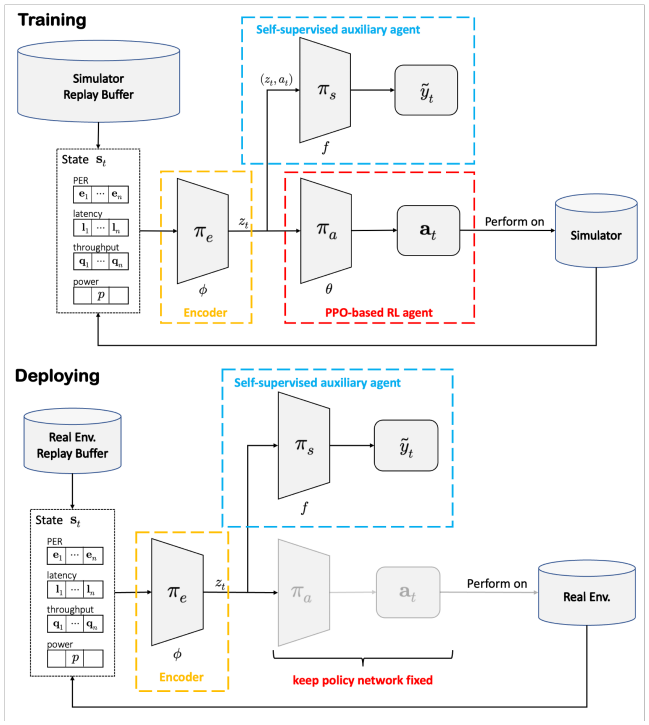


**Figure 3.** Self-supervised reinforcement learning model for policy learning and adaptation.

The inverse dynamic prediction utilize the existing data generated during the interaction between RL agent and environment. At each step, RL agent collects a state $s_t$ and makes a decision $a_t$. After conducting $a_t$, RL agent could again observe a new state $s_{t+1}$ for next iteration. Therefore, RL agent could always observe a transition sequence $(s_t, a_t, s_{t+1})$ at each iterative round. Inverse dynamic prediction takes $s_t$ and $s_{t+1}$ as input and predicts the action $a_t$ to take. The objective of inverse dynamic prediction in this project can be derived formally as

$$L(\phi, f) = l(a_t, \pi_f(\pi_\phi(s_t), \pi_\phi(s_{t+1}))) \tag{4}$$

Since our actions are defined in discrete space, the output $a_t$ is actually a soft-max logits, which represent the probability distribution of taking each action. The loss function $l(\bullet)$ is defined as cross-entropy loss. Hence, the training and testing/deployment process can be decribed as follows.

**Training.** During training, we jointly train both RL policy network and self-supervised learning network so that RL agent can make right decisions, and meanwhile self-supervision agent can make right predictions. The corresponding objective during training process can be written as

$$\min_{\phi,\theta,f} J(\phi, \theta) + \alpha L(\phi, f) \tag{5}$$

**Testing.** During testing/deployment, the distribution of $s_t$ and $s_{t+1}$ could be changed, and the mapping $s_t \rightarrow a_t$ (consequently $(s_t, s_{t+1}) \rightarrow a_t$) could be changed. During deployment we only minimize self-supervised learning loss since we only need small scale refinement and reward is inaccessible in some cases. The objective during testing/deployment process can be written as

$$\min_{\phi, f} L(\phi, f) \tag{6}$$

## 4 Experiments

In this section, we illustrate our experiments on validating how well the RL agent could dynamically configure mmWave mesh network and how well the self-supervised learning model could adapt policy to a new deployment environment.

### 4.1 Experimental Scenario

The experimental mmWave mesh network topology is shown in Fig. 4. The traffic generated at the source node (Node 1) needs several relays to reach the destination node (Node 5). Three different paths are considered in this scenario:

- 2-hop: Node 1→Node 3→Node 5
- 3-hop: Node 1→Node 2→Node 4→Node 5
- 4-hop: Node 1→Node 2→Node 3→Node 4→Node 5

We investigate a scenario where a car stops by and causes the failure of a blocked link for a while by setting a blockage randomly at a link. Upon blockage happens, the RSS and throughput of the blocked link will decrease significantly. So the RL agent needs to dynamically adapt routing topology to avoid the blocked link for optimal QoS based on the variation of observed RSS and throughput.

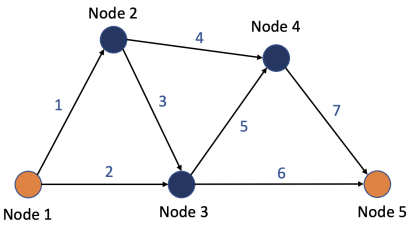


**Figure 4.** mmWave mesh network topology.

### 4.2 Experiments in NS3 Simulator

We utilize NS3 simulator to generate abundant data and train the self-supervised RL agent in simulations first. During each episode, we simulate the 600 Mbps traffic from Node 1 to Node 5 for 40 seconds. The RL agent collects a state $s_t$ and decides an action $a_t$ every 2 seconds. The maximum length of each episode is 20. We train the policy for 25000 time steps (1250 episodes). The implementation of our neural network architecture is illustrated in Fig. 5. All hidden layers of NNs are fully connected layers. The hidden dimension of policy network is 64, while the hidden dimension of encoder network is 32. The batch size is 128, which means the self-supervised RL agent are updated every 128 steps.

**Table 1.** Average test reward with/without self-supervised (SSL) policy adaptation.

| Strategy | Test reward | Average throughput |
|---|---|---|
| RL | 11.23 | 336.9 Mbps |
| RL+SSL | **19.56** | **586.8** Mbps |

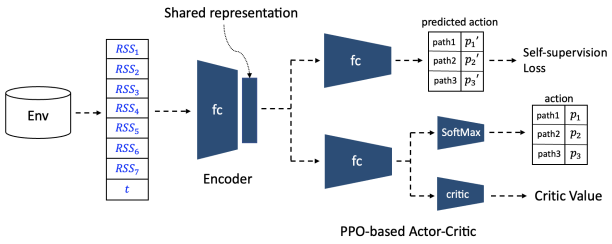


**Figure 5.** Implementation of neural network architecture.

Fig. 6 shows the learning curve of RL policy. It starts from a low average reward, and finally increases to the maximum average reward, which means that the RL agent could learn a good policy: make right decisions to avoid blockage link and maintains high-rate data transmission.

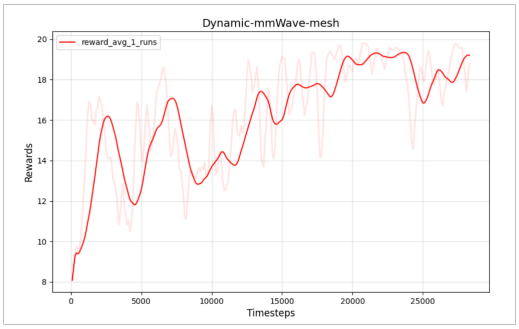


**Figure 6.** Learning curve of RL.

After training a good policy in training environment, we deploy the policy in a new environment where Node 1 is further from Node 2 and Node 3. So the observed RSS of link 1 and link 2 is lower than what is observed in the training environment, which is the domain gap between training and deployment environment.

The result of pre-trained policy in testing environment is shown in Table 1. We generate a 600 Mbps data flow from Node 1 to Node 5. Directly deploying the original policy to the new environment cannot get high throughput. But with the help of SSL, policy can easily adapt to the new

environment. The average throughput increases by more than 70%. Fig. 7 shows the actual throughput variation with and without self-supervised policy adaptation. Actually, the throughput is extremely unstable without SSL policy adaptation, and could easily be degraded to 0. However, if we use SSL policy adaptation, the throughput could stably keep at a very high level (600 Mbps). Therefore, our self-supervised policy adapation model could help bridge the domain gap when we apply our offline trained policy to a new environment.

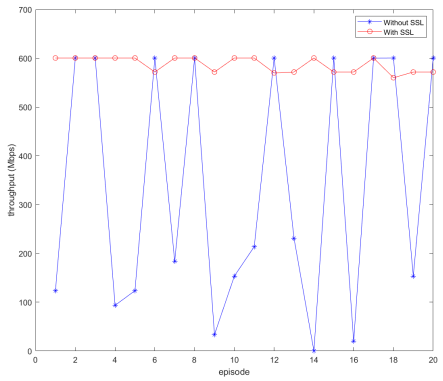


**Figure 7.** The episodic end-to-end throughput during testing/deployment.

### 4.3 Experiments in Real Testbed

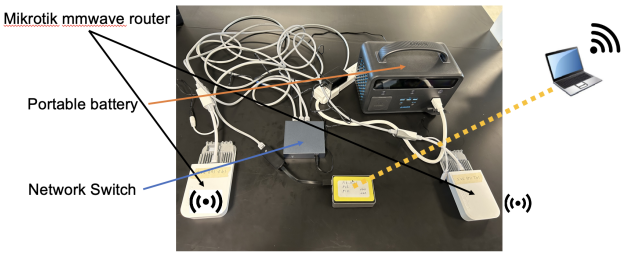


**Figure 8.** Node structure in real testbed.

Fig. 8 shows the structure of each node. In each node, we have multiple Mikrotik mmwave routers. The number of the routers depends on the number of the nodes that are connected to each node. Each router uses mmwave wireless link to communicate with another router which is equipped on another node. This is how two nodes are connected. The throughput of mmwave link can reach to more than 1.5 Gbps. In each node we have a raspberry Pi. The raspberry Pi is used for receiving controlling information from a laptop by WiFi signal, and it is also used for sending controlling information to routers by cable. The laptop here acts as a controlling center, which can manage the whole topology, such as route selection and pulling RSS and throughput information by Remote Procedure Call (RPC). The laptop uses WiFi signal to connect to the raspberry Pi in each node so that the whole network can be controlled. In each node we also have a portable battery for power supply.

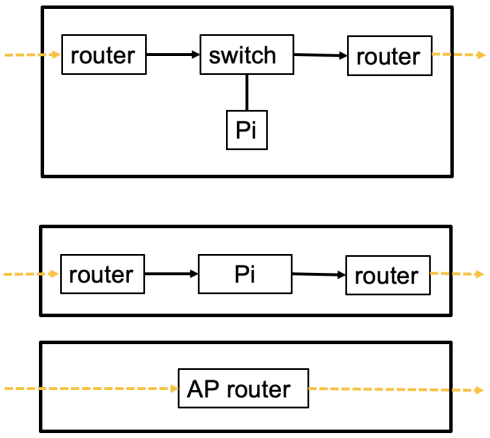


**Figure 9.** Block diagram of testbed node structure.

Fig. 9 is our current design to forward data. Previously, we have tried two different node structure design. We have tried to use raspberry Pi (see Fig. 9) to build a bridge between a pair of routers, however, the throughput is limited by the raspberry Pi, the whole link can only reach to 300 Mbps. We have also tried to just use an AP router for forwarding data (see Fig. 9), but it turned out that the throughput is limited at 500 Mbps. We think that this is caused by the TDMA MAC layer protocol. In our current design, the throughput of the link is higher than 500 Mbps, but it is limited at 1 Gbps, this is because the maximum throughput of the cable link between router and switch is only 1 Gbps, so this is a simulation-to-reality gap and our self-supervised learning model need to deal with this gap.

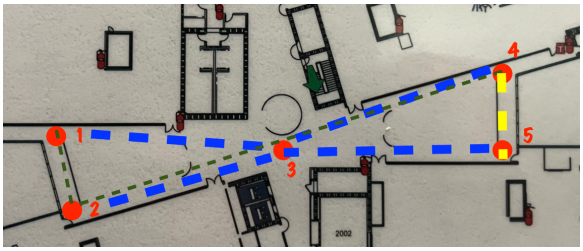


**Figure 10.** Node structure in real testbed.

We deployed the whole network at the second floor of a research hall (see Fig. 10). We have tried to deploy the network indoors, but it turned out that the router kept switching at different beam directions. We think that this is because there

are so many selectable beams, and they have similar network conditions.

To know more about the physical layer of mmwave communications, we deployed two nodes outdoors and connected them to each other. The distance between them was near 100 meters. We issued an UDP traffic to test the link performance. We found out that in outdoor environment, the beam direction was fixed, instead of randomly changing among multiple beams indoors. This is consistent with the design of mmwave network. We also found out that when a car or a person passed through the link, the RSS of the link suddenly dropped down significantly and returned to the normal value immediately after the blockage was removed. This verifies the blockage issue of mmwave communication.

Our experiments still have some points that could be improved in future. When there was no blockage, although it did not happen frequently, our model may switch to another route and then switch the route back immediately, we think that this is because the real test bed is not stable enough so that sometimes the throughput of a route can suddenly go down to nearly zero. For the same reason, the controlling center may suddenly stop working or get stuck at one code line. The future work about this is, we will go through the whole test bed step by step and figure out whether hidden problem exists in each part, which can make the network unstable.

When there was a blockage on one route, our model kept selecting the previous route, without flexibly selecting other better-throughput routes. A possible reason is, the throughput of multi-hop routes are extremely low, so that throughput changes were not significant enough when we blocked these routes. Based on our debugging, this was caused by signal collisions due to multipath effect, so we will deploy the whole network outdoor to see if this problem can be fixed. Another possible reason is, we neglected some simulation-to-reality gaps, such as MAC-layer differences, and these gaps cannot be eliminated by self-supervised learning model. To solve this issue, we will try building more sophisticated simulation models, which involve MAC-layer settings and queuing of each node.

## 5 Conclusion

In this project, we firstly did comprehensive paper reading and material learning about how to establish RL model, self-supervised learning model, and how to combine them together so that we can learn from simulations and use the knowledge on real scenarios by eliminating sim2real gap. The test results in simulations indicate that the RL agent can learn a policy to avoid blockage links and self-supervised learning model can help to eliminate domain gaps. Secondly, we built a testbed with Mikrotik mmwave devices and Respberry Pis. Our testbed can establish multiple routes and the topology can be controlled effectively. Thirdly, we successfully ran the simulation-trained RL policy and self-supervision on real testbed.

## 6 Self-contribution

- **Wenyuan Zhao.** Design the project objective as solving mmWave mesh network topology management. Propose the RL model to dynamically configure routing path. Propose the SSL model to bridge the domain gap. Write the code of RL and SSL algorithms. Train RL policy in network simulations and refine the model based on simulation results. Validate the performance of SSL policy adaptation. Help build the physical mesh network and the interface between real testbed and RL agent.
- **Zheng Liu.** Define project objective. Build a physical mmWave mesh network to do experiments. Modify and refine testbed structure based on testing results. Conduct experiments in different scenarios: indoor and outdoor. Investigate challenges of mmWave mesh communication: beamforming, blockage pattern. Work on improve testbed capability.
- **Ziyan Cui.** Background research on mesh network, and help define project objective. Investigate simulation tools (NS3) and machine learning tools (pytorch). Learn how to adapt WiGig module to perform end-to-end simulation and build the simulation platform. Model refinement and hyperparamter tuning. Train RL policy in network simulations and evaluate simulation results.

## References

[1] Dusit Niyato and Ekram Hossain. 2009. Cognitive radio for next-generation wireless networks: An approach to opportunistic channel selection in IEEE 802.11-based wireless mesh. *IEEE Wireless Communications* 16, 1 (2009), 46–54.

[2] David Oriol, Thi Mai Trang Nguyen, and Guy Pujolle. 2012. Available bandwidth estimation in Wireless Mesh Networks using neural networks. In *2012 Third International Conference on The Network of the Future (NOF)*. IEEE, 1–5.

[3] Ashwini Patil and HK Sawant. 2012. Technical specification group services and system aspects IP multimedia subsystem (IMS). *Int. J. Electron. Commun. Comput. Eng.* 3, 2 (2012), 234–238.

[4] Philipp Schulz, Maximilian Matthe, Henrik Klessig, Meryem Simsek, Gerhard Fettweis, Junaid Ansari, Shehzad Ali Ashraf, Bjoern Almeroth, Jens Voigt, Ines Riedel, et al. 2017. Latency critical IoT applications in 5G: Perspective on the design of radio interface and network architecture. *IEEE Communications Magazine* 55, 2 (2017), 70–78.

[5] Junyang Shi, Mo Sha, and Xi Peng. 2021. Adapting wireless mesh network configuration from simulation to reality via deep learning based domain adaptation. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 887–901.

[6] Huanhuan Zhang, Anfu Zhou, Jiamin Lu, Ruoxuan Ma, Yuhan Hu, Cong Li, Xinyu Zhang, Huadong Ma, and Xiaojiang Chen. 2020. OnRL: improving mobile video telephony via online reinforcement learning. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. 1–14.