Unboxing MAC Protocol Design Optimization Using Deep Learning

Hannaneh Barahouei Pasandi, Tamer Nadeem
Dept. of Computer Science, Virginia Commonwealth University, Richmond, VA 23284, USA
barahoueipash, tnadeem@vcu.edu

***Abstract*—Evolving amendments of 802.11 standards feature a large set of physical and MAC layer control parameters to support the increasing communication objectives spanning application requirements and network dynamics. The significant growth and penetration of various devices come along with a tremendous increase in the number of applications supporting various domains and services which will impose a never-before-seen burden on wireless networks. The challenge however, is that each scenario requires a different wireless protocol functionality and parameter setting to optimally determine how to tune these functionalities and parameters to adapt to varying network scenarios. The traditional trial-error approach of manual tuning of parameters is not just becoming difficult to repeat but also sub-optimal for different networking scenarios. In this paper, we describe how we can leverage a deep reinforcement learning framework to be trained to learn the relation between different parameters in the physical and MAC layer and show that how our learning-based approach could help us in getting insights about protocol design optimization task.**

***Index Terms*—MAC Protocol, Deep Reinforcement Learning, Wireless Networks**

I. Introduction

With the advent of microprocessors, the number of connected wireless devices continues to grow at a steady pace. A very recent forecast from International Data Corporation (IDC) estimates that there will be 41.6 billion connected wireless devices in 2025 [2]. This significant growth and penetration of various devices come along with a tremendous increase in the number of applications supporting various domains and services. Hence, it is widely accepted that this impressive scale of devices and applications will impose a never-before-seen burden on wireless networks. To cope with the emergence of various device characteristics and application requirements, complex and custom design of high performance networking protocols is needed. Networking protocols such as WiFi and Bluetooth, traditionally, are manually designed as "general-purpose" protocols for different network characteristics and scenarios through long-time and hard-work human efforts. However, while this approach is increasingly becoming difficult to repeat, these designed protocols are deeply rooted in inflexible, cradle-to-grave designs, and thus unable to address the demands of different network characteristics and scenarios. Therefore, it has now become crucial to re-engineer protocols designing process and shift toward a vision of an intelligent designing process that adapts and optimizes network protocols under various environment contexts such as device characteristics, application requirements, user objectives, and network conditions. In case of only physical layer, no single physical-layer design can work well under all scenarios, hence the natural response of the standards bodies has been to specify designs with a large number of control parameters ranging from modulation order and coding rate, to OFDM sub-carrier spacing and cyclic prefix length, to transmit power, etc., such that a medium can be tuned to the specific deployment scenario in the field. Each of these parameters has numerous settings leading to a large number of choices, and it becomes extremely difficult for domain experts to design a control algorithm that chooses the right algorithm depending on the scenario and the varying network conditions.

Deep learning (DL) techniques have recently been applied to various protocol and radio optimization tasks including routing [5], congestion control [11] and MAC protocol [12], just to name a few. Applying DL techniques can reduce manual human-based efforts to tune protocol parameters. Joseph et al. [6] show how to design a DL-based control algorithm to jointly control two parameters namely modulation order and transmit power scaling. In their work, they show that applying DL technique may work well to control the two aforementioned parameters, but depending on the context (different devices, throughput targets, etc.,) it becomes extremely complicated to get enough insights about how black-box DL technique works, although they only tune two parameters from a large set of available control parameters. Such observations reveal why it is extremely hard for domain experts to manually design control algorithms that could capture optimal solution for each scenario.

To the best of our knowledge, the current efforts in applying DL to enhance protocol performance focus only on *tuning* or *controlling* protocol parameters. Table I points to a few of the recent DL-based approaches proposed in different layers of the network stack. However, we believe that optimizing a protocol performance goes beyond individual protocol parameter *tuning*. In this paper, we propose a novel Deep Reinforcement Learning (DRL)-based framework, that is not only capable of tuning protocol parameters, but also optimizing the main functionalities for each protocol. In the proposed framework, a protocol is decoupled into a set of parametric modules as DRL inputs, each representing a main protocol functionality referred

This material is based upon work partially supported by the US National Science Foundation under Grant No. CNS-1764185.

TABLE I: Example approaches of using DL for communication protocol parameter tuning in different network stack layers

Network Layer	Function/ Sub-Layer	Objective	Learning Algorithm	Model Input	Control Parameter	Ref.
Data Link	MAC	Maximizing the sum throughput Allocation fairness	ResNet	State of Channel	Transmition of a packet	[12]
Network	Routing	Maximizing the minimum allocated bandwidth between possible source destination pairs in the network	Graph-Based Deep Learning	Graph of network topology	Each router locally controls which output interface to be used	[5]
Transport	Congestion Control	Maximizing the overall utility (e.g., goodput, delay, α-fairness)	DRL using LSTM	States of all active TCP flows	Congestion window	[11]

as *Building Blocks (BBs)*. This modularization technique helps to better understand the generated protocols and optimize the protocol design and analyze them in a systematic fashion. We feed into DRL agent, a high-level specification for a scenario, including the communication objective, the protocol BBs, measurements, and network configuration. The DRL agent then is able to learn what protocol blocks (components) are important to be included or to be neglected in the protocol design. Therefore, this framework could provide a tool for protocol designers to re-think the blocks used in a designed protocol. In addition, our framework can be utilized as a multi-variant optimization tool that helps in alleviating the current protocol design process. When designing a protocol, domain experts should keep different application requirements, user objectives, device constraint and network conditions in mind. Considering these parameters all together is a daunting task as discussed in [6].

As a case study, we narrow down our focus to propose a DRL-based framework for designing MAC protocols *hereafter* `DeepMAC`. In `DeepMAC` framework, MAC protocols are decoupled into a set of parametric modules, each representing a main functionality across popular flavors of 802.11 WLANs (IEEE 802.11 a/b/g/n/ac amendments). As we showcase in Section III-B, the DRL agent learns that when the load of the network is very low, it could eliminate control and sensing mechanisms (ACK and Carrier Sensing blocks, respectively) to increase the throughput of the channel by reducing the bandwidth overhead and waiting time introduced in these mechanisms. Therefore, this framework could serve as a tool for protocol designers to re-think the blocks used in a designed protocol. In addition, our framework could be utilized as a multi-variant optimization tool that helps in alleviating the current protocol design process. When designing a protocol, domain experts should keep different application requirements, user objectives, device constraint and network conditions in mind. Considering these parameters all together is a daunting task. By using this framework, domain experts provide the required specifications (objective) for a specific scenario as DRL input and could identify/capture the role that each protocol component (block) plays in varying scenarios for different objectives. It could also help domain experts to get insights about the relation between different protocol components for different objectives, although such components may not have a direct dependency/relation on each other if considered alone.

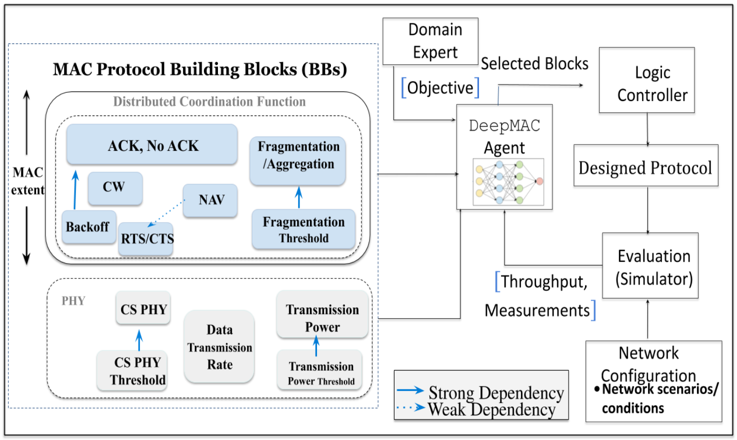

Fig. 1: `DeepMAC` framework

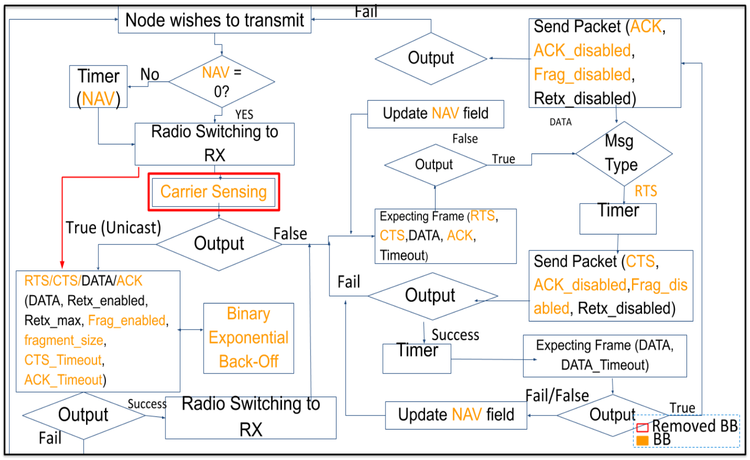
Node wishes to transmit
Fail
Timer (NAV)
No
NAV = 0?
YES
Radio Switching to RX
Carrier Sensing
True (Unicast)
Output
False
RTS/CTS/DATA/ACK (DATA, Retx_enabled, Retx_max, Frag_enabled, fragment_size, CTS_Timeout, ACK_Timeout)
Binary Exponential Back-Off
Output
Success
Fail
Radio Switching to RX
Output
Send Packet (ACK, ACK_disabled, Frag_disabled, Retx_disabled)
Update NAV field
False
Output
True
DATA
Msg Type
RTS
Timer
Expecting Frame (RTS, CTS,DATA, ACK, Timeout)
Send Packet (CTS, ACK_disabled,Frag_disabled, Retx_disabled)
Fail
Output
Success
Timer
Expecting Frame (DATA, DATA_Timeout)
Fail/False
Update NAV field
Output
True
Removed BB
BB

Fig. 2: Realizing IEEE 802.11 DCF using coarse-grained MAC blocks and their corresponding parameters

II. DeepMAC Framework

MAC protocols are required to be designed with a rich set of requirements to satisfy the needs of the overlaying applications (e.g., Augmented/ Virtual Reality, video conferencing) and scenarios. Due to the limited channel resources and a large number of devices accessing the channel, it is desirable that the MAC protocol minimizes the time wasted due to collisions or exchange of control messages.

In our previous work [8], we proposed a Reinforcement Learning (RL)-based framework to optimize the MAC protocol using a simple set of functionalities. However, we discovered that the RL-based approach may face instability since the agent has to find a balance between exploration and exploitation. In [7], [9] we provided a complete overview of the whole protocol design framework using machine learning techniques. We described the key design considerations for the learning agent (e.g., centralized, distributed or hybrid agents) and explained how these agents should communicate with one another. We then expanded our framework [10] to use deep architecture along with new set of building blocks. In this work, we use the same framework as [10] but with the objective to get more insights about protocol design through a deeper analysis of the DRL agent. Figure 1 illustrates `DeepMAC` framework for optimizing the design of wireless MAC protocols. We describe the key modules of this framework in the following.

A. Building Blocks and Logic Controller

Building blocks A network protocol is structured into several layers. Each layer is broken into a set of blocks with its own specific functionality. As described, building blocks are a set of separated parametric modular components, each of which is in charge of one (or several) specific well-defined functionality [1], [4]. The combination of different building blocks and the interactions between them determine the overall behavior of a network protocol for a given environment. In our framework, we have extracted a set of MAC protocol blocks from Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications [3] which includes MAC functionalities across all 802.11 amendments. Figure 1 shows the extracted blocks and instances of their dependencies captured based on non-Directional Multi-Gigabit (non-DMG) MAC architecture.

Once the blocks and their interactions are established, the network protocol could be represented as a graph, where the parameterized blocks are the vertices and the edges connecting the blocks represent the transition between them. Conducting the operations of individual blocks in an appropriate order, we are able to implement the protocol mechanisms. As an example, Figure 2 shows how IEEE 802.11 DCF is realized using the extracted coarse-grained blocks. The network protocol is operated under a variety of conditions and environments, which trigger events causing the protocol to act. Therefore, when describing a building block, we should also capture the dynamic behavior of a protocol caused by different events. Building blocks should react to incoming events, conducting their main operation while interacting with each other. The dynamic behavior of a BB could be estimated if the input events are known since the behavior of the BBs is deterministic. To be exact, we could describe a building block and its dynamic behavior as the following tuple:

$$Block :< E, P, S, F, D > \tag{1}$$

Where E is the **E**vent that triggers the block, P is a **P**arameter inside the block that could be adjusted, S is the internal **S**tate of the block, F is the main **F**unction that is executed in the block, and D represents the possible internal **D**ependencies between a block with other blocks. In our framework, the logic controller is responsible to check the sanity of a generated protocol. To show an example using tuple 1, lets consider Backoff mechanism as a single building block. A tuple that describes this block could be: <ACK_timeout, CW, Freeze/Countdown, Avoid Collision, ACK>.

Logic Controller In each iteration, `DeepMAC` agent takes a numerical vector of building blocks, objective and network measurement as input as described in more detail later, and outputs a set of selected blocks that logic controller uses to generate the protocol. In Figure 2 example, if `DeepMAC` agent decides that for the underlying scenario carrier sensing mechanism (framed by red color) should be excluded from the design, then it is the logic controller's responsibility to rewire a new version of DCF without carrier sensing. In addition, some functional blocks are dependent on each other. The logic controller is also in charge of checking the block execution sequences, their interdependencies, and interaction rules between blocks to ensure logically correct protocol design. We extracted the interdependencies between blocks from PHY and MAC specification [3] and incorporated them into the logic controller using if-then-else rules. In our design, all dependencies are uni-directional meaning if *Block A* depends on *Block B* it only shows restrictions of $A \to B$ but not $B \to A$. In the Backoff example, this mechanism is *strongly* dependent on ACK block; if there is no ACK, there will be no ACK timeout to signal frame retransmission. However, ACK mechanism can be used without Backoff. Sometimes blocks are *weakly* dependent on each other (see Figure 1).

B. DeepMAC As A Reinforcement Learning Problem

`DeepMAC` uses RL to learn the best set of protocol blocks for different scenarios. In `DeepMAC`, we consider a *centralized* agent for the design of 802.11 MAC protocols. This centralized agent, in practice, can be based on a single supernode (e.g., the Access Point) that periodically updates its model. Meaning it decides the selected set of MAC layer blocks and parameters to be used with all other nodes in the network. The reward function can be any objective function that is required to be optimized. The *reward* function in `DeepMAC`, is the *average throughput* of the link. The *state* of the agent is a vector of numerical representation of the set of the building blocks, and a history with a fixed length of the average link throughput values which are used as the input to `DeepMAC` agent. In this set, a value except 0 indicates that the corresponding block is included in the protocol design (each of the elements in the input vector can have different values which indicate what parameter or algorithm/method/mechanism should be used in the design), while 0 means the component is completely excluded from the design. The *action* in this framework, is the act of choosing the next state among all the available states from the current state such that the reward is maximized.

DRL agent architecture The neural network we have adopted is equipped with three hidden layers and an output layer. We find through experiments that this simple architecture can provide satisfactory performance, and increasing the complexity of the neural network does not contribute to performance improvements. The data is flattened before going through the hidden layers which utilize Relu as the activation function. The output layer consists of multiple neurons, each producing the Q-value of the corresponding action.

C. Building Block Design

Following the modular design principle in context of protocol design, two main branches exist on how to divide protocols and define the components: FSM and Data Flow. FSMs are graphical formalism that have become widely used in specifications of embedded and reactive systems. Their main drawback is that even for a moderate complicated system, they result in large diagrams. In order to support the dynamic design of the MAC protocols and the flexibility in selecting the optimum set of building blocks (components) by the RL agent in designing efficient protocols, capturing the interactions and dependencies between components is of crucial importance. Therefore, a desirable solution would be a flexible modular design that captures all possible interactions, dependencies, and relations of all possible design options. Yet among the main challenges to develop flexible and reusable set of building blocks is how to decide on the level of granularity of each block, and to evaluate different block granularity levels of the same function. One approach could be a brute-force in which the interactions, relations, dependencies, and conflicts between every couple of build blocks is defined by design experts. However, this approach is complex and time-consuming. Therefore, to address these issues, there should be further research on exploring how to reduce this complexity through approximation techniques, as well as whether this process could be automated.

D. Reward Function Design

Another important design decision is how to design and optimize the reward function. In a global optimization, both centralized and distributed agents work towards optimizing the same goal, while in local optimization, distributed nodes can optimize their own goals. Each of these approaches have their own challenges. Different applications have different performance requirements. Therefore, defining the "right" global optimization objective is not straightforward. Optimizing the objective function relies on the assumptions that all end-hosts employ the same prescribed protocol. Thus, there is a limited support for network heterogeneity, as well as, fulfilling different applications' objectives. On the other hand, each node in a distributed optimization tries to optimize its own objective function in which it might not converge.

III. DEEPMAC EVALUATION

Performance Metrics This section presents the numerical results and evaluation of `DeepMAC` regarding *block selection* by the agent under different scenarios. Before we delve into the experimental evaluation of our analysis, we clarify that we run the pre-trained DRL agent for every scenario. After training our DRL agent on a MacBook Pro with 2.9 GHz Intel Core i5 with 16 GB of memory, the agent took on average 1 ms to execute [10]. We assume that the supernode (centralized agent) uses hardware accelerators which can reduce the execution time by an order of magnitude and comfortably meet the time constraint requirements. We have not considered the convergence time of the DRL agent as a performance metric to evaluate since we have already shown the convergence of the agent in [8].

A. Simulation Configuration

We consider an ad-hoc network where individual nodes communicate with each other directly. To carry out our simulations, we use our own C++ event-driven simulator. Table III includes the blocks and their corresponding algorithm, mechanism, or parameters that are used by `DeepMAC` framework for the experiments. Without loss of generality, we assume that each node has always a packet to transmit, and the packet generation rate follows a Poisson process. In our experiments, we consider eight different networking scenarios described in Table IV.

B. Selected Blocks in Different Scenarios

The selected blocks by the agent are shown in Table V. In the following, we divide our observations about DeepMAC behavior in two parts and discuss each in more detail.

Low load with/without noise In scenarios with the low load when the noise is absent (Scenario #1) no control packet such as ACK or RTS/CTS is selected by the agent. This is justifiable. Even though the control packets are much smaller than the data packets, the time spent for control packet transmission is not negligible.

Therefore, when the network is under-saturated, and the number of competing stations are small, the DRL agent avoids control packet overheads to maximize the throughput. Intuitively, to reduce the relative percentage of the time loss due to packet overhead and MAC coordination, frame aggregation is also selected by the agent. While for the same scenario, when the noise is present, the agent adds Career Sensing (CS) block. This could be because the agent learns such a mechanism can be useful when the throughput drops. For scenarios with the average level of noise (Scenario #3,4) except common ACK mechanism selection, there is no obvious pattern.

High and saturated load with/without noise We discuss the following observations for this set of scenarios: 1) The first observation for Scenario #5 to 8 is the ACK mechanism selection by the agent. Intuitively, this could be because the agent learns such a mechanism can contribute to prevent more number of collisions and corresponding retransmissions to enhance the throughput. 2) When comparing scenario 5 with 6, we observe that the agent uses the Fragmentation block. The size of the sub-frames in practice plays an important factor that can influence network throughput performance for a given channel condition. The larger fragments, possibly the higher Packet Error Rate (PER) which would cause throughput drop due to a large number of retransmissions. 3) When the network is saturated, the agent selects protection mechanisms

TABLE II: Simulation configuration

Parameters	Values
Frame Size	1500Bytes (Default)
Time Slot	0.2 msec
Channel Capacity	10 Mbps
Learning Rate (α)	1
History Length (H_t)	15
Discount Factor (γ)	0.8

TABLE III: Blocks and their associated algorithm/mechanism/parameter

Building Block	Algorithm/Parameter	Default
Backoff	BEB, EIED	BEB
ACK	No ACK, ACK	ACK
Fragmentation (Fr)	Packet Size =200, 500, 1000 bytes	Packet Size = 1500 bytes
Aggregation (Ag)	Packet Size =2000 bytes	Packet Size =1500 bytes
RTS/CTS	Enabled/Disabled	N/A
CW	0-1023	$CW_{min} = 15$
Carrier Sense (CS)	Enabled/Disabled	N/A
Data Transmission Rate (DR)	6/9/12/24/36/48/54 (Mbps)	54 Mbps

TABLE IV: Simulation scenarios

Scenario	Nodes	Load	Noise
1	5	Low	No
2	5	Low	Yes
3	15	Average	No
4	15	Average	Yes
5	20	High	No
6	20	High	Yes
7	50	Saturated	No
8	50	Saturated	Yes

TABLE V: Blocks selected by DeepMAC agent

#	DR	BEB	EIEB	CS	CW	No ACK	ACK	Fr	Ag	RTS/ CTS
1	⬚ 54	⬚	⬚	⬚	🟧 31	🟧	⬚	⬚ 1500	🟧 2000	⬚
2	🟧 24	⬚	⬚	🟧	🟧 31	🟧	⬚	⬚ 1500	⬚ 1500	⬚
3	⬚ 54	⬚	⬚	🟧	⬚ 15	⬚	🟧	⬚ 1500	⬚ 1500	⬚
4	🟧 48	⬚	🟧	⬚	⬚ 15	⬚	🟧	⬚ 1500	⬚ 1500	🟧
5	⬚ 54	⬚	🟧	🟧	⬚ 15	⬚	🟧	⬚ 1500	⬚ 1500	🟧
6	🟧 24	🟧	⬚	⬚	⬚ 15	⬚	🟧	🟧 1000	⬚ 1500	⬚
7	🟧 36	🟧	⬚	⬚	⬚ 15	⬚	🟧	🟧 500	⬚ 1500	🟧
8	🟧 24	🟧	⬚	⬚	⬚ 15	⬚	🟧	🟧 500	⬚ 1500	🟧

🟧 Selected BB

⬚ Inactive or With Default Value BB

such as ACK and RTS/CTS along with smaller frame sizes and lower bitrate. However, it is not clearly obvious if the smaller frames contribute much to enhance the throughput. This is due to the fact that small fragments with the extra introduced overhead could also decrease the throughput performance. The varying results reveal why it is extremely hard for an algorithm based on manually-specified rules and thresholds to capture the optimal solution, and why it is instead better to use such a deep learning-based tool to optimize the design of control algorithms and get insights about what functionality is useful under what scenario.

IV. Conclusion

In this paper, we proposed and evaluated a framework for MAC protocol design optimization using a DRL-based approach. We have shown that by observing the decisions of the DeepMAC agent and using a method such as input modularization (protocol decomposition into building blocks), it is possible to extract information about the associated component selection by the agent. We envision this method could offer useful insights, especially to protocol designers to build deeper perception about the significance of an individual or a set of protocol blocks (functions) under different scenarios. This could help them focusing on enhancements/ modifications of important components than focusing on the whole protocol performance in order to enhance the protocol design and performance.

References

[1] Fan Bai, Ganesha Bhaskara, and Ahmed Helmy. Building the blocks of protocol design and analysis: challenges and lessons learned from case studies on mobile ad hoc routing and micro-mobility protocols. *ACM SIGCOMM Computer Communication Review*, 34(3):57–70, 2004.

[2] David Reinsel Carrie MacGillivray. Worldwide global datasphere iot device and data forecast. 2019–2023.

[3] IEEE Computer Society LAN MAN Standards Committee et al. Part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. 2016.

[4] Christian Doerr, Michael Neufeld, Jeff Fifield, Troy Weingart, Douglas C Sicker, and Dirk Grunwald. Multimac-an adaptive mac framework for dynamic radio networking. In *First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, 2005. DySPAN 2005.*, pages 548–555. IEEE, 2005.

[5] Fabien Geyer and Georg Carle. Learning and generating distributed routing protocols using graph-based deep learning. In *Proceedings of the 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, pages 40–45, 2018.

[6] Samuel Joseph, Rakesh Misra, and Sachin Katti. Towards self-driving radios: Physical-layer control using deep reinforcement learning. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, pages 69–74, 2019.

[7] Hannaneh Barahouei Pasandi. Towards a machine learning-based framework for automated design of networking protocols. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, page 433–434. IEEE, 2019.

[8] Hannaneh Barahouei Pasandi and Tamer Nadeem. Challenges and limitations in automating the design of mac protocols using machine-learning. In *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pages 107–112. IEEE, 2019.

[9] Hannaneh Barahouei Pasandi and Tamer Nadeem. Poster: Towards self-managing and self-adaptive framework for automating mac protocol design in wireless networks. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, pages 171–171. ACM, 2019.

[10] Hannaneh Barahouei Pasandi and Tamer Nadeem. Mac protocol design optimization using deep learning. In *2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*. IEEE, 2020.

[11] Zhiyuan Xu, Jian Tang, Chengxiang Yin, Yanzhi Wang, and Guoliang Xue. Experience-driven congestion control: When multi-path tcp meets deep reinforcement learning. *IEEE Journal on Selected Areas in Communications*, 37(6):1325–1336, 2019.

[12] Yiding Yu, Taotao Wang, and Soung Chang Liew. Deep-reinforcement learning multiple access for heterogeneous wireless networks. *IEEE Journal on Selected Areas in Communications*, 37(6):1277–1290, 2019.