Towards a Learning-Based Framework for Self-Driving Design of Networking Protocols

HANNANEH BARAHOUEI PASANDI, (Student Member, IEEE),
AND TAMER NADEEM, (Senior Member, IEEE)
Department of Computer Science, Virginia Commonwealth University, Richmond, VA 23284, USA
Corresponding author: Hannaneh Barahouei Pasandi (barahoueipash@vcu.edu)

This work was supported in part by the U.S. National Science Foundation under Grant CNS-1764185.

ABSTRACT Networking protocols are designed through long-standing and hard-working human efforts. Machine Learning (ML)-based solutions for communication protocol design have been developed to avoid manual effort to adjust individual protocol parameters. While other proposed ML-based methods focus mainly on tuning individual protocol parameters (e.g. contention window adjustment), our main contribution is to propose a new Deep Reinforcement Learning (DRL) framework to systematically design and evaluate networking protocols. We decouple the protocol into a set of parametric modules, each representing the main protocol functionality that is used as a DRL input to better understand and systematically analyze the optimization of generated protocols. As a case study, we introduce and evaluate DeepMAC a framework in which the MAC protocol is decoupled into a set of blocks across popular 802.11 WLANs (e.g. 802.11 a/b/g/n/ ac). We are interested to see which blocks are selected by DeepMAC across different networking scenarios and whether DeepMAC is capable of adapting to network dynamics.

INDEX TERMS Communication protocols, deep learning, machine-generated algorithm, protocol design, reinforcement learning.

I. INTRODUCTION
The proliferation of existing Internet and mobile communications networked devices, systems and applications has contributed to increasingly large, heterogeneous, dynamic and systematically complex networks. A very recent forecast by the International Data Corporation (IDC) estimates that there will be 41.6 billion connected wireless devices in 2025 [10]. This significant growth and penetration of various devices is accompanied by an enormous increase in the number of applications supporting different domains and services. The increasing availability and performance requirements of these applications suggest that “general-purpose” protocol stacks are not always adequate and need to be replaced by application tailored protocols. Ideally, protocols (e.g. for routing, congestion control, video streaming, etc) will perform well across the entire range of environments in which they may operate. Unfortunately, this is usually not the case; a protocol may fail to achieve good performance if the network conditions deviate from assumptions implicitly or explicitly underlying its design, or due to specific implementation choices made by domain experts. However, while this approach is becoming increasingly difficult to repeat, these designed protocols are deeply rooted in rigid, cradle-to-grave designs and are therefore unable to meet the requirements of different network characteristics and scenarios [5], [8]. We believe that MAC protocols have the potential to outperform their human-designed process in some scenarios as shown in [4]. ML-based protocols are not limited to human intuitions and may be capable of optimizing control-plane traffic and channel access in ways that are not yet seen.

Unlike others, we focus on decoupling protocol features/ functionalities in different technologies and scenarios. We use control parameters to accommodate our idea. Taking the physical layer as an example, no single physical-layer design can work well under all scenarios, therefore 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, OFDM subcarrier spacing and cyclic prefix length, to transmit power, etc., so that the medium can be tuned to the specified deployment scenario in the field. Each of these parameters has a number of settings that lead to a large number of choices, and it is extremely difficult for domain experts to design a control algorithm that...
chooses the correct algorithm depending on the scenario and the different network conditions. As a result of the heterogeneity and dynamically changing characteristics of networks (e.g., IoT), network protocol design requires a new approach in which control rule optimizations are not only based on a closed-form analysis of isolated protocols, but are based on high-level policy objectives and a comprehensive view of the underlying components. It has therefore become crucial to re-engineer the protocol design process and migrate towards a vision of an intelligent design process that adapts and optimizes network protocols in a variety of environment contexts, such as device characteristics, application requirements, user objectives and network conditions.

Given that the networking field often deals with complex problems that require efficient solutions, exploiting Machine Learning (ML) techniques for solving these problems looks promising. The value that such techniques can bring to the protocol design process stems from the flexibility of the protocol design process to design protocols that can learn from their past experience and the ability to respond to network dynamics and environmental conditions in real-time. Among ML techniques, Reinforcement Learning (RL) is suitable for the unknown environments where decision-making ability is crucial. Recently, Deep learning (DL) and Deep Reinforcement Learning (DRL) [29] techniques have been applied to various protocol and radio optimization tasks including routing [19], congestion control [62], MAC [46], [63] and frequency estimation in PHY layer [69], just to name a few.

To the best of our knowledge, current efforts in the application of DL to improve protocol performance focus only on tuning or controlling protocol parameters. Applying such DL-based techniques can reduce human-based manual efforts to tune protocol parameters. However, the black-box nature of DL-based techniques leaves us little insight into how they work. Joseph et al. [27] show how to design a DL-based control algorithm to jointly control two parameters, namely modulation order, and power scaling. In their work, they show that applying the DL technique can work well to control the two parameters mentioned above but depending on the context of different devices, throughput targets, etc.), it is extremely complex to get enough insight into how the black-box DL technique works, even if only to adjust two parameters from a large set of available control parameters. We believe that the optimization of protocol performance goes beyond the individual protocol parameter tuning.

In this article, we propose a novel DRL-based framework that is not only capable of tuning protocol parameters but also optimizes the main functionality of each protocol. In the proposed framework, a protocol is decoupled into a set of parametric modules as DRL inputs, each representing the main protocol functionality referred to as Building Blocks (BBs). This modularization technique helps to better understand the protocols generated, optimize the protocol design and analyze them systematically. These BBs and a set of other parameters are fed as input into the DRL agent. The DRL agent then be able to learn what protocol blocks (components) are important for inclusion in or exclusion from the protocol design. Note that our proposed framework is generic in the design of networking and communication protocols across all layers of the network stack. We narrow down our focus to propose a DRL-based framework for designing wireless MAC protocols hereafter DeepMAC as a version of the proposed framework.

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 show in Section VI, 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. This framework could therefore 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 multivariate optimization tool that helps in alleviating the current protocol design process. 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 gain insight into the relation between different protocol components for different objectives, although such components may not have a direct dependency/relation to each other if considered alone.

Contributions: This article is built on top of our prior works [47], [48]. We summarize the main contributions of the paper in the following, and we discuss the detailed contributions and differences between this article and our prior works in Section III.

i) We motivate a novel Reinforcement Learning (RL)-based protocol design approach that not only tunes the protocol parameters but also optimizes the protocol design across all network stack layers by leveraging the concept of demodulating a protocol into its set of main functionalities referred to as building blocks.

ii) In order to show the feasibility of our framework, we propose DeepMAC, a novel deep reinforcement learning-based framework that targets the design of 802.11 MAC protocols based on the given networking scenario. Evaluation results show that DeepMAC can intelligently select the optimum protocol design for a given objective (higher throughput) under different scenarios and outperforms conventional protocols e.g., CSMA/CA. By using the demodulating concept, we are able to interpret DeepMAC behavior under different scenarios.

iii) We present the future trends and opportunities that such a novel RL-based framework can bring to protocol design approaches that are more robust and adaptive to varying network conditions, application requirements, and heterogeneous device characteristics.
II. BACKGROUND

Reinforcement learning is a machine learning technique where the agent interacts with a time-variant environment that can be modeled as a Markov Decision Process (MDP), a Partially Observable MDP (POMDP), a game, etc. The core components of the RL technique are environment, reward (r), possible set of actions (A) and states (S). The state is the perception of the environment by the agent and is defined based on the sensory information of the agent. The agent selects an action from the given state and receives a reward. Actions are the agent’s methods that allow it to interact and change its environment, and thus transfer between states. The policy \( \pi \) prescribes actions to be taken in a given state. We can then value a given state \( s \) and a policy \( \pi \) in terms of expected future rewards.

Each RL algorithm follows the policy \( \pi \) in order to decide which actions to perform in each state. RL algorithms are categorized mainly into on-policy and off-policy reinforcement learning models. Algorithms that concern about the policy which yielded past state-action decisions are referred to as on-policy algorithms, while those ignoring it are known as off-policy. In other words, there are two main methods for solving RL problems: calculating the value functions or Q-values of each state and choosing actions according to those, or directly computing a policy that defines the probabilities each action should be taken depending on the current state, and acting on it. A well-known off-policy algorithm is Q-learning [60], as its update rule uses action that will yield the highest Q-value, while the actual policy used may restrict the action or choose another. The on-policy variation of Q-learning is known as SARSA, where the update rule uses the action chosen by the followed policy. For a more in-depth review of reinforcement learning algorithms, the interested reader is referred to [35], [54]. There are many different variations and assumptions that change the methods in an RL problem; the focus here is on Q-learning.

Q-learning is one of the most popular off-policy algorithms in RL. It is also regarded as temporal difference learning that learns an action-value function to find a Q-value for each state-action pair. Q-learning agent learns its optimal policy by exploring and exploiting the environment. At each time instant \( t \), the agent observes the current state \( s_t \) and chooses a proper available action \( a_t \) from this state to maximize the cumulative reward in time instant \( t + 1 \). More formally, the Q-value of \( (s_t, a_t) \) from the policy \( \pi \) which is denoted as \( Q^\pi(s_t, a_t) \) is the sum of discounted reward received at time \( t + 1 \) when action \( a_t \) is taken in state \( s_t \), and it follows the optimal policy \( \pi^* \), thereafter. The Q-values are updated using the following rule known as one of the Bellman equation forms:

\[
Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]
\]  

where \( \alpha \) is the learning rate and \( \gamma \) is the discount factor. Intuitively, the above equation adjusts the long-term or delayed rewards for a given state, action, and future state \( s_{t+1} \) by weighting the previous Q-value estimate, the reward received and the best possible long-term reward obtained in the future state. The Q-values estimate can be adjusted for any delayed reward desired. For instance, to seek short-term rewards exclusively, we can set \( \gamma = 0 \), while to estimate the history of all rewards it would be \( \gamma = 1 \).

The ability of traditional RL techniques is limited by the curse of dimensionality, and it will be inapplicable in large-scale systems [56]. Deep Reinforcement Learning (DRL) [40] has therefore been proposed to overcome this problem. DRL methods are obtained when deep neural networks are used to approximate reinforcement learning components, including value function, policy, and model. By integrating DL into RL, DRL uses Deep Neural Nets (DNNs) to overcome the curse of dimensionality and hence is able to solve large-scale problems effectively. When using neural networks with Q-learning, the Q-learning’s table is replaced by the neural network that is referred to as the approximator function that is denoted as \( Q(s,a; \theta) \), where \( \theta \) represents the trainable weights of the network. In deep Q-learning, the state is given as the input and the Q-value of all possible actions is generated as the output. In this case, the Bellman Equation is used as the cost function which is the squared error of the predicted Q-value and the target Q-value as follows.

\[
\text{cost} = [Q(s_t, a_t; \theta) - (r(s_t, a_t) - \gamma \arg \max_a Q(s_{t+1}, a_{t+1}; \theta’))|^2 \]  

(2)

Multi-agent reinforcement learning (MARL) [18] is an emerging paradigm in RL that allows multiple learners to analyze the environment at the same time. MARL is the integration of multi-agent systems (MASs) with RL and it is suitable for distributed systems. Besides issues in RL such as convergence, there are new issues like multiple equilibria and even fundamental issues like what is the question for multi-agent learning to solve, whether convergence to an equilibrium is an appropriate goal, etc. In MASs, each agent not only learns to operate independently but also cooperates with others to obtain the best joint reward. The direct strategy is to apply independent Q-learning in each agent and consider other agents as a part of the environment. However, this approach limits the number of agents because it is computationally expensive to train every agent in the system. There are different approaches to design a MARL framework, but they are beyond the scope of this article.

III. RELATED WORK

Recent breakthroughs in ML techniques have drawn the network community’s attention. By having the ability to interact with complicated environments and decision making, ML techniques provide promising solutions for higher network performance. Today, networks are getting more complicated, and traditional protocol optimization methods are becoming less effective and inefficient. More specifically,
RL has been utilized in tuning and optimizing many network sub-fields [37] such as Wi-Fi [21], [28] in PHY [20], [41], [44], MAC [32], [46], and upper layers [6]. Since this article focuses on communication protocols that apply RL/DRL techniques, readers are referred to [53] for a general survey of applications of machine learning in communications and networking. In the following, we elaborate on the RL and DRL applications in the network stack in a top-down manner as follows. We first start by discussing RL-based congestion control algorithms in the transport layer. We then review the applications of RL in routing protocol design. Finally, we delve into the RL-based PHY/MAC protocols.

There have been several efforts that applied RL to congestion control in specialized domains. Prior work in [52] employed RL to create a cooperative congestion control controller for multimedia networks. RL has also been used to solve congestion problems in wireless sensor networks [67]. Authors in [2] explored designing a TCP-style congestion control algorithm using Q-learning. The recent effort [26] proposed a DRL-based adaptive framework for congestion control. In this model, state is bounded histories of network statistics (e.g., sending rate, latency), action is periodically tuning the sending rates, and the reward is a linear function that rewards throughput while penalizing loss and latency. A promising approach to multipath congestion control is to use reinforcement learning. Deep Reinforcement Learning Congestion Control (DRL-CC) [62] algorithm jointly sets the congestion window for all active flows and all paths and achieves high fairness in a wired network scenario with multiple active flows. The results showed that DRL-CC significantly outperformed a few of the conventional congestion control algorithms in terms of goodput without sacrificing fairness. Authors in [61] proposed and evaluated Remy tool that generates congestion control algorithms to run at the endpoints rather than manually formulate each endpoint’s reaction to congestion signals. Remy is a heuristic search algorithm that maintains rule tables in which states are mapped to actions. Unlike RL-based approaches, Remy is static meaning if the actual network conditions change, performance could potentially degrade substantially.

The literature abounds with a variety of attempts to utilize RL for optimizing routing protocols optimization in different contexts. These efforts span different levels of routing management that is a crucial aspect for traffic control/engineering as the poorly chosen paths can lead to network congestion. The prior work [57], for instance, studied ML-guided routing by examining the classical environment of intradomain traffic engineering for the optimization of routing within a single, self-administered network. Some of the RL-based routing protocols focus on customizable, and fine-grained routing management for the SDN. For instance, DROM mechanism [65] was proposed to optimize the routing in the SDN by utilizing the Deep Deterministic Policy Gradient (DDPG). DROM simplified the network operation and maintenance by improving the network performance, such as delay and throughput, with a black-box optimization in continuous time. RL is also used for designing routing protocols in WSNs. Authors in [23] proposed QELAR an RL-based routing protocol for routing underwater sensor networks with the main focus on energy efficiency as a major design concern for sensor networks. In QELAR the definition of system states is related to individual packets i.e., if a node has a packet the system state depends on that packet. The action is to forward the packet, and the reward function is based on the fact that each packet forwarding attempt consumes energy, occupies channel bandwidth, and contributes to the number of hops to the destination (i.e., delay). For a comprehensive survey on RL for routing, interested readers are referred to [36].

Similarly, RL techniques are playing a critical part in the MAC layer design for WLANs and Wireless Sensor Network (WSNs). Recent years have witnessed a wide study on deep reinforcement learning (DRL) in the field of Dynamic Spectrum Access (DSA) problems in wireless networks. In particular, Naparstek and Cohen [42] considered the problem of DSA for network utility maximization in multi-channel wireless networks. The authors proposed a novel distributed dynamic spectrum access algorithm based on deep multi-agent reinforcement learning. In their mechanism, the objective is to maximize a certain network utility in a distributed manner without online coordination or message exchanges between users/agents. In [59], the DSA problem was formulated as a POMDP with unknown system dynamics. In this framework, a sensor at each time slot selects a channel to transmit data and receive a reward based on the success or failure of the transmission. To investigate the multi-channel problem, a DRL-based adaptive modulation and coding scheme was developed in [68] for primary users to learn the interference pattern of secondary users in cognitive radio networks. The authors in [12], [71] also investigated the multi-channel access problem where the users aim to maximize their own throughput by learning the channel characteristics and the transmission patterns of the “primary” users.

RL has been employed to develop medium access control schemes in WSNs and wireless networks. Unlike this article, these works do not leverage the recent advances in DRL. One of the early works in RL-based MAC protocol optimization is S-MAC [64] that is a MAC protocol developed for duty-cycled WSNs which aims to minimize idle listening time to reduce energy consumption. To this end, it applied RL to negotiate a schedule among nodes that specified when nodes are awake and asleep within a time frame. ALOHA-QIR [13] is another example of RL-based MAC protocol in WSNs which combined slotted ALOHA and Q-learning. In ALOHA-QIR, each node stores a Q-value for individual time slots in which the agent’s action is to select the slot with the highest Q-value for its next transmission, and the reward function is a simple numerical value. ALOHA-QIR obtains significant improvements in terms of throughput, delay, and energy efficiency when compared to basic slotted ALOHA. Similarly, the work in [7] used Q-learning to enable wireless nodes to learn their own optimal transmission
strategy using the historical sensory information that defines the node state including the number of transmission in the previous slot, number of consecutive idle or useless slots, and number of consecutive collisions. The strategy of an agent is defined as the probability of transmission. Authors assumed that agents do not have prior information about the network and learn their optimal strategy using the number of collisions or successful transmissions’ information. Recently, Alfredo et al. [14] proposed a policy-based RL approach to improve slotted ALOHA in terms of fairness. Unlike ALOHA-QIR, they considered a dynamic time frame, and each node learns the best time slot to transmit a packet based on its local policy tree. Authors assumed that the nodes can detect the state of the time slot, and thus, the fate of their transmissions. The proposed algorithm was shown to be better than slotted ALOHA with exponential backoff and framed slotted ALOHA with Q learning (ALOHA-Q) in terms of throughput and fairness.

Few prior works focused on DRL-based MAC protocol optimization in wireless networks. Authors in [66] investigated a DRL-based MAC protocol for heterogeneous wireless networking. This work considered the scenario of a number of networks operating different MAC protocols including TDMA, ALOHA, and the proposed DRL-based protocol all trying to access the time slots of a common wireless medium. They assumed that the DRL agent has no prior knowledge about other nodes. The proposed method objective was to maximize the sum throughput and maximize α-fairness among all networks’ nodes that access the shared medium. In [3], the conventional CSMA/CA protocol was improved for densely deployed WiFi networks, in which DRL was adopted to learn the optimal CW for each WiFi node to improve overall throughput. Cao et al. [9] proposed a DRL-based MAC protocol to assist the backscatter communications for IoT networks, where the DRL was introduced to learn the reserved information and make decisions accordingly.

Although the aforementioned mechanisms differ in the details, their common objective is to optimize a protocol by tuning and/or controlling the protocol parameters. Our proposed approach is different from these works in the following respects. First, we argue that designing methods to boost protocol performance is not only about parameter tuning, but also to decide what functionality to include or exclude from the design. The novelty of our approach resides in the way our framework constructs a protocol from a set of building blocks. By decomposing the protocol into the set of mechanistic building blocks, we aim to better understand the design, the interdependencies among different protocol building blocks, and to ease the analysis of the protocols. In theory, a self-managed protocol design framework would identify where and when to execute design tasks. This process includes re-configuring the protocol by choosing the right set of building blocks when necessary, predicting performance problems due to unpredictable network changes. In addition, our framework supports multi-variant communication objectives that could be explicitly defined by domain experts. By using this framework, domain experts provide the required specifications for a specific scenario as DRL input and could identify/capture the role that each protocol block plays in varying scenarios for different objectives. It could also help to get insights into the relationship between different protocol components for different objectives even when such components may not have a direct relation to each other if considered alone.

In our previous works [47], [48] we targeted IEEE 802.11 MAC protocol design. MAC protocol was decoupled into its set of building blocks and then the agent selected the appropriate protocol blocks based on varying network conditions. In this article, we build and expand on our earlier works [47], [48] in different aspects. Mainly, in this article, we propose a general Reinforcement Learning (RL)-based framework that could be adapted to optimize the design of not only MAC protocols but of communication protocols across all network stack. In the proposed framework, we expand the concept of protocol building blocks and describe the challenges such as the level of granularity and the dependency of blocks on each other. More formally, we propose to model a building block as a tuple that captures the main functionality of the block, its internal state, and dependency status to other blocks. Furthermore, we thoroughly describe the architecture of the RL agent, the different design decisions of the RL agent i.e. the centralized or distributed RL agent, their pros and cons. We also describe the impact of the reward function and whether it should be optimized locally or globally. Unlike our previous work that only focuses on MAC protocols, this article comprehensively reviews the main RL-based state-of-the-art techniques and architectures for protocol optimization across different layers of the network stack. We also extend our earlier version of DeepMAC design by expanding its building blocks and demonstrating how the DCF protocol can be designed using the proposed building block concept. In addition, we extend on our earlier evaluations and investigate the convergence and adaptability of the RL agent based on past experience in subsections VI-B and VI-C. Finally, we discuss the opportunities that such a novel RL-based framework can bring to protocol design approaches that are more robust and adaptive to varying network conditions, application requirements, and heterogeneous device characteristics.

In Table 1 we summarize additional works similar to what we have discussed about RL formulation for protocol enhancement in different network stack layers.

IV. FRAMEWORK FOR NETWORKING PROTOCOLS DESIGN

A. FRAMEWORK OVERVIEW

In this subsection, we describe our proposed reinforcement learning-based framework that can be generalized to design different types of protocols with different levels of complexity for any layer in the networking stack. Figure 1 illustrates the proposed framework overview.
As shown, the RL agent selects a set of building blocks from the existing blocks, along with the network configuration, and the reward feedback as input, and designs a protocol accordingly. The designed protocol is then evaluated (e.g., through simulation), and a reward (e.g., mean throughput of the link) is calculated and sent back as feedback that signals the agent about the current protocol performance. This process iterates until a protocol with an optimum set of building blocks for the current network/environment configuration is designed.

In the following subsections, we explain the main components of the proposed framework.

### B. PROTOCOL BUILDING BLOCKS

A network protocol is structured into several layers. Each layer is broken into a set of blocks with its own specific functionality. Building blocks are a set of separated parametric modular components, each of which is in charge of one (or several) specific well-defined functionality [15], [38], [55]. The combination of different blocks and the interactions between them determine the overall behavior of a network protocol for a given environment. Once blocks and their interactions are established, 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. By conducting the operations of individual blocks in an appropriate order, we are then able to implement the protocol mechanisms.

Yet among the main challenges to developing a flexible and reusable set of building blocks is 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 brute-force in which the interactions, relations, dependencies, and conflicts between every couple of blocks are
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, and whether this process could be automated.

Another main challenge is to check the sanity and correct execution order of the selected blocks that will be used in protocol design. Such a sanity check requires understanding network condition impact on the selection of the blocks as well as logical dependencies among the blocks themselves to design a proper protocol. Therefore, we need to capture both the effects when formulating the building blocks which we discuss in more detail in the following.

The network protocol is operated under a variety of conditions and environments, which trigger events causing the protocol to act. Following the modular design principle in the context of protocol design, two main branches exist on how to divide protocols and define the components: Finite State Machine (FSM) and Data Flow. FSMs are formalisms 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. To overcome this, in the literature, prior works [55], [58], [70] use extended FSMs to model MAC protocols based on the block(modular)-based concept to add flexibility to protocol design, and reusability of the components. The existing (block-based) frameworks consider either a static way of connecting the components together or add a limited degree of dynamics in binding the components together. Therefore, these models do not fully capture the interactions, relations, dependencies, and conflicts between different components.

In order to support the dynamic design of the 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 building blocks is of crucial importance. 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 in proper order. To check the sanity and correct execution of the blocks, one approach that we have used is to design a logic controller module in which the predefined rules are embedded using if-then-else statements. 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 not only based on its main functionality, but also by capturing events and dependencies among blocks as the following tuple:

\[
\text{Block} : < E, P, S, F, D >
\]

where \( E \) is the Event is a set of signals either provided by the hardware interrupt block, or coming from the upper layers, which triggers the block. \( P \) is a set of Parameters inside the block that could be tuned, \( S \) is the State of the block, \( F \) is the main Function that is executed in the block, and \( D \) represents the possible Dependencies between a block with other blocks.

In our approach, the dependencies are uni-directional meaning if Block \( A \) depends on Block \( B \) it only shows restrictions of \( A \) to \( B \) but not \( B \) to \( A \). Blocks can have different types of dependencies between them that we have mainly categorized into: strong, weak, and conditional dependency. In our framework, a strong dependency is between those blocks that are tightly wired together and must be selected together in order to deliver their functionality properly. A weak dependency is identified, when a block can be updated/executed based on the output of the other block. However, if the latter block does not exist in the protocol design, the former block may use a predefined specification for its execution. Conditional dependency captures the logical order of the execution of the blocks. For example, if we have a block called Sending Packet to transmit a packet then it can be executed as it is, but if we have a Carrier Sensing block present, then the Sending Packet block is executed after Carrier Sensing.

To show an example using Tuple 3, let’s consider the Backoff mechanism as a single building block. A tuple that describes this block could be:

\[
< \text{ACK\_timeout}, \text{CW}, \text{Freeze}/\text{Countdown}, \text{BEB}, \text{ACK} >
\]

Acknowledgment time out referred to as ACK_timeout in the above tuple is the event that triggers the Backoff block, since it indicates that the frame transmission was unsuccessful. Contention Window (CW) is the parameter that can be tuned to different values to set the Backoff counter range, the state is the state of the Backoff counter which could either be counting down when the medium is sensed to be idle, or it could be frozen when the channel is sensed busy. The last component of the tuple is dependency which indicates whether the Backoff mechanism is dependent on any other block. In this example, the Backoff mechanism is strongly dependent on the ACK block (Backoff \( \rightarrow \) ACK), since if there is no ACK block present, then there is no ACK_timeout event to trigger the Backoff. Although the other direction does not hold, meaning we can use ACK without having a Backoff mechanism in a designed protocol.

C. RL AGENT

The agent takes both the protocol building blocks and the network/environment configuration as inputs in order to utilize the building blocks and their interactions in designing the optimum protocol that fits with the network requirements using reinforcement learning mechanisms.

In RL, the agent training continues with each Q-value function update. The offline RL training happens when the agent runs in a variety of scenarios and learned Q-values are stored such that the agent uses them when making new decisions. In online RL, typically, the agent starts from an initial state, and its training evolves over time meaning the agent does not have any prior knowledge (Q-value) when it starts from the initial state.

RL agent can be implemented in centralized or distributed approaches. A centralized agent means there is a single agent
responsible for managing the protocol design task, and then the designed protocol is enforced to be used by all the nodes in the network. The decentralized approach assumes that multiple agents perform the task of learning based on their own knowledge, including what actions to take based on the current state and expectation of other agents’ actions. Although in a distributed approach (i.e., multi-agent environment) each agent has the flexibility to design its own optimal protocol based on its characteristics and application requirements, instability throughout the network could happen as some agents may take random actions that can affect the learning process of other agents. However, having different protocols makes it impractical for agents to communicate with each other since they all have to agree on the same protocol to be able to talk to each other. On the other hand, while a centralized approach is simple and easy to control and manage, it becomes computationally expensive when the number of nodes in a network grows or the state space becomes large. Moreover, a centralized approach is not suitable for a heterogeneous environment where different nodes have different objectives.

Combining the benefits of both centralized and distributed approaches, a hybrid approach could be designed for RL agents [22]. The work by Kraemer and Banerjee [30] proposes a hybrid solution in the context of Decentralized partially observable Markov decision processes (Dec-POMDPs) for modeling multi-agent planning and decision-making under uncertainty. More specifically, they proposed a centralized approach, where a group of agents can be guided at the same time by using a centralized algorithm via an open communication channel. Then, after the training, agents are allowed to communicate over a channel, and thus can operate freely in a decentralized manner. A similar approach could be adopted for a hybrid solution in protocol design context. A design strategy could classify the building blocks and their interactions into global and local blocks. Global blocks are the ones mainly responsible for designing the main skeleton of the protocol in which all nodes have to adapt/use the same selected blocks. On the other hand, local blocks are the ones that could be selected, adapted, tuned, and configured individually based on the local context of each node.

Another design challenge especially for distributed and hybrid approaches is the communication strategy between agents in order to collectively improve their performance. The communication mechanisms between multiple agents could be mainly categorized into two main approaches; individual peer-to-peer channels and all-to-all channels. Peer-to-peer channels will enable peer agents experiencing similar conditions and targeting similar rewards to exchange their information with each other in order to speed-up the learning process and be able to converge to the optimum protocol within time constraints. Additionally, if the design includes a Long Short Term Memory (LSTM) that stores previous experiences and observations of agents over time, an agent that is currently learning a new protocol may utilize information from another agent who dealt with similar tasks in the past. All-to-all channels will allow sending crucial information that will affect all agents in the system in order to adapt the protocol design such as changes in time constraints or discovery of new patterns in data.

D. REWARD FUNCTION
The essential goal of RL agent is to learn a policy to select actions that maximize its expected rewards for state-action pairs. The agent learns the policy by interacting with the environment and observing the reward function in every state. Reward function varies from a simple performance metric such as total network throughput to a complex formula. Using different reward functions will generate different protocol designs for the same network scenario. Therefore, the reward function should be designed carefully based on device characteristics and application requirements. For example, with embedded devices such as Internet-of-Things (IoT), power consumption would be more important than throughput, and hence a protocol design that minimizes the retransmission is needed. In this case, the reward function needs to consider the number of collisions and packet corruption in its formula in order to minimize the retransmissions.

Another important design decision is how to design and optimize the reward function which can be optimized globally or locally. 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 has its 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 assumption 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. As an example, if each node wants to maximize its own throughput, it sends as much packet as possible without having any idea how happy or unhappy the other nodes are. In [51], authors bring a symmetric parking-lot topology example as a classical congestion control problem. They use PCC [16] to generate congestion control protocols in which each flow explicitly performs local optimization of an objective function. In this example, authors show that the link allocation found by PCC can vary widely, depending on the initial rates or which flow starts first, even though they all last forever once they get started which leads to unfairness in link allocation. Therefore, it is very important to understand which reward optimization works better depending on the given task.

V. DeepMAC: A CASE STUDY FOR 802.11-BASED MAC PROTOCOL DESIGN
MAC protocols need to be designed with a rich set of requirements in order to satisfy the needs of the overlaying applications and network conditions. The huge diversity of possible network conditions implies that even a protocol that works well across a wide variety of network conditions may suffer
from bad performance on other networks. In the case of MAC protocols, 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 addition, it is required that the effective throughput remain high irrespective of the traffic levels. Overwhelmingly, the main challenge is the dynamicity of network conditions (e.g., nodes entering and leaving). Thus, it is imperative that the MAC protocol can be easily scalable and adjusted delicately to the changing environment with little or no control information exchange.

To overcome the aforementioned challenges, we implement and evaluate DeepMAC which is a DRL-based framework that optimizes MAC protocols for a given networking scenario. In Q-learning, if the combinations of states and actions are too large, the memory and the computation requirement for Q will be too high. Therefore, we use a deep Q network (DQN) to approximate the Q-value function. Figure 2 shows DeepMAC framework and its key modules that aim to optimize the design of wireless MAC protocols. We describe the key modules in the following subsections.

**FIGURE 2.** DeepMAC framework.

### A. DeepMAC BUILDING BLOCKS

In our framework, we have extracted a set of MAC protocol blocks from Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications [24] which includes the basic MAC DCF functionalities across all 802.11a/b/g/n/ac amendments that are shown in Figure 2. The realization of IEEE 802.11 DCF using these coarse-grained MAC blocks (namely Carrier Sensing (CS), ACK, Backoff, RTS/CTS, Fragmentation, Packet Transmission) and their corresponding parameters is shown in Figure 3. The modular concept of building blocks makes protocol design more flexible. As shown in the figure, if the RL agent does not select the Carrier Sensing block in the protocol design for a given scenario, this block can be removed and the protocol execution sequence switches to the next logically selected block. As mentioned earlier and described later in more detail, in our framework the logic controller checks the sanity and execution order of the blocks using embedded if-then-else rules. Having established a number of potential building blocks, the DRL agent takes these blocks and history of the average channel throughput as its main inputs as described later.

### B. DeepMAC AS A REINFORCEMENT LEARNING PROBLEM

DeepMAC uses Reinforcement Learning (RL) along with a deep architecture to learn the best set of protocol blocks for different scenarios. In DeepMAC, we consider a centralized learning agent for the design of 802.11 MAC protocols. This centralized agent, in practice, can be placed on a single supernode (e.g., the Access Point) that periodically updates its model. The supernode decides the selected set of MAC layer blocks and parameters to be used by all the other nodes in the network. We elaborate on the DRL agent’s states, actions, and rewards, below.

1) **STATE, ACTION**

The **state** of the agent is a vector of numerical representation of the set of the building blocks, and a history with a fixed link of the average link throughput values which are used as part of the input for DeepMAC agent. In this set, for each block, 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 block 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. Given the input, the output of the simulator is the average throughput of the channel which is considered as the reward of the DRL agent for the selected building blocks at the current step. We assume that all nodes employ the same prescribed protocol using the selected blocks by the centralized agent. The agent takes both the protocol design blocks and history of the reward as inputs and outputs the best combination of building blocks for the current scenario that maximizes the reward.
2) REWARD FUNCTION
The designers need to specify the communication objective according to the corresponding use case, device category, etc., under different scenarios. In the DRL framework, this defines the reward function. For example, for a battery-constrained IoT device that cares about maximizing the throughput while minimizing spent energy as discussed in [31], [49], a designer may specify to optimize the following reward function:

\[
W_0 \text{ (number of successful transmitted bits)} - W_1 \text{ (energy spent per bit)}
\]

where \(W_0\) and \(W_1\) control the relative tradeoff between these two conflicting components. In DeepMAC, the reward function is the average throughput of the link. Although such reward objectives can change based on the provided scenario by the protocol designer.

3) DRL AGENT ARCHITECTURE
The neural network we adopted is equipped with three hidden layers and an output layer. We find through our experiments that this simple architecture can yield satisfactory performance, and increasing the complexity of the neural network does not contribute to performance improvements while inducing more training overload. 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. LOGIC CONTROLLER
Regarding network protocols, some functional blocks are dependent on each other. In our framework, the logic controller is responsible to check the sanity of a generated protocol. More specifically, the designed logic controller checks a) the blocks’ execution sequences b) their interdependencies, and c) interaction rules among blocks to ensure logically correct protocol design. We extracted the interdependencies among different blocks from PHY and MAC specification [24] and incorporated them in the logic controller using if-then-else rules. We provide the following examples to describe some of such dependencies in the following. As shown in Figure 2, NAV (Network Allocation Vector) which is considered as a virtual CS mechanism is conditionally dependent on RTS/CTS block. There are two methods to set the NAV parameter: a) by reservation information distributed through the “RTS/CTS Duration” field and b) by information provided in the Duration/ID field in individually addressed frames. Therefore, in our framework, NAV is set based on the RTC/CTS block if this block is available (selected by the agent). Otherwise, it is set based on the latter approach.

D. NETWORK CONFIGURATION AND DESIGNED PROTOCOL EVALUATION
Integrated with protocol design elements would be the network scenarios and conditions, such as communication medium types and node mobility. Different scenarios have different assumptions and requirements that need to be captured when designing a protocol. In DRL, an agent repeatedly observes the environment’s (simulator) state, performs an action, and then observes the reward for the performed action. Our goal is to train DRL that outputs network protocols that trigger and exploit flaws in existing protocols. Training the network on a wide range of protocols ideally aids in selecting a protocol that shows the best performance for different network configurations. Observing blocks selected by the DRL agent for different network conditions provides some insight into what protocol flaw is being exploited. To evaluate DeepMAC framework, we developed an event-driven simulator using C++, while having the ns-3 design in mind. Our simulator mimics the MAC protocol of ns-3, but it is flexible to support the decomposed building blocks, and consequently the design of MAC protocols. Each building block is considered as a module and the agent decides about the inclusion and exclusion of the block as a part of protocol design. As input, the simulator takes the values of building blocks from the DRL agent that passed the logic controller check of finding any type of conflict or interdependency between them. It also receives the network configuration parameters including the number of nodes, level of noise, etc.

VI. DeepMAC EVALUATION
This section presents the numerical evaluations of DeepMAC in terms of a) convergence b) average throughput enhancement and c) block selection by the agent under different scenarios, respectively. 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 real-time requirement. We clarify that our RL agent uses an online approach where it has no prior knowledge about the underlying environment.

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 event-driven simulator. Table 2 summarizes the simulation configuration parameters used in our experiments. The nodes are static and are randomly scattered in a 200 × 200m area. In our experiments, we consider eight different networking scenarios described in Table 3. The low load scenarios correspond to an under-saturated network with 5 nodes, and an average packet generation rate of 8 packets per second, while scenarios with high load represent close to saturated, and saturated networks with 20 to 50 nodes, and an average packet generation rate of 470 packets per second. With regards to noise, when the noise is not present, the received packets are assumed to be delivered with no error with probability 1, while when the noise is present a fixed bit error rate (BER) of 0.0001 is considered. Scenario 1, for

1 High-end simulation tools (such as Opnet, NS-3, etc.) have the ability to reproduce with an accuracy of implementation. However, such tools do not support our building block decomposition concept properly.
example, corresponds to a network having 5 nodes with a low traffic load that represents an under-saturated network while noise is absent. Table 4 includes the blocks and their associated algorithm, mechanism, or parameters and the default values that are used by DeepMAC framework for the experiments. Some blocks have different algorithms or parameters. As an example, if the fragmentation block is not selected by the agent, then the frame size remains 1500 bytes in the corresponding scenario. Otherwise if selected, the frame size varies. In order to see what blocks are selected by the agent in different scenarios, the evaluation for each scenario is performed 20 times. We then collect those blocks that are selected together more frequently than others by the agent over 20 rounds of repeating each scenario.

### B. LEARNING CONVERGENCE RATE

We first evaluate the learning convergence rate of the RL agent to the goal state. We measure the convergence rate based on the number of times that the agent selects the goal state which includes those building blocks that contribute to the highest throughput of the channel for a given scenario. Although the goal is unknown to the agent, as a ground truth we assume that we know the goal state in order to count how many times the agent visits this goal state in every episode. Each episode contains 100 steps, and each step is considered as the action of selecting a state and updating the corresponding Q-value.

In addition, network channel condition is dynamic which makes the previous optimal policy learned by the agent inefficient. Therefore, the agent must find a new optimum policy to adapt to new changes. As it is not efficient to retrain the RL agent each time the characteristics of the channel change, we want to investigate whether the previously learned Q-values are helpful or harmful for converging to the new optimum policy. We evaluate RL convergence under eight different scenarios before and after the network is trained for that particular scenario. We carry out a set of experiments in which we compare the convergence of the agent for all combinations of different scenarios. Due to the space limitation, we only show how to analyze the results of Scenario #1 and #5, while the results of other scenarios resemble similar behavior. To evaluate the convergence rate when the agent starts to learn from an initial state, we let the agent learn the optimum goal states in Scenario #1 and #5, respectively. We then initialize agent Q-Values with those values learned from a different scenario namely Scenario #4 to evaluate whether the learned Q-values from Scenario #4 helps the agent to converge faster or slower compared to the case when all Q-values are set to zero.

Figures 4 and 5 show the convergence of the RL agent in Scenario #1 and #5 without any prior learned Q-values and with Q-values learned from Scenario #4, respectively. As shown in Figure 4, we can observe that the previously learned Q-values can actually help the agent for faster convergence to the goal state, while the agent takes more steps to converge when it is not trained (all Q-values are set to 0 initially). In the second scenario shown in Figure 5 the previously learned Q-values from Scenario #4 are used to train the agent in Scenario #5. As we observe, using the previously trained values slows down the convergence of the agent compared to a scenario without training. This is because the reward values of these two scenarios have a significant difference meaning the agent needs an additional number of steps to get out of the sub-optimal states and find the new goal state. Therefore, when there is a drift in network condition we first should be able to measure the significance of the drift and decide whether the agent should start from an initial state or it is helpful to make decisions based on the previous observations. However, identifying significant network drifts requires to be carefully investigated for a wide range of scenarios that we
plan to address in our future work. In addition, as shown in the graphs, the RL agent converges approximately after 50 episodes in both scenarios. The RL agent number of visits per episode fluctuates after convergence and never goes all the way up to 100 visits (which is the maximum number of visits in an episode). These oscillations are caused by the exploration versus exploitation ($\epsilon$-greedy) action selection policy. This behavior is expected since Q-learning with $\epsilon$-greedy approach will converge to the Bellman equation, but its behavior will not converge due to random action selection with a frequency of $\epsilon$ ($\epsilon = 0.05$). In the literature, one approach to smooth the curve in the graph is $\epsilon$-decay which decreases $\epsilon$ value over time. However, in our approach, we have selected a fixed $\epsilon$ value experimentally.

C. DeepMAC VERSUS ALOHA

In ALOHA, a node with a packet to send simply transmits. However, the simplicity of Pure ALOHA (referred to as ALOHA hereafter) comes at the price of poor performance, with a maximum throughput of only 18% of the available bandwidth. As a result, several variants of ALOHA have evolved over the years to allow more efficient sharing of common channels in untethered networks. We are interested to see whether DeepMAC framework is able to design optimum simple ALOHA-like protocols for a given scenario. We compare the performance of ALOHA under eight different scenarios with DeepMAC. To design ALOHA-like protocols, DeepMAC is set to use a limited set of building blocks (namely ACK, Backoff, Contention Window, Carrier Sensing, Data Transmission Rate) that could only construct an ALOHA protocol. ALOHA is originally designed for wired networks and does not include ACK. However, we have modified ALOHA to fit the wireless network configuration. The intuition behind this comparison is to see whether the agent is able to perform better when it has the same or even less number of functionalities (building blocks) compared to ALOHA. To this end, Figure 6 illustrates higher throughput gains of DeepMAC against ALOHA in all the scenarios. Due to its self-adaptive characteristics, DeepMAC uses the set of blocks in different scenarios to enhance the throughput. For example, in Scenario #1 where the network load is low, DeepMAC removes ACK and Backoff control mechanisms to gain a higher throughput, while in Scenario #6 where the traffic the load is high and the noise is present it includes ACK and Backoff to enhance the number of successful transmissions. Therefore, this consideration shows that having an intelligent agent that decides under what scenario which components could be beneficial is important in terms of performance gain.

D. DeepMAC VERSUS IEEE 802.11

IEEE 802.11 is one of the most popular wireless protocols that is based on CSMA/CA (Carrier Sense Multiple Access/Collision Avoidance) mechanism, which is the contention-based medium access control base for many of the current wireless protocols. In this section, we are comparing the performance of DeepMAC with IEEE 802.11 and consequently CSMA/CA mechanism.

IEEE 802.11 uses Binary Exponential Backoff (BEB) technique to randomize each node attempt of transmitting in order to reduce collisions. However, IEEE 802.11 random Backoff is decentralized and unable to efficiently handle collisions. Therefore, the network throughput degrades when the number of competing nodes increases. Given that collisions cannot be detected, IEEE 802.11 uses ACK mechanism to determine the successful reception of a packet. In addition, the RTS/CTS mechanism is introduced that can effectively ameliorate the hidden node problem. Although exchanging these control packets (i.e., ACK, RTS/CTS) is useful for successful packet transmission, they could introduce extra overhead for bandwidth utilization. In the following, we describe the throughput gains of DeepMAC versus IEEE 802.11 DCF (and CSMA/CA) in two different traffic loads: Low and High with different number of nodes and the assumption of no noise. Such assumption helps to simplify the effect of avalanche rate when multiple transmission rates are available.

1) LOW TRAFFIC LOAD

As illustrated in Figure 7(a), the low load traffic corresponds to a varying number of nodes from 1 to 15 that covers scenarios #1 and #3 in Table 3. In these scenarios, every 3 seconds a new node joins the network, and the simulation duration lasts for 45 seconds. As illustrated in Figure 7(a), IEEE 802.11 fails to fully utilize the channel bandwidth, while
DeepMAC protocol effectively adapts to the network load changes by selecting the appropriate set of building blocks. Looking closely at the graph, when there is only one node, DeepMAC performs slightly better than IEEE 802.11 since it removes extra overheads (ACK and CS) from the design. Overall, DeepMAC improves channel throughput by ∼6%.

By looking closely at the selected blocks by DeepMAC in Table 5 when the number of nodes is 3, we observe that the “No ACK” mechanism is selected along with “Aggregation” that both can enhance the throughput by reducing extra control frame overhead. Interestingly, the DRL agent learned that when the load of the network is low, it could eliminate control packets (e.g., ACK) to increase the throughput of the channel. These observations may look trivial for a human expert, but it makes it interesting when a DRL agent is able to learn such intuition on its own. In our implementation, when the “No ACK” mechanism is selected we assume that the transmission reliability (e.g., ACK) is handled in the upper layers (e.g., TCP protocol in Transport layer). However, this raises the issue of cross-layer optimization since an unreliable MAC may impact TCP performance in which TCP assumes a packet loss is due to congestion, while it could be due to wireless link interference. We discuss cross-layer design optimization as one of our future directions later in Section VII.

TABLE 5. Selected blocks by DeepMAC under different networking load.

| # of Nodes | Traffic Load | Blocks Selected by DeepMAC |
|------------|--------------|----------------------------|
| 3          | Low          | No ACK, Aggregation         |
| 15         | Average      | ACK, Fragmentation, BEB, CW |
| 45         | High         | EIED, CW, CS, ACK, Fragmentation, RTS/CTS |

2) HIGH TRAFFIC LOAD

In the second experiment, we consider high load traffic that corresponds to scenarios #5 and #7 in Table 3. At the beginning of the experiment 25 nodes are competing for the channel. At every 2 seconds, a new node joins until the number of contending nodes reaches 50 as shown in Figure 7(b). The starting point of this figure is the continuation of Figure 7(a). As we observe, DeepMAC performs better than IEEE 802.11 even when the number of the nodes increases to 50 (∼2% higher throughput) when the performance of both of the approaches degrades. By looking at Table 5, we observe that DeepMAC has selected EIED (Exponential Increase Exponential Decrease) over BEB. This can be because the slower reduction rate helps improve saturation throughput. Besides, control packets are also selected by the agent to likely avoid collisions and consequent retransmissions of large data packets.

E. ANALYSIS OF THE SELECTED BLOCKS UNDER DIFFERENT SCENARIOS

This subsection focuses on understanding the reasons behind the selected blocks by the agent under different scenarios. The selected blocks are shown in Table 6 in which the highlighted cells indicate the selected blocks in each scenario along with their selected values (if the block has a tunable parameter), while white cells indicate that the corresponding block is not selected in the given scenario. In the following, we divide our observations about DeepMAC behavior into three parts and further discuss each case individually.

TABLE 6. Blocks selected by DeepMAC agent.

| # | OB | BEB | EIBE | CS | CW | No ACK | ACK | Fr | Ag | RTS/CTS |
|---|----|-----|------|----|----|--------|-----|----|----|--------|
| 1 | 24 | 15  | 31   | 31 | 15 | 5000   | 200 |    |    |        |
| 2 | 24 | 15  | 31   | 31 | 15 | 5000   | 200 |    |    |        |
| 3 | 24 | 15  | 31   | 31 | 15 | 5000   | 200 |    |    |        |
| 4 | 24 | 15  | 31   | 31 | 15 | 5000   | 200 |    |    |        |
| 5 | 24 | 15  | 31   | 31 | 15 | 5000   | 200 |    |    |        |
| 6 | 24 | 15  | 31   | 31 | 15 | 5000   | 200 |    |    |        |
| 7 | 24 | 15  | 31   | 31 | 15 | 5000   | 200 |    |    |        |

1) LOW LOAD WITH/WITHOUT NOISE

In scenarios with the low load when the noise is absent (e.g., Scenario #1), as shown in Table 6, cells corresponding to control packets such as ACK or RTS/CTS are not selected by the agent. This observation is justifiable. Even though the control packets are much smaller than the data packets, the time spent on control packet transmission is not negligible. Therefore, when the network is undersaturated, and the number of competing nodes 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, it adds Career Sensing (CS) block. This may be due to the fact that the agent learns such a mechanism could be useful when the throughput drops.

2) AVERAGE LOAD W/O NOISE

For scenarios with the average level of noise (Scenario #3 and #4) except common ACK mechanism selection, there is no obvious pattern. This observation could be either because such scenarios are not able to capture the useful information of what specific blocks should be selected, or it is simply because selecting different blocks does not provide a significant difference in the achieved throughput in such scenarios.
3) HIGH AND SATURATED LOAD WITH/WITHOUT NOISE
We divide our observations into three parts for the following scenarios: (1) The first observation in the high and saturated scenarios (Scenario #5 to #8) is the ACK mechanism that is selected by the agent. Intuitively, this could be because the agent learns such a mechanism can contribute to prevent more number of collisions and retransmissions to enhance the throughput. (2) When comparing scenario 5 to 6, we observe that the agent activates the Fragmentation block. The size of the sub-frames in practice plays an important factor that can impact the network throughput performance for a given channel condition. The larger packets could contribute to 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 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 since 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. The results demonstrate instead it is helpful to use machine learning techniques to optimize the design of control algorithms as well as, to get insights into what functionality (block) is useful under what scenario.

VI. DISCUSSION/FUTURE DIRECTIONS
The proposed protocol design framework opens up the possibility of future research directions in different dimensions. In the following, we discuss some of these directions.

A. BUILD A LIBRARY OF PROTOCOL ELEMENTS
One of our main future directions is to develop and build an open library/dataset of protocol design building blocks to be utilized by the networking and Machine Learning (ML) research community to enhance the communication field. We aim to target to identify and develop the basic building blocks of popular variants IEEE 802.11 standards (i.e., a/b/g/n/ac/ax).

B. GET NEW INSIGHTS INTO PROTOCOL DESIGN
One of the most attractive aspects of ML techniques lies in their ability to discover new knowledge that could go beyond the current human understanding and perception. Current protocol design approaches are limited to human perception and understanding of this field, thus limiting the potential for extracting new and unexpected insights during the design process. Therefore, by applying ML techniques to protocol design, one may clearly see an exceptional potential in developing more efficient, robust, and autonomous systems for protocol design. Additionally, such a combination of two fields could lead to new understandings of ML techniques applied to challenging and evolving data, as well as gaining invaluable insights into how ML may advance the future of the network communication field.

C. DEAL WITH HETEROGENEITY
Today, billions of wireless devices each of which has its own specification and characteristics are competing for spectrum, creating complex systems in which existing human-driven network design strategies become inefficient. There are some recent approaches towards managing such heterogeneous networking environments (e.g., DARPA Spectrum Collaboration Challenge (SC2) [1]), but these visions focus only on a narrow spectrum. We envision a framework that not only considers the MAC and PHY layer, but the network protocol stack as a whole, allowing devices to collectively learn from each other and transfer knowledge among each other. This will form intelligent and adaptive systems for protocol design that are capable of self-management, even when dealing with a plethora of diverse devices.

D. CROSS-LAYER DESIGN OPTIMIZATION
In our implementation, we assume that the transmission reliability (e.g., ACK) is handled in the upper layers (e.g., TCP protocol in Transport layer). An unreliable MAC protocol may impact the performance of upper-layer protocols (e.g., TCP, as it assumes a packet loss, is due to the congestion while it could be caused by wireless link poor quality). Therefore, it would be more efficient to have a cross-layer design optimization where constraints from different layers are enforced to select certain blocks to optimize the whole network stack.

E. MAKE PROTOCOLS MORE ROBUST
How well can the learned policy perform in conditions never seen during training? This is a question about the generalization capability of the learning algorithm [45]. The vast diversity of possible network conditions implies that even a protocol that works well across a wide variety of network conditions may suffer from bad performance on other networks. Thus, enhancing the robustness of protocols is clearly desirable. This typically involves identifying scenarios that result in poor performance by the protocol and using these scenarios for debugging and guiding changes to the protocol. Bad protocol behavior might be triggered by complex sequences of changes in network conditions, making identifying such examples challenging. Network conditions that have been shown to induce bad performance for a protocol will also contain hints regarding where the problem lies, i.e., the demonstrated problem should be explainable. One possible way to identify bad protocol performance is to leverage RL to generate adversarial network traces for an input protocol by observing protocol behavior and changing its network conditions to harm its performance relative to the optimal. The adversarial network traces generated by the adversarial framework can then be used to train more robust RL-based protocols.

The ultimate goal of this study is to train network nodes to build their own customized new MAC protocols. Recent
research [25], [33] suggests this is a hard goal to achieve when all agents (i.e., nodes) start with no previous knowledge. Consequently, scheduled training that alternates between supervised learning and self-play as suggested in [34] seems promising to emerge fully new protocols.

VIII. CONCLUSION
In this article, we have motivated the importance of a shift from the human-driven protocol design process to a machine-based design. 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 block selection by the agent. We envision this approach could provide useful insights, in particular to protocol designers to build a deeper perception of the significance of an individual or a set of protocol blocks (functions) under different scenarios. This could help them focus on enhancements/modifications of essential components, rather than focusing on the whole protocol performance in order to enhance the protocol design and performance. Although employing ML techniques for automating the design of networking protocols is a promising paradigm, it is accompanied with some challenges that we highlighted in this article. We also pinpointed the opportunities and future directions to leverage RL in protocol design domain.

REFERENCES
[1] (2018). The DARPA SC2 Website. Accessed: Jul. 2020. [Online]. Available: https://archive.darpa.mil/sc2/
[2] A. Agrawal, “Xavier: A reinforcement-learning approach to TCP congestion control,” Tech. Rep., 2016.
[3] R. Ali, N. Shahin, Y. B. Zikria, B.-S. Kim, and S. W. Kim, “Deep reinforcement learning paradigm for performance optimization of channel observation-based MAC protocols in dense WLANs,” IEEE Access, vol. 7, pp. 3500–3511, 2018.
[4] F. A. Aoudia and J. H. Hoydis, “Model-free training of end-to-end communication systems,” IEEE J. Sel. Areas Commun., vol. 37, no. 11, pp. 2503–2516, Nov. 2019.
[5] V. Arun and H. Balakrishnan, “Copa: Practical delay-based congestion control for the Internet,” in Proc. 15th USENIX Symp. Netw. Syst. Design Implement. (NSDI), 2018, pp. 329–342.
[6] R. Atallah, C. Assi, and M. Khabbaz, “Deep reinforcement learning-based scheduling for roadside communication networks,” in Proc. 15th Int. Symp. Modeling Optim. Mobile, Ad Hoc, Wireless Netw. (WiOpt), May 2017, pp. 1–8.
[7] H. Bayat-Yeganeh, V. Shah-Mansouri, and H. Kebricaei, “A multi-state Q-learning based CSMA MAC protocol for wireless networks,” Wireless Netw., vol. 24, no. 4, pp. 1251–1264, May 2018.
[8] X. Cai, B. Li, M. Yang, Z. Yan, B. Yang, Y. Jin, and X. Li, “Fractional backoff algorithm for the next generation WLAN,” in Proc. Int. Wireless Internet Conf. Springer, 2018, pp. 24–31.
[9] X. Cao, Z. Song, B. Yang, X. Du, L. Qian, and Z. Han, “Deep reinforcement learning MAC for backscatter communications relying on Wi-Fi architecture,” in Proc. IEEE Global Commun. Conf. (GLOBECOM), Dec. 2019, pp. 1–6.
[10] D. R. Carrie MacGillivray, Worldwide Global Datasphere IoT Device and Data Forecast, 2019–2023.
[11] U. Challita, L. Dong, and W. Saad, “Proactive resource management for LTE in unlicensed spectrum: A deep learning perspective,” 2017, arXiv:1702.07051. [Online]. Available: http://arxiv.org/abs/1702.07051
[12] H.-H. Chang, H. Song, Y. Yi, J. Zhang, H. He, and L. Liu, “Distributive dynamic spectrum access through deep reinforcement learning: A reservoir computing-based approach,” IEEE Internet Things J., vol. 6, no. 2, pp. 1938–1948, Apr. 2019.
[13] Y. Chu, P. D. Mitchell, and D. Grage, “ALOHA and Q-learning based medium access control for wireless sensor networks,” in Proc. Int. Symp. Wireless Commun. Syst. (ISWCS), Paris, France, Aug., 2012, pp. 511–515.
[14] L. de Alfarro, M. Zhang, and J. Garcia-Luna-Aceves, “Approaching fair collision-free channel access with slotted aloha using collaborative policy-based reinforcement learning,” in Proc. IFIP Netw. Conf. (Netw.), Jun. 2020, pp. 262–270.
[15] C. Doert, M. Neufeld, R. Kiesel, T. Weingart, D. C. Sicker, and D. Grunwald, “Multi-MAC—an adaptive MAC framework for dynamic radio networking,” in Proc. 1st IEEE Int. Symp. Frontiers Dyn. Spectr. Access Netw. (DisSPAN), 2005, pp. 548–555.
[16] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira, “PCC vivace: Online-learning congestion control,” in Proc. 15th USENIX Symp. Netw. Syst. Design Implement. (NSDI), 2018, pp. 343–356.
[17] I. Dowling, E. Curran, R. Cunningham, and V. Cahill, “Using feedback in collaborative reinforcement learning to adaptively optimize MANET routing,” IEEE Trans. Syst., Man, Cybern., A, Syst. Humans, vol. 35, no. 3, pp. 360–372, May 2005.
[18] M. Egorov, “Multi-agent deep reinforcement learning,” in Convolutional Neural Networks for Visual Recognition. 2016.
[19] F. Fazeli and G. Carle, “Learning and generating distributed routing protocols using graph-based deep learning,” in Proc. Workshop Big Data Anal. Mach. Learn. Data Commun. Netw., Aug. 2018, pp. 40–45.
[20] T. Gruber, S. Cammerer, J. Hoydis, and S. T. Brink, “On deep learning-based channel decoding,” in Proc. 51st Annu. Conf. Inf. Sci. Syst. (CISS), Mar. 2017, pp. 1–6.
[21] Y. L. Gwon and H. Kung, “Inferring origin flow patterns in Wi-Fi with deep learning,” in Proc. 11th Int. Conf. Autonomic Comput. (ICAC), 2014, pp. 73–83.
[22] H. Hu, A. Lerer, A. Peysakhovich, and J. Foerster, “Other-play” for zero-shot coordination,” 2020, arXiv:2003.02979. [Online]. Available: http://arxiv.org/abs/2003.02979
[23] T. Hu and Y. Fei, “QELAR: A machine-learning-based adaptive routing protocol for energy-efficient and lifetime-extended underwater sensor networks,” IEEE Trans. Mobile Comput., vol. 9, no. 6, pp. 796–809, Jun. 2010.
[24] IEEE Computer Society LAN MAN Standards Committee and others Part II: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, 2016.
[25] N. Jaques, A. Lazaridou, E. Hughes, C. C. G ülçehre, P. A. Ortega, D. Strouse, J. Z. Leibo, and N. de Freitas, “Social influence as intrinsic motivation for multi-agent deep reinforcement learning,” in Proc. ICML, 2019, pp. 3040–3049.
[26] N. Jay, N. H. Rotman, P. Brighten Godfrey, M. Schapira, and A. Tamar, “Internet congestion control via deep reinforcement learning,” 2018, arXiv:1810.03259. [Online]. Available: http://arxiv.org/abs/1810.03259
[27] S. Joseph, R. Misra, and S. Katti, “Towards self-driving radios: Physical-layer control using deep reinforcement learning,” in Proc. 20th Int. Workshop Mobile Comput. Syst. Appl., 2019, pp. 69–74.
[28] N. Kato, Z. M. Fadullnah, B. Mao, F. Tang, O. Akashi, T. Inoue, and K. Mizutani, “The deep learning vision for heterogeneous network traffic control: Proposal, challenges, and future perspective,” IEEE Wireless Commun., vol. 24, no. 3, pp. 146–153, Jun. 2017.
[29] B. Kiumarsi, K. G. Vamvoudakis, H. Modares, and F. L. Lewis, “Optimal and autonomous control using reinforcement learning: A survey,” IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 6, pp. 2042–2062, Jun. 2018.
[30] L. Kraemer and B. Baraneej, “Multi-agent reinforcement learning as a rehearsal for decentralized planning,” Neurocomputing, vol. 190, pp. 82–94, May 2016.
[31] N. Kumar, F. Fazeli, M. Stojanovic, and S. S. Naryanan, “Online rate adjustment for adaptive random access compressed sensing of time-varying fields,” EURASIP J. Adv. Signal Process., vol. 2016, no. 1, p. 48, Dec. 2016.
[32] L. Liu, Y. Cheng, L. Cai, S. Zhou, and Z. Niu, “Deep learning based optimization in wireless network,” in Proc. IEEE Int. Conf. Commun. (ICC), May 2017, pp. 1–6.
[33] R. Lowe, J. Foerster, Y.-L. Boureau, J. Pineau, and Y. Dauphin, “On the pitfalls of measuring emergent communication,” in Proc. 18th Int. Conf. Auto. Agents MultiAgent Syst., 2019, pp. 693–701.
[34] R. Lowe, A. Gupta, J. Foerster, D. Kiela, and J. Pineau, “On the interaction between supervision and self-play in emergent communication,” in Proc. Int. Conf. Learn. Represent., 2019.

[35] T. V. Maia, “Reinforcement learning, conditioning, and the brain: Successes and challenges,” Cognit. Affect. Behav. Neurosci., vol. 9, no. 4, pp. 343–364, Dec. 2009.

[36] Z. Mammeri, “Reinforcement learning based routing in networks: Review and classification of approaches,” IEEE Access, vol. 7, pp. 55916–55950, 2019.

[37] Q. Mao, F. Hu, and Q. Hao, “Deep learning for intelligent wireless networks: A comprehensive survey,” IEEE Commun. Surveys Tuts., vol. 20, no. 4, pp. 2593–2621, 4th Quart., 2018.

[38] D. G. Messerschmitt, “Rethinking components: From hardware and software to systems,” Proc. IEEE, vol. 95, no. 7, pp. 1473–1496, Jul. 2007.

[39] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing Atari with deep reinforcement learning,” 2013, arXiv:1312.5602. [Online]. Available: http://arxiv.org/abs/1312.5602.

[40] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, and J. Veness, “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, p. 529, 2015.

[41] E. Nachmani, Y. Be’ery, and D. Burshtein, “Learning to decode linear processes and challenges,” IEEE J. Sel. Areas Commun., vol. 37, no. 6, pp. 1325–1336, Jun. 2019.

[42] N. Papapetrou and K. Cohen, “Deep multi-user reinforcement learning for distributed dynamic spectrum access,” IEEE Trans. Wireless Commun., vol. 18, no. 1, pp. 310–323, Jan. 2019.

[43] P. Nurmi, “Reinforcement learning for routing in ad hoc networks,” in Proc. 5th Int. Symp. Modeling Optim. Mobile, Ad Hoc Wireless Netw. Workshops, Apr. 2007, pp. 1–8.

[44] T. O’Shea and J. Huydis, “An introduction to deep learning for the physical layer,” IEEE Trans. Cogn. Commun. Netw., vol. 3, no. 4, pp. 563–575, Dec. 2017.

[45] C. Packer, K. Gao, J. Kos, P. Kranenbuhl, V. Koltun, and D. Song, “Assessing generalization in deep reinforcement learning,” Tech. Rep., 2018.

[46] H. B. Pasandi and T. Nadeem, “Challenges and limitations in automating the design of MAC protocols using machine-learning,” in Proc. Int. Conf. Artif. Intell. Inf. Commun. (ICACI), Feb. 2019, pp. 107–112.

[47] H. B. Pasandi and T. Nadeem, “MAC protocol design optimization using deep learning,” in Proc. Int. Conf. Artif. Intell. Inf. Commun. (ICACI), Feb. 2020, pp. 709–715.

[48] H. B. Pasandi and T. Nadeem, “Unboxing MAC protocol design optimization using deep learning,” in Proc. IEEE Int. Conf. Pervas. Comput. Commun. Workshops (PerCom Workshops), Mar. 2020, pp. 1–5.

[49] C. Pellei, D. Zucchetto, A. Zanella, and M. Zorzi, “An interference-aware modulation and coding scheme selection in cognitive heterogeneous networks,” IEEE Trans. Wireless Commun., vol. 18, no. 6, pp. 3281–3294, Jun. 2019.

[50] M. Zhang, H. Wang, J. Li, and H. Gao, “Learned sketches for frequency estimation,” Inf. Sci., vol. 367, pp. 365–385, Jan. 2019.

[51] X. Zhang, J. Ansari, G. Yang, and P. Mahonen, “TRUMP: Efficient and flexible realization of medium access control protocols for wireless networks,” IEEE Trans. Mobile Comput., vol. 15, no. 10, pp. 2614–2626, Oct. 2016.

[52] C. Zhong, Z. Lu, M. C. Gursoy, and S. Velipasalar, “Actor-critic deep reinforcement learning for dynamic multichannel access,” in Proc. IEEE Global Conf. Signal Inf. Process. (GlobalSIP), Nov. 2018, pp. 590–603.

HANNAH B. PASANDI

(Ph.D. candidate, IEEE) received the master’s degree from Univeristé Grenoble Alpes. She is currently pursuing the Ph.D. degree with the Department of Computer Science, Virginia Commonwealth University. Her research interests include wireless communication, with an emphasis on the design and implementation of data-driven MAC protocols that not only consider conventional factors but also extract and include important environmental factors in designing protocols for IEEE WLANs.

TAMER NADEEM

(Senior Member, IEEE) received the M.Sc. and Ph.D. degrees in computer science from the University of Maryland, College Park. He is currently an Associate Professor with the Department of Computer Science, Virginia Commonwealth University (VCU). He is also the Assistant Director of the VCU Cybersecurity Center and the Founder of the Mobile Systems and Intelligent Communication (MuSIC) Laboratory, Department of Computer Science, VCU. Prior to VCU, he was the Senior Research Scientist of Siemens Corporate Research (SCR), Princeton, NJ, USA. His research interests include several aspects of wireless networking and mobile computing systems, including smart wireless systems, mobile and edge computing, software-defined networks, machine learning in networking, network security and privacy, the Internet of Things and smart city systems, vehicular networks, and intelligent transport systems. He holds five U.S. patents and has more than 100 publications in peer-reviewed top scholarly journals and conference proceedings. He serves as a member of the technical and organizing committees for various ACM and IEEE conferences. He has been serving on the Journal Editorial Board of IET Communications journal and Sensors (MDPI).