IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 24 June 2023, accepted 12 July 2023, date of publication 24 July 2023, date of current version 2 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3298102

RESEARCH ARTICLE

Learning-Based Adaptive Sliding-Window RLNC for High Bandwidth-Delay Product Networks

SHAHZAD[1], (Student Member, IEEE), RASHID ALI[2], (Member, IEEE), AMIR HAIDER[1], AND HYUNG SEOK KIM[1], (Member, IEEE)

[1]Department of Intelligent Mechatronics Engineering, Sejong University, Seoul 05006, South Korea
[2]Department of Information and Communication Technologies (DTIC), Universitat Pompeu Fabra (UPF), 08018 Barcelona, Spain

Corresponding authors: Amir Haider (amirhaider@sejong.ac.kr) and Hyung Seok Kim (hyungkim@sejong.ac.kr)

This work was supported in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) Grant by the Korean Government through the Ministry of Science and Information Communication Technology (MSIT) under Grant 2022-0-00331, and in part by the National Research Foundation of Korea (NRF) Grant by the Korean Government through MSIT under Grant 2022R1F1A1063662.

ABSTRACT Sliding-window random linear network coding (RLNC) is a good fit for achieving low in-order delivery delay in future-generation networks characterized by lossy links. In high bandwidth-delay product networks, however, the issue of integrating RLNC with transmission control protocol (TCP) flow and congestion control poses a significant challenge. In this paper, we propose an innovative reinforcement learning (RL) framework that addresses this issue by decoupling the RLNC sliding window from TCP and dynamically adjusting it to enhance network performance in terms of goodput, in-order delivery delay, and decoding complexity. By employing RL, we enable autonomous decision-making for adjusting the sliding window of RLNC, which operates independently of TCP. This decoupling allows RLNC to adapt dynamically to the varying conditions of the network, without prior knowledge of its characteristics. By leveraging the benefits of RLNC and TCP separately, we achieve more efficient and effective utilization of network resources. The results highlight significant improvements in goodput, in-order delivery delay, and decoding complexity. These improvements are crucial because in network coding, there is always a trade-off between goodput, delay, and decoding complexity, and minimizing this trade-off is very challenging. Using RL and decoupling of RLNC sliding window from TCP, we address this challenge and minimize the trade-off significantly. Goodput is improved by up to 11%, the in-order delivery delay is reduced by a factor of 9%, and coding complexity shows an improvement of up to 45% compared to the state-of-the-art.

INDEX TERMS High BDP networks, in-order delay, reinforcement learning, RLNC, TCP sliding window.

I. INTRODUCTION

Reliable low-latency communication is the core of future-generation networks such as the industrial Internet of things and non-terrestrial networks. These networks generally have a high bandwidth-delay product (BDP) and are considered crucial to the development of 6th generation communication systems [1]. Random linear network coding (RLNC) [2] is an important alternative to conventional retransmission-based loss-recovery mechanisms for achieving low end-to-end delivery delay, especially in high-BDP networks [3].

RLNC is a network coding (NC) process used in network communication to increase data transmission reliability and effectiveness. RLNC linearly combines packets at the sender side to add redundancy to the transmitted data. The coded packets produced by this process are linear combinations of the original packets [4]. These coded packets are then transmitted over the network. Fig. 1 shows a network model with RLNC encoder and decoder on the sender and receiver devices respectively.[1] The RLNC encoder generates coded

The associate editor coordinating the review of this manuscript and approving it for publication was Hang Shen.

[1]The RLNC encoder and decoder can also reside at R1 and R2.

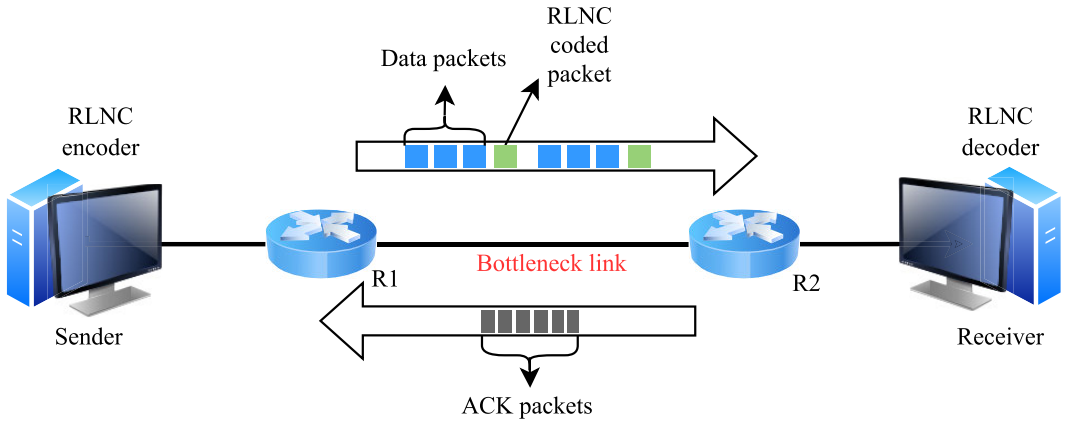

FIGURE 1. A network model with an RLNC encoder at the sender device and an RLNC decoder at the receiver device.

packets and sends them to the receiver via the bottleneck link between router 1 (R1) and router 2 (R2). The sender can choose to send only data packets (No NC), coded packets, or a mix of both depending on the user's requirement.

On the receiver side, the RLNC decoder allows for the decoding of the original packets using a subset of received coded packets. After decoding a coded packet, the receiver sends an acknowledgment (ACK) packet to the sender to confirm the receipt of said coded packet. Instead of relying on receiving all the original packets individually, the receiver can use a set of coded packets to reconstruct the original packets. This property of RLNC provides robustness against packet loss and network errors, as any subset of coded packets is potentially sufficient for decoding.

RLNC has several advantages in lossy network environments, such as wireless networks or networks with high congestion. It can improve the reliability of data transmission by reducing the impact of packet loss and increasing the likelihood of successful data recovery. Additionally, RLNC can enhance network efficiency by reducing retransmissions and enabling opportunistic data forwarding.

RLNC can be broadly categorized into *block* RLNC and *sliding-window* RLNC. Block RLNC operates on fixed-size blocks of packets, where a block typically consists of a predetermined number of packets. The block size is predetermined and remains constant throughout the transmission. Encoding is performed across the blocks of packets. The sender applies RLNC to the collected packets in the block. It randomly generates coefficients and linearly combines the packets using these coefficients to create coded packets. Each coded packet contains a linear combination of the original packets in the block. This creates blocking delay which is not good for applications expecting in-order delivery. Unlike block RLNC, which operates on fixed-size blocks, sliding-window RLNC adapts to varying network conditions and packet arrivals. The sender and receiver both initialize a sliding window. The window size represents the number of packets that can be encoded or decoded at a given time.

The sender collects a certain number of original packets based on the current window size. The number of packets collected depends on the window size and can vary as the window slides. The receiver collects the received coded packets within its window size. The number of received packets in the window can vary due to network conditions and potential packet loss. The receiver needs a sufficient number of linearly independent coded packets within the window to successfully decode the original packets. After decoding, both the sender and receiver slide their windows to accommodate new packets. The window slides by a certain number of packets, which can be a fixed amount or dynamically adjusted based on network conditions or feedback from the receiver.

Sliding-window RLNC avoids blocking delay by encoding across "yet not-acknowledged (non-ACKed) packets", which also form the transmission control protocol (TCP) sliding window (TCP_w). As new acknowledgments (ACKs) arrive, the sliding window moves forward, also moving the set of data packets that will be encoded here onward. This allows the receiver to decode as soon as it has sufficient encoded packets to reconstruct a data packet. RLNC provides redundancy for reliability, while tuning the TCP sliding window affords better flow and congestion control.

Sliding-window RLNC provides flexibility in terms of adding new data packets for encoding while removing older ones. In technical terms, this is known as closing the window. Previously, algorithms would adopt an infinite window, where every encoded packet contains all previously transmitted data packets [5], [6], [7]. These algorithms, however, are not practical because of excessive memory usage and computational complexities. Later on, a finite-window approach was introduced where the number of data packets included in the encoding process does not exceed a predefined limit, which happens to be TCP_w in general, e.g., [8], [9], and [10].

Most finite-window algorithms are also systematic, which means that they place encoded packets among a stream of data packets (as shown in Fig 1). Fig. 2 shows an example of the systematic finite sliding-window operation. $\{s_1, s_2, \ldots, s_n\}$

represent data packets, and $\{c_1, c_2, c_3, \ldots, c_n\}$ represent encoded packets. The coding rate (R) determines the insertion of an encoded packet into the stream after every k data packets; hence,

$$R = k/(k+1). \tag{1}$$

In this example (Fig. 2) $R = 3/4$, which means that an encoded packet is inserted after every $k = 3$ data packets. The red cross shows a packet loss during transmission. The dotted and dashed lines show the range of data packets involved in the encoding process of each c_i.

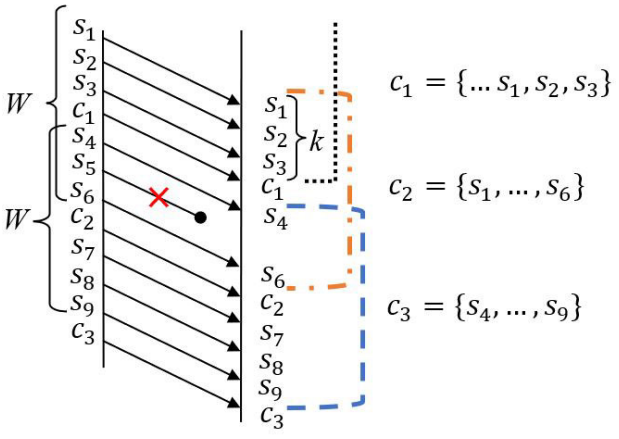

FIGURE 2. Sliding-window RLNC with $TCP_W = 6$, coding rate(R) = 3/4, and number of data packets (k) = 3. Dashed and dotted lines show the range of data packets involved in each encoding process.

In this paper, we highlight the issue of either having a common window for TCP flow control and encoding or having an independent but constant encoding window i.e., E_w. We propose an adaptive learning-based sliding-window RLNC framework called LS-RLNC to solve the decision problem of how the E_w should evolve over time to cope with the changing network environment. LS-RLNC utilizes reinforcement learning (RL) to achieve this. The goal is to maximize the overall goodput while keeping the in-order delivery delay as low as possible.

RL is a branch of machine learning that focuses on training an agent to make sequential decisions in an environment to maximize a long-term reward. It involves training an agent by interacting with an environment and getting positive or negative feedback as a reward for the actions taken by the agent.

In the context of network communication, RL can be applied to optimize various aspects of network performance, such as throughput, latency, energy efficiency, and resource allocation as suggested by the literature [11], [12], [13], [14], [15]. In NC, RL is used in decision-making scenarios in general. For example [11] uses RL to solve the decision problem of when should the sender transmit a coded packet in a systematic NC scenario in non-terrestrial networks. This implies that the prospect of learning the ideal action-value function in RL, by engaging with the environment without a priori knowledge is crucial to NC. This is appealing to our decision problem because mathematically modeling it under changing network conditions is complex. Therefore we choose RL to dynamically evolve the E_w.

This paper focuses on sliding-window RLNC in specific and other sliding-window NC schemes in general. LS-RLNC can be modified to accommodate other sliding-window NC variations (e.g., [16], [17], [18], [19]). We highlight our contributions in this paper as follows:

- A practical and adaptive learning-based sliding-window RLNC scheme called LS-RLNC with congestion and delay feedback is proposed. We carefully devise the state space and reward function to maximize goodput and reduce in-order delay.
- LS-RLNC utilizes the explicit congestion notification (ECN) feedback and the receiver's ACK feedback to evolve the E_w carefully.
- We have implemented and evaluated LS-RLNC in Mininet and Python 3. LS-RLNC outperforms the state-of-the-art in goodput, in-order delay, and decoding complexity.

II. RELATED WORK

Over the years, several sophisticated schemes have shown better overall performance of the sliding-window approach by carefully designing the placement mechanism of encoded packets [5], [11], [20]. Authors in [21] use an adaptive algorithm for sliding-window RLNC to estimate channel conditions and adjust the retransmission rates to improve the overall performance. Caterpillar RLNC (CRLNC) [22] is another sliding-window RLNC variant that does not rely on feedback and focuses on decoding simplification. CRLNC with feedback (CRLNC-FB) [10] and [23] are continuations of CRLNC that embed selective-repeat automatic repeat request (ARQ) and multihop support into CRLNC respectively.

Authors in [3] propose a lightweight network-coded ARQ protocol for ultra-reliable low latency communication (URLLC). In this work, authors decouple the RLNC sliding window from TCP and show through simulations that it outperforms selective-repeat ARQ and CRLNC-FB [10].

Compared to retransmission-based schemes, sliding-window RLNC schemes are efficient in minimizing the overall end-to-end in-order delivery delay, which is crucial to most future network delay-sensitive applications [5], [7].

In general, these schemes (except [3]) trade throughput for improved delay by placing multiple encoded packets as redundancy. However, the problem of using the same TCP_W remains intact. This is problematic because i) it deeply impacts the encoding process and ii) it impacts the complexity–performance trade-off. It determines the nature of each encoded packet that is created, i.e., the maximum number of data packets involved in its encoding. In addition, it determines the size of the decoding matrix, i.e., the maximum number of packets required to perform decoding. Further, it impacts the size of the encoding vector of each encoded packet, consequently affecting the encoding quality.

Apart from added complexity and delay, there is another major issue in using a common window for both flow control and encoding: the inability of TCP_w to cope with bursts of errors for encoding. This problem is highlighted in Fig. 3. The data packets s_5 and s_6 are lost during the transmission. Two linearly independent encoded packets are required to recover the lost data packets. In the case of common TCP_w (Fig. 3a), the decoder has to wait for both c_2 and c_3 because they include s_5 and s_6 in their composition. However, when c_3 is received, the sliding window has moved forward and s_1 and s_2 are already deleted (delivered). In this scenario, c_2 and c_3 are not enough for decoding. This issue can be handled with an independent encoding window (E_w), as shown in Fig.3(b). Here the size of E_w is 4 instead of 6. In this case, c_2 and c_3 can be utilized for decoding to obtain the missing data packets because s_5 and s_6 are still within range and the dependability on previous data packets is limited.

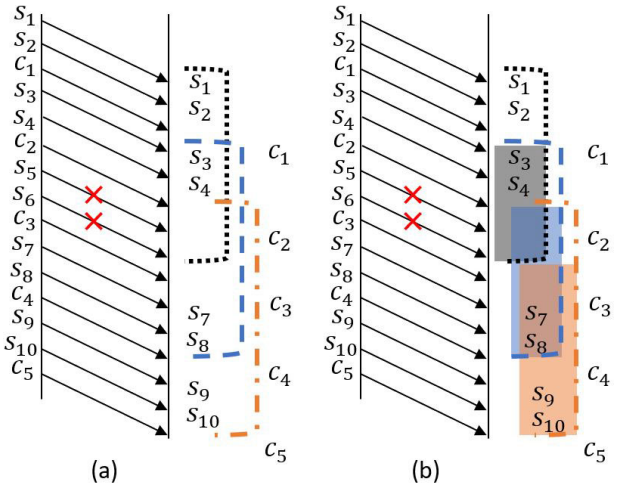

FIGURE 3. Burst error effect on sliding-window RLNC. (a) A common window for both congestion and encoding. (b) Separation of TCP_W and encoding window (E_W).

III. LS-RLNC SYSTEM MODEL

We outline the LS-RLNC system model in this section. The system model comprises two modules named the network module and the RL module. Fig. 4 shows the workflow of LS-RLNC and the integration of its modules in relation to the network model (Fig. 1). The details of each module are as follows.

A. NETWORK MODULE

This section explains the theory and network components of LS-RLNC. We propose an independent learning-based E_w that can adapt well to changing network conditions. In general, $E_w \leq TCp_w$ According to this relation, the current sliding-window RLNC schemes can be considered a unique case where $E_w = TCP_w$. However, it is also true that we can define E_w as

$$E_w = \begin{cases} d \times k, & \text{for active encoding,} \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

where d is defined as the encoding depth, i.e., the number of k-packet groups participating in the encoding process and k represents the number of data packets as indicated in (1).

Because the decoding complexity is $O(M^3)$, where M is the decoding matrix size [3], it is apparent that the size of M depends on TCP_w. Therefore, having E_w limited to a subset of TCP_w is beneficial because i) it reduces the decoding complexity consequently improving decoding delay and ii) it also simplifies the encoding process by shortening the encoding vector size. However, the size of E_w is given by (2). Hence, E_w is affected by d and k. This implies that E_w is dependant on (1) as well. This becomes a decision problem of how E_w should evolve and is the central focus of this paper.

Consider that data from an application are buffered in the TCP sender queue as $\{s_0, s_1 \ldots s_n\}$. Each data packet comprises K equal bits. Whenever the TCP_w allows a sending opportunity, a data or encoded packet is sent according to (1). Let's say a data packet $s_{i_{sq}+1}$ is chosen for transmission; then, i_{sq} is the index of the most recent data packet that was sent. If an encoded packet, e.g., c_k is next in line to be transmitted, then it is created by encoding all data packets present in E_w as

$$c_k = \sum_{x=l_E}^{u_E} g_x^{(k)} s_x^{(k)}, \tag{3}$$

where l_E is the index of the last non-ACKed data packet in E_w, the upper limit of c_k is $u_E = i_{sq}$ and $g_x^{(k)}$ are coefficients randomly chosen over a finite field $GF(2^m)$.[2] l_E, index of c_k, and u_E are carried in the packet header for identification.

At the receiver end, arriving packets (data/encoded) are buffered. At the application level, data packets are in-order delivered. Let i_{od} represent the index of the most recent in-order delivered data packet. $s_{i_{od}+1}$ gets delivered to the application when it becomes accessible; otherwise, the lost data packet is retrieved from available encoded packets by initiating the decoding process. During this time, arriving data packets are halted in the buffer until the lost data packet is recovered and delivered to the application. During this period, more data packets can potentially be lost., e.g., in case of burst error or congestion control kicking in. Let

$$l_D = i_{od} + 1 \tag{4}$$

and

$$u_D = u_E. \tag{5}$$

Hence, (4) and (5) form the limits of the decoding window (D_w) which is given by

$$D_w = \begin{cases} u_D - l_D + 1 \geq 2, & \text{for active decoding,} \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

[2]To avoid the overhead of carrying the coefficients, we adopt the pseudo-random number generator strategy implemented in [11].

The decoding is done using on-the-fly Gaussian elimination [24]. When the number of encoded packets in D_w equals the number of lost data packets, the decoder will most likely succeed given a large enough $GF(2m)$. The application then receives the recovered data packets, and the index of the most recent in-order data packet is modified as $i_{od} = u_D$.

TCP_w is directly affected by incipient congestion (IC) because it controls the sender's rate. We utilize this information for intelligent decision-making about E_w. We adopt enhanced explicit congestion notification (EECN) [25] to obtain an early insight into the network condition. The EECN feedback mechanism is given in Algorithm 1.

Algorithm 1 EECN Feedback Mechanism

```
// Egress pipeline
if ECN capable packet then
    if queue_depth > threshold then
        Set ECN register
        Store port number in register_port
        Store source IP in register_source
    end
end
if (ACK packet && destination IP == register_source
  && ingress_port == register_port && ECN register
  is set) then
    Set ECN-Echo bit
    Clear ECN register
end
if (ACK packet && CWR bit is set && source IP ==
  source register) then
    Clear CWR bit
end
```

EECN is our prior research work which addresses the limitations of the traditional TCP congestion control mechanisms in high BDP networks by introducing an enhanced Explicit Congestion Notification (ECN) technique using P4 programming [26]. The EECN mechanism dynamically adjusts the ECN markings on packets, providing more precise and accurate feedback to the sender. EECN informs the sender of incipient congestion without waiting for the receiver's feedback. This is done by exploiting programmable network devices to send statistics to the sender directly using software-defined networks (SDN).

From [20], we obtain the expected decoding delay for all in-flight packets in TCP_w as

$$\widetilde{T}_t^E = 1/(f - p_e), \tag{7}$$

where f is the fraction of encoded packets in TCP_w and p_e is the packet loss rate. We denote $\bar{T}_t^E$ as the expected decoding delay for all in-flight packets in E_w. Hence, in our case, as we have decoupled E_w from TCP_w, we can modify f as the fraction of encoded packets in E_w. This is valid because the encoding is performed according to (2) instead of TCP_w. Note that both f and p_e can be estimated. We estimate f as

$$f_t = \frac{\text{number of encoded packets in } E_w}{\text{number of all packets in } E_w} \tag{8}$$

This intrinsically allows LS-RLNC to have a lower in-order delay compared to typical sliding-window RLNC schemes. Similarly, we can estimate p_e. From this discussion, we can write the expected decoding delay as

$$T_t^E = \begin{cases} \widetilde{T}_t^E, & \text{for } TCP_w \\ \bar{T}_t^E, & \text{for } E_w \end{cases} \tag{9}$$

Finally, as a safety net, we also implement a retransmission policy that is triggered only when $E_w = W$ and the decoding process fails to recover the lost packet in a certain time period T.

B. REINFORCEMENT LEARNING MODULE

In this section, we explain the use of RL in LS-RLNC. The integration of the LS-RLNC's RL module with the network module is shown in Fig. 4. The goal is to evolve E_w such that goodput is maximized and in-order delay is kept as low as possible. To achieve this, we first record some network information. The sender keeps track of the timestamps of each data and encoded packet, the total number of each type of packet sent up to each timestamp, and the packets comprising E_w. The sender also stores information from ACK packets, namely, the ID of the last received packet, the value of i_{od}, E_w, and the number of acknowledged data and encoded packets. We now formulate the RL design for LS-RLNC. This RL design comprises an action space, a state space, and a reward function.

1) ACTION SPACE

Our action space is rather simple, i.e.,

- increase E_w
- decrease E_w
- no change

2) STATE SPACE

We design the state space of LS-RLNC as follows:

$$S_t = (D_{w_t}, E_{w_t}, IC_t), \tag{10}$$

where D_{w_t} is D_w, E_{w_t} is E_w, and IC_t is the level of incipient congestion at time t respectively. Eq. 10 gives us the necessary information to estimate the end-to-end in-order delay. IC is zero if there is no congestion and nonzero when congestion is anticipated; hence, the effect of queuing delay is captured. Further, D_w and T^E give us the overall decoding delay. Additionally, note that these variables are linked to goodput as well. Recall from Section III that data are in-order delivered to the application when $s_{i_{od}+1}$ is received or when a coded packet completes the decoding process. In both cases, goodput is realized when D_w is 0. Consequently, the RL agent can track the goodput and in-order delay using ACK feedback and the rest of the stored records at the sender. From the above discussion, we infer that $D_w = 0$ being true after a decoding

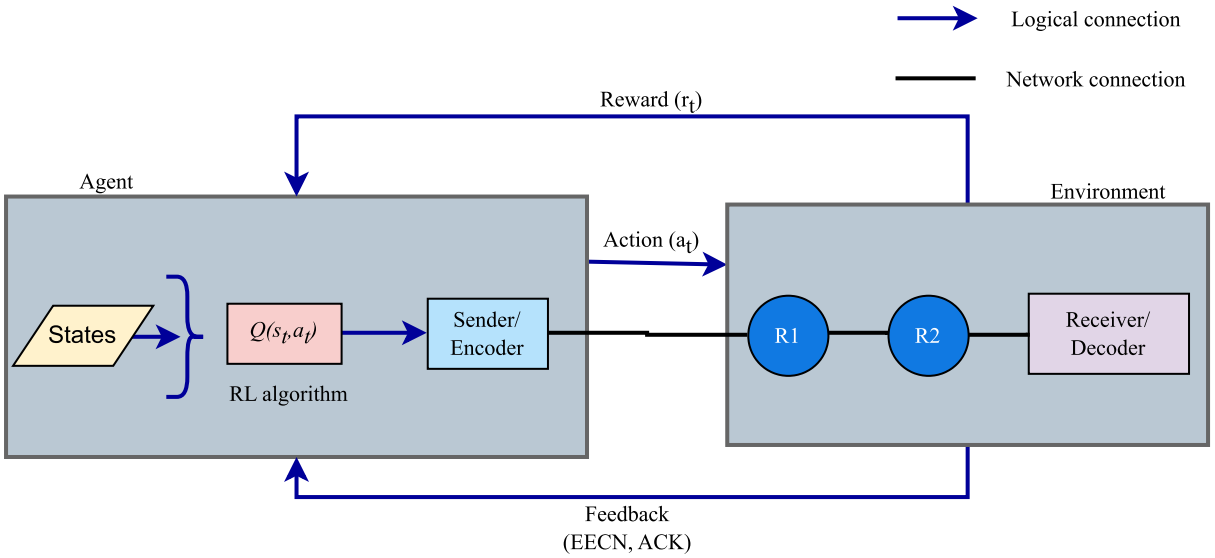

FIGURE 4. LS-RLNC workflow in relation to the network model (Fig. 1).

process, results in an episode of LS-RLNC. Any state S_t with zero D_w is a terminating state. After each episode, the increase in the number of $s_{i_{od}}$ from that at the start of the episode divided by the episode duration yields the realized goodput. If decoding is performed and/or the value of i_{od} does not increase, a negative reward relative to T^E and D_w is given.

3) REWARD FUNCTION

The formulation of the action space and the state space lead us to formulate our reward function as follows:

$$r_t = \begin{cases} a \times goodput_t, & D_{w_t} = 0 \text{ and } i_{od_t} > \text{last } i_{od}, \\ -b \times T^{E_t}, & D_{w_t} = 0 \text{ and } i_{od_t} = \text{last } i_{od}, \\ -c \times (T^{E_t} + D_{w_t}), & D_{w_t} > 0. \end{cases} \quad (11)$$

where a, b, and c are tunable nonzero values. Note that a negative reward is given when D_w is zero but i_{or} is not increased. This is because it would be considered a wasted transmission (encoded packet) with no goodput.

We adopted Q-learning in this work because it uses a simple value iteration update method, which is feasible for online learning. Q-learning updates the function as

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a_t) - Q(s_t, a_t)], \quad (12)$$

where α is the learning rate, γ is the discount factor, and r_{t+1} is the immediate reward. Further, we adopted the ϵ-*greedy* method to ensure exploration during the initial phase of learning. With ϵ-*greedy*, a random action a_t is taken with probability ϵ; otherwise, a greedy action

$$a = maxQ_t(a) \quad (13)$$

is taken with probability $1 - \epsilon$. The RL-related parameters, given in Table. 1, were used following common RL practices and exhaustive trial and error experimentation. No function approximation was used in this work because D_w, E_w, and IC are discrete and do not exceed their limits. Therefore, using tabular methods for RL is feasible because the state space is not large.

IV. SIMULATION SETUP

In this section, we discuss the simulation setup to evaluate our proposed scheme: LS-RLNC. We evaluate the performance of LS-RLNC by comparing it with rapidARQ [3] and traditional sliding-window RLNC.[3] Sliding RLNC uses the same coding rate R obtained from (1) and encodes packets across the entire TCP_w instead of E_w. However, rapidARQ uses a constant E_w and relies on the selection of d for better performance. We decide the best value of d by exhaustive simulations.

We evaluated LS-RLNC in Mininet and Python 3. Mininet is a Python-based network simulation tool that creates a realistic virtual network, running real kernel, switch, and application code, on a single machine. Because it is Python-based, the RL and EECN modules, also written in Python, are called by the Mininet script as Python functions. We used a dumbbell topology similar to Fig 1, where an application at the left leaf communicates with an application at the right leaf. The application transmits at a rate of 20 Mbps, while the bottleneck link's bandwidth is set to 10 Mbps to observe congestion. Congestion is generated by inducing non-ECN traffic to the network randomly.[4] The edge router implements ECN-enabled queuing, where packets are marked when a

[3]We refer to it as sliding RLNC in the graphs and here onward to save space.

[4]Similar congestion patterns are generated for each scheme to observe fairness.

TABLE 1. Simulation parameters.

Parameter	Value	
Propagation delay (t_{PD})	50-300 ms	
	Scenario 1	50-300 ms
	Scenario 2	100 ms
	Scenario 3	200 ms
	Scenario 4	100 ms
Bandwidth (BW)	Bottleneck link	10 Mbps
	Edge link	20 Mpbs
Packet error rate (p_e)	1-20%	
	Scenario 1	20%
	Scenario 2	1-15%
	Scenario 3	10%, 20%
	Scenario 4	5-20%
Channel burstiness (B)	1-3	
	Scenario 1	1
	Scenario 2	1
	Scenario 3	1-3
	Scenario 4	1
Timeout (T)	1.5 {round trip time (RTT)}	
Traffic type	TCP	
Packet size	1500 bytes	
Coding rate (R)	According to channel capacity	
Learning rate (α)	0.1	
Discount factor (γ)	0.95	
Exploration rate (ϵ)	0.05	

certain queue threshold is reached. In case of overflow, the queue is flushed.

We trained our agent in the aforementioned environment by running 4000 simulations. In each simulation, the application ran until 4000 data packets were in-order delivered with p_e uniformly randomly chosen from the given range. We categorized our simulations into four scenarios. All simulation scenarios follow the simulation parameters given in Table. 1. Scenario 1 deals with different propagation delays and its associated results are shown in Fig. 5. Scenario 2 looks at the effects of dynamic p_e on the cumulative goodput, and its associated results are given in Fig. 6. Scenario 3 is created to analyze the behavior of LS-RLNC under a bursty p_e model. The results of this scenario are given in Fig. 7. Scenario 4 is devised to observe the decoding complexity of LS-RLNC in terms of decoding window size and the average goodput as well. The results of scenario 4 are shown in Fig. 8 and 9.

Fig. 5 shows the performance of all tested schemes in terms of delay with uniform p_e. We tested the schemes for propagation delays 50 ms, 100 ms, 200 ms, and 300 ms in each subplot respectively. The cumulative moving average (MA) of in-order delivery delay incurred by each data packet is plotted. The MA is calculated over a window of 100 packets. Both LS-RLNC and rapidARQ showed lower in-order delay compared to sliding RLNC. However, LS-RLNC showed better performance than both even though we selected d for rapidARQ through exhaustive simulations. This is because the other schemes did not consider IC, T^E, and D_w. Early insight into the network congestion state helps reduce packet loss at the cost of a lower sending rate (explained later). The separation of E_w from TCP_w provides better protection to data packets and reduces the decoding delay. Recall from Sec. III how T^E and D_w affect the in-order delay. We reduced T^E by limiting f, which relies on E_w. This led to a shorter D_w in general (as we show later) resulting in a lower in-order delivery delay.

We show the packet drop rate in each scenario as well. The packet drops occur mainly owing to two reasons: i) p_e and ii) IC. LS-RLNC and rapidARQ showed good resilience to IC compared to sliding RLNC. However, LS-RLNC showed even lower packet drops than rapidARQ. This is because i) LS-RLNC got an early indication of IC (due to the use of EECN [25] framework) and ii) our agent dynamically changed E_w to provide better protection to recent data packets.

Fig. 6 shows the in-order cumulative goodput of each scheme. Cumulative goodput is measured as the amount of in-order data delivered divided by the elapsed time. p_e was dynamically randomly chosen from $1 - 15\%$. All the schemes experienced similar loss patterns and attempted to deliver 4000 data packets. The propagation delay, in this case, was $100ms$. The overall cumulative goodput of all schemes was similar. This is mainly because all schemes used a similar coding rate R given in (1). We could still observe some differences, where rapidARQ showed slightly better goodput and completed the transmission a little earlier. This is because LS-RLNC provides better resilience to IC at the cost of a lower sending rate, which impacts the overall goodput. We tested the cumulative goodput results for other propagation delays and found similar results. However, this is only when a uniform p_e model is used, as we will see later that in bursty error models, LS-RLNC shows improved goodput.

In Fig. 7, we highlight the effects of a bursty p_e model on each scheme. We adopted a similar on-off model as [3] for this purpose. In this model, there is an "on" period and an "off " period. During these "on" and "off" periods the p_e is given by

$$p_{e_{on}} > p_e \tag{14}$$

and

$$p_{e_{off}} = 0. \tag{15}$$

During an on-off cycle, the average p_e is similar to that of a uniform channel. The burstiness of the channel is controlled by

$$B = (T_{on} + T_{off})/T_{on}, \tag{16}$$

where T_{on} and T_{off} are the time periods for $p_{e_{on}}$ and $p_{e_{off}}$ respectively. As the level of burstiness increases, the average goodput is reduced. However, compared to the other schemes LS-RLNC shows better goodput. We also tested the burstiness in relation to the packet drop rate and found similar results.

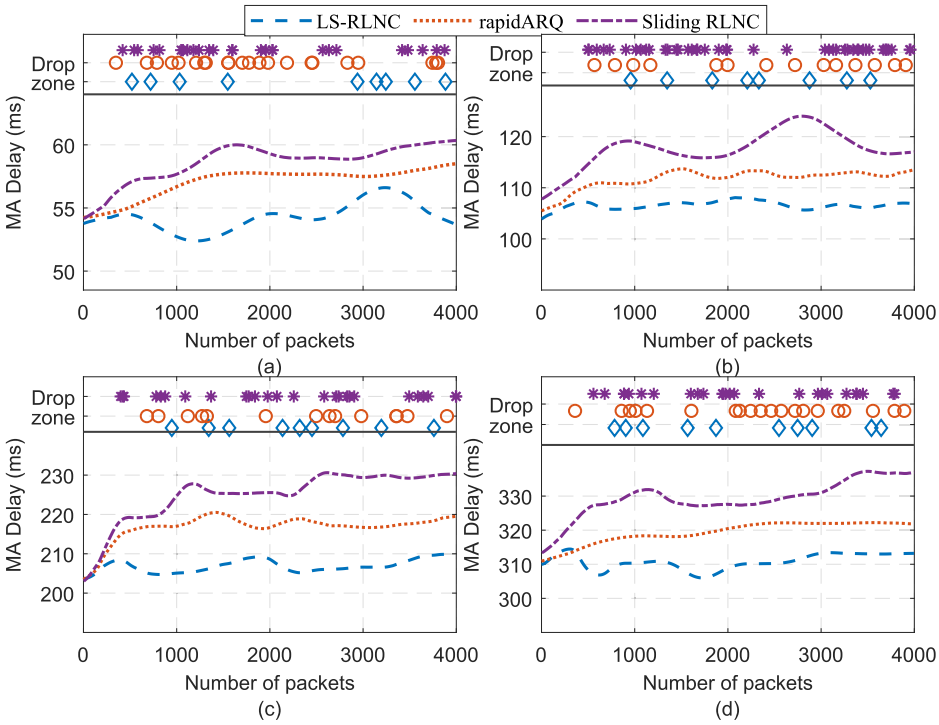

FIGURE 5. Cumulative moving average of the end-to-end in-order delivery delay incurred to all data packets in each scheme with propagation delays of (a) 50*ms* (b) 100*ms* (c) 200*ms* and (d) 300*ms*.

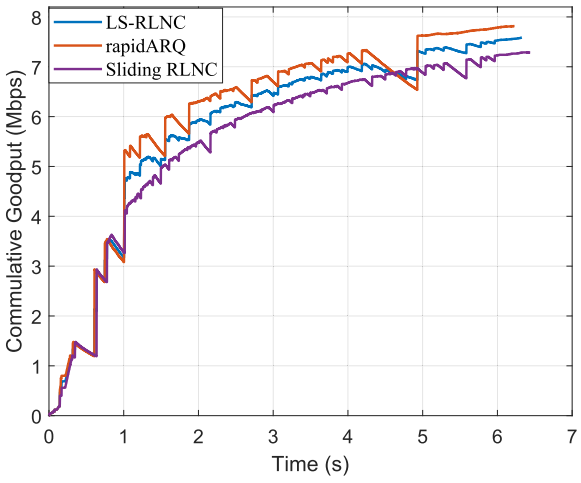

FIGURE 6. Cumulative goodput (Mbps) in dynamic p_e environment. p_e is dynamically randomly chosen from range [1 − 15%].

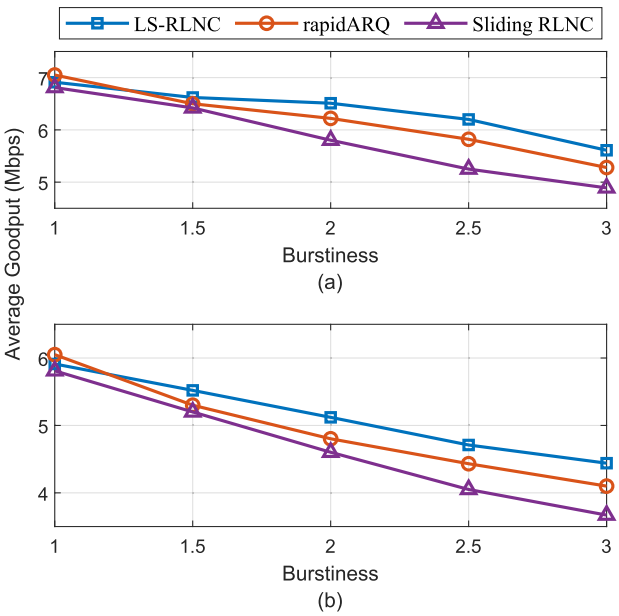

FIGURE 7. Performance analysis in terms of average goodput (Mbps) under a bursty p_e model. (a) p_e = 10% (b) p_e = 20%.

Fig. 8 shows the decoding complexity in terms of the average decoding window size. We chose a different R for each p_e in this case to meet the channel capacity (1-p_e). We also chose the value of d that generated the max average goodput through exhaustive simulations. LS-RLNC shows better decoding complexity by keeping a low decoding window. This is because the agent only allows the E_w to expand if it yields a better r. LS-RLNC has a lower D_w mainly due to the separation of E_w and TCP_w. E_w generates a lower T^E compared to TCP_w because f is reduced, which consequently lowers the decoding complexity. LS-RLNC reduces the decoding complexity by an average of 35% compared to rapidARQ and by an average of 78.5% compared to sliding RLNC.

Fig. 9 shows the average goodput observed in relation to the decoding complexity (Fig. 8) for each p_e in scenario 4.

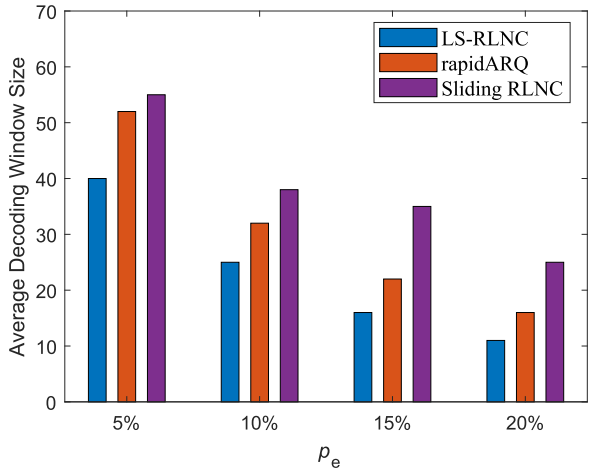

FIGURE 8. Decoding complexity in terms of decoding window size for p_e between [5 – 20%].

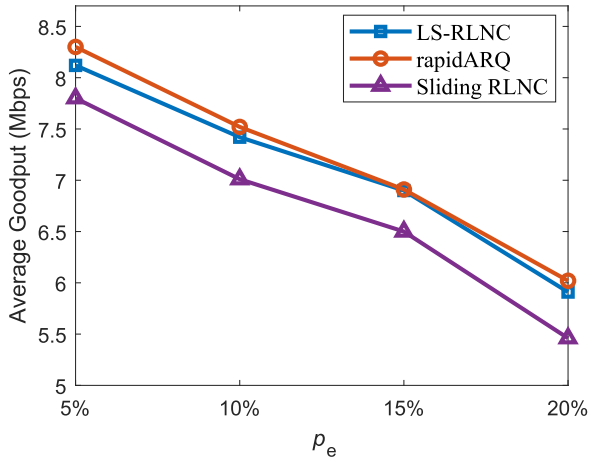

FIGURE 9. Average goodput in relation to the decoding complexity (Fig. 8) for p_e between [5 – 20%].

rapidARQ has slightly higher goodput when p_e is low but as p_e increases LS-RLNC shows better goodput. This is the trade-off of LS-RLNC where it improves the decoding complexity at the expense of goodput. However, as the p_e increases, this trade-off becomes minimal and LS-RLNC gives better decoding complexity as well as average goodput.

V. CONCLUSION

LS-RLNC is an adaptive sliding window RLNC for high bandwidth-delay product networks using reinforcement learning. LS-RLNC decouples the encoding window (E_w) from TCP sliding window (TCP_w) and uses network and receiver feedback to optimize the value of E_w. LS-RLNC shows that a carefully designed RL scheme to dynamically evolve E_w can achieve high goodput with low in-order delivery delay and reduced decoding complexity. Further, we show through simulations that LS-RLNC has better overall performance than state-of-the-art sliding RLNC schemes. LS-RLNC improves the goodput by up to 6-10%. In-order delivery delay is reduced by up to 11% and decoding complexity is reduced between 28-45%. These improvements in performance are crucial to the utility of RLNC because in network coding there is always a trade-off between goodput, delay, and decoding complexity. The results show that LS-RLNC minimizes this trade-off effectively. The results also verify that in scenarios with a bursty error model, LS-RLNC shows better resilience and improved goodput compared to state-of-the-art schemes while maintaining low in-order delivery delay.

REFERENCES

[1] M. Giordani and M. Zorzi, "Non-terrestrial networks in the 6G era: Challenges and opportunities," *IEEE Netw.*, vol. 35, no. 2, pp. 244–251, Mar. 2021.
[2] T. Ho, M. Medard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *IEEE Trans. Inf. Theory*, vol. 52, no. 10, pp. 4413–4430, Oct. 2006.
[3] F. Karetsi and E. Papapetrou, "Lightweight network-coded ARQ: An approach for ultra-reliable low latency communication," *Comput. Commun.*, vol. 185, pp. 118–129, Mar. 2022.
[4] K. H. Lee, J. H. Kim, and S. Cho, "RLNC in practical wireless networks," in *Wireless Algorithms, Systems, and Applications*. Cham, Switzerland: Springer, 2014, pp. 194–204.
[5] M. Karzand, D. J. Leith, J. Cloud, and M. Médard, "Design of FEC for low delay in 5G," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 8, pp. 1783–1793, Aug. 2017.
[6] J. Cloud and M. Medard, "Multi-path low delay network codes," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–7.
[7] A. Garcia-Saavedra, M. Karzand, and D. J. Leith, "Low delay random linear coding and scheduling over multiple interfaces," *IEEE Trans. Mobile Comput.*, vol. 16, no. 11, pp. 3100–3114, Nov. 2017.
[8] J. Cloud, D. Leith, and M. Médard, "A coded generalization of selective repeat ARQ," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 2155–2163.
[9] Y. Lin, B. Liang, and B. Li, "SlideOR: Online opportunistic network coding in wireless mesh networks," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–5.
[10] F. Gabriel, S. Wunderlich, S. Pandi, F. H. P. Fitzek, and M. Reisslein, "Caterpillar RLNC with feedback (CRLNC-FB): Reducing delay in selective repeat ARQ through coding," *IEEE Access*, vol. 6, pp. 44787–44802, 2018.
[11] F. Zhang, Y. Li, J. Wang, and T. Q. S. Quek, "Learning-based FEC for non-terrestrial networks with delayed feedback," *IEEE Commun. Lett.*, vol. 26, no. 2, pp. 306–310, Feb. 2022.
[12] R. Ali, Y. B. Zikria, S. Garg, A. K. Bashir, M. S. Obaidat, and H. S. Kim, "A federated reinforcement learning framework for incumbent technologies in beyond 5G networks," *IEEE Netw.*, vol. 35, no. 4, pp. 152–159, Jul. 2021.
[13] Z. Qin, Z. Fei, J. Huang, Y. Wang, M. Xiao, and J. Yuan, "Reinforcement-learning-based overhead reduction for online fountain codes with limited feedback," *IEEE Trans. Commun.*, vol. 71, no. 7, pp. 3977–3991, Jul. 2023.
[14] Y. Wang, S. Fu, C. Yao, H. Zhang, and F. R. Yu, "Caching placement optimization in UAV-assisted cellular networks: A deep reinforcement learning based framework," *IEEE Wireless Commun. Lett.*, early access, May 9, 2023, doi: 10.1109/LWC.2023.3274535.
[15] M. Z. Islam, Shahzad, R. Ali, A. Haider, and H. S. Kim, "Reinforcement learning-aided edge intelligence framework for delay-sensitive industrial applications," *Sensors*, vol. 22, no. 20, p. 8001, Oct. 2022.
[16] P. Garrido, D. Gómez, J. Lanza, and R. Agüero, "Exploiting sparse coding: A sliding window enhancement of a random linear network coding scheme," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–6.
[17] E. Tasdemir, V. Nguyen, G. T. Nguyen, F. H. P. Fitzek, and M. Reisslein, "FSW: Fulcrum sliding window coding for low-latency communication," *IEEE Access*, vol. 10, pp. 54276–54290, 2022.
[18] J. Yang, Z.-P. Shi, C.-X. Wang, and J.-B. Ji, "Design of optimized sliding-window BATS codes," *IEEE Commun. Lett.*, vol. 23, no. 3, pp. 410–413, Mar. 2019.

[19] S. Jayasooriya, J. Yuan, and Y. Xie, "An improved sliding window BATS code," in *Proc. 11th Int. Symp. Topics Coding (ISTC)*, Aug. 2021, pp. 1–5.
[20] Y. Li, F. Zhang, J. Wang, T. Q. S. Quek, and J. Wang, "On streaming coding for low-latency packet transmissions over highly lossy links," *IEEE Commun. Lett.*, vol. 24, no. 9, pp. 1885–1889, Sep. 2020.
[21] A. Cohen, G. Thiran, V. B. Bracha, and M. Médard, "Adaptive causal network coding with feedback for multipath multi-hop communications," *IEEE Trans. Commun.*, vol. 69, no. 2, pp. 766–785, Feb. 2021.
[22] S. Wunderlich, F. Gabriel, S. Pandi, F. H. P. Fitzek, and M. Reisslein, "Caterpillar RLNC (CRLNC): A practical finite sliding window RLNC approach," *IEEE Access*, vol. 5, pp. 20183–20197, 2017.
[23] P. Schwenteck, E. Tasdemir, R. Radeke, and F. H. P. Fitzek, "Performance analysis of caterpillar RLNC for multi-hop communication," in *Proc. IEEE 32nd Annu. Int. Symp. Pers., Indoor Mobile Radio Commun. (PIMRC)*, Sep. 2021, pp. 1439–1444.
[24] V. Bioglio, M. Grangetto, R. Gaeta, and M. Sereno, "On the fly Gaussian elimination for LT codes," *IEEE Commun. Lett.*, vol. 13, no. 12, pp. 953–955, Dec. 2009.
[25] S. Shahzad, E.-S. Jung, J. Chung, and R. Kettimuthu, "Enhanced explicit congestion notification (EECN) in TCP with P4 programming," in *Proc. Int. Conf. Green Human Inf. Technol. (ICGHIT)*, Feb. 2020, pp. 35–40.
[26] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

SHAHZAD (Student Member, IEEE) received the B.S. degree in computer science from COMSATS University Islamabad (CUI), Pakistan, in 2015, and the master's degree in computer engineering from Hongik University, Sejong, South Korea, in 2020. He is currently pursuing the Ph.D. degree with the Department of Intelligent Mechatronics Engineering, Sejong University, Seoul, South Korea. Along with his study, he is a Research Assistant with the Mobile INtelligent Embedded System (MINES) Laboratory, Sejong University. From 2016 to 2018, he was a Laboratory Engineer with CUI Attock Campus, Pakistan. His research interests include network coding, software-defined networks, congestion control algorithms, machine learning applications in networking, and P4 programming.

RASHID ALI (Member, IEEE) received the B.S. degree in information technology (IT) from Gomal University, Pakistan, in 2007, the M.S. degree in computer science (advanced network design), under the supervision of Dr. Stanislav Belenki, and the M.S. degree in informatics from University West, Sweden, under the supervision of Dr. Maria Spante, in 2010 and 2013, respectively, and the Ph.D. degree in information and communication engineering from the Department of Information and Communication Engineering, Yeungnam University, South Korea, in February 2019. From 2007 and 2009, he was with Wateen Telecom Private Ltd., Pakistan, as a WiMAX Engineer with the Operations Research Department. From July 2013 to June 2014, he was with COMSATS University, Pakistan, as a Lecturer. He has more than five years of experience in research, academia, and industry, in the field of information and communication engineering, and computer science. He was a Postdoctoral Research Fellow of the Department of Information and Communication Engineering, Yeungnam University. He was an Assistant Professor with the School of Intelligent Mechatronics Engineering, Sejong University, Seoul, South Korea. He is currently a Marie Skodowska-Curie Research Fellow of the Wireless Networking Group, Department of Information and Communication Technologies, Universitat Pompeu Fabra, Barcelona. His research interests include the Internet of Things (IoT), 5G, future WLANs, wireless sensor networks (WSNs), transport/network/MAC layer protocols, information-centric networking (ICN), embedded systems/smart systems, network-on-a-chip, blockchain, machine/deep learning, reinforcement learning, and federated reinforcement learning for wireless networks

AMIR HAIDER received the B.S. degree in electronics engineering from International Islamic University, Islamabad, Pakistan, in 2012, the M.S. degree in electrical engineering from the University of Engineering and Technology, Taxila, Pakistan, in 2014, and the Ph.D. degree from the Division of Electronics and Electrical Engineering, Dongguk University, Seoul, South Korea, in 2019. He was a Lecturer with the Department of Electrical and Computer Engineering, COMSATS University Islamabad, Pakistan, from August 2014 to August 2015, and later from September 2019 to February 2020, after completing the Ph.D. degree. He was a Postdoctoral Researcher with the School of Intelligent Mechatronics Engineering, Sejong University, Seoul, from March 2020 to February 2021, where he is currently an Assistant Professor. His research interests include URLLC for beyond 5G mobile networks, artificial intelligence-aided wireless communication systems, MIMO antenna design, optical communication systems, and wireless positioning technologies.

HYUNG SEOK KIM (Member, IEEE) received the B.S. degree from the School of Electrical Engineering, Seoul National University, South Korea, in 1996, and the M.S. and Ph.D. degrees from the School of Electrical Engineering and Computer Engineering, Seoul National University, in 1998 and 2004, respectively. In 2003 and 2004, he held a Visiting Researcher positions with the University of Virginia. From 2004 to 2006, he was with the Telecommunication Research and Development Center, Samsung Electronics. He is currently a Faculty Member of the Department of Intelligent Mechatronics Engineering, Sejong University. His research interests include networking with special emphasis on wireless communication and machine learning.