Journal of Reliable Intelligent Environments (2023) 9:283–301
https://doi.org/10.1007/s40860-023-00207-7

ORIGINAL ARTICLE

DeepSHARQ: hybrid error coding using deep learning

Pablo Gil Pereira[1] · Kai Vogelgesang[1] · Moritz Miodek[1] · Andreas Schmidt[2] · Thorsten Herfet[1]

Received: 10 January 2023 / Accepted: 17 May 2023 / Published online: 14 June 2023
© The Author(s) 2023

Abstract
Cyber-physical systems operate under changing environments and on resource-constrained devices. Communication in these environments must use hybrid error coding, as pure pro- or reactive schemes cannot always fulfill application demands or have suboptimal performance. However, finding optimal coding configurations that fulfill application constraints—e.g., tolerate loss and delay—under changing channel conditions is a computationally challenging task. Recently, the systems community has started addressing these sorts of problems using hybrid decomposed solutions, i.e., algorithmic approaches for well-understood formalized parts of the problem and learning-based approaches for parts that must be estimated (either for reasons of uncertainty or computational intractability). For DeepSHARQ, we revisit our own recent work and limit the learning problem to block length prediction, the major contributor to inference time (and its variation) when searching for hybrid error coding configurations. The remaining parameters are found algorithmically, and hence we make individual contributions with respect to finding close-to-optimal coding configurations in both of these areas—combining them into a hybrid solution. DeepSHARQ applies block length regularization in order to reduce the neural networks in comparison to purely learning-based solutions. The hybrid solution is nearly optimal concerning the channel efficiency of coding configurations it generates, as it is trained so deviations from the optimum are upper bound by a configurable percentage. In addition, DeepSHARQ is capable of reacting to channel changes in real time, thereby enabling cyber-physical systems even on resource-constrained platforms. Tightly integrating algorithmic and learning-based approaches allows DeepSHARQ to react to channel changes faster and with a more predictable time than solutions that rely only on either of the two approaches.

Keywords Error control · Transport layer · Hybrid error coding · Machine learning

1 Introduction

The natural component of networked cyber-physical systems (CPS) is resource-constrained devices [1] and dynamic communication channels that do not provide performance guarantees [2, 3]. As a result, fulfilling the application demands in terms of tolerable delay and packet loss rate, packet sizes, and sending data rate is a challenging task [1, 4–7]. A changing environment implies frequent changes to communication parameters [8], such as end-to-end delay, loss rate, or data rate, meaning that key control functions, such as congestion [5, 6, 8–10], rate [9], or error control [11–14], should be able to react quickly enough to adapt to these changes.

Ensuring reliable and timely communication for these demanding applications in changing environments is done using *hybrid error coding*, often referred to as HARQ [15, 16] (Hybrid ARQ). However, to apply HARQ, an appropriate coding configuration must be computed based on the given application and measured channel parameters [13, 17–19]. Under changing conditions, this computation must even be repeated at regular intervals—incurring non-negligible computing overhead on constrained devices. When applied to the physical layer of cellular networks [20], typically only

✉ Pablo Gil Pereira
gilpereira@cs.uni-saarland.de

Kai Vogelgesang
vogelgesang@cs.uni-saarland.de

Moritz Miodek
miodek@cs.uni-saarland.de

Andreas Schmidt
andreas.schmidt@cs.uni-saarland.de

Thorsten Herfet
herfet@cs.uni-saarland.de

[1] Telecommunications Lab, Saarland Informatics Campus, C6.3, Saarbrücken 66123, Germany

[2] Dependable Systems and Software, Saarland Informatics Campus, E1.3, Saarbrücken 66123, Germany

the code rate, as part of the selected modulation and coding scheme (MCS), is adapted. The incremental redundancy follows a fixed schedule with a fixed number and sequence of redundancy versions (RVs). However, on higher layers, specifically the transport layer, this parameterization needs to consider and fulfill application requirements and hence it is a complex task. Finding this configuration has been well-understood mathematically for the last decades [17, 18, 21, 22]—including finding optimal configurations that fulfill application requirements and minimizing redundancy overhead. However, this task does not allow for a closed-form representation whose complexity is independent of the channel parameters. Instead, it is a search problem with a complexity dependent on its input parameters—e.g., a linear increase in round-trip time leads to a more than linear increase of configurations to evaluate. Executing the search for realistic channel parameters on realistic CPS computing devices proved intractable [23].

Based on an efficient, but still intractable, reimplementation of the full search [21], we set out to bring hybrid error coding to resource-constrained devices. In one branch, we approached the problem using machine learning [23], in particular using supervised learning with deep neural networks. In a second branch, we have been successful in decomposing the search problem in stages and improving individual stages algorithmically—achieving optimal redundancy efficiency but shorter inference time [24]. In this article, we look at the decomposed search and combine both algorithmic as well as learning approaches to build DeepSHARQ: a search with minimized run-time but high efficiency.

The contribution of this article is threefold:

(a) We describe a decomposition of the HARQ coding configuration search, allowing for optimizations at different stages.
(b) We implement the search algorithm DeepSHARQ that leverages both algorithmic and learning-based approaches to infer efficient coding configurations in real time.
(c) We evaluate DeepSHARQ and compare it against existing solutions—showing its usability on resource-constr ained devices.

The remainder of this article is structured as follows: first, we describe related approaches to our work (Sect. 2) and give background on error control at the transport layer of packet networks (Sect. 3). How optimal HARQ configurations can be determined is explained in Sect. 4. Our approach, DeepSHARQ, is described in detail in Sect. 5. This is extended by a description of the model training process (Sect. 6) and an evaluation of the search (Sect. 7). Section 8 outlines directions for future research and Sect. 9 concludes the paper.

2 Related work

The end-to-end design paradigm [25] has led to many proposals to complement error coding in the lower layers with coding at the transport layer in order to improve reliability without prohibitively increasing the delay [11, 12, 21, 22, 26–30]. Maximum Distance Separable (MDS) block codes ensure that the number of correctable losses equals the number of transmitted parity packets. MDS codes have been used to provide predictable reliability under time constraints [21], reduce delay in multimedia communication [28], and avoid feedback implosion in multicast [16, 31]. Despite their high loss rate floor, and hence a redundancy transmission overhead to achieve the same performance as MDS codes, binary codes have also been a mechanism of choice due to their reduced coding complexity [11, 29, 30]. Finally, making the end-to-end delay independent of the block length is possible with windowed Random Linear Codes (RLC), which evenly distribute the parity packets over the source packets. RLC codes have proved to reduce the in-order delay, and hence the tail delay in fully reliable protocols [15, 32]. However, this delay reduction in RLC codes comes at the cost of lower code rates than block codes [33], and the run-time complexity of their matrix inversion function hinders their deployment in packetized layers [29, 34, 35]. Michel et al. [12] have extended QUIC with the three aforementioned code families, showing that RLC codes achieve the lowest delay. Although, in this paper, we have opted for block codes, which in principle have a larger delay, we have done so because i) we implement a delay-aware scheme, which ensures that the delay of no packet exceeds the application's target delay, and ii) we target code configurations that approach the theoretical minimum under timing constraints [36] and windowed RLC codes are limited from the code rate standpoint [33].

Like in almost any other field, the significant advances in Deep Learning (DL) have made their way into networked communications [37]—e.g., adaptive video streaming [38, 39], channel state information prediction [27, 40], congestion control [6, 8, 10], and protocol optimization [26, 27, 41]. In the context of error control, Chen et al. [26] use reinforcement learning to select the code rate of an FEC scheme in order to improve the quality-of-experience in the context of real-time video streaming. Cheng et al. [27], implement an LSTM network that predicts the future loss pattern in a block of data packets, and based on it, selects the amount of redundancy to transmit. Hu et al. [19] also use LSTM networks to predict loss patterns, but propose a model compression method to enable fast inference and compensate for the large complexity of LSTM networks.

Non-learning-based approaches have also been proposed to implement adaptive error control [13, 17, 18, 22]. Tickoo et al. [22] implement loss-tolerant TCP that uses an adaptive FEC scheme based on MDS codes that, similar to our

approach, adjusts the transmitted redundancy to the channel characteristics. Adaptive, RLC-based error control is proposed in [17], and the authors show that the proposed mechanism is on par with pure ARQ in throughput- and delay-bound scenarios. [13] proposes a new code construction for low-delay stream codes and presents an adaptive algorithm that outperforms MDS codes. Michel at al. [18] implemented adaptive FEC in QUIC, and evaluated the algorithm's performance for applications with different requirements, showing the benefit of FEC over QUIC's purely reactive error control.

3 Background

Error control is a key function in the most common transport protocols, as it compensates losses in the lower layers in order to provide the desired reliability level. This section introduces the different building blocks in error control.

3.1 Transport layer error control

Networked systems experience packet losses for multiple reasons, e.g., buffer overflows in congested links, channel noise, and fading, and medium access collisions. PHY/MAC layers already implement error correction mechanisms that transmit some form of redundancy that allows for loss recovery. However, these mechanisms fail to provide predictable reliability and end-to-end guarantees [20]. Therefore, error control in the upper layers must complement them [25].

Automatic Repeat reQuest (ARQ) has traditionally been the scheme of choice in the most widely deployed transport protocols—i.e., TCP and QUIC. ARQ requires a feedback mechanism to signal either the reception of packets with acknowledgments (ACK) or packet losses with negative acknowledgments (NAK). TCP implements cumulative ACKs referring to the last, correctly received byte, whereas QUIC implements a selective packet-based mechanism in which every received and processed packet is ACKed. Although an ACK could be issued for every packet, both TCP and QUIC implement ACK aggregation mechanisms that reduce the receiver-side traffic—e.g., see delayed ACKs in TCP [42] and ACK aggregation in QUIC [43]. On the other hand, NAKs have been typically implemented for multicast [44, 45] to avoid the feedback implosion problem—i.e., the sender in a multicast group is overwhelmed by the ACKs from all receivers, both in terms of received traffic and processing time [16]. When packet retransmissions are triggered depends on the implemented loss detection algorithm [14, 46–48]. TCP was originally designed with a purely time-based retransmission mechanism. However, more recent algorithms use duplicate ACKs/NAKs as packet loss signals as well, which provides faster reactions than timers at the risk of wrongly deeming a packet as lost due to packet reordering in the network. Regardless of the implemented algorithm, retransmissions are never triggered before the round-trip time (RTT) that is required to collect feedback for a packet, and hence we say that ARQ's delay is RTT-dependent.

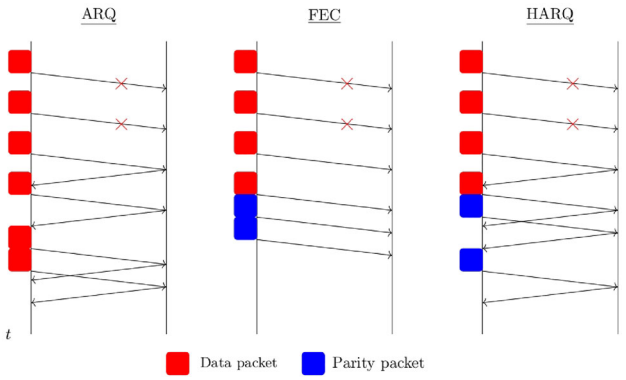

Fig. 1 Comparison of the different redundancy transmission schemes for error control

Obtaining feedback is not always possible if i) the application's target delay is not large enough to wait for feedback, or ii) a feedback channel does not exist (e.g., television broadcasting). In such cases, Forward Error Coding (FEC) is more suitable for the task. Unlike ARQ, FEC proactively transmits redundancy information (RI). As no information about lost packets is available at the time of transmitting the redundancy, FEC must encode *parity packets*, which are a linear combination of data packets, so that losses can be recovered by solving a linear equation system at the receiver (see Sect. 3.2 for a detailed description of how these packets are encoded). As a result, the loss recovery delay is no longer RTT-dependent, but it is proportional to the source packet intervals that the sender must wait to collect packets before encoding.

As the ARQ and FEC delays differ in nature, it stands to reason that both approaches should be combined to provide optimal predictable reliability under delay constraints. When combined, the optimal balance between proactive (FEC) and reactive (ARQ) can be found such that the transmitted RI is minimized. Hybrid ARQ (HARQ) implements precisely that behavior: parity packets can be transmitted in the proactive or reactive cycles, and the sender stops transmitting redundancy when the receiver signals it has enough to recover the losses or until it is too late to recover them in time. Figure 1 provides a graphical comparison of the three aforementioned schemes.

3.2 Packet coding

When implemented in the transport layer, HARQ transmits parity packets—or, more generally, *parity symbols*—to recover the losses. A block code $\mathcal{C}(n, k) : \mathbb{F}_q^k \rightarrow \mathbb{F}_q^n$ trans-

forms a message vector $\vec{m}$ into a code word $\vec{c} \in \mathcal{C}$. The finite field $\mathbb{F}_q$ has size q. Typically, the field is selected from the family of Galois Fields $GF(2^m)$ for binary representation, where m is the number of bits per symbol in the alphabet. Here, k is the block length—number of symbols in $\vec{m}$—, and n the codeword length—number of symbols in $\vec{c}$. The symbols are encoded by performing a matrix–vector multiplication with the generator matrix G ($\vec{c} = \vec{m} \cdot G$). At the receiver, the original message vector is recovered by performing the inverse operation ($\vec{m} = \hat{\vec{c}} \cdot \hat{G}^{-1}$). $\hat{G}$ is a $k \times k$ submatrix of G, whose columns have been selected based on the position of the received symbols $\hat{\vec{c}}$. Figure 2 shows how the encoding operation is performed. We assume a systematic code is used—i.e., the $k \times k$ identity matrix is part of G, and thus the code word contains a verbatim copy of the message vector. Systematic codes reduce the coding complexity as only $p = n - k$ symbols are encoded instead of n, achieve better error correction capabilities: if the linear system cannot be solved—e.g., it is undetermined because fewer than k packets were received—, they can still forward the received verbatim data without decoding, and they also allow for data transmission before all the k packets are collected for encoding, which reduces the end-to-end delay.

While the physical layer performs the coding operation at the symbol level—i.e., directly in bits—, IP networks are packetized erasure channels, meaning that full packets are lost in the network because packets with uncorrectable bit flips are not forwarded to the upper layer, or full packets are dropped due to buffer overflows. As a result, HARQ at the transport layer must be capable of recovering full packets. Assume an IP packet is MTU[1] bytes long. With virtual interleaving the packets can be split into smaller symbols of m bits, k packets are grouped in the interleaver buffer, and the coding operations are iterated throughout the complete packet length. In [29], we showed that the packetization directly impacts the complexity of the system: while the matrix inversion has typically dominated the run-time complexity of coding in the physical layer, the matrix–vector multiplication dominates the packetized layers. As a result, a different code construction may be the best option depending on the channel conditions and platform the protocol runs on.

3.3 Code construction

Three different families of codes have been proposed for the transport layer: MDS [21, 22, 31], binary [11, 29, 30], and RLC codes [18, 34, 35]. They vary in error correction capabilities, underlying field size, and generator matrix construction, and they have different algorithmic tools at their disposal for efficient implementation [31, 49].

[1] MTU is the network's maximum transmission unit of the underlying medium, e.g., 1,500 bytes in Ethernet.

Maximum Distance Separable (MDS) codes [31] guarantee that the minimum distance between codewords is $d_{min} = e + 1$—i.e., they meet the Singleton Bound with equality—, where $e = n - k$ is the number of correctable erasures [50]. For this property to hold true, any $k \times k$ submatrix of G must be invertible. The Cauchy and Vandermonde matrices fulfill this same property, and thus they are frequently used to construct this type of code, usually in $GF(2^8)$ so that symbols are one-byte long.

The matrix inversion is, at the symbol level, the main contributor to the run-time complexity. Binary codes [51–53] overcome this limitation by decoding without an explicit matrix inversion. However, operating in $GF(2)$ does not guarantee the invertibility of every square submatrix. As a result, the loss rate floor is lifted from MDS codes—conversely, binary codes require excess parity packets to achieve the same loss rate as MDS codes. It can be shown that the excess portion of the transmitted redundancy reduces for very large block lengths [52]. Hence, binary codes have dominated physical layer deployments—e.g., LDPC [51] in 4 G and 5 G, or polar [53] codes in 5 G—, where such large block lengths are common. However, they can also perform well in the transport layer when running on resource-constrained devices. Since most CPUs do not directly support operations in high-order Galois Fields, binary codes, which can be implemented with simple XORs, can significantly reduce the run-time complexity [29].

Finally, random linear codes (RLC) follow a random code construction— similar to some binary codes [51, 52], which actually are a sub-family of RLC codes—, in high-order Galois Fields to have a high probability of obtaining linearly independent rows and hence decrease the loss rate floor of random codes. However, these codes need many resources for matrix inversion [34, 35], and it is still an open research question as to whether they can be efficiently used on embedded devices, the natural component of CPS.

In the following, this paper assumes systematic MDS codes are used. However, the presented algorithms are code-agnostic as long as the probability of losing a packet and triggering retransmission rounds (see Eqs. 8 and 2 in Sect. 4.1) are adapted to model other code's properties (e.g., random binary, polar or RLC codes, and non-systematic codes).

4 Predictably reliable, delay-aware error control

Providing predictably reliable, delay-aware error control is only possible with precise models of the communication channel, which must be used to find the optimal configuration subject to application and network constraints. In this

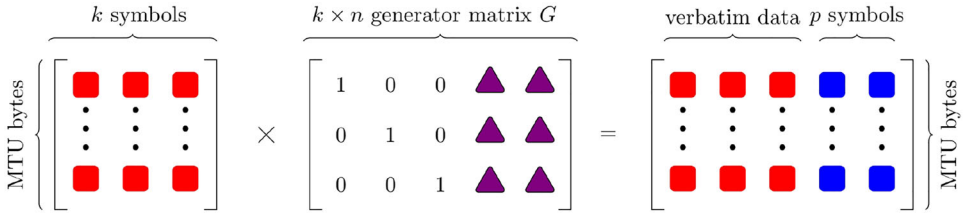

Fig. 2 Encoding process of a systematic code with a block length k, p parity packets and a generator matrix G. Symbols are packets of MTU bytes

section, we introduce SHARQ, an algorithm that finds the optimal configuration in polynomial time.

4.1 Problem statement

The performance of every HARQ scheme is governed by two parameters: the block length k, or how many data packets are encoded, and the repair schedule N_P, which dictates how the p parity packets[2] are distributed among the N_C repair cycles (see Fig. 3). The objective is to find the HARQ configuration that minimizes the transmitted RI (see Eq. 1) while meeting the application and network constraints at the same time. Minimizing the RI is essential for any communication system, otherwise resources—i.e., energy and bandwidth—are wasted due to the throughput increase, which is unfair to the other systems the communication channel is shared with. Formally,

$$
\begin{aligned}
k^*, N_P^* &= \arg\min_{k,N_P} RI(k, N_P) \\
such\ that &: D_{HARQ}(k, N_P) \leq D_T \\
&\quad PLR_{HARQ}(k, \|N_P\|_1) \leq PLR_T \\
&\quad R_{HARQ}(k, N_P) \leq R_C
\end{aligned}
$$

which considers three constraints: (i) every data packet must be received within the application target delay, (ii) the average number of loss packets cannot be greater than the application target loss rate, and (iii) the transmission data rate should not increase beyond the bottleneck data rate of the communication channel.

The redundancy information is a weighted sum over the entries of the repair schedule N_P (see Eq. 1). The weight is the probability of that cycle being required:

$$
\begin{aligned}
&RI(k, N_P) \\
&\quad = \frac{1}{k} N_P[0] + \quad \frac{1}{k} \sum_{c=1}^{N_C} w^R[p[c-1]] \cdot N_P[c]
\end{aligned} \tag{1}
$$

where $p[c] = n[c] - k$ is the cumulative number of parity packets until round c and $w^R[c]$ the weight for $N_P[c]$—i.e.,

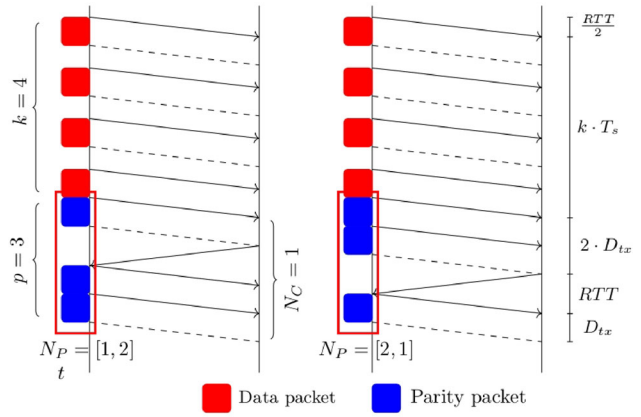

Fig. 3 HARQ delay budget. We analyze the impact of the repair schedule N_P on the achievable capacity of HARQ in the transport layer

the probability of cycle c to be triggered in a multicast group with R receivers.[3] Formally,

$$
w[i] = \sum_{j=max(0,k-p+i)}^{k-1} \binom{k+i}{j} (1-p_e)^j p_e^{k+i-j} \tag{2}
$$

and consequently $w^R[i] = 1 - (1 - w[i])^R$. It can be shown that, for sufficiently large block lengths, the probability of triggering a new retransmission in a binary erasure channel decreases exponentially with the number of cycles. In such a case, the optimal repair schedule can be straightforwardly built: N_P is an all-ones vector except for the last entry, which is $p - N_C + 1$. However, in the short block length regime, such a naive repair schedule construction may be suboptimal [24]: if the probability of cycle $N_C - 1$ to fail is sufficiently high, accumulating packets in later rounds approaches FEC behavior—i.e., all parity packets are transmitted with very high probability. In such cases, parity packets should be brought forward to reduce the probability of latter cycles in the schedule—see Sect. 4.3 for an algorithm that efficiently finds the optimal schedule.

While the FEC delay Eq. 3 depends on the source packet interval T_s to collect k data packets before encoding, the ARQ delay Eq. 4 is RTT-dominated due to the ACK-triggered

[2] The number of parity packets is the 1-norm of the repair schedule vector ($p = \|N_P\|_1$).

[3] For this paper, we limit ourselves to the unicast scenario and therefore $R = 1$. Extending the models to consider multiple receivers is left as future work, for which transfer learning may be an excellent candidate.

Table 1 Model parameters

Parameter	Definition
PLR_T	Target packet loss rate
D_T	Target delay
p_e	Channel erasure rate
RTT	Round trip time
T_s	Inter-packet time
D_{PL}	Loss detection delay
D_{RS}	Processing delay
D_{tx}	Transmission delay
R_C	Channel data rate
P_L	Packet length
k	Block length
p	# of parity packets
n	Codeword length
N_C	# of repair cycles
N_P	Repair schedule

retransmission process.[4] The HARQ delay Eq. 5 can be represented as the combination of its FEC and ARQ components, as depicted in Fig. 3. D_{RS} is the response delay of the system and models operating system delays—e.g., packet management or scheduling. Although a more precise adaptation can be achieved by feeding dynamic response delays into the algorithm [54], we have opted for a rather conservative constant value ($D_{RS} = 1$ ms) to reduce the dimensions of the input dataset—see Sec. 6.1. The model also considers the upper bound to the time required to detect that a packet is lost (D_{PL}), which is the maximum time the system needs to mark a packet as lost after its transmission and hence determines when a new retransmission round is triggered. D_{PL} solely depends on the loss detection algorithm implemented in the transport protocol [14, 46–48]. For the remainder of the paper, $D_{PL} = 4.5 \cdot T_s$, which assumes the mechanism in [46] is implemented (see Sect. 5.4 for more details on why this is the case):

$$\begin{aligned} D_{FEC}(k, N_P) = & \frac{RTT + D_{RS}}{2} \\ & + k \cdot T_s \\ & + N_P[0] \cdot D_{tx} \end{aligned} \tag{3}$$

$$\begin{aligned} D_{ARQ}(N_P) = & \frac{RTT}{2} \\ & + N_C(RTT + D_{RS} + D_{PL}) \\ & + (\|N_P\|_1 - N_P[0]) \cdot D_{tx} \end{aligned} \tag{4}$$

[4] The FEC and ARQ delay expressions here defined assume a symmetric RTT and hence the $\frac{RTT}{2}$ components. For non-symmetric channels, more precise timing information can be obtained e.g., by making the transport protocol time-aware. See [54] for an example of such a transport protocol.

$$\begin{aligned} D_{HARQ}(k, N_P) = & D_{FEC}(k, N_P) \\ & + D_{ARQ}(N_P) \\ & - \frac{RTT}{2} \end{aligned} \tag{5}$$

The error control presented in this paper assumes some periodicity in the application data arrival—i.e., video streaming with a constant frame rate or sensors in CPS with a constant sampling rate. Equation 5 accordingly considers that the inter-packet time is constant for the optimization time window D_T. However, the proposed mechanisms can also be applied to bursty, time-aware traffic: the T_s estimation function must detect a burst—e.g., when the application does not provide further data after one T_s—, in which case a new constraint is added that caps k to the maximum achievable block length for such a burst. The model also considers symmetrical network delay for simplicity. However, in the future, we intend to integrate DeepSHARQ in the time-aware protocol introduced in [55] to also provide predictable error control over networks with asymmetrical delays.

The packet loss rate is given in (6), where $P(I_k = i)$ is the probability of being unable to decode exactly i data packets—i.e., the loss rate as seen by the application—when a systematic MDS code is used and $b = max(p + 1, i)$. Although we have already applied the framework here presented to channels with memory, such as the Gilbert–Elliot channel [21], in this paper, we limit ourselves to the more tractable i.i.d. channels in order to support intuition and plausibility for the reader. This is motivated by the fact that, if the protocol reacts to channel changes fast enough, the underlying channel can be modeled as a binary erasure channel with packet loss probability p_e:

$$PLR_{HARQ}(k, p) = \frac{1}{k} \sum_{i=1}^{k} i \cdot Pr(I_k = i) \tag{6}$$

$$Pr(I_k = i) = \sum_{e=b}^{p+i} \binom{n}{e} \cdot p_e^e \cdot (1 - p_e)^{n-e} \cdot p_d\binom{e}{i} \tag{7}$$

$$p_d\binom{e}{i} = \frac{\binom{k}{i}\binom{n-k}{e-i}}{\binom{n}{e}} \tag{8}$$

While the previous two constraints deal purely with application constraints, the data rate constraint avoids network congestion by ensuring that the transmitted data rate (9) is below the bottleneck data rate of the network R_C:

$$R_{HARQ}(k, N_P) = (1 + RI(k, N_P)) \cdot \frac{P_L}{T_s} \tag{9}$$

Once the formal model is defined, an algorithm must be implemented that finds the optimum fast enough to react to

Algorithm 1 SHARQ

Require: k_{max}, p_{max}
Ensure: $k^*, N_P^* = \arg\min_{k,N_P} RI(k, N_P)$
  $p \leftarrow p_{max}$
  for $k = k_{max} \rightarrow 1$ **do**
    while $PLR_{HARQ}(k, p-1) \leq PLR_T$ **do**
      $p \leftarrow p - 1$
    end while
    $N_C \leftarrow N_{C,max}(k, p)$
    if $N_C < 0$ **then**
      continue
    end if
    $(N_P, RI) \leftarrow$ graph_search(k, p, N_C)
    if $RI < RI(k^*, N_P^*)$ **then**
      $(k^*, N_P^*) \leftarrow (k, N_P)$
    end if
  end for
  return (k^*, N_P^*)

changes within the channel coherence time—i.e., the time the channel properties remain unchanged.

4.2 SHARQ

Scheduled HARQ (SHARQ) is a search algorithm that, given the application delay and loss rate constraints, and the channel state information, finds (k, N_P) that minimizes the RI. SHARQ's algorithm—see Alg. 1—takes as input the maximum block length (k_{max}) and the number of parity packets (p_{max}). Given the maximum block lengths allowed by the delay and loss rate constraints, k_{max} is the minimum of the two: $k_{max} = min(k_{lim}^{D_T}, k_{lim}^{PLR_T})$, where

$$k_{lim}^{D_T} = \left\lfloor \frac{D_T - \frac{RTT+D_{RS}}{2}}{T_s} \right\rfloor$$

$$k_{lim}^{PLR_T} = max\{k \mid PLR_{HARQ}(k, 255-k) \leq PLR_T\}$$

Given $p_{opt}(k) = min\{p \mid PLR_{HARQ}(k, p) \leq PLR_T\}$ the optimal number of parity packets for a block length k to fulfill the packet loss rate constraint, it can be shown that it is a monotonically increasing function. The loss rate constraint solely depends on k and p—see Eq. 6. Therefore, as the block length increases, the RI decreases if p is kept constant: $RI(k, p) > RI(k+1, p)$. Conversely, the PLR increases because the same number of parity packets carries information from more data packets: $PLR_{HARQ}(k, p) < PLR_{HARQ}(k+1, p)$. As a result, $p_{opt}(k-1) \leq p_{opt}(k) \forall k \in [1, k_{max}]$, with the equality holding true if the PLR increase is not large enough to surpass PLR_T. It directly follows that the maximum number of parity packets is $p_{max} = p_{opt}(k_{max})$. Due to the monotonically increasing nature of the PLR, k_{max} and p_{max} can be found with a binary search with a run-time complexity $\mathcal{O}(m \cdot \mathcal{C}_{PLR})$, with m the number of bits per symbol in the Galois Field, and $\mathcal{C}_{PLR} = \mathcal{O}(k + log(p))$ the complexity of obtaining the PLR (see Appendix A for more details on the PLR complexity).

If (k, p) is known, N_C can be directly obtained: as long as there are enough p's to fill later cycles and the delay budget allows it, this cycle can only reduce the RI because every newly transmitted parity packet reduces the probability of later cycles. Therefore, N_C can be directly obtained as the maximum number of cycles that fit in the remaining of the delay budget:

$$N_{c,max}(k, p) = \left\lfloor \frac{D_T - k \cdot T_s - p \cdot \frac{P_L}{R_C} - \frac{RTT+D_{RS}}{2}}{RTT + D_{RS} + D_{PL}} \right\rfloor$$

SHARQ clearly decouples the delay and PLR constraints, resulting in a more structured and efficient exploration of the search space. For every block length, p solely depends on the PLR constraint, whereas N_C solely depends on the delay constraints. Finally, the graph search in Sect. 4.3 is used to find the optimal N_P. The graph search has run-time complexity $\mathcal{C}_{GS} = \mathcal{O}(p^2 N_C)$, and hence the run-time complexity of the SHARQ search algorithm is in $\mathcal{O}(N_{C,max} \cdot k_{max} \cdot p_{max}^2)$.

4.3 Graph search

Algorithm 2 Graph Search

Require: k, p, N_C
Ensure: $N_P^* = \arg\min_{N_P} RI(k, N_P)$
  obtain w^R
  initialize $D[x, y]$ to ∞ for $0 \leq x \leq p$ and $0 \leq y \leq N_C$
  $lower \leftarrow 0$; $upper \leftarrow (p - N_C)$
  for $x = lower \rightarrow upper$ **do**
    $D[x, 0] \leftarrow x$; $Parent[x, 0] \leftarrow x$
  end for
  for $y = 1 \rightarrow N_C$ **do**
    increase $lower$ and $upper$ by 1
    for $x = lower \rightarrow upper$ **do**
      for $x' = (lower - 1) \rightarrow x - 1$ **do**
        $step \leftarrow x - x'$
        $current \leftarrow D[x', y-1] + step \cdot w^R[x']$
        if $current < D[x, y]$ **then**
          $D[x, y] \leftarrow current$
          $Parent[x, y] \leftarrow step$
        end if
      end for
    end for
  end for
  $x \leftarrow p$
  for $y = N_C \rightarrow 0$ **do**
    $step \leftarrow Parent[x, y]$
    $N_P^*[y] \leftarrow step$
    decrease x by $step$
  end for
  $RI = \frac{1}{k} \cdot D[p, N_C]$
  return (N_P^*, RI)

The objective of the graph search algorithm is to find the schedule N_P with minimum RI, given a (k, p) pair and N_C. As seen in Eq. 1, the RI is a weighted sum over the entries of N_P. Each weight is the probability that the corresponding retransmission round is required. This structure creates a trade-off: packets in the later rounds are less likely to be transmitted and hence have a lower cost in terms of RI. However, putting fewer packets into the early rounds increases the probability that the later rounds are needed.

The key observation to efficiently find the optimal schedule is that the weight for round c only depends on the *number* of packets in rounds before c, but not *how* they are scheduled. In other words, if we have already scheduled x packets into y rounds, the cost of assigning dx packets to the next round is the same regardless of how the x packets were scheduled before. This structure can be expressed as a graph Eq. 10:

$$\begin{aligned} G &= (V, E, w_E) \\ V &= \{start\} \cup \left\{(x, y) \mid 0 \le x \le p, 0 \le y \le N_C\right\} \\ E &= \left\{(start, (x, 1)) \mid 0 \le x \le p - N_C\right\} \\ &\quad \cup \left\{((x', y), (x, y+1)) \mid 0 \le y < N_C - 1 \right. \\ &\qquad \left. \wedge\, x - x' \ge 1 \wedge y \le x' \le p - N_C + y\right\} \\ &\quad \cup \left\{((x, N_C - 1), (p, N_C)) \mid N_C - 1 \le x < p\right\} \end{aligned} \tag{10}$$

with edge weights reflecting the RI cost Eq. 11:

$$\begin{aligned} w_E(((x', y-1), (x, y))) &= \frac{1}{k} \cdot w^R[x'] \cdot (x - x'), \\ w_E((start, (x, 1))) &= \frac{1}{k} \cdot x \end{aligned} \tag{11}$$

The edges are chosen to enforce that every retransmission round (i.e., $N_P[c]$ for $c > 0$) is assigned at least one packet. Consequently, we also need to ensure that we do not assign too many packets to one round, as we need at least one packet for every following round. An example of the resulting graph is shown in Fig. 4.

Each path through the graph from the *start* node to (p, N_C) corresponds to a schedule. Since the edge weights are equal to the required RI, the schedule achieving the minimal RI corresponds to the shortest path. This graph can be computed using a dynamic programming approach, shown in Alg. 2. For each layer, we relax the nodes between the *lower* and *upper* bound for the number of packets admissible for the corresponding round as per the restriction above. We store both the minimum distance in D and a parent pointer, allowing us to reconstruct the shortest path in the end.

The edge weights can be obtained in $\mathcal{O}(p)$. Each layer has $\mathcal{O}(p)$ nodes with $\mathcal{O}(p)$ predecessors each. Since there are N_C layers, the time complexity is $\mathcal{O}(p^2 N_C)$.

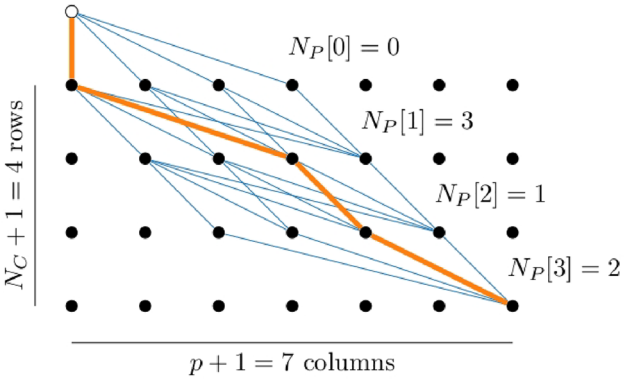

Fig. 4 Graph for $p = 6$ and $N_C = 3$. Each path represents a choice of N_P. The edge weights are set such that the N_P with the lowest RI corresponds to the shortest path. The highlighted edges represent $N_P = [0, 3, 1, 2]$

5 DeepSHARQ

Based on SHARQ's search structure, DeepSHARQ applies learning algorithms to estimate the block length and implements a simple schedule construction to reduce the run-time complexity compared to algorithms that use purely learning and algorithmic solutions.

5.1 Design principles

DeepSHARQ is designed with two main principles in mind: i) in contrast to purely learning-based approaches, DeepSHARQ exploits SHARQ's search structure to simplify the learning problem, thereby requiring smaller neural networks to achieve similar inference accuracy, and ii) DeepSHARQ relaxes the optimality constraint to achieve predictably low inference times.

SHARQ quickly finds p with a binary search and N_C with a closed-form expression. Thanks to SHARQ's structured search, it becomes apparent that the iteration over all possible block lengths and the graph search are responsible for most of the inference time. DeepSHARQ tackles the problem by inferring the block length with a neural network and using a simple repair schedule construction that, despite being suboptimal, does not produce significant RI increases.

5.2 Output space regularization

The quantization of the output space makes small variations in the input produce significantly different block lengths in the output—e.g., how to use an increase in the delay budget? It is possible that the extra time is enough to use yet another retransmission cycle, which may significantly drop the maximum block length that fits in the remaining time. Figure 5 shows how significant the block variations are. For each of the figures, a different input parameter is linearly increased,

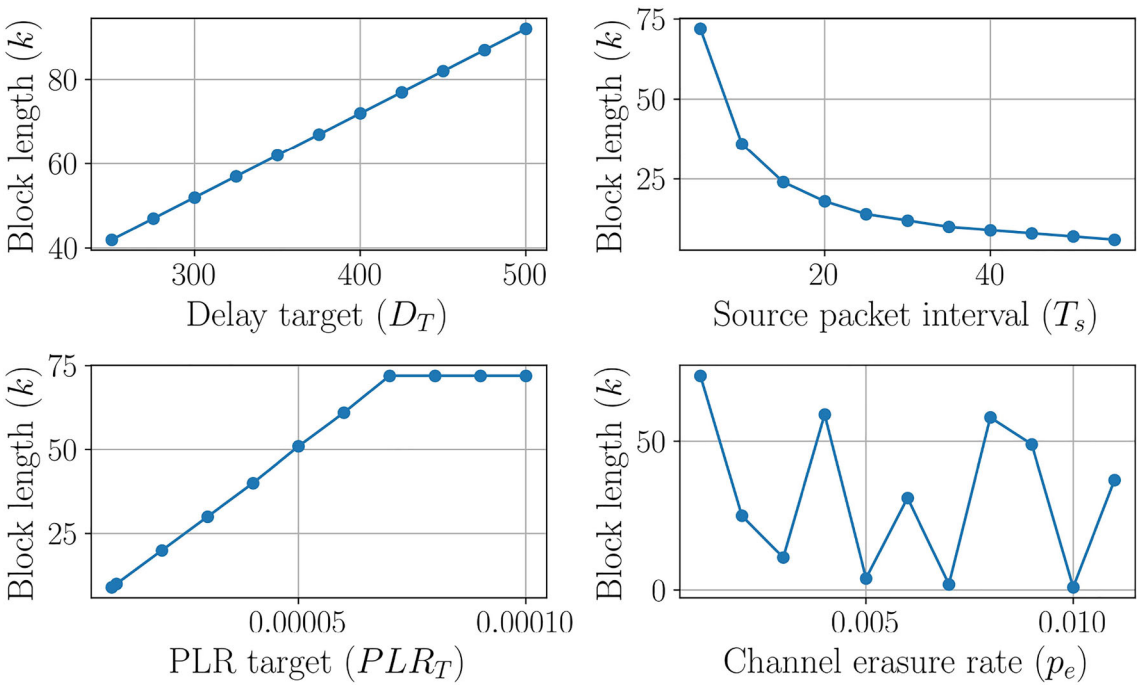

Fig. 5 Optimal block length as a function of the input parameters showing the output space does not react smoothly to small changes in the input space. A different input parameter is linearly increased in each figure, while the baseline is kept constant ($PLR_T = 0.0001$, $D_T = 400$ ms, $R_C = 6$ Mbps, $p_e = 0.001$ and $T_s = 5$ ms). Particularly, it can be observed how linearly increasing p_e produces a quasi-random behavior in the output

while the baseline is kept the same ($PLR_T = 0.0001$, $D_T = 400$ ms, $R_C = 6$ Mbps, $p_e = 0.001$ and $T_s = 5$ ms). Changes in D_T, PLR_T, and T_s produce relatively smooth variations in k that could be easily learned. However, a linear increase in p_e results in a quasi-random behavior in the optimal block length. Although the block length variations may differ for other application and channel models, Fig. 5 clearly illustrates how difficult learning the output space can be. We propose a different training mechanism that tackles this problem by simplifying the output space via regularization. Instead of predicting the optimal label for each configuration, we train networks that predict any block length out of a set of valid block lengths. The set of valid k's is selected so that the RI deviation from the optimal RI is within certain limits. Formally, given the set $\mathcal{K}_v$ of all block lengths that fulfill the requirements—see Sect. 4.1—, the neural network is allowed to predict any block length in the set $\mathcal{K}_v^\delta \subset \mathcal{K}_v$ such that $RI(k, p_{opt}(k)) \leq (1+\delta) \cdot RI_{opt} \, \forall k \in \mathcal{K}_v^\delta$.

5.3 Repair schedule construction

It can be proved that, when error control is provided without any timing constraints, the probability of decoding failure decreases exponentially with every newly transmitted parity packet. The repair schedule in such a case is an all-ones vector so that the contribution of every new packet to the RI decreases exponentially as well—see Eq. 1 in Sect. 4. SHARQ's simple schedule is based on this theoretical optimum: for $N_C = 0$, p packets are transmitted in the FEC cycle, whereas for $N_C \in [1, p]$ the FEC cycle is set to 0, followed by the all-ones and $p - N_C + 1$ in the last cycle. Despite being suboptimal [24], such a naive repair schedule has the advantage that it can be constructed in $\mathcal{O}(1)$ and, as we prove in Sect. 7.3, the RI increase it produces is negligible.

5.4 System architecture

DeepSHARQ's pipeline is depicted in Fig. 6, where the neural network has 4 hidden layers with 150 neurons and leaky ReLU activation function, and a softmax output layer (see Fig. 7). DeepSHARQ inherits some of its algorithmic components from SHARQ [24], namely the binary search for p, the closed-form expression for N_C, and the constraint fulfillment check once the configuration is found in order to notify the application that the channel supports its requirements. On the other hand, the graph search in Sect. 4.3 is substituted by the simple repair schedule construction in Sect. 5.3, so that the run-time complexity of finding the schedule is reduced from $\mathcal{O}(p_{max}^2 N_{C,max})$ to $\mathcal{O}(1)$, and the block length selection goes from a full search for $k \in [1, k_{max}]$ in SHARQ to neural network prediction with run-time complexity $\mathcal{O}(1)$ in DeepSHARQ. As a result, the major contributors to DeepSHARQ's complexity are the binary search for p, with $\mathcal{O}(m \cdot (k_{max} + log(p_{max})))$, and the RI calculation, with

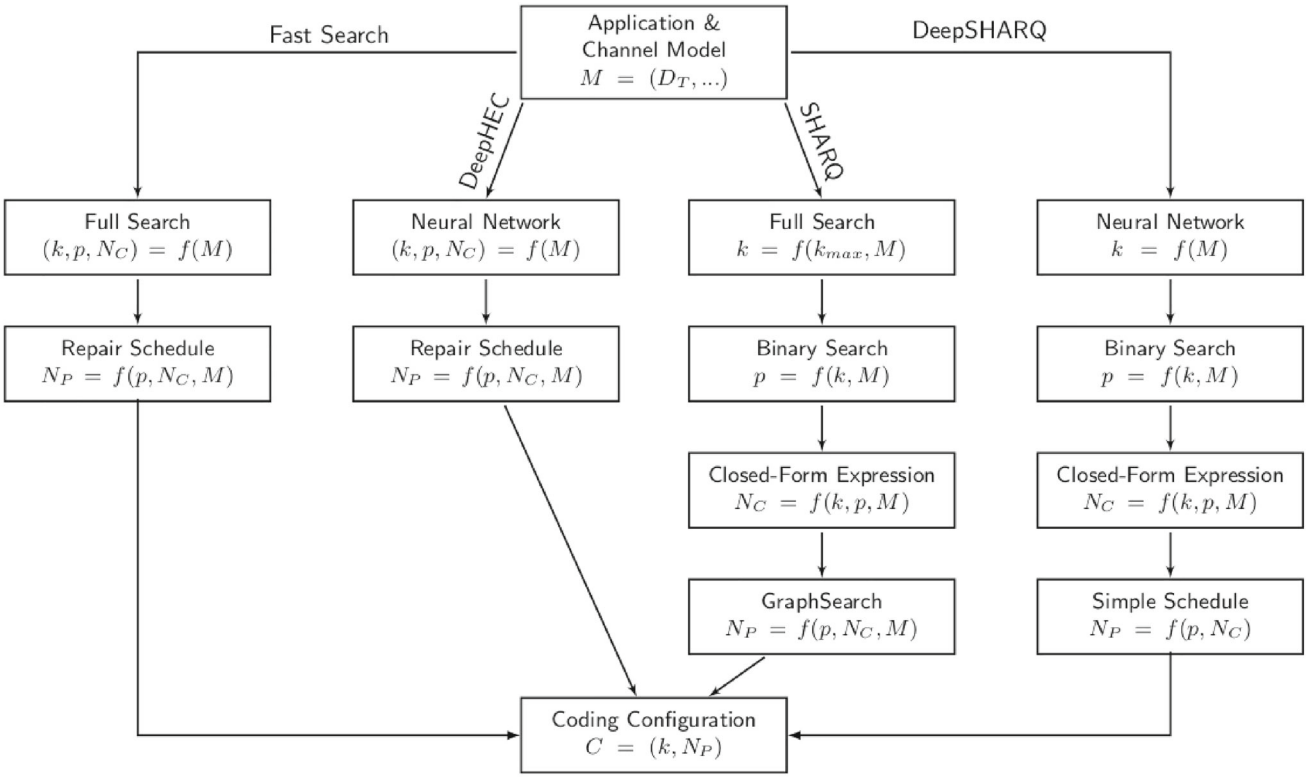

Fig. 6 Architecture and Information Flow for the different Search Algorithms. Each column represents one algorithm implementation. *M* represents a data structure that includes the application constraints and channel model

$\mathcal{O}(k_{max} + p_{max})$. Here m is the maximum number of steps in the binary search, which directly depends on the number of elements in the employed Galois Field $GF(2^m)$—see Sect. 3.2. In the transport layer, $m = 8$ so that symbols are one-byte long and $k_{max} = p_{max} = 2^m$,[5] resulting in DeepSHARQ's run-time complexity $\mathcal{O}(m \cdot k_{max} + p_{max})$. Table 2 shows how the run-time complexity of the search has been reduced with every newly proposed algorithm.

Although DeepSHARQ has not been designed for a specific transport protocol, it assumes the implemented transport layer functions fulfill certain requirements. In the following, we describe such assumptions and how DeepSHARQ interacts with the other transport functions.

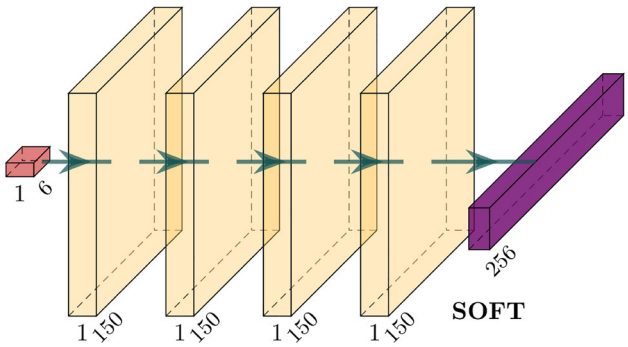

Fig. 7 DeepSHARQ's neural network architecture

Loss detection

DeepSHARQ triggers the repair cycles in the schedule N_P if losses are detected in a block. This paper assumes the algorithm presented in [46] is implemented, which maintains a packet loss count at the receiver that is increased if (i) an out-of-order packet arrives, or (ii) a packet timeout expires. The timeout is configured between 1 and 2 times the inter-packet time (T_s). A new cycle is triggered when the loss count reaches a configurable threshold. The higher the threshold, the higher the algorithm's robustness against in-network packet reordering. For time-bound scenarios with target delays in the same orders of magnitude as the RTT and T_s (see Table 3), packet reordering is equivalent to packet losses if the packets arrive outside of the time budget—i.e., D_T milliseconds after the transmission of the first packet in the block. Therefore, packet reordering has little impact in such scenarios. We consider a low threshold of three loss

[5] Given a systematic code in $GF(2^m)$, it is theoretically possible to construct a (k_{max}, p_{max}) with $k_{max} = p_{max} = 2^m$. However, the MDS coder implementation considered in this paper enforces the code (k, p) to fulfill that $k + p \leq 2^m$. Therefore, both variables must be independently treated in the complexity analysis.

Table 2 Search algorithms run-time complexity

Algorithm	Complexity
Fast Search	$\mathcal{O}(N_{C,max}^2 k_{max}^2 \cdot max(k_{max} p_{max}, p_{max}^2, N_{C,max} k_{max}, N_{C,max} p_{max}))$
DeepHEC	$\mathcal{O}(N_{C,max} k_{max} \cdot max(k_{max} p_{max}, p_{max}^2, N_{C,max} k_{max}, N_{C,max} p_{max}))$
SHARQ	$\mathcal{O}(N_{C,max} k_{max} p_{max}^2)$
DeepSHARQ	$\mathcal{O}(m \cdot k_{max} + p_{max})$

counts and a packet timeout of $1.5 \times T_s$, which results in $D_{PL} \leq 4.5 \cdot T_s$. The delay model in Sect. 4.1 considers the worst-case detection delay to ensure the parity packets arrive at the receiver in time. Recently, new algorithms have been proposed that perform better in channels with significant packet reordering [47, 48]. In future work, we plan to integrate more recent algorithms into our model for faster and more accurate loss detection.

Congestion control

DeepSHARQ ensures the transmitted data rate does not exceed the channel data rate (see Eq. 9 in Sect. 4.1). However, it does not implement any mechanisms to sample the bottleneck data rate and ensure it is not exceeded but relies on congestion control for that. Congestion control is available in most transport layer protocols because it is key for a fair share of the available network resources. Although DeepSHARQ is congestion-control-agnostic and it could in principle coexist with any of the many proposed algorithms, we recommend BBR-like algorithms [9, 54] that try to operate at the Bandwidth-Delay Product (BDP). Operating at the BDP is crucial for CPS as it keeps network buffers empty, thereby minimizing the end-to-end delay while the data rate is close to the bottleneck data rate.

Channel estimation

DeepSHARQ's ability to fulfill the application requirements depends on the precision of the estimated channel model. Another benefit of implementing BBR-like congestion control is that it provides an estimate of two of DeepSHARQ's input parameters: RTT and R_C [56]. An estimation of the remaining parameter, the channel loss rate, is proposed in [21], which uses gaps in the data stream to estimate p_e. In addition, the tolerated delays in CPS are so small that they are typically in the same order of magnitude as the channel coherence time, or even smaller. In other words, the channel can be considered constant during the time budget and [21, 56] provide an estimation precise enough for most IP deployments. Nevertheless, fast-changing, dynamic channels can have a coherence time in the single-millisecond range which pose a more challenging scenario. Machine-learning-based solutions seem promising for such a small granularity as well [26, 27]. We believe this is an interesting parallel research path that could enable DeepSHARQ even in the most demanding channels.

6 Model training

Finding the right hyperparameters is essential to achieve good performance in data-intensive tasks. This section analyzes the different components used in the learning process to shed some light on the model selection process, as well as to ensure the results are reproducible.

6.1 Dataset generation

We have designed the dataset with two objectives in mind: i) it must represent current deployments faithfully, and ii) it must generalize for any of the included deployments. The model uses six input parameters:

- Application parameters: target erasure rate PLR_T, target delay D_T, and source packet interval T_s.
- Network parameters: channel data rate R_C, channel erasure rate p_e, and round-trip time RTT.

We have considered traces obtained in the wild for the most common network deployments—i.e., broadband,[6] 4 G [57], 5 G [58, 59], and WiFi [60, 61] deployments. For application-related parameters, we have used delay and reliability constraints of traditional [62][7] as well as more demanding applications still under deployment [7, 63]. For each of the parameters, an order of magnitude is selected from Table 3 with equal probability, and a randomly selected number between 1 and 9 is prepended to that order of magnitude. Finally, Alg. 1 is executed for the input with a slight modification: not only the optimal block length k_{opt} is logged, but k_{min} and k_{max} are also obtained, which respectively are the minimum and maximum block lengths that ensure the RI only deviates δ from the optimal RI (see Sect. 5.2). The

[6] Federal Communications Commission, "Raw Data - Measuring Broadband America", https://www.fcc.gov/oet/mba/raw-data-releases, 2020.

[7] Youtube, "Recommended upload encoding settings", https://support.google.com/youtube/answer/1722171?hl=en, 2021.

Table 3 Selected orders of magnitude for the generation of the parameter dataset

Parameter	Orders of Magnitude	Unit	Reference
PLR_T	$10^{-3}, 10^{-4}, 10^{-5}$	Rate	[62]
D_T	$10^0, 10^1, 10^2$	ms	[7, 62, 63]
p_e	$10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}$	Rate	[59, 60]
RTT	$10^0, 10^1, 10^2$	ms	[59, 61]
R_C	$10^{-1}, 10^0, 10^1, 10^2, 10^3$	Mbps	[57, 59, 61][6]
T_s	10^{-1},$10^0, 10^1$	ms	[7]

complete dataset consists of 2.5 million inputs, 42.4% of which do not have an empty solution space—i.e., there is at least one configuration fulfilling all the constraints at the same time, see Sect. 4.1. The neural networks have only been trained with these 1,060,527 inputs without an empty solution space, split into training (60%), validation (20%), and test (20%) sets. Such a dataset simplification reduces the time and resources spent in training without a negative impact on the system's performance, as DeepSHARQ nevertheless discards any predicted k that does not meet the constraints.

The models in Sect. 4 consider other three parameters that are not included in the dataset: the packet length P_L, the processing delay D_{RS}, and the loss detection delay D_{PL}. We assume the packet length is fixed to the MTU, and hence $P_L = 1,500$ bytes. We also considered a rather conservative constant value for the processing delay $D_{RS} = 1$ ms to reduce the dimensions of the dataset. Finally, D_{PL} is linearly dependent on T_s, and hence it adds no new information as an input.

6.2 Loss definition

Unlike common classification problems, in which the neural network learns the mapping from the input to a single valid output, DeepSHARQ's neural network is trained to accept as correct any label within a range. Therefore, we propose a new loss that accounts for the fact that the true label is a set and not a single value. Given the true label k_i, and the neural network prediction $\hat{k}_i$, the proposed loss is based on the binary cross-entropy $H(k_i, \hat{k}_i)$, which is defined as follows:

$$H(k_i, \hat{k}_i) = p[k_i \in \mathcal{K}_v^\delta] \cdot log(p[\hat{k}_i \in \mathcal{K}_v^\delta]) + (1 - p[k_i \in \mathcal{K}_v^\delta]) \cdot log(1 - p[\hat{k}_i \in \mathcal{K}_v^\delta])$$

where $p[\hat{k}_i \in \mathcal{K}_v^\delta]$ is the probability that the neural network predicts any block length that belongs to the accepted range, and $p[k_i \in \mathcal{K}_v^\delta] = 1$ because the true label always belongs to that range by definition:

$$L(k, \hat{k}) = -\frac{1}{N} \sum_{i=1}^{N} log(p[\hat{k}_i \in \mathcal{K}_v^\delta]) \quad (12)$$

The loss $L(k, \hat{k})$ in Eq. 12 is evaluated for every batch of size N. In Sect. 7, we show that this loss allows the model to correctly learn the mapping from input parameters to any label in the set of valid labels.

6.3 Ablation study

Tuning the learning rate hyperparameter is instrumental to successfully training neural networks. PyTorch implements various learning rate policies that can be configured to schedule the learning rate, such as *plateu*[8] or *super-convergence*[9] [64]. Both policies benefit from large maximum learning rates that allow for a longer exploration phase and low learning rates to fine-tune the model weights. The super-convergence scheduler begins with a rising phase that goes from $start\_lr$ to max_lr, after which it decays towards end_lr, which is substantially lower than $start\_lr$. The plateau learning rate policy monitors the validation loss to estimate the effectiveness of the current learning rate (i.e., if after $patience$ epochs the validation loss did not decrease by at least $threshold$ amount, it decays the learning rate by a constant $factor$ until the min_lr has been reached). The super-convergence policy requires an optimizer with momentum, and hence we trained all the models with momentum-enabled stochastic gradient descent. Super-convergence varies the momentum between 0.85 and 0.95, while it is constant at 0.9 for plateau.

Figure 8 shows both policies' accuracy and learning rate evolution with DeepSHARQ's neural network limited to 1,000 epochs. The plateau policy reaches lower learning rates faster than super-convergence resulting in higher initial accuracy, but convergence towards lower accuracy in the second half of the training. On the other hand, super-convergence surpasses plateau in accuracy for the last hundred epochs due to its extended high learning rate exploration phase. We selected super-convergence with $max\_lr = 0.04$, as it achieves the best performance.[10]

Training the models for 1,000 epochs takes approximately 24 h on a PC with an Intel Core i7-7700 CPU at 3.6 GHz and 8 cores, with an average core load of approximately 50%. The main bottleneck is the calculation of the loss function in Sect. 6.2, which, unlike the traditional cross-entropy loss, must be independently evaluated for every sample in

[8] https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html.

[9] https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.OneCycleLR.html.

[10] Results for $max\_lr = 0.05$ or higher are not included, as such large learning rates overshoot and achieve worse performance.

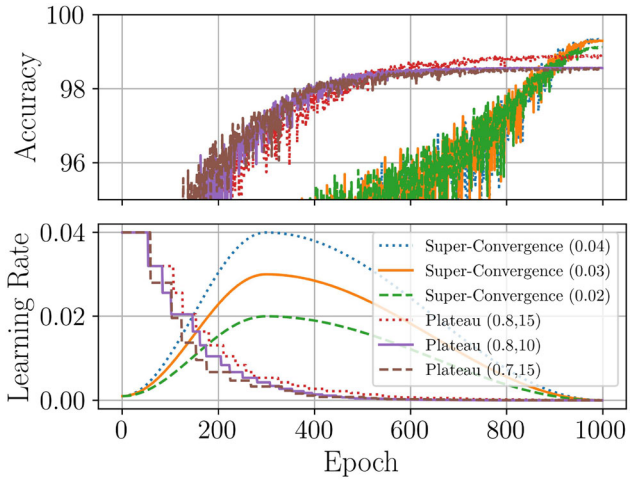

Fig. 8 Validation accuracy and learning rate evolution for different learning rate policies. Three maximum learning rates have been used for super-convergence (i.e., 0.02, 0.03, and 0.04), whereas patience (10 and 15) and learning rate reduction factors (0.7 and 0.8) have been used for plateau

the batch because every input may consider a different $\mathcal{K}_v^\delta$ set. However, the significantly smaller models (see Sect. 7.2) counteract the impact of the longer training phase when DeepSHARQ is deployed at scale on a significant number of end devices. In addition, thanks to the broad set of channels and applications considered in Sect. 6.1, DeepSHARQ is readily deployable on the most common networks nowadays without a lengthy re-training for fine-tune adaptation.

Table 4 presents the ablation study we performed to select the final hyperparameters. All the presented results are for models trained for a range of valid labels for $\delta = 0.3$ (see Sect. 6.2). The regularization factor is a key parameter for super-convergence, as high learning rates already act as a form of regularization [64] and, combining it with L2 regularization with a high factor can be detrimental for performance (see Table 4 rows 2 and 6). We also tested multiple epochs and selected 1,000 as it strikes the right balance between good performance and training time.

For DeepSHARQ, we have selected the model with 4 hidden layers and 150 neurons because, for the selected δ, it achieves a good compromise between accuracy and model size. However, the method here proposed is flexible enough to allow for different model selections: while a large model with close-to-optimal accuracy may be a good option on powerful PCs, smaller models—e.g., using larger δ's, and hence at the cost of an RI increase—may be desirable for resource-constrained platforms.

Table 4 Ablation study. The super-convergence learning rate policy with $max\_lr = 0.04$ has been used for all the models. The accuracy is calculated for a range of valid labels and $\delta = 0.3$

# of Parameters	Layers	Neurons	Epochs	Reg. Factor	Accuracy
56,856	4	100	1000	10^{-5}	99.24
56,856	4	100	1000	10^{-4}	97.21
107,656	4	150	1000	10^{-5}	99.57
107,656	4	150	1000	10^{-4}	97.20
107,656	4	150	500	10^{-5}	98.99
107,656	4	150	1500	10^{-5}	99.77
213,656	5	200	1000	10^{-5}	99.9
317,006	5	250	1000	10^{-5}	99.94

7 Evaluation

In the following, we evaluate the newly proposed neural network approach, as well as DeepSHARQ's real-time response to channel changes.

7.1 Methodology

All the models evaluated in this section have been trained following Sect. 6 and PyTorch 1.12.1. Only the test dataset has been used to generate the accuracy and Cumulative Distribution Functions (CDF) here presented, and the algorithms have been executed on a PC running Ubuntu 22.04 LTS on an Intel Core i7-7700 CPU at 3.6 GHz and 32 GB RAM. For inference time evaluation, the PyTorch-trained models have been ported to TensorFlow 2.11 using the TensorFlow Backend for ONNX[11] and executed with TensorFlow Lite with the `tflite`[12] Rust crate. Two different neural networks are considered: i) k_{opt}, trained to predict the optimal block length, and ii) k_{range}, trained to predict any block length in a range of valid block lengths—see Sect. 5.

7.2 Model performance

Figure 9 compares k_{opt} and two k_{range} versions ($\delta = 0.1$ and $\delta = 0.3$) in terms of validation accuracy and loss. The models were trained for 1,000 epochs, using the super-convergence learning rate policy configured with $max\_lr = 4 \cdot 10^{-3}$, $start\_lr = 2 \cdot 10^{-2}$, and $min\_lr = 2 \cdot 10^{-9}$. The smoother output space allows the models to faster converge towards high accuracy—conversely, low loss–, and experience a lower variance within the high learning rate in super-convergence—see Sect. 6. This result is somehow expected since more labels are accepted as "correct" for k_{range} models, and hence the accuracy as defined in the learning process is increased. However, extending the range of

[11] https://github.com/onnx/onnx-tensorflow.

[12] https://crates.io/crates/tflite.

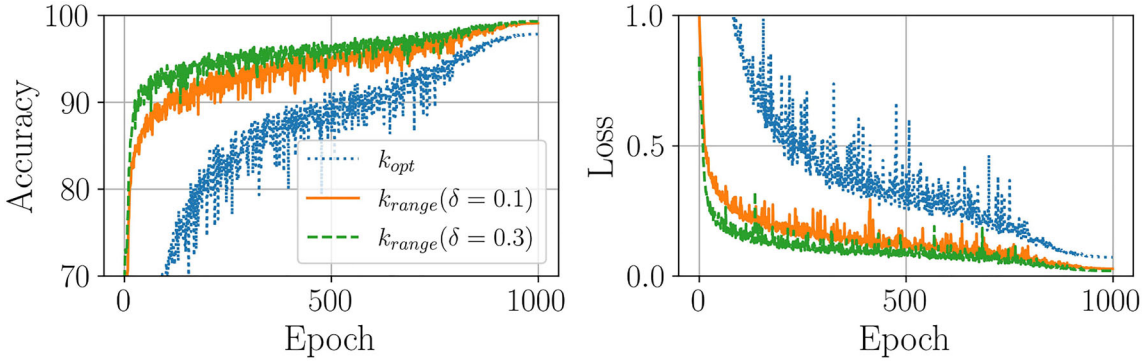

Fig. 9 Evolution of the validation accuracy and loss for a neural network with 4 hidden layers with 150 neurons each

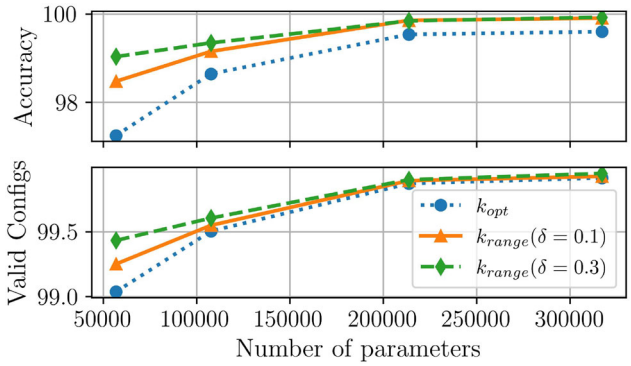

Fig. 10 Model test accuracy as a function of the number of parameters. Four models are depicted with different hidden layers and neurons: (4,100), (4,150), (5,200), and (5,250)

valid labels also improves the model performance from an information theoretical standpoint (see Fig. 10). If a *valid configuration* is defined as any configuration with an information rate below the time-bound channel capacity—i.e., any configuration meeting all the constraints—then k_{range} models are able to find more valid configurations over the test dataset the larger the range of valid labels is. Such a trade-off between RI optimality and model size is particularly beneficial for resource-constrained devices, in which the bottleneck is the CPU rather than the network, both in terms of processing speed and energy consumption [29], especially when connected to 5 G networks [59]. In contrast, the DeepHEC model presented in [23] needs 5 hidden layers with 250 neurons each to reach 99.59% valid configurations in the test dataset. Bear in mind the aforementioned DeepHEC model predicts k, p and N_C with 99.75%, 99.82%, 99.94% accuracy, respectively. In other words, DeepHEC needs 3.54× more parameters than DeepSHARQ (i.e., 107,656 DeepSHARQ vs. 381,63 DeepHEC) in order to support the same configurations.

7.3 The cost of optimality

DeepSHARQ slightly deviates from the optimization problem defined in Sect. 4.1, as it introduces two sources of suboptimal configurations: i) the neural network, which can either be directly trained to allow for suboptimal block lengths, or produce misclassifications even when trained to predict the optimum, and ii) the simple schedule construction. Figure 11 compares k_{opt} with 5 layers, 250 neurons, and k_{range} with 4 layers, 150 neurons, each model implementing two different repair schedule constructions: the graph search

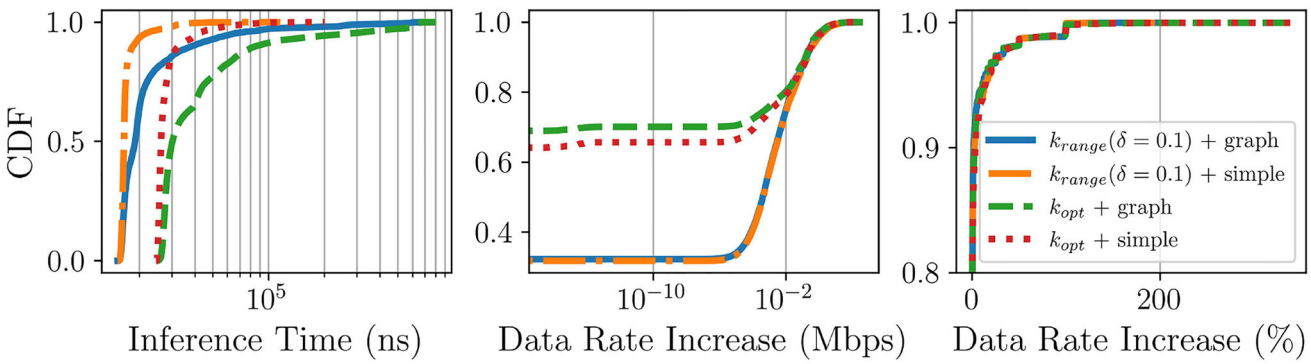

Fig. 11 Cumulative Distribution Function (CDF) of DeepSHARQ's inference time (log scale), absolute data rate increase (log scale), and data rate increase as a percentage of the optimum data rate (linear scale)

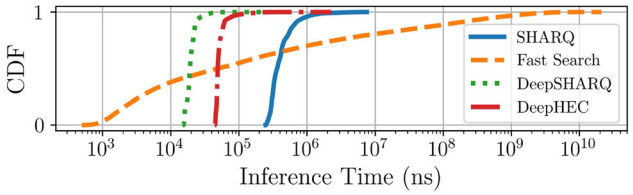

Fig. 12 Cumulative Distribution Function (CDF) of the inference time. DeepSHARQ's neural network has 4 hidden layers and 150 neurons per layer, and it has been trained with a dataset ensuring that $RI = (1+\delta) \cdot RI_{opt}$ with $\delta = 0.1$

Table 5 Mean, median, standard deviation, and 99th percentile of the inference time in microseconds for the different algorithms

Algorithm	Mean	Median	SD	99th per.
DeepSHARQ	20.11	19.09	7.48	39.18
DeepHEC	57.40	48.63	233.21	151.34
SHARQ	462.8	363.74	388.07	1.88×10^3
Fast Search	111.09×10^3	50.287	528.94×10^3	2.5×10^6

in Sect. 4.3 (*graph*) and the simple repair schedule in Sect. 5 (*simple*). The results show that allowing for suboptimal configurations reduces the tail and average inference time, and increases its predictability. SHARQ's graph search for optimal repair schedules results in a long tail in the inference time due to its quadratic complexity, while increasing the unpredictability of the system at the same time. On the other hand, k_{range} reduces the average delay by 40% in comparison to k_{opt} due to the smaller NN.

The faster inference time comes at the cost of a data rate increase. As expected, k_{opt} produces no increase in significantly more cases than k_{range}—69% vs. 32%. However, in both cases, the increase is below 100 kbps in 87% of the cases and below 1 Mbps for the 97th percentile, and thus they do not seem prohibitively large when looking at the data rates in current deployments [57, 59, 61]. When it comes to the tail data rate increase, k_{opt} performance is worse than k_{range}, which shows that learning a range instead of a single label acts as a regularization mechanism that improves generalizability. Finally, although the graph search makes a slight difference for k_{opt}, it does not make any significant difference for k_{range}, which further supports the design decision of opting for a suboptimal but faster scheduler.

7.4 Inference time

Figure 12 compares DeepSHARQ's inference time with three previously published algorithms: SHARQ [24], Fast Search [23], and DeepHEC [23]. Although Fast Search outperforms any other model in 43% of the cases, it also has a tail delay—i.e., the largest inference time experienced by the algorithm—6 orders of magnitude higher than DeepSHARQ's. The three other models trade some inference time in the lower percentiles to provide a much more predictable inference time over the complete dataset. More precise statistics on the inference time are collected in Table 5, which shows that not only DeepSHARQ outperforms the other algorithms in terms of the mean and median inference time, but its standard deviation is at least two orders of magnitude smaller.

SHARQ shows that a purely algorithmic solution is able to achieve high predictability. However, deep learning solutions are the only ones able to consistently achieve low delay—see DeepHEC and DeepSHARQ. DeepSHARQ outperforms all other models in terms of tail inference time and predictability thanks to i) its smaller neural network, which is the component consuming most of the delay budget, and ii) its simple schedule, which avoids spending precious time in finding a better schedule that nevertheless produces no significant RI reduction—see Sec. 7.3.

The results presented here show that modeling the problem purely with deep learning results in excessively large models. DeepHEC learns (k, p, N_C), and hence needs larger neural networks to achieve similar performance in terms of supported coding configurations (5 hidden layers and 250 neurons per layer, see [23]). DeepSHARQ halves the inference time compared to DeepHEC, and its tail delay is an order of magnitude smaller. The repair schedule construction is the only algorithmic component that remains in DeepHEC. Although it could be learned as well—e.g., applying reinforcement learning—, the neural network is expected to grow even larger due to the increased complexity of the problem to solve. Combining both approaches, which DeepSHARQ does, simplifies the problem enough for small neural networks to learn it while minimizing the time spent to derive the remaining parameters and the verification of the constraint fulfillment. DeepSHARQ's average inference time is an order of magnitude smaller than the end-to-end delay requirements of tactile applications [7], so that it can react to channel changes faster than they are detected even for the most demanding applications delay-wise. Two orders of magnitude until the target delay of the most demanding applications is reached leaves plenty of room for increased inference time when the algorithm is executed on more constrained devices.

8 Future work

While initial works have evaluated the coding configuration search in a concrete transport protocol [21], this and other recent articles looked at the search problem in isolation. Future work would be to integrate DeepSHARQ into existing transport layer protocols (e.g., QUIC [43] or CoAP [65]) and evaluate the performance under changing channel conditions. These practical evaluations would evaluate the

ability of the protocol to meet the application constraints in a changing environment, and they would also involve the use of embedded devices—testing if they are capable to run the search in real time. In addition, we intend to further integrate DeepSHARQ with state-of-the-art loss detection algorithms [47, 48].

In [29], we proved that a priori there is no single code that is optimal in terms of energy efficiency, but this depends on the hardware it is executed on, as well as channel conditions and application requirements. Given the recently increasing interest in energy-aware systems [66, 67], a further line of research would involve making DeepSHARQ energy-aware as well. This involves changing the search problem to incorporate energy as an input metric, i.e., an application-defined energy limit per application packet, and an output metric, i.e., how much energy a coding configuration demands.

From a theoretical communication perspective, a further line of research involves results on finite block coding [36]. These results allow computing the optimal block length based on channel parameters. Central to this is, however, the computation of the *dispersion* metric—something that has not been done practically in network protocols.

9 Conclusion

In this article, we presented DeepSHARQ, an approach to finding optimal hybrid error coding configurations in real time. Coming from the (computationally complex) search problem, we presented a decomposed search algorithm that improves the complexity of the algorithm using algorithmic as well as learning-based methods. We propose a new training methodology that, exploiting the quantized nature of the HARQ configurations, improves the neural network performance in learning (i.e., achieved accuracy) as well as communication (i.e., yielded configurations supported by the channel capacity) terms. Our evaluations show that DeepSHARQ is delivering both on the efficiency of the inferred configurations as well as on being fast in executing the inference. To the best of our knowledge, this is the best approach so far to finding coding configurations and makes it possible to execute this demanding task on devices with limited resources—a common trait of cyber-physical system hardware.

Author Contributions PG, KV, and TH developed the algorithms. PG, KV, AS, and TH conceptualized the experiments. PG, KV, and MM carried out the experiments. PG and AS wrote the main manuscript. KV contributed to Sect. 4. MM contributed to Sect. 6. All the authors have reviewed the text and agree with its content.

Funding Open Access funding enabled and organized by Projekt DEAL. This work was funded by the German Research Foundation (DFG) grants 315036956 as part of SPP 1914—e.LARN (see http://larn.systems) and 389792660 as part of TRR 248—CPEC (see https://perspicuous-computing.science).

Data availability statement The datasets, algorithm implementation, and scripts used to generate the results presented here are available through the GitLab webpage of some of the authors (https://git.nt.uni-saarland.de/open-access/deepsharq) as well as Zenodo (https://doi.org/10.5281/zenodo.8026445).

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose. The authors have no conflicts of interest to declare that are relevant to the content of this article. All the authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript. The authors have no financial or proprietary interests in any material discussed in this article.

Ethical approval Not applicable.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

Appendix A

The run-time complexity of the HARQ packet loss rate in Eq. 6 can be reduced to linear complexity if some optimizations are taken into account. Splitting the PLR into the two components in Eq. A1, it can be shown that each of the components can be calculated in $\mathcal{O}(k_{max} + log(p_{max}))$ (see Eq. A2 and Eq. A4). Bear in mind that the derivations here presented do consider the alternative expression for the probability of i packet losses in systematic MDS codes presented in Eq. A5:

$$PLR_{HARQ}(k,p) = \underbrace{\frac{1}{k}\sum_{i=1}^{min(k,p)} i \cdot Pr(I_k = i)}_{PLR_1(k,p)} + \underbrace{\frac{1}{k}\sum_{i=p+1}^{k} i \cdot Pr(I_k = i)}_{PLR_2(k,p)} \tag{A1}$$

$$PLR_1(k,p) = p_e^p (1-p_e)^k \sum_{i=1}^{min(k,p)} i \cdot \binom{k}{i} \cdot PS_p(p,i) \tag{A2}$$

Algorithm 3 PLR

Require: k, p, p_e
Ensure: $PLR = \frac{1}{k}\sum_{i=1}^{k} i \cdot Pr(I_k = i)$
  $ps_p \leftarrow 0$
  $plr_1 \leftarrow 0$
  for $i = 0 \rightarrow min(k, p)$ **do**
    $plr_1 \leftarrow plr_1 + i \cdot \binom{k}{i} \cdot ps_p$
    $ps_p \leftarrow \frac{p_e}{1-p_e} \cdot \left(\binom{p}{p-i} + ps_p\right)$
  end for
  $plr_1 \leftarrow plr_1 \cdot p_e^p \cdot (1-p_e)^k$
  $plr_2 \leftarrow 0$
  if $p \leq k$ **then**
    $pow \leftarrow p_e^p \cdot (1-p_e)^{k-p}$
    for $j = p+1 \rightarrow k$ **do**
      $pow \leftarrow pow \cdot \frac{p_e}{1-p_e}$
      $plr_2 \leftarrow plr_2 + j \cdot \binom{k}{j} \cdot pow$
    end for
  end if
  return $\frac{(plr_1+plr_2)}{k}$

$$PS_p(p,i) = \frac{p_e}{1-p_e}\left(\binom{p}{i} + PS_p(p,i-1)\right) \tag{A3}$$

$$PLR_2(k,p) = \frac{1}{k}\sum_{i=p+1}^{k} i \cdot \binom{k}{i} p_e^i (1-p_e)^{k-i} \tag{A4}$$

$$Pr(I_k = i) = \begin{cases} \binom{k}{i} p_e^i (1-p_e)^{k-i} & i > p \\ \binom{k}{i} \sum_{f=1}^{i} \binom{p}{p-i+f} p_e^{p+f} (1-p_e)^{k-f} & i \leq p \end{cases} \tag{A5}$$

The complete algorithm to obtain the PLR is shown in Alg. 3, where $PS_P(p,0) = 0$.

References

1. Lee EA (2008) Cyber physical systems: Design challenges. In: 2008 11th IEEE International Symposium on Object and Component-oriented Real-time Distributed Computing (ISORC), pp. 363–369 . IEEE
2. Bernat G, Burns A, Liamosi A (2001) Weakly hard real-time systems. IEEE Trans Comput 50(4):308–321
3. Josephrexon BJ, Maggio M (2022) Experimenting with networked control software subject to faults. In: Proceedings of the 61st IEEE Conference on Decision and Control . IEEE
4. Baillieul J, Antsaklis PJ (2007) Control and communication challenges in networked real-time systems. Proc IEEE 95(1):9–28
5. Dong M, Meng T, Zarchy D, Arslan E, Gilad Y, Godfrey B, Schapira M (2018) PCC Vivace: Online-Learning Congestion Control. In: 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), pp. 343–356
6. Ma Y, Tian H, Liao X, Zhang J, Wang W, Chen K, Jin X (2022) Multi-objective congestion control. In: Proceedings of the Seventeenth European Conference on Computer Systems, pp. 218–235
7. The Tactile Internet - ITU-T Technology Watch Report. Technical report, ITU (August 2014)
8. Tian H, Liao X, Zeng C, Zhang12 J, Chen K (2022) Spine: An Efficient DRL-based Congestion Control with Ultra-low Overhead. In: Proceedings of the 18th International Conference on Emerging Networking EXperiments and Technologies
9. Cardwell N, Cheng Y, Gunn CS, Yeganeh SH, Jacobson V (2017) BBR: congestion-based congestion control. Commun ACM 60(2):58–66
10. Jay N, Rotman N, Godfrey B, Schapira M, Tamar A (2019) A deep reinforcement learning perspective on internet congestion control. In: International Conference on Machine Learning, pp. 3050–3059 . PMLR
11. Garrido P, Sanchez I, Ferlin S, Aguero R, Alay O (2019) rQUIC: Integrating FEC with QUIC for robust wireless communications. In: 2019 IEEE Global Communications Conference (GLOBECOM), pp. 1–7 . IEEE
12. Michel F, De Coninck Q, Bonaventure O (2019) QUIC-FEC: Bringing the benefits of Forward Erasure Correction to QUIC. In: 2019 IFIP Networking Conference (IFIP Networking), pp. 1–9 . IEEE
13. Emara S, Fong SL, Li B, Khisti A, Tan W-T, Zhu X, Apostolopoulos J (2021) Low-latency network-adaptive error control for interactive streaming. IEEE Trans Multimedia 24:1691–1706
14. Stevens WR (1997) TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. RFC 2001, RFC Editor (January). http://www.rfc-editor.org/rfc/rfc2001.txt
15. Huang Y-Z, Mehrotra S, Li J (2009) A hybrid FEC-ARQ protocol for low-delay lossless sequential data streaming. In: 2009 IEEE International Conference on Multimedia and Expo, pp. 718–725 . IEEE
16. Rubenstein D, Kurose J, Towsley D (2001) A study of proactive hybrid FEC/ARQ and scalable feedback techniques for reliable, real-time multicast. Comput Commun 24(5–6):563–574
17. Cohen A, Malak D, Bracha VB, Médard M (2020) Adaptive causal network coding with feedback. IEEE Trans Commun 68(7):4325–4341
18. Michel F, Cohen A, Malak D, De Coninck Q, Médard M, Bonaventure O (2022) FlEC: Enhancing QUIC with application-tailored reliability mechanisms. IEEE/ACM Transactions on Networking
19. Hu H, Cheng S, Zhang X, Guo Z (2021) LightFEC: Network adaptive FEC with a lightweight deep-learning approach. In: Proceedings of the 29th ACM International Conference on Multimedia, pp. 3592–3600
20. Dahlman E, Parkvall S, Skold J (2020) 5G NR: The Next Generation Wireless Access Technology. Academic Press, Radarweg 29, 1043 NX Amsterdam, The Netherlands
21. Gorius M (2012) Adaptive Delay-constrained Internet Media Transport. PhD thesis, Saarland Informatics Campus
22. Tickoo O, Subramanian V, Kalyanaraman S, Ramakrishnan K (2005) LT-TCP: End-to-end framework to improve TCP performance over networks with lossy channels. In: International Workshop on Quality of Service, pp. 81–93 . Springer
23. Pereira PG, Schmidt A, Herfet T (2022) DeepHEC: Hybrid Error Coding using Deep Learning. In: 2022 18th European Dependable Computing Conference (EDCC), pp. 17–24 . IEEE
24. Vogelgesang K, Pereira PG, Herfet T (2023) SHARQ: Scheduled HARQ for Time- and Loss-Rate-Sensitive Networks. In: 2023 IEEE 20th Annual Consumer Communications & Networking Conference (CCNC). IEEE
25. Saltzer JH, Reed DP, Clark DD (1984) End-to-end arguments in system design. ACM Transactions on Computer Systems (TOCS) 2(4):277–288
26. Chen K, Wang H, Fang S, Li X, Ye M, Chao HJ (2022) RL-AFEC: adaptive forward error correction for real-time video communication based on reinforcement learning. In: Proceedings of the 13th ACM Multimedia Systems Conference, pp. 96–108
27. Cheng S, Hu H, Zhang X, Guo Z (2020) DeepRS: Deep-learning Based Network-Adaptive FEC for Real-Time Video Communi-

cations. In: 2020 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5 . IEEE
28. Palmer M, Krüger T, Chandrasekaran B, Feldmann A (2018) The QUIC Fix for Optimal Video Streaming. In: Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC, pp. 43–49
29. Pereira PG, Herfet T (2022) Polar Coding for Efficient Transport Layer Multicast. In: 2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC), pp. 313–318. IEEE
30. Ferlin S, Kucera S, Claussen H, Alay Ö (2018) MPTCP Meets FEC: Supporting latency-sensitive applications over heterogeneous networks. IEEE/ACM Trans Networking 26(5):2005–2018
31. Rizzo L (1997) Effective erasure codes for reliable computer communication protocols. ACM SIGCOMM computer communication review 27(2):24–36
32. Karzand M, Leith DJ, Cloud J, Medard M (2017) Design of FEC for low delay in 5G. IEEE J Sel Areas Commun 35(8):1783–1793
33. Roca V, Teibi B, Burdinat C, Tran T, Thienot C (2017) Less Latency and Better Protection with AL-FEC Sliding Window Codes: A Robust Multimedia CBR Broadcast Case Study. In: 2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), pp. 1–8 . IEEE
34. Shojania H, Li B (2009) Random network coding on the iPhone: fact or fiction? In: Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video, pp. 37–42
35. Wunderlich S, Cabrera JA, Fitzek FH, Reisslein M (2017) Network coding in heterogeneous multicore IoT nodes with DAG scheduling of parallel matrix block operations. IEEE Internet Things J 4(4):917–933
36. Polyanskiy Y, Poor HV, Verdú S (2010) Channel coding rate in the finite blocklength regime. IEEE Trans Inf Theory 56(5):2307–2359
37. Eliyahu T, Kazak Y, Katz G, Schapira M (2021) Verifying learning-augmented systems. In: Proceedings of the 2021 ACM SIGCOMM 2021 Conference, pp. 305–318
38. Mao H, Netravali R, Alizadeh M (2017) Neural adaptive video streaming with Pensieve. In: Proceedings of the Conference of the ACM Special Interest Group on Data Communication, pp. 197–210
39. Huang T, Zhou C, Zhang R-X, Wu C, Yao X, Sun L (2019) Comyco: Quality-aware adaptive video streaming via imitation learning. In: Proceedings of the 27th ACM International Conference on Multimedia, pp. 429–437
40. Xiao S, He D, Gong Z (2018) Deep-Q: Traffic-driven QoS inference using deep generative network. In: Proceedings of the 2018 Workshop on Network Meets AI & ML, pp. 67–73
41. Zappone A, Di Renzo M, Debbah M, Lam TT, Qian X (2019) Model-aided wireless artificial intelligence: Embedding expert knowledge in deep neural networks for wireless system optimization. IEEE Veh Technol Mag 14(3):60–69
42. Braden R (1989) Requirements for internet hosts - communication layers. STD 3, RFC Editor October
43. Iyengar J, Thomson M (2021) Quic: A udp-based multiplexed and secure transport. RFC 9000, RFC Editor May 2021
44. Ott J, Wenger S, Sato N, Burmeister C, Rey J (2006) Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF). RFC 4585, RFC Editor July
45. Adamson B, Bormann C, Handley M, Macker J (2004) Negative-acknowledgment (NACK)-Oriented Reliable Multicast (NORM) Protocol. RFC 3940, RFC Editor November
46. Oh B, Han J, Lee J (2010) Timer and sequence based packet loss detection scheme for efficient selective retransmission in DCCP. In: International Conference on Future Generation Communication and Networking, pp. 112–120 . Springer
47. Cheng Y, Cardwell N, Dukkipati N, Jha P (2021) The RACK-TLP Loss Detection Algorithm for TCP. RFC 8985, RFC Editor February
48. Iyengar J, Swett I (2021) QUIC Loss Detection and Congestion Control. RFC 9002, RFC Editor May
49. Shojania H, Li B (2010) Tenor: making coding practical from servers to smartphones. In: Proceedings of the 18th ACM International Conference on Multimedia, pp. 45–54
50. MacWilliams FJ, Sloane NJA (1977) The Theory of Error Correcting Codes vol. 16. Elsevier, Radarweg 29, 1043 NX Amsterdam, The Netherlands
51. Gallager R (1962) Low-density parity-check codes. IRE Transactions on information theory 8(1):21–28
52. Luby M (2002) LT codes. In: The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings., pp. 271–271 . IEEE Computer Society
53. Arikan E (2009) Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. IEEE Trans Inf Theory 55(7):3051–3073
54. Schmidt A, Reif S, Pereira PG, Honig T, Herfet T, Schroder-Preikschat W (2019) Cross-layer pacing for predictably low latency. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 1–6. IEEE
55. Pereira PG, Schmidt A, Herfet T (2018) Cross-layer effects on training neural algorithms for video streaming. In: Proceedings of the 28th ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video, pp. 43–48
56. Cheng Y, Cardwell N, Yeganeh SH, Jacobson V (2022) Delivery Rate Estimation. Internet draft, RFC Editor March
57. Raca D, Quinlan JJ, Zahran AH, Sreenan CJ (2018) Beyond throughput: A 4G LTE dataset with channel and context metrics. In: Proceedings of the 9th ACM Multimedia Systems Conference, pp. 460–465
58. Narayanan, A, Ramadan E, Carpenter J, Liu Q, Liu Y, Qian F, Zhang Z-L (2010) A first look at commercial 5G performance on smartphones. In: Proceedings of The Web Conference 2020, pp. 894–905
59. Xu D, Zhou A, Zhang X, Wang G, Liu X, An C, Shi Y, Liu L, Ma H (2020) Understanding operational 5G: A first measurement study on its coverage, performance and energy consumption. In: Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication, pp. 479–494
60. Sheshadri RK, Koutsonikolas D (2017) On Packet Loss Rates in Modern 802.11 Networks. In: IEEE INFOCOM 2017-IEEE Conference on Computer Communications, pp. 1–9. IEEE
61. Bhartia A, Chen B, Wang F, Pallas D, Musaloiu-E R, Lai TT-T, Ma H (2017) Measurement-based, Practical Techniques to Improve 802.11ac Performance. In: Proceedings of the 2017 Internet Measurement Conference, pp. 205–219
62. ITU-T Y.1541: Network performance objectives for IP-based services. Technical report, ITU December 2011
63. Mangiante S, Klas G. Navon A, GuanHua Z, Ran J, Silva MD (2017) VR is on the Edge: How to Deliver 360 Videos in Mobile Networks. In: Proceedings of the Workshop on Virtual Reality and Augmented Reality Network, pp. 30–35
64. Smith LN, Topin N (2019) Super-convergence: Very fast training of neural networks using large learning rates. In: Artificial Intelligence and Machine Learning for Multi-domain Operations Applications, vol. 11006, pp. 369–386. SPIE

65. Shelby Z, Hartke K, Bormann C (2014) The Constrained Application Protocol (CoAP). RFC 7252, RFC Editor June
66. Anand V, Xie Z, Stolet M, De Viti R, Davidson T, Karimipour R, Alzayat S, Mace J (2022) The Odd One Out: Energy is not like Other Metrics. In: HotCarbon 2022: 1st Workshop on Sustainable Computer Systems Design and Implementation
67. Siegmund N, Dorn J, Weber M, Kaltenecker C, Apel S (2022) Green configuration: Can artificial intelligence help reduce energy consumption of configurable software systems? Computer 55(3):74–81

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.