Vincent Latzko, Christian Vielhaus, Frank H. P. Fitzek

Usecase Driven Evolution of Network Coding Parameters Enabling Tactile Internet Applications

Erstveröffentlichung in / First published in:

IEEE International Conference on Communications, Dublin 07. – 11.06.2020. IEEE Explore. ISBN 978-1-7281-5089-5

DOI: https://doi.org/10.1109/ICC40277.2020.9149269

Usecase Driven Evolution of Network Coding Parameters Enabling Tactile Internet Applications

Vincent Latzko*, Christian Vielhaus*, Frank H. P. Fitzek*†
*Deutsche Telekom Chair, Technische Universität Dresden, 01062 Dresden, Germany
†Centre for Tactile Internet with Human-in-the-Loop (CeTI), Technische Universität Dresden, 01062 Dresden, Germany

***Abstract*—Present-day and future network protocols that include and implement Forward Error Correction are configurable by internal parameters, typically incorporating expert knowledge to set up. We introduce a framework to systematically, objectively and efficiently determine parameters for Random Linear Network Codes (RLNC). Our approach uses an unbiased, consistent simulator in an optimization loop and utilizes a customizable, powerful and extendable parametric loss function. This allows to tailor existing protocols to various use cases, including ultra reliable, low latency communication (URLLC) codes. Successful configurations exploring the search space are under evolutionary pressure and written into a database for instant retrieval. We demonstrate three examples, Full Vector Coding, tail RLNC, and PACE with different focus for each.**

I. Introduction

In all networking environments, channel losses are a crucial element of error. Typically, these losses are addressed by forward error correction (FEC), using various coding schemes (cf. Figure 1). Among these, Random Linear Network Codes (RLNC) have emerged as a powerful method to tackle these losses, even in the absence of knowledge of the state of the underlying media. In fact, RLNC has been shown to be an optimal erasure code. Selecting optimal parameters for the schemes however, is an unresolved and ongoing research topic. The field of channel estimation has made significant progress in the past, but still the characteristics of individual operation remain a challenge. Adding to the task, the parameter space of RLNC is quite vast, and other codes have similar complexity in choice. Optimal parameters are not a unique configuration either, as different use cases require different prioritization of the performance indicators. For example, Tactile Codes as outlined in [1] or interactive high-definition content (e.g. Google Stadia) require minimal latency and maximum robustness above all else, whereas video streaming can often deal with latency and jitter without degradation of user experience, as long as (short term mean) bandwidth requirements are met. Similar arguments can be made for other use cases. Optimizing for latency is an important, challenging and pressing task, since bandwidth can often be addressed by additional channels and is primarily a commercial problem in these cases, whereas latency tends to be fixed for a deployed network. To balance coding, the amount and placement of redundancy as well as computational complexity is the task of the coding scheme.

Currently, determining parameters of any given coding scheme, such as RLNC Full Vector Coding (FVC), tail RLNC, or PACE (cf. Section III) is based on heuristics and human expertise. Exhaustively searching the space of parameters to find an optimal choice is computationally unattractive, especially if a new protocol or scheme is designed and subsequently introduced. The resulting equally extensive database would also be cumbersome to search in real time, when a satisfying combination of parameters needs to be found quickly as the channel changes. Instead, we build the foundation of a novel system to adapt coding parameters to the use case, conditioned on the channel. In this publication, we present a framework as the backbone that allows to place different emphasis on key parameters using weights (e.g. allowing to focus on latency before other performance parameters). It then iteratively evaluates a simulator with diverse configurations that are enforced to satisfy the requirements (cf. Figure 4).

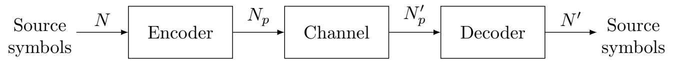

Figure 1. Considered problem with encoder, Binary Erasure Channel (BEC) and decoder. Redundancy gives $N_p > N$ and packet drops $N'_p \leq N_p$. In the case of possible decoding, $N' \equiv N$, as subsequent packets are unnecessary

Subsequently, this article will look at related work in Section II and briefly introduce the background of RLNC in Section III. Section IV looks at our methodology in-depth, Section V shows our results. In Section VI we conclude the paper with a summary and outline future research.

II. Previous Work

In [2], the authors look at how to select optimal density for coding in wireless sensor networks, where energy consumption per transmitted bit must be minimized. This optimization problem is analyzed comprehensively with respect to invested energy, including the power draw for memory access, XOR'ing the packets' content and physical network interface of a specific chip, but does not include considerations beyond energy efficiency. With a different focus, more parameters are under observation in [3] for an examination of coding overhead. Their focus is on minimizing the coding impact on the packet size, looking at generation size, field size and density (a measure of how many elements of the coding vector are non-zero). The result indicates that for minimal overhead and optimal recoding ability, the binary field should almost always be chosen, provided that generation size is large enough ($G \gtrsim 64$).

III. RLNC Background

RLNC is based on Galois fields of various size q, where q typically is a power of a prime $p, \left(q = p^k\right)$. In the digital world, $q = 2^k$ is convenient and consequently the fields are $GF\left(q\right)$ or $\mathbb{F}_q$ in literature to denote its origin as a finite field. As such, source packets or *symbols* are formed from the original data at the sender's encoder to be transmitted over a lossy channel with drop probability ε, i.e. a channel that may or may not drop any of the symbols during transit (cf. Figure 1). The receiver will try and decode the symbols into the original data, reversing the encoding process even in the presence of losses. We consider end to end losses $e = 1 - \frac{N'}{N}$. Ergo, the code needs to add some redundancy to counter the channel losses.

As noted in e.g. [4], [5] or [6], *linear* network coding suffices to achieve maximum flow, which corresponds to operations that are linear combinations in $\mathbb{F}_q$. In addition, the coefficients of those operations may be chosen randomly to allow for decoding with probability if the cardinality of the field is large enough to almost always produce linearly independent combinations.

The source data is separated by the encoder into symbols, each of size ψ, and G consecutive symbols form a generation. This results in a symbol matrix $\mathbf{S}$ consisting of G rows and ψ columns for each generation (see Table I for abbreviations). The coding ratio $C = \frac{N_p}{G} \geq 1$ refers to the ratio of the encoder's output and input, (refer to Figure 1). The total number of redundancy packets per generation is $r = \lceil (C-1)\,G \rceil$, as the determination depends on the desired code rate. Consequently, the absolute number of packets for a given generation is then $N_p = G + r$. The actual encoding is performed by multiplication of a linear local encoding function matrix $\mathbf{L}$ in $\mathbb{F}_q$. The packets are given by the rows of $\mathbf{T} = \mathbf{LS}$, where the coding vectors c_i are drawn from $\mathbb{F}_q$ arranged as rows in the $N_p \times G$ matrix $\mathbf{L}$. $\mathbf{T}$ is simply the $N_p \times \psi$ matrix of transmission symbols put on the channel.

For decoding after transmission, the receiver gathers at least G packets and forms the matrix $\mathbf{R}$ along with the coefficient matrix $\mathbf{C}_r$. If at least G linearily independent symbols are received, $\mathbf{C}_r$ has full rank, inversion is possible and the packets are recovered by $\mathbf{S} = \mathbf{C}_r^{-1}\mathbf{R}$.

Table I
Summary of notations

Notation	Definition	Type
G	Generation Size	Integer
C	Coding Ratio	Real
ε	Channel loss probability	Real
r	# of Redundancy packets	Integer
N_p	# of packets for Generation, incl. r	Integer
ψ	Symbol size	Integer

A. Full vector coding

In the case of full vector coding (also referred to as dense network coding), every single symbol on the channel is coded, that is, all G symbols are combined to yield N_p coded symbols. Implicitly, all packets are of the same value. This gives rise to an all-or-nothing-principle: either the full generation can be decoded once the final symbol arrives at the receiver's end, or the full generation has to be dismissed. This can be decided instantly by inspecting the rank of the decoding matrix or simply counting the arriving packets, provided the field size is high enough. Coding delay is present on the encoder since all packets of a generation need to be present for combination. Note also that inverting the matrix $\mathbf{C}_r$ becomes computationally costly for high generation sizes, as inverison complexity grows with $\mathcal{O}\left(G^3\right)$. FVC's symbol loss probability is given by

$$e_{\text{FVC}}\left(G, r, \varepsilon\right) = 1 - \sum_{i=G}^{G+r} \left(\begin{array}{c} G+r \\ i \end{array}\right) \varepsilon^{G+r-i} \left(1-\varepsilon\right)^i .$$

The problem of delay is addressed in [7] by trying to iteratively form block matrices of the leading stump of $\mathbf{C}_r$ as the packets are received, decoding along a sliding window. This results in lower total inversion time.

B. Tail RLNC

The main innovation of [8] uses the encoder in a bypass manner: most packets are delivered uncoded (systematically) and only after the generation has finished, an additional r symbols are coded by the encoder, spanning all the packets of the generation (see Figure 2, top). This results in a symbol loss probability of

$$e_{\text{tailRLNC}}\left(G, r, \varepsilon\right) = \varepsilon + \varepsilon\left(e_{\text{FVC}}\left(G, r-1, 1-\varepsilon\right)\right).$$

The scheme allows for zero decoding effort in a loss-free case. Additionally, there is no encoding delay, since the systematic packets are sent out immediately and do not need to be held back to be combined with additional packets. Formally, for a generation of G packets with r redundancy, the transmission matrix $\mathbf{T} = \left[\mathbf{I}\,|\mathbf{C}\right]^T$ is an $G \times G$ identity matrix extended by a $r \times G$ matrix of coding coefficients sampled from $\mathbb{F}_q$. If $s < r$ packets are dropped by the channel, recovery is possible after s of the coded packets are received. However, upon any loss, the decoder enters the blocking state and cannot forward data to higher OSI layers until reconstruction is possible. This is motivated by compatibility with layers that expect in-order delivery. The scheme performs better with small generation sizes, when redundancy is not many slots removed from a possible lost symbol.

C. PACE

While tail RLNC improves over FVC in terms of latency and computational load (cf. [8]), there is a fair amount of optimization potential. [9] introduce PACE-uniform with a placement of redundancy in a similar manner to tail RLNC in the way that coded packets are placed between systematic ones, but with the difference that a coded symbol does not terminate a generation. Instead, after every $\sigma_s \ll G < N_p$ source symbols, exactly $k = 1$ coded packets follow, which contain random linear combinations of *all* the previous symbols, not just the previous σ_s (cf. Figure 2, bottom). This

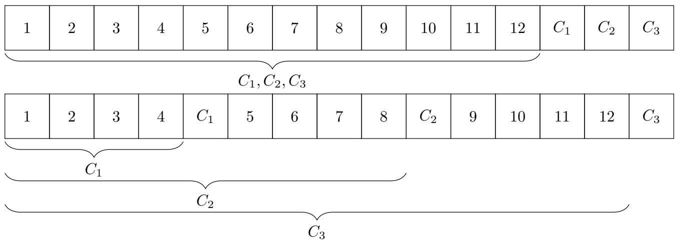
1 2 3 4 5 6 7 8 9 10 11 12 C_1 C_2 C_3
C_1, C_2, C_3
1 2 3 4 C_1 5 6 7 8 C_2 9 10 11 12 C_3
C_1
C_2
C_3

Figure 2. Top: tail RLNC uses coded packets to protect the generation, shown with $G = 12, r = 3$. This results in $C = \frac{15}{12} = 1.25$. Bottom: PACE with its different distribution scheme, $G = 12, C = 1.25$.

forms a sub-generation and allows for quick recovery if at most one of these packets is dropped. Like in tail RLNC, if any symbols necessary for operation are lost, the decoder enters the blocking state. However, once recovery is successful with subsequent redundancy of later sub-generations, normal operation continues. Should N_p not be an integer multiple of σ_s, the remainder is distributed among the first $\left\lfloor \frac{N_p}{\sigma_s} \right\rfloor$ generations. Note that this increasing coverage of the redundancy symbols with growing number of subgenerations introduces a peculiarity: early packets of a generation are protected by more redundancy compared with later ones, and later coded packets have higher value due to their construction. To address the asymmetry of packets' protection, the authors allow for additional tail redundancy r_t, which like the tail RLNC case protects the generation as a whole. In essence, the distribution of the redundancy is optimized for sporadic packet drops. The extension PACE-burst allows for $k \neq 1$, but has not been investigated here.

In contrast to FVC and tail RLNC, no closed form expression is known for PACE's symbol loss probability. A recent approach based on one finite state machine each for systematic and coded packets allows for a recursive estimation of decoding probabilities[1].

IV. Methodology

Applicability of code in network applications is driven by a few key performance indicators. Good code configurations maximize resilience and throughput, while they minimize both coding latency and the impact of coding complexity on devices. For example, mindlessly adding redundancy helps resilience, but does so at the cost of goodput and latency. With the previous chapter, one would expect tail RLNC ahead of FVC, especially in the case of non-negligible losses, provided the redundancy is dimensioned adequately. We subsequently optimize the configuration of a chosen coding scheme for given and fixed transmission channel parameters.

The NCKernel library features RLNC functionality using the KODO library [10] and implements various protocols. It exposes an API and can be used as a consistent, unbiased simulator (cf. Figure 3 for the noise power $s_e \propto 1/\sqrt{N}$). It can be configured to include random packet drops and

[1]Private communication with authors

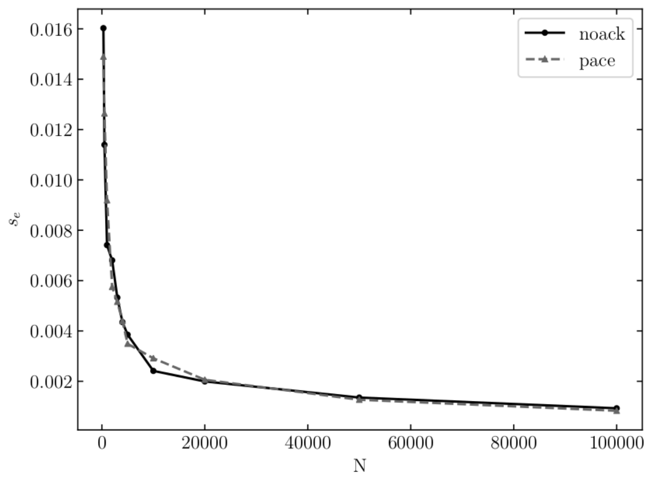

Figure 3. The consistency of NCKernel as a simulator shown as diminishing noise power s_e versus the number of simulated packets N

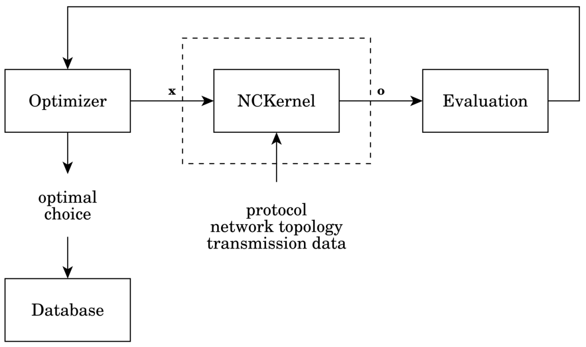

Figure 4. Optimization framework

the simulator will accurately deliver non-innovative (linearly dependent) RLNC packets. Essentially, it acts as a stochastic function mapping from the discrete configuration space $\mathcal{P}$ to the output space O, which consists of the three-tuple code rate $(\eta \in [0, 1])$, symbol loss $(e \in [0, 1])$, and average symbol latency $(\ell \in [\ell_c, \infty))$, where ℓ_c is the channel latency and infinite delay is used for dropped packets. Code rate is a measure how efficient the code is, $\eta = \frac{N}{N_p}$. We also limit analysis to data filling a single generation, i.e. $N = G$.

Due to the large amount of both channel conditions and parameter configurations for any of the protocols introduced in Section III, an exhaustive table is neither tractable nor desired. Instead, diverse samples of high quality are suitable to cover a range of requirements. The simulator's input configuration is determined and evaluated by an optimization framework based on an evolutionary strategy (ES). Some parameters are integers and the results of sampling in $\mathcal{P}$ need to be mapped to viable configurations before a simulation can start. ES strategies are typically sampling-heavy, however our implementation pools computational resources if multiple individuals map to the same RLNC-configuration. This is especially helpful a few (≈ 5) iterations after simulation start, where this grouping further enhances accuracy of the simulator run. Additionally, the amount of packets sent through NCKernel is increased automatically in later, user definable stages for higher reliability. The sampler also filters candidate configurations that are prohibited by the designer.

A. Loss & Performance Function

Because different use cases lead to different emphasis that a flexible code needs to address, a customizable function is introduced that serves as a loss, cost or fitness, depending on preferred terminology. This function maps from a point in the parameter space to a finite value that acts as a measure for the optimization loop: $f : \mathbb{R}^n \to \mathbb{R},$ where n is the number of free parameters in the code.

This is a composite function that first calls the simulator with a configuration $\mathbf{x},$ which results in the simulator output $\mathbf{o} = (\eta, e, \ell)^T$ then uses an inner performance function $\mathbf{p}(\mathbf{o}) = \left((1-\eta), e, \left(1 - \frac{\ell_c}{\ell}\right)\right)^T$ together with user-defined weights to combine the result into a scalar value,

$$\hat{f}(\mathbf{w}, \mathbf{o}) = \mathbf{w} \cdot \mathbf{p}(\mathbf{o}) = w_1 (1-\eta) + w_2 e + w_3 \left(1 - \frac{\ell_c}{\ell}\right),$$

which will be minimized. The weights sum to one, so f is bounded to $[0, 1)$. Furthermore, we use an additive pentalty constraint to steer the optimizer away from undesired areas in $O,$ i.e. we supply some *maximum acceptable result* $\hat{\mathbf{o}}$ to form

$$\beta(\mathbf{o}) = \frac{1}{n} \sum_{i=1}^{n} \max(0, p_i(\mathbf{o}) - p_i(\hat{\mathbf{o}})).$$

The final $f = \hat{f}(\mathbf{w}, \mathbf{o}) + \mu_e \beta(\mathbf{o})$ (with some μ_e) implicitly depends on $\mathbf{x}$ through the random, but consistent mapping by the simulator, thus we will write $f(\mathbf{x})$ subsequently.

In total, this loss function allows users to prohibit configurations that are known a-priori to be unsuitable for the use case and also balances the three key norms of performance limitations. It is heavily used in our optimization scheme and can also be easily extended by simply adding additional elements in a key-value fashion, providing a scalar weight and performance measure, respectively.

B. Optimization using CMA-ES

Here, we broadly paint an overview of how the Covariance Matrix Adaptation ES (CMA-ES) performs optimization. For an more in-depth look, we refer to [11]. CMA-ES is well suited for high dimensional problems, is gradient-free and robust against noisy objectives. Like any ES, optimization takes place by way of generations of individuals (in this case, individual parameter configurations) that are mutated and subsequently evaluated by the fitness function. In every generation $g,$ initial sampling is performed from a multivariate GAUSSian distribution, yielding configurations $x_i \sim \mathbf{m}_g + \sigma_g \mathcal{N}(\mathbf{0}, \mathbf{C}_g), i = 1, \ldots, \lambda$. The covariance matrix can be seen as an n-dimensional ellipsoid with the eigenvalues' square root $\sqrt{\nu_j}$ as the length of its principal axes, since covariance matrices are positive semi-definite and eigendecompositions are possible.

All λ individuals are then simulated by NCKernel, evaluated and the best μ individuals are selected to contribute to updating the mean vector for the next generation's sampling step, $\mathbf{m}_{g+1} = \sum_{i=1}^{\mu} w_i x_i,$ where w_i are descending weights that sum to one and x_i are the best-ranked candidates. We use the best practice $\mu = \lambda/2,$ ([12]). The covariance matrix is updated using the maximum likelihood approach, explaining the μ best encountered data points:

$$\mathbf{C}_{g+1}^{(\mu)} = \sum_{i=1}^{\mu} w_i (\mathbf{x}_i - \mathbf{m}_g)(\mathbf{x}_i - \mathbf{m}_g)^T.$$

A rank-$\mu-$ update is used to add a momentum term,

$$\mathbf{C}_{g+1}^{(r\mu)} = (1 - c_\mu) \mathbf{C}_g + c_\mu \mathbf{C}_{g+1}^{(\mu)},$$

which acts as a moving average with exponential decay over past generations. Additionally, an evolution path is constructed that influences step size σ_g and guides evolution towards successful individuals, balancing the explore-exploit dilemma by effectively de-correlating search steps.

We furthermore added local restart strategies to avoid premature convergence and higher coverage of the search space. In this case, CMA-ES is restarted with all previous individuals as seeds and a higher total population count.

C. Database for fast retrieval

We also store resulting parameters after successfully completed runs in a small database, to relieve end devices from continuous evolutionary iteration. This is done with diversity of solutions in mind to maximize applicability in different scenarios. At run time, a simple look up of the previously optimized parameters is done and in case of missing or mismatched values, interpolation is possible. The pseudo-code 1 summarizes the process for completeness and reference. The actual implementation is available.

Algorithm 1 Pseudo-code for optimization loop

```
Initialization mean vector, covariance matrix
while termination criterion not reached:
    sample individuals and group configurations
    run simulations
    evaluate fitness, penalize violations
    sort population vector
    adapt mean vector, covariance matrix, stepsize
finally:
    store and return mean vector, covariance matrix
```

V. Experiments & Results

We show the output of optimization for three exemplary use cases, one for each of the three protocols outlined above. In all cases, the channel properties are set to $\varepsilon = 0.1, l_c = 5$ms, and 10^5 packets are simulated for each of the $\lambda = 30$ starting individuals. In all experiments, we keep field size fixed at $\mathbb{F}_{2^8},$ as this is typically chosen based on hardware restrictions. It can however, easily be included as a free parameter in CMA-ES. In all following cases, the performance becomes very close after ≈ 15 iterations of evolution.

1) Full vector coding: FVC is the most vanilla approach using evolutionary strategies to coding parameter optimization, whereas both tail RLNC and PACE display more capabilities of our framework. We analyzed FVC in the space of

$$\mathcal{P} = \left\{ \mathbf{x} = (G, r)^T : 1 \leq G \leq 80, 1 \leq r \leq 10 \right\},$$

with an objective weighting of $\mathbf{w} = (0.2, 0.7, 0.1)^T$, focussing on minimal symbol error, and thus a resilient code. CMA-ES converges after 19 iterations to the correct analytical optimum $\mathbf{x}^* = (10, 4)^T$, cf. Figure 5, which results in a code ratio of $C = 1.4$. Each dot corresponds to one individual configuration. The covariance matrix is displayed via an ellipse with its principal axes according to the eigendecomposition. The main focus driving the optimization is symbol loss and consequently, the symbol error is $e \approx 0.92\%$. At this point, additional redundancy only deteriorates coding efficiency η.

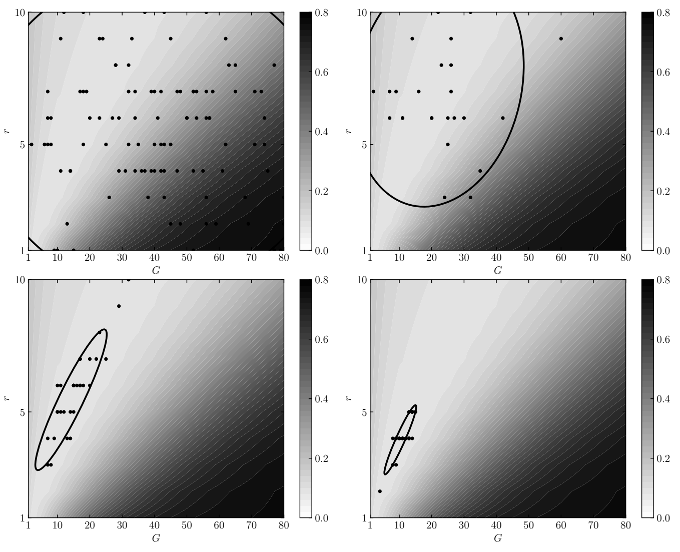
Figure 5. Four examples of CMA-ES used on Full Vector Coding (displaying evolutionary generations $1, 5, 10$ and 15, top left to bottom right): CMA-ES converges to the global optimum $x^* = (10, 4)^T$. Lighter color indicates better performance in the fitness landscape. Best viewed on screen.

2) Tail RLNC optimized for goodput: In our second experiment we analyzed tail RLNC optimized for goodput with a constraint on the symbol loss, $e \leq 0.02$, neglecting latency completely. The fitness can in that case be expressed as $\hat{f}(\mathbf{o}) = 1 - (1 - e)\eta$. No weights are employed in this case. The parameter space is extended to

$$\mathcal{P} = \left\{ \mathbf{x} = (G, r)^T : 1 \leq G \leq 80, 1 \leq r \leq 20 \right\},$$

both allowing and expecting larger generation sizes and more redundancy symbols to satisfy the requirements of a bandwidth-oriented use case. Figure 6 shows the convergence of CMA-ES, again recovering the analytic optimum, while in Table II numerical results are displayed. We give the two performance indicators η and e, the objective $\hat{f}(\mathbf{o})$ and the total fitness $f(\mathbf{x})$. Note how penalties $\mu_e \beta(\mathbf{o})$ (with $\mu_e = 4$) are only present for violations of symbol loss rate, strongly driving loss as desired, and that the configurations with best objective values $\hat{f}(\mathbf{o})$ at the bottom are not chosen due to these violations.

Table II
NUMERICAL RESULTS OF THE TAIL RLNC EXPERIMENT WITH CONSTRAINTS

$\boldsymbol{x} = (G, r)$	$\eta\,[\%]$	$e\,[\%]$	$\hat{f}(\boldsymbol{o})$	$\mu_e \beta(\boldsymbol{o})$	$f(\boldsymbol{x})$
$(80, 12)$	86.9565	1.9659	0.1475	0	0.1475
$(79, 12)$	86.8132	1.8649	0.1481	0	0.1481
$(78, 12)$	86.6667	1.7664	0.1486	0	0.1486
$(80, 13)$	86.0215	1.2766	0.1508	0	0.1508
$(79, 13)$	85.8696	1.2000	0.1516	0	0.1516
$(79, 11)$	87.7778	2.7514	0.1464	0.0301	0.1765
$(80, 11)$	87.9121	2.8749	0.1462	0.0350	0.1812

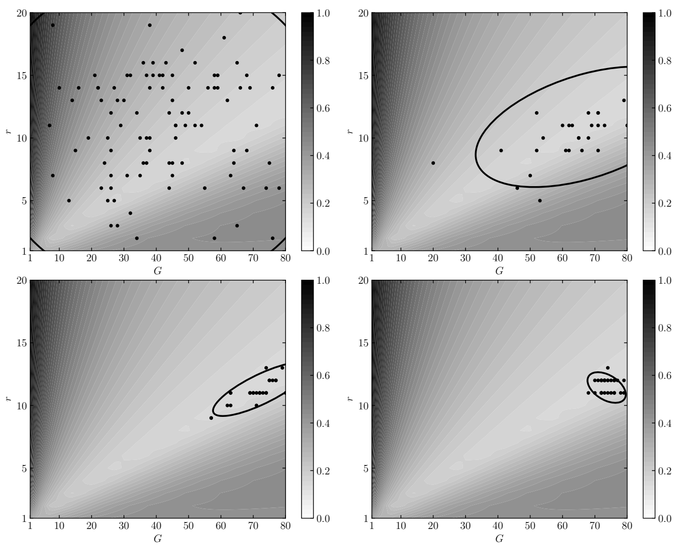
Figure 6. CMA-ES used on tail RLNC and converging to the global optimum $x^* = (80, 12)^T$, again with generations $1, 5, 10$ and 15.

3) PACE with efficiency constraint: We further analyzed PACE with an output constraint of $\hat{\mathbf{o}} = (0.8, 1, \infty)$, which limits the allowed coding ratio to $C \leq \frac{1}{0.8} = 1.25$. This corresponds to an enforced channel efficiency via the penalties, so weights $\mathbf{w} = (0, 1, 0)$ are a sufficient choice. Thus,

$$\mathcal{P} = \left\{ \mathbf{x} = (G, C)^T : 1 \leq G \leq 40, 1 \leq C \leq 1.25 \right\}.$$

Note that the rounding operation in PACE's definition of the number of packets introduces uncertainty and may penalize some configurations more than their pure performance values would indicate. The results are given in Figure 7 and Table III. Penalties $\mu_e \beta(\mathbf{o})$ occur for violations of efficiency in this case, even though the two last configurations achieve best symbol loss. The rugged, piece wise appearance of the fitness landscape contains many plateaus which may pose problems for gradient-based optimization and also many local optima. As expected, due to its design, CMA-ES copes with these problems and converges to the optimal value of $(40, 1.23)^T$. This configuration is equivalent to $(40, 1.24)^T$

Table III
NUMERICAL RESULTS OF THE PACE EXPERIMENT WITH CONSTRAINTS

$\mathbf{x} = (G, C)$	$\eta\,[\%]$	$e\,[\%]$	$w_2 p_2(e)$	$\mu_e \beta(\boldsymbol{o})$	$f(\boldsymbol{x})$
$(40, 1.23)$	80.0000	1.1255	0.0113	0	0.0113
$(38, 1.22)$	80.8511	1.2129	0.0121	0	0.0121
$(40, 1.21)$	81.6327	1.2743	0.0127	0	0.0127
$(32, 1.22)$	80.0000	1.3450	0.0134	0	0.0134
$(39, 1.25)$	79.5918	0.9451	0.0095	0.0163	0.0258
$(38, 1.24)$	79.1667	0.8159	0.0082	0.0332	0.0414

and $(40, 1.25)^T$ due to the rounding in PACE, resulting in the same number of redundancy $r = 10$ and has been found with the recursive approach for PACE's symbol loss probability.

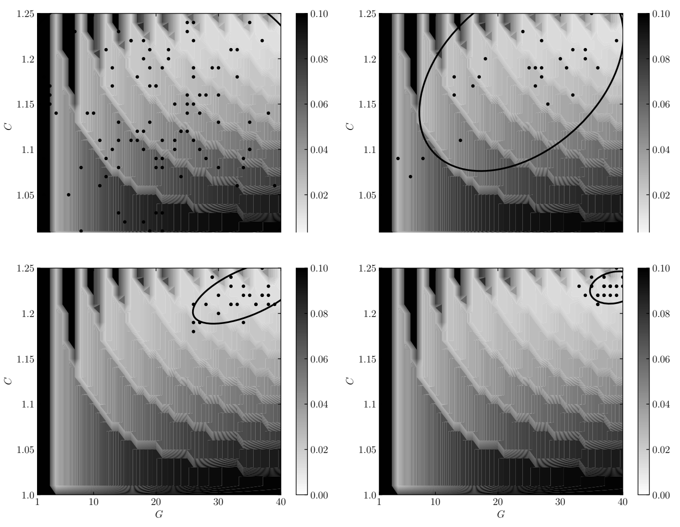

Figure 7. CMA-ES on constrained PACE converging on the correct global optimum $x^* = (40, 1.25)^T$.

VI. Conclusion and Future Works

This article puts forward a framework for an objective, explainable and optimal selection of random linear network coding parameters. We take into consideration user-definable, flexible goals for the code configuration using a novel weighting factor for our newly introduced loss function, allowing to put special emphasis on reliability, latency or bandwidth (mapped from symbol loss, symbol delay and code rate). Special constraints allow for minimal values to be specified that need to be achieved by the code. Our approach successfully recovers known optima for two of the cases demonstrated where analytical results exist, and does so without prior knowledge. In the case of PACE, our method quickly converges to meaningful configurations that coincide with state of the art recursive approaches. This demonstrates the ability of our framework to both verify existing coding configurations as well as to find new optimal parameters even for future use cases and protocols. The extensibility of the framework allows for usage of arbitrary codes and the inclusion of more parameters (e.g. field size) that we did not investigate. Furthermore, the modular nature allows for drop-in replacements for any of our used tools. For easier deployment, also on computationally weak devices, the integration of the database allows to benefit from systematic application of this research.

However, we limited our research to BECs, and consequently, research into non-binary erasure channels is promising. These channels are more realistic in the way that losses are often correlated in reality, resulting in a bursty nature. For speedups, different optimizers like BFGS may decrease runtime considerably.

Acknowledgment

This research has been supported in part by European Union H2020 - ECSEL - AutoDrive (Grant Agreement number: 737469), by Deutsche Telekom and by the German Research Foundation (DFG, Deutsche Forschungsgemeinschaft) as part of Germany's Excellence Strategy - EXC 2050/1 - Project ID 390696704 - Cluster of Excellence "Centre for Tactile Internet with Human-in-the-Loop" (CeTI) of Technische Universität Dresden. Further funding granted by BMBF and the Free State of Saxony (grant number 16ESE0258S).

References

[1] M. Simsek, G. P. Fettweis, and C.-L. I, "Tactile internet," 2019.

[2] J. Heide, Q. Zhang, and F. H. P. Fitzek, "Selecting Optimal Parameters of Random Linear Network Coding for Wireless Sensor Networks," in *2013 IEEE 78th Vehicular Technology Conference (VTC Fall)*. Las Vegas, NV, USA: IEEE, Sep. 2013, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6692403

[3] J. Heide, M. V. Pedersen, F. H. P. Fitzek, and M. Medard, "On Code Parameters and Coding Vector Representation for Practical RLNC," in *2011 IEEE International Conference on Communications (ICC)*. Kyoto, Japan: IEEE, Jun. 2011, pp. 1–5. [Online]. Available: http://ieeexplore.ieee.org/document/5963013/

[4] A. Montanari and R. Urbanke, "Coding for Network Coding," *arXiv:0711.3935 [cs, math]*, Nov. 2007, arXiv: 0711.3935. [Online]. Available: http://arxiv.org/abs/0711.3935

[5] C. Fragouli, J.-Y. Le Boudec, and J. Widmer, "Network coding: an instant primer," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, p. 63, Jan. 2006. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1111322.1111337

[6] T. Ho, M. Medard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong, "A Random Linear Network Coding Approach to Multicast," *IEEE Transactions on Information Theory*, vol. 52, no. 10, pp. 4413–4430, Oct. 2006. [Online]. Available: http://ieeexplore.ieee.org/document/1705002/

[7] S. Wunderlich, F. Gabriel, S. Pandi, and F. H. Fitzek, "We don't need no generation - a practical approach to sliding window RLNC," in *2017 Wireless Days*. Porto, Portugal: IEEE, Mar. 2017, pp. 218–223. [Online]. Available: http://ieeexplore.ieee.org/document/7918148/

[8] R. Prior and A. Rodrigues, "Systematic network coding for packet loss concealment in broadcast distribution," in *The International Conference on Information Networking 2011 (ICOIN2011)*. Kuala Lumpur, Malaysia: IEEE, Jan. 2011, pp. 245–250. [Online]. Available: http://ieeexplore.ieee.org/document/5723187/

[9] S. Pandi, F. Gabriel, J. A. Cabrera, S. Wunderlich, M. Reisslein, and F. H. P. Fitzek, "PACE: Redundancy Engineering in RLNC for Low-Latency Communication," *IEEE Access*, vol. 5, pp. 20 477–20 493, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/8003268/

[10] M. Pedersen, J. Heide, and F. Fitzek, "Kodo: An open and research oriented network coding library," vol. 6827. Springer Publishing Company, 5 2011, pp. 145–152, in Proceedings of the 2011 Networking Workshops. (eds.) Casares-Giner, Vicente, Manzoni, Pietro & Pont, Ana.

[11] T. Bäck, C. Foussette, and P. Krause, *Contemporary Evolution Strategies*. Springer Publishing Company, Incorporated, 2013.

[12] N. Hansen, "The CMA Evolution Strategy: A Tutorial," *arXiv:1604.00772 [cs, stat]*, Apr. 2016, arXiv: 1604.00772. [Online]. Available: http://arxiv.org/abs/1604.00772