entropy

MDPI

Article

BAR: Blockwise Adaptive Recoding for Batched Network Coding †

Hoover H. F. Yin [1,2,*], Shenghao Yang [3,*], Qiaoqiao Zhou [4], Lily M. L. Yung [5] and Ka Hei Ng [5]

[1] Department of Information Engineering, The Chinese University of Hong Kong, Shatin, New Territories, Hong Kong, China
[2] Department of Electronic and Computer Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, China
[3] School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen, Shenzhen 518172, China
[4] Department of Computer Science, School of Computing, National University of Singapore, Singapore 119391, Singapore; zhouqq@comp.nus.edu.sg
[5] Independent Researcher, Hong Kong, China; lily@link.cuhk.edu.hk (L.M.L.Y.); kaheicanaan@link.cuhk.edu.hk (K.H.N.)
* Correspondence: hfyin@ie.cuhk.edu.hk (H.H.F.Y.); shyang@cuhk.edu.cn (S.Y.)
† This paper is an extended version of our papers published in Yin, H.H.F.; Yang, S.; Zhou, Q.; Yung, L.M.L. Adaptive Recoding for BATS Codes. In Proceedings of the 2016 IEEE International Symposium on Information Theory (ISIT), Barcelona, Spain, 10–15 July 2016; pp. 2349–2353; and Yin, H.H.F.; Ng, K.H. Impact of Packet Loss Rate Estimation on Blockwise Adaptive Recoding for Batched Network Coding. In Proceedings of the 2021 IEEE International Symposium on Information Theory (ISIT), Melbourne, VIC, Australia, 12–20 July 2021; pp. 1415–1420.

Abstract: Multi-hop networks have become popular network topologies in various emerging Internet of Things (IoT) applications. Batched network coding (BNC) is a solution to reliable communications in such networks with packet loss. By grouping packets into small batches and restricting recoding to the packets belonging to the same batch; BNC has much smaller computational and storage requirements at intermediate nodes compared with direct application of random linear network coding. In this paper, we discuss a practical recoding scheme called blockwise adaptive recoding (BAR) which learns the latest channel knowledge from short observations so that BAR can adapt to fluctuations in channel conditions. Due to the low computational power of remote IoT devices, we focus on investigating practical concerns such as how to implement efficient BAR algorithms. We also design and investigate feedback schemes for BAR under imperfect feedback systems. Our numerical evaluations show that BAR has significant throughput gain for small batch sizes compared with existing baseline recoding schemes. More importantly, this gain is insensitive to inaccurate channel knowledge. This encouraging result suggests that BAR is suitable to be used in practice as the exact channel model and its parameters could be unknown and subject to changes from time to time.

Keywords: linear network coding; batched network coding; adaptive recoding

Citation: Yin, H.H.F.; Yang, S.; Zhou, Q.; Yung, L.M.L.; Ng, K.H. BAR: Blockwise Adaptive Recoding for Batched Network Coding. *Entropy* **2023**, *25*, 1054. https://doi.org/10.3390/e25071054

Academic Editor: Boris Ryabko

Received: 31 May 2023
Revised: 23 June 2023
Accepted: 28 June 2023
Published: 13 July 2023

1. Introduction

Noise, interference and congestion are common causes of packet loss in network communications. Usually, a packet has to travel through multiple hops before it can arrive at the destination node. Traditionally, the intermediate nodes apply the store-and-forward strategy. In order to maintain a reliable communication, retransmission is a common practice. A feedback mechanism is applied so that a network node can acknowledge that a packet is lost. However, due to the delay and bandwidth consumption of the feedback packets, retransmission schemes come with a cost of degraded system performance.

In the era of the Internet of things (IoT), there is a diversity of devices and network topologies. Embedded devices or microcomputers have been heavily deployed due to their mobility, lightweight design, and low power consumption. Multi-hop wireless networks

have become a common network topology, highlighting the issues in reliable transmission as wireless links are more vulnerable to packet loss. The packet loss at each link accumulates and the chance of successfully receiving a packet at the destination drops exponentially. Fountain codes, such as in [1–3], can recover the lost packets without the need for feedback because of their ratelessness property. However, the throughput still degrades quickly if there is packet loss at each network link unless link-by-link feedback and retransmission are adopted.

1.1. Network Coding-Based Approaches

Random linear network coding (RLNC) [4,5], a simple realization of network coding [6–8], can achieve the capacity of multi-hop networks with packet loss even without the need for feedback [9,10]. Unfortunately, a direct application of RLNC induces an enormous overhead for the coefficient vectors, as well as high computational and storage costs in network coding operations at intermediate nodes, where the intermediate nodes are usually routers or embedded devices with low computational power and storage space.

Generation-based RLNC was proposed in [11] to resolve these issues. The input packets of the file are partitioned into multiple subsets called the generations, and RLNC is applied to each generation independently. This approach, however, cannot achieve an optimal theoretical rate. Practical concerns and solutions have been further investigated to improve this type of RLNC, such as decoding delay and complexity [12–20], packet size [21–25], and coefficient overhead [26–28].

Instead of partitioning into disjoint subsets, overlapped subsets were investigated in [29–32]. To further reduce the computational costs, the use of RLNC was restricted to small subsets of coded packets generated from the input packets in [33–38]. Example codes used for generating the coded packets include LDPC and fountain codes. This combination of coding theory and network coding is another variant of RLNC called batched network coding (BNC). BATS codes [38,39], a class of BNC, have a close-to-optimal achievable rate where the achievable rate is upper bounded by the expectation of the rank distribution of the batch transfer matrices that model the end-to-end network operations (packet erasures, network coding operations, etc.) on the batches [40]. This hints that the network coding operations, also known as recoding, have an impact on the throughput of BNC.

1.2. Challenges of Recoding in Practice

Baseline recoding is the simplest recoding scheme which generates the same number of recoded packets for every batch. Due to its simple and deterministic structure, baseline recoding appears in many BNC designs and analyses, such as [41–46]. However, the throughput of baseline recoding is not optimal with finite batch sizes [47]. The idea of adaptive recoding, aiming to outperform baseline recoding by generating different numbers of recoded packets for different batches, was proposed in [47] without truly optimizing the numbers. Two adaptive recoding optimization models for independent packet loss channels were then formulated independently in [48,49]. A unified adaptive recoding framework was proposed in [50], subsuming both optimization models and supporting other channel models under certain conditions.

Although adaptive recoding can be applied distributively with local network information, it is a challenge to obtain accurate local information when deploying adaptive recoding in real-world scenarios. Adaptive recoding requires two pieces of information: information distribution remaining in the received batches and the channel condition of the outgoing link.

The first piece of information may change over time if the channel condition of the incoming link varies. One reason for the variation is that the link quality can be affected by interference from users of other networks around the network node. We proposed a simple way to adapt to this variation in [49], grouping a few batches into a block and observing the distribution of received batches in this block. This approach was later called blockwise adaptive recoding (BAR) in [51,52].

The second piece of information may also vary from time to time. In some scenarios, such as deep-space [53–55] and underwater communications [56–58], feedback can be expensive or is not available; therefore, a feedbackless network is preferred. Without feedback, we cannot update our knowledge on the channel condition of the outgoing link. Although we may assume an unchanged channel condition and measure information such as the packet loss rate of the channel beforehand, this measurement, however, can be inaccurate due to observational errors or precision limits.

1.3. Contributions

In this paper, we focus on the practical design of applying BAR in real-world applications. Specifically, we answer the following questions in this paper:

1. How does the block size affect the throughput?
2. Is BAR sensitive to an inaccurate channel condition?
3. How can one calculate the components of BAR and solve the optimization efficiently?
4. How can one make use of link-by-link feedback if it is available?

The first question is related to the trade-off between throughput and latency: a larger block induces a longer delay but gives a higher throughput. We show by numerical evaluations that a small block size can already give a significant throughput gain compared with baseline recoding.

For the second question, we demonstrate that BAR performs very well with an independent packet loss model on channels with dependent packet loss. We also show that BAR is insensitive to an inaccurate packet loss rate. This is an encouraging result as this suggests that it is feasible to apply BAR in real-world applications.

The third question is important in practice as BAR is suppose to run at network nodes, usually routers or IoT devices with limited computational power, but also they may need to handle a huge amount of network traffic. Furthermore, by updating the knowledge of the incoming link from a short observation, we need to recalculate the components of BAR and solve the optimization problem again. In light of this, we want to reduce the number of computations to improve the reaction time and reduce the stress of congestion. We answer this question by proposing an on-demand dynamic programming approach to build the components and some implementation techniques to speed up the algorithm for BAR.

Lastly, for the fourth question, we consider both a perfect feedback system (e.g., the feedback passes through a side-channel with no packet loss) and a lossy feedback system (e.g., the feedback uses the reverse direction of the lossy channel for data transmission). We investigate a few ways to estimate the packet loss rate and show that the throughput can be further boosted by using feedback. Furthermore, a rough estimation is sufficient to catch up the variation in the channel condition. In other words, unless there is another application which requires a more accurate estimation on the packet loss rate, we may consider using an estimation with low computational cost, e.g., the maximum likelihood estimator.

1.4. Paper Organization and Nomenclature

The paper is organized as follows. We first formulate BAR in Section 2. Then, we discuss the implementation techniques for solving BAR efficiently and evaluate the throughput of different block sizes in Section 3. In Section 4, we demonstrate that BAR is insensitive to inaccurate channel models and investigate the use of feedback mechanisms. Lastly, we conclude the paper in Section 5.

Some specific terminology and notations appear frequently throughout the paper. We summarize some of the important terminology and frequently used notations in Tables 1 and 2, respectively.

Table 1. Terminology used for batched network coding (BNC).

Terminology	Description
Batch	A small set of coded packets.
Batch size	The number of coded packets in a batch.
Rank of a batch	A measure of "useful" information (linearly independent packets) retained in the batch.
Expected rank of a batch	The expectation of the rank of the batch at the next network node.
Incoming rank distribution	The distribution of the ranks of the batches arriving at the current network node.
Throughput	The expectation of the rank distribution of the batches arriving at the destination node.
Recoding	The network coding operations restricted to the packets belonging to the same batch.
Recoded packets	The coded packets generated by recoding.
Recoder	The module that performs recoding.
Baseline recoding	A strategy that generates the same number of recoded packets per batch.
Adaptive recoding	A strategy that generates different number of recoded packets per batch.
Block	A set of batches.
Blockwise adaptive recoding	Applying adaptive recoding block by block.

Table 2. Frequently used notations in this paper.

Notation	Description
M	Batch size.
r_b	The rank of the batch b.
t_b	The number of recoded packets for batch b.
$E(r_b, t_b)$	The expected rank of batch b when its rank is r_b at the current node and t_b recoded packets are sent.
$(h_0, \ldots, h_M)$	The incoming rank distribution.
p	The packet loss rate in the independent packet loss model.
$\mathcal{L}$	A block.
$t^{\mathcal{L}}_{\max}$	The total number of recoded packets in block $\mathcal{L}$.
$t^{b}_{\max}$	The maximum number of recoded packets allowed for batch b.
$\mathsf{Binom}(n, p)$	The binomial distribution.
$B_p(t, i)$	The probability mass function of the binomial distribution $\mathsf{Binom}(t, 1-p)$.
$\beta_p(t, r)$	The sum of the first r probability masses of $\mathsf{Binom}(t, 1-p)$.
$\mathsf{Beta}(a, b)$	The beta distribution.
$I_x(a, b)$	The regularized incomplete beta function.

2. Blockwise Adaptive Recoding

In this section, we briefly introduce BNC and then formulate BAR.

2.1. Network Model

As some intermediate nodes may be hardware-implemented routers or not easily reachable for an upgrade, it is not required to deploy a BNC recoder at every intermediate node. The nodes that do not deploy a recoder are transparent to the BNC network as no network coding operations are performed. In the following text, we only consider intermediate nodes that have deployed BNC recoders.

It is not practical to assume every intermediate node knows the information of the whole network; thus, a distributed scheme that only requires local information is desirable. For example, the statistics of the incoming batches, the channel condition of the outgoing link, etc. In a general network, there may be more than one possible outgoing link to reach the destination. We can assign one recoder or one management unit for each outgoing link at an intermediate node [59,60]. In this way, we need a constraint to limit the number of recoded packets of certain batches sent via the outgoing links. The details are discussed in Section 2.4. In other words, we consider each route from the source to the destination separately as a line network.

Line networks are the fundamental building blocks of a general network. Conversely, a recoding scheme for line networks can be extended to general unicast networks and certain multicast networks [38,48]. A line network is a sequence of network nodes where the network links only exist between two neighbouring nodes. An example of a line network is illustrated in Figure 1. In this paper, we only consider line networks in our numerical evaluations.

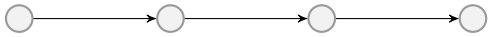


Figure 1. A three-hop line network. Network links only exist between two neighbouring nodes.

2.2. Batched Network Coding

Suppose we want to send a file from a source node to a destination node through a multi-hop network. The file is divided into multiple input packets, where each packet is regarded as a vector over a fixed finite field. A BNC has three main components: the encoder, the recoder and the decoder.

The encoder of a BNC is applied at the source node to generate batches from the input packets, where each batch consists of a small number of coded packets. Recently, a reinforcement learning approach to optimize the generation of batches was proposed in [61]. Nevertheless, batches are commonly generated using the traditional approach as follows. To generate a batch, the encoder samples a predefined degree distribution to obtain a degree, where the degree is the number of input packets that constitute the batch. Depending on the application, there are various ways to formulate the degree distribution [62–65]. According to the degree, a set of packets is chosen randomly from the input packets. The size of the input packets may be obtained via certain optimizations, such as in [66], to minimize the overhead. Each packet in the batch is formed by taking random linear combinations on the chosen set of packets. The encoder generates M packets per batch, where M is known as the batch size.

Each packet in a batch has a coefficient vector attached to it. Two packets in a batch are defined as linearly independent of each other if and only if their coefficient vectors are linearly independent from each other. Immediately after a batch is generated, the packets within it are assigned as linearly independent from each other. This is accomplished by suitably choosing the initial coefficient vectors [59,67].

A recoder is applied at each intermediate node, performing network coding operations on the received batches to generate recoded packets. This procedure is known as recoding. Some packets of a batch may be lost when they pass through a network link. Each recoded packet of a batch is formed by taking a random linear combination of the received packets in a given batch. The number of recoded packets depends on the recoding scheme. For example, baseline recoding generates the same number of recoded packets for every batch. Optionally, we can also apply a recoder at the source node so that we can have more than M packets per batch at the beginning. After recoding, the recoded packets are sent to the next network node.

At the destination node, a decoder is applied to recover the input packets. Depending on the specific BNC, we can use different decoding algorithms, such as Gaussian elimination, belief propagation and inactivation [68,69].

2.3. Expected Rank Functions

The rank of a batch at a network node is defined by the number of linearly independent packets remaining in the batch, a measure of the amount of information carried by the batch. Adaptive recoding aims to maximize the sum of the expected value of the rank distribution of each batch arriving at the next network node. For simplicity, we called this expected value the expected rank.

For batch b, we denote its rank by r_b and the number of recoded packets to be generated by t_b. The expectation of r_b at the next network node, denoted as $E(r_b, t_b)$, is known as the expected rank function. We have

$$E(r,t) = \sum_{i=0}^{t} \Pr(X_t = i) \sum_{j=0}^{\min\{i,r\}} j\zeta_j^{i,r}, \tag{1}$$

where X_t is the random variable of the number of packets of a batch received by the next network node when we send t packets for this batch at the current node, and $\zeta_j^{i,r}$ is the probability that a batch of rank r at the current node with i received packets at the next network node has rank j at the next network node. The exact formulation of $\zeta_j^{i,r}$ can be found in [38], which is $\zeta_j^{i,r} = \frac{\zeta_j^i \zeta_j^r}{\zeta_j^j q^{(i-j)(r-j)}}$, where q is the field size for the linear algebra operations and $\zeta_j^m = \prod_{k=0}^{j-1}(1-q^{-m+k})$. It is convenient to use $q = 2^8$ in practice as each symbol in this field can be represented by 1 byte. For a sufficiently large field size, say $q = 2^8$, $\zeta_j^{i,r}$ is very close to 1 if $j = \min\{i,r\}$, and is very close to 0 otherwise. That is, we can approximate $\zeta_j^{i,r}$ by $\delta_{j,\min\{i,r\}}$ where $\delta_{\cdot,\cdot}$ is the Kronecker delta. This approximation has also been used in the literature, see, e.g., [45,70–76].

Besides generating all recoded packets by taking random linear combinations, systematic recoding [39,47,59,67], which concerns received packets as recoded packets, can be applied to save computational time. Systematic recoding can achieve a nearly indistinguishable performance compared with methods which generate all recoded packets by taking random linear combinations [39]. Therefore, we can also use (1) to approximate the expected rank functions for systematic recoding accurately.

For the independent packet loss model with packet loss rate p, we have $X_t \sim \mathsf{Binom}(t, 1-p)$, a binomial distribution. If $p = 1$, then a store-and-forward technique can guarantee the maximal expected rank. If $p = 0$, then no matter how many packets we transmit, the next network node must receive no packets. Thus, we assume $0 < p < 1$ in this paper. It is easy to prove that the results in this paper are also valid for $p = 0$ or 1 when we define $0^0 := 1$, which is a convention in combinatorics such that $\mathsf{Binom}(t, 0)$ and $\mathsf{Binom}(t, 1)$ are well-defined with correct interpretation. In the remaining text, we assume $\zeta_j^{i,r} = \delta_{j,\min\{i,r\}}$. That is, for the independent packet loss model, we have

$$E_{\mathsf{indep}}(r,t) = \sum_{i=0}^{t} \binom{t}{i} (1-p)^i p^{t-i} \min\{i,r\}. \tag{2}$$

A demonstration of the accuracy of the approximation $\zeta_j^{i,r} \approx \delta_{j,\min\{i,r\}}$ can be found in Appendix A.

We also consider the expected rank functions for burst packet loss channels modelled by Gilbert–Elliott (GE) models [77,78], where the GE model was also used in other BNC literature such as [52,55,70]. A GE model is a two-state Markov chain, as illustrated in Figure 2. In each state, there is an independent event to decide whether a packet is lost or not. We define $f(s,i,t) := \Pr(S_t = s, X_t = i)$, where S_t is the random variable of the state of the GE model after sending t packets of a batch. By exploiting the structure of the GE model, computation of f can be performed by dynamic programming. Then, we have

$$E_{\mathsf{GE}}(r,t) = \sum_{i=0}^{t} (f(\mathbf{G},i,t) + f(\mathbf{B},i,t)) \min\{i,r\}. \tag{3}$$

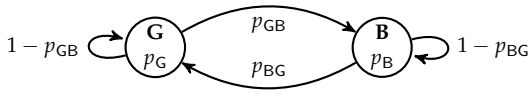


Figure 2. A Gilbert–Elliott (GE) model. In each state, there is an independent event to decide whether a packet is lost or not.

It is easy to see that it would take more steps to compute (3) than (2). Therefore, a natural question to ask is that for burst packet loss channels, is the throughput gap small between adaptive recoding with (2) and (3)? We demonstrate in Section 4.2 that the gap is small so we use (2) any time a nice throughput is received. Therefore, in our investigation we mainly focus on (2).

In the rest of this paper, we refer to $E(r,t)$ as $E_{\mathsf{indep}}(r,t)$ unless otherwise specified. From [50], we know that when the loss pattern follows a stationary stochastic process, the expected rank function $E(r,t)$ is a non-negative, monotonically increasing concave function with respect to t, which is valid for arbitrary field sizes. Further, $E(r,0)=0$ for all r. However, we need to calculate the values of $E(r,t)$ or its supergradients to apply adaptive recoding in practice. To cope with this issue, we first investigate the recursive formula for $E(r,t)$.

We define the probability mass function of the binomial distribution $\mathsf{Binom}(t,1-p)$ by

$$B_p(t,i)=\begin{cases}\binom{t}{i}(1-p)^i p^{t-i} & \text{if } 0\le i\le t,\\ 0 & \text{otherwise.}\end{cases} \tag{4}$$

For integers $r\ge 0$ and $t\ge -1$, we define

$$\beta_p(t,r)=\begin{cases}1 & \text{if } t\le r-1,\\ \sum_{i=0}^{r-1}\binom{t}{i}(1-p)^i p^{t-i}=\sum_{i=0}^{r-1}B_p(t,i) & \text{otherwise.}\end{cases} \tag{5}$$

When $t\ge 0$, the function $\beta_p(t,r)$ is the partial sum of the probability masses of a binomial distribution $\mathsf{Binom}(t,1-p)$. The case where $t=-1$ is used in the approximation scheme in Section 3 and is discussed in that section.

The regularized incomplete beta function, defined as $I_x(a,b):=\frac{\int_0^x t^{a-1}(1-t)^{b-1}\,dt}{\int_0^1 t^{a-1}(1-t)^{b-1}\,dt}$ ([79], Equation 8.17.2), can be used to express the partial sum of the probability masses of a binomial distribution. When $t\ge r>0$, we can apply ([79], Equation 8.17.5) and obtain

$$\beta_p(t,r)=\sum_{i=0}^{r-1}\binom{t}{i}(1-p)^i p^{t-i}=I_p(t-r+1,r). \tag{6}$$

There are different implementations of $I_p(\cdot,\cdot)$ available for different languages. For example, the GNU Scientific Library [80] for C and C++, or the built-in function betainc in MATLAB. However, most available implementations consider non-negative real parameters and calculate different queries independently. This consideration is too general for our application, as we only need to query the integral points efficiently. In other words, this formula may be sufficient for prototyping or simulation, but it is not efficient enough for real-time deployment on devices with limited computational power. Nevertheless, this formula is useful for proving the following properties:

Lemma 1. *Assume $0<p<1$. Let Λ be an index set.*

(a) $B_p(t+1,i)=(1-p)B_p(t,i-1)+pB_p(t,i)$ *for* $i=0,1,\ldots,t$;
(b) $\beta_p(t+1,r)\le\beta_p(t,r)$ *where the equality holds if and only if* $t+1<r$ *or* $t\ge r=0$;
(c) $\beta_p(t,r)\le\beta_p(t+1,r+1)$ *where the equality holds if and only if* $t<r$;
(d) $\beta_p(t,r+1)\ge\beta_p(t,r)$ *where the equality holds if and only if* $t<r$;
(e) $1\ge\max_{b\in\Lambda}\beta_p(t_b,r_b)\ge\beta_p(t_a+s,r_a)$ *for all* $a\in\Lambda$ *and any non-negative integer* s;

(f) $0 \leq \min_{b\in\Lambda} \beta_p(t_b, r_b) \leq \beta_p(t_a - s, r_a)$ *for all* $a \in \Lambda$ *and any non-negative integer* s *such that* $t_a - s \geq -1$.

Proof. See Appendix B. □

With the notation of $\beta_p(t,r)$, we can now write the recursive formula for $E(r,t)$.

Lemma 2. $E(r,t+1) = E(r,t) + (1-p)\beta_p(t,r)$, *where* t *and* r *are non-negative integers.*

Proof. Let Y_i be independent and identically distributed Bernoulli random variables, where $\Pr(Y_i = 1) = 1 - p$ for all i. When $Y_i = 1$, the i-th packet is received by the next hop.

When we transmit one more packet at the current node, Y_{t+1} indicates whether this packet is received by the next network node or not. If $Y_{t+1} = 0$, i.e., the packet is lost, then the expected rank will not change. If $Y_{t+1} = 1$, then the packet is linearly independent from all the already received packets at the next network node if the number of received packets at the next network node is less than r. That is, the rank of this batch at the next network node increases by 1 if $\sum_{i=1}^{t} Y_i < r$. Therefore, the increment of $E(r,t)$ is $\Pr(Y_{t+1} = 1, \sum_{i=1}^{t} Y_i < r)$. Note that $\sum_{i=1}^{t} Y_i \sim \mathsf{Binom}(t, 1-p)$. As Y_i are all independent and identically distributed, we have $\Pr\big(Y_{t+1} = 1, \sum_{i=1}^{t} Y_i < r\big) = (1-p)\beta_p(t,r)$. □

The formula shown in Lemma 2 can be interpreted as a newly received packet that is linearly independent of all the already received packets with a probability tends to 1 unless the rank has already reached r. This can also be interpreted as $\zeta_j^{i,r} = \delta_{j,\min\{i,r\}}$ with a probability tends to 1. The above lemma can be rewritten in a more useful form as stated below.

Lemma 3. *Let* t *and* r *be non-negative integers.*

(a) $E(r,t+1) = E(r,t) + (1-p)$ *if* $t < r$;

(b) $E(r,t) = (1-p)\sum_{j=0}^{t-1} \beta_p(j,r) = (1-p)\Big(\min\{r,t\} + \sum_{j=r}^{t-1} \beta_p(j,r)\Big)$.

Proof. See Appendix C. □

2.4. Blockwise Adaptive Recoding

The idea of adaptive recoding was presented in [47], and then independently formulated in [48,49]. The former formulation imposes an artificial upper bound on the number of recoded packets and then applies a probabilistic approach to avoid integer programming. The latter investigates the properties of the integer programming problem and proposed efficient algorithms to directly tackle this problem. These two formulations were unified in [50] as a general recoding framework for BNC. This framework requires the distribution of the ranks of the incoming batches, also called the incoming rank distribution. This distribution, however, is not known in advance, and can continually change due to environmental factors. A rank distribution inference approach was proposed in [81], but the long solving time hinders its application in real-time scenarios.

A more direct way to obtain up-to-date statistics is to use the ranks of the few latest batches, a trade-off between a latency of a few batches and the throughput of the whole transmission. This approach was proposed in [49], and later called BAR in [51,52]. In other words, BAR is a recoding scheme which groups batches into blocks and jointly optimizes the number of recoded packets for each batch in the block.

We first describe the adaptive recoding framework and its relation to BAR. We fix an intermediate network node. Let $(h_0, h_1, \ldots, h_M)$ be the incoming rank distribution, t_r the number of recoded packets to be sent for a batch of rank r, and t_{avg} the average number of recoded packets to be sent per batch. The value of t_r is a non-negative real number that is interpreted as follows. Let $\epsilon = t_r - \lfloor t_r \rfloor$ be the fractional part of t_r. There is an ϵ chance to transmit $\lfloor t_r \rfloor + 1$ recoded packets, and a $1 - \epsilon$ chance to transmit $\lfloor t_r \rfloor$ packets. That is, the

fraction is the probability of transmitting one more packet. Similarly, $E(r, t_r)$ is defined as the linear interpolation by $(1-\epsilon)E(r, \lfloor t_r \rfloor) + \epsilon E(r, \lfloor t_r \rfloor + 1)$. The framework maximizes the expected rank of the batches at the next node, which is the optimization problem:

$$\max_{t_r \geq 0, \forall r \in \{0,1,\ldots,M\}} \sum_{r=0}^{M} h_r E(r, t_r) \quad \text{s.t.} \quad \sum_{r=0}^{M} h_r t_r = t_{\text{avg}}. \tag{7}$$

For BAR, the incoming rank distribution is obtained from the recently received few batches. Let a block be a set of batches. We assume that the blocks at a network node are mutually disjoint. Suppose the node receives a block $\mathcal{L}$. For each batch $b \in \mathcal{L}$, let r_b and t_b be the rank of b and the number of recoded packets to be generated for b, respectively. Let $t^{\mathcal{L}}_{\text{max}} = t_{\text{avg}}/|\mathcal{L}|$ be the number of recoded packets to be transmitted for the block $\mathcal{L}$. The batches of the same rank are considered individually with the notations r_b and t_b, and the total number of packets to be transmitted for a block is finite; therefore, we assume t_b for each $b \in \mathcal{L}$ is a non-negative integer, and $t^{\mathcal{L}}_{\text{max}}$ is a positive integer. By dividing both the objective and the constraint of the framework by $|\mathcal{L}|$, we obtain the simplest formulation of BAR:

$$\max_{t_b \in \{0,1,2,\ldots\}, \forall b \in \mathcal{L}} \sum_{b \in \mathcal{L}} E(r_b, t_b) \quad \text{s.t.} \quad \sum_{b \in \mathcal{L}} t_b = t^{\mathcal{L}}_{\text{max}}. \tag{8}$$

To support scenarios with multiple outgoing links for the same batch, e.g., load balancing, we may impose an upper bound on the number of recoded packets per batch. Let t^b_{max} be a non-negative integer that represents the maximum number of recoded packets allowed to be transmitted for the batch b. This value may depend on the rank of b at the node. Subsequently, we can formulate the following optimization problem based on (8):

$$\begin{aligned} \max_{t_b \in \{0,1,2,\ldots\}, \forall b \in \mathcal{L}} \quad & \sum_{b \in \mathcal{L}} E(r_b, t_b) \\ \text{s.t.} \quad & \sum_{b \in \mathcal{L}} t_b = t^{\mathcal{L}}_{\text{max}} \\ & t_b \leq t^b_{\text{max}}, \forall b \in \mathcal{L}. \end{aligned} \tag{9}$$

Note that we must have $\sum_{b \in \mathcal{L}} t^b_{\text{max}} \geq t^{\mathcal{L}}_{\text{max}}$. In the case where this inequality does not hold, we can include more batches in the block to resolve this issue. When t^b_{max} is sufficiently large for all $b \in \mathcal{L}$, (9) degenerates into (8).

The above optimization only depends on the local knowledge at the node. The batch rank r_b can be known from the coefficient vectors of the received packets of batch b. As a remark, the value of $t^{\mathcal{L}}_{\text{max}}$ can affect the stability of the packet buffer. For a general network transmission scenario with multiple transmission sessions, the value of $t^{\mathcal{L}}_{\text{max}}$ can be determined by optimizing the utility of certain local network transmissions [82,83].

Though we do not discuss such optimizations in this paper, we consider solving BAR with a general value of $t^{\mathcal{L}}_{\text{max}}$.

On the other hand, note that the solution to (9) may not be unique. We only need to obtain one solution for recoding purpose. In general, (9) is a non-linear integer programming problem. A linear programming variant of (9) can be formulated by using a technique in [81]. However, such a formulation has a huge amount of constraints and requires the values of $E(r_b, t)$ for all $b \in \mathcal{L}$ and all possible t to be calculated beforehand. We defer the discussion of this formulation to Appendix H.

A network node will keep receiving packets until it has received enough batches to form a block $\mathcal{L}$. A packet buffer is used to store the received packets. Then, the node solves (9) to obtain the number of recoded packets for each batch in the block, i.e., $\{t_b\}_{b \in \mathcal{L}}$. The node then generates and transmits t_b-recoded packets for every batch $b \in \mathcal{L}$. At the same time, the network node continually receives new packets. After all the recoded packets for the block $\mathcal{L}$ are transmitted, the node drops the block from its packet buffer and then repeats the procedure by considering another block.

We do not specify the transmission order of the packets. Within the same block, the ordering of packets can be shuffled to combat burst loss, e.g., [43,44,52]. Such shuffling can reduce the burst error length experienced by each batch so that the packet loss events are more "independent" from each other. On the other hand, we do not specify the rate control mechanism, as it should be separated as another module in the system. This can be reflected in BAR by choosing suitable expected rank functions, e.g., modifying the parameters in the GE model. BAR is only responsible for deciding the number of recoded packets per batch.

The size of a block depends on its application. For example, if an interleaver is applied to L batches, we can group the L batches as a block. When $|\mathcal{L}| = 1$, the only solution is $t_b = t^{\mathcal{L}}_{\max}$, which degenerates into baseline recoding. Therefore, we need to use a block size of at least 2 in order to utilize the throughput enhancement of BAR. Intuitively, it is better to optimize (9) with a larger block size. However, the block size is related to the transmission latency as well as the computational and storage burdens at the network nodes. Note that we cannot conclude the exact rank of each batch in a block until the previous network node finishes sending all the packets of this block. That is, we need to wait for the previous network node to send the packets of all the batches in a block until we can solve the optimization problem. Numerical evaluations in Section 3.5 show that $|\mathcal{L}| = 2$ already has obvious advantage over $|\mathcal{L}| = 1$, and it may not be necessary to use a block size larger than eight.

3. Implementation Techniques for Blockwise Adaptive Recoding

In this paper, we focus on the implementation and performance of BAR. Due to the non-linear integer programming structure of (9), we need to make use of certain properties of the model in order to solve it efficiently. The authors of [49] proposed greedy algorithms to solve (9), which were then generalized in [50] to solve (7). The greedy algorithms in [50] have an potential issue when certain probability masses in the incoming rank distribution are too small, as they may take too many iterations to find a feasible solution. The number of iterations is in the order of $\sum_{r=0}^{M} t_r$, depending on the solution to (7). That is, we cannot establish a bound on the time complexity as the incoming rank distribution can be arbitrary.

For BAR, we do not have this issue because the number of recoded packets in a block, $t^{\mathcal{L}}_{\max}$, is fixed.

In this section, we first discuss the greedy algorithm to solve (9) in Section 3.1. Then, we propose an approximation scheme in Section 3.2, and discuss its application to speed up the solver for practical implementations in Section 3.3. The algorithms in Sections 3.1 and 3.3 are similar to that in [50], but they are modified to optimize BAR. Note that the algorithms in [50] are generalized from [49], so the correctness of the aforementioned modified algorithms is inherited directly from the generalized proofs in [50]. For the approximation scheme in Section 3.2, which did not appear in [50], a more detailed discussion is provided in this section.

The algorithms in this section frequently query and compare the values of $(1-p)\beta_p(t,r)$ for different $t \in \{-1, 0, 1, \ldots, t^{\mathcal{L}}_{\max}\}$ and $r \in \{0, 1, 2, \ldots, M\}$. We suppose a lookup table is constructed so that the queries can be performed in $\mathcal{O}(1)$ time. The table is reusable if the packet loss rate of the outgoing link is unchanged. We only consider the subset $\{-1, 0, 1, \ldots, t^{\mathcal{L}}_{\max}\} \times \{0, 1, 2, \ldots, M\}$ of the β_p domain because

1. the maximum rank of a batch is M;
2. any t_b cannot exceed $t^{\mathcal{L}}_{\max}$ as $\sum_{b\in\mathcal{L}} t_b = t^{\mathcal{L}}_{\max}$.

The case $t = -1$ will be used by our approximation scheme so we keep it in the lookup table. We can build the table on-demand by dynamic programming, discussed in Section 3.4.

3.1. Greedy Algorithm

We first discuss the case $t^{\mathcal{L}}_{\max} \leq \sum_{b\in\mathcal{L}} \min\{r_b, t^b_{\max}\}$. This condition means that the value of $t^{\mathcal{L}}_{\max}$ is too small such that the node has just enough or even not enough time to forward the linearly independent packets received. It is trivial that every $\{t_b\}_{b\in\mathcal{L}}$ satisfying $0 \leq t_b \leq \min\{r_b, t^b_{\max}\}$ and $\sum_{b\in\mathcal{L}} t_b = t^{\mathcal{L}}_{\max}$ is a solution to (9), because every such recoded

packet gains $1-p$ to the expected rank by Lemma 3(a),where this gain is maximal according to the definition of $\beta_p(t,r)$.

For $t^{\mathcal{L}}_{\max} > \sum_{b\in\mathcal{L}} \min\{r_b, t^b_{\max}\}$, we can initialize t_b by $\min\{r_b, t^b_{\max}\}$ for every $b \in \mathcal{L}$ as every such recoded packet gains the maximal value $1-p$ to the expected rank. After this, the algorithm chooses the batch that can gain the most expected rank by sending one more recoded packet, and assigns one more recoded packet to it. The correctness is due to the concavity of the expected rank functions.

The above initialization reduces most iterations in the algorithm, as in practice, the difference between the number of recoded packets and the rank of the batch is not huge. Algorithm 1 is the improved greedy algorithm. Unlike the version in [50], the complexity of Algorithm 1 does not depend on the solution.

Algorithm 1: Solver for BAR.

Data: $t^{\mathcal{L}}_{\max}; \{r_b\}_{b\in\mathcal{L}}$
Result: An assignment $\{t^*_b\}_{b\in\mathcal{L}}$ solving (9)
$t \leftarrow t^{\mathcal{L}}_{\max}$; $t_b \leftarrow 0, \forall b \in \mathcal{L}$;
foreach $b \in \mathcal{L}$ **do**
  if $\min\{r_b, t^b_{max}\} \geq t$ **then**
    $t_b \leftarrow t$;
    return The assignment $\{t_b\}_{b\in\mathcal{L}}$;
  else
    $t_b \leftarrow \min\{r_b, t^b_{\max}\}$; $t \leftarrow t - t_b$;
while $t > 0$ **do**
  $T \leftarrow \{b \in \mathcal{L} : t_b = t^b_{\max}\}$;
  $b \leftarrow$ an element in $\arg\max_{b\in\mathcal{L}\setminus T} \beta_p(t_b, r_b)$;
  $t_b \leftarrow t_b + 1$;
  $t \leftarrow t - 1$;
return The assignment $\{t_b\}_{b\in\mathcal{L}}$;

Theorem 1. *Algorithm 1 can be ran in* $\mathcal{O}(|\mathcal{L}| + \max\{0, t^{\mathcal{L}}_{max} - \sum_{b\in\mathcal{L}} \min\{r_b, t^b_{max}\}\} \log|\mathcal{L}|)$ *time.*

Proof. There are totally $\max\{0, t^{\mathcal{L}}_{\max} - \sum_{b\in\mathcal{L}} \min\{r_b, t^b_{\max}\}\}$ iterations in the **while** loop. The query of $\mu = \arg\max_{b\in\mathcal{L}\setminus T} \beta_p(t_b, r_b)$ can be implemented by using a binary heap. The initialization of the heap, i.e., heapify, takes $\mathcal{O}(|\mathcal{L}|)$ time, which can be performed outside the loop. Each query in the loop takes $\mathcal{O}(1)$ time. The update from $\beta_p(t_\mu, r_\mu)$ into $\beta_p(t_\mu + 1, r_\mu)$, if $t_\mu < t^\mu_{\max} - 1$, takes $\mathcal{O}(\log|\mathcal{L}|)$ time. For $t_\mu = t^\mu_{\max} - 1$, we may remove the entry from the heap, taking the same time complexity as the update above. As $\sum_{b\in\mathcal{L}} \min\{r_b, t^b_{\max}\} \geq t^{\mathcal{L}}_{\max}$ by assumption, the algorithm will not query an empty heap. Therefore, the overall time complexity is $\mathcal{O}(|\mathcal{L}| + \max\{0, t^{\mathcal{L}}_{\max} - \sum_{b\in\mathcal{L}} \min\{r_b, t^b_{\max}\}\} \log|\mathcal{L}|)$. □

In the algorithm, we assume that a lookup table for $\beta_p(t,r)$ is pre-computed. The table can be reused unless there is an update on the outgoing channel condition. Nevertheless, we will discuss an efficient way to construct the lookup table in Section 3.4, and the insignificance of the measurement or prediction errors of the loss probability of the outgoing channel in Section 4.

As the query $\arg\max_{b\in\mathcal{L}\setminus T} \beta_p(t_b, r_b)$ is run repeatedly and an update is performed after every query, we can use a binary heap as described in the proof in real implementation. Note that by Lemma 1(b), $\beta_p(t_\mu, r_\mu) \geq \beta_p(t_\mu + 1, r_\mu)$, so the update is a decrease key operation in a max-heap. In other words, a Fibonacci heap [84] cannot benefit from this operation here.

3.2. Equal Opportunity Approximation Scheme

Algorithm 1 increases t_b step by step. From a geometric perspective, the algorithm finds a path from the interior of a compact convex polytope that models the feasible solu-

tions to the facet $\mathcal{H}$: $\sum_{b\in\mathcal{L}} t_b = t^{\mathcal{L}}_{\max}$. If we have a method to move a non-optimal feasible point on $\mathcal{H}$ towards an optimal point, together with a fast and accurate approximation to (8) or (9), then we can combine them to solve (9) faster than using Algorithm 1 directly. This idea is illustrated in Figure 3. A generalized tuning scheme can be found in [50] based on the algorithm in [49]. However, there is no approximation scheme proposed in [50].

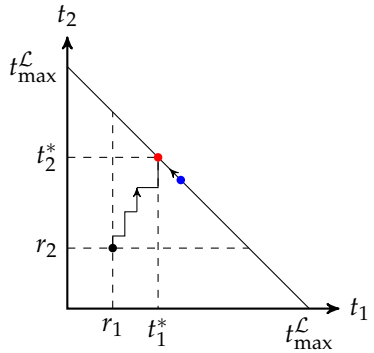


Figure 3. This figure illustrates the idea of modifying the output of an approximation scheme with two batches, where $\mathcal{L} = \{1,2\}$, $t^{\mathcal{L}}_{\max} \geq r_1 + r_2$ and $t^1_{\max}, t^2_{\max} \geq t^{\mathcal{L}}_{\max}$. The red and blue dots represent the optimal and approximate solutions on the facet $t_1 + t_2 = t^{\mathcal{L}}_{\max}$, respectively. Algorithm 1 starts the search from an interior point (r_1, r_2), while a modification approach starts the search from the blue dot.

We first give an approximation scheme in this subsection. The approximation is based on an observation of the solution for (8) that does not impose an upper boundary on t_b: A batch of higher rank should have more recoded packets transmitted than a batch of lower rank. Unless most $t^b_{\max}$ are too small, the approximation for (8) is also a good approximation for (9).

Theorem 2. *Let $\mathcal{L}$ be a block where $|\mathcal{L}| \geq 2$. If $\{t_b\}_{b\in\mathcal{L}}$ solves (8) and $t_b \geq r_b$ for all $b \in \mathcal{L}$, then $t_m < t_n$ for all $m, n \in \mathcal{L}$ such that $r_m < r_n$.*

Proof. See Appendix D. □

As we cannot generate any linearly independent packets for a batch of rank 0, we have $E(0,\cdot) = 0$. Therefore, we can exclude batches of rank 0 from $\mathcal{L}$ before we start the approximation. We define $\mathfrak{L} = \{b \in \mathcal{L} : r_b > 0\} \subseteq \mathcal{L}$. When $t^{\mathcal{L}}_{\max} > \sum_{b\in\mathcal{L}} r_b$, we have $t_b \geq r_b$ for all $b \in \mathcal{L}$. An easy way to obtain an approximation is to assign $\{t_b\}_{b\in\mathcal{L}}$ following the guidelines given in Theorem 2 by:

- $t_b = 0$ for all $b \in \mathcal{L} \setminus \mathfrak{L}$;
- $t_b = r_b + \ell$ for all $b \in \mathfrak{L}$.

where $\ell = (t^{\mathcal{L}}_{\max} - \sum_{b\in\mathcal{L}} r_b)/|\mathfrak{L}|$. In the case where ℓ is not an integer, we can round it up for batches with higher ranks and round it down for those with lower ranks.

The above rules allocate the unassigned packets to batches equally after r_b packets have been assigned to each batch b. Thus, we call this approach the equal opportunity approximation scheme. The steps of this scheme are summarized in Algorithm 2.

Note that we do not need to know the packet loss rate p to apply this approximation. That is, if we do not know the value of p, we can still apply this approximation to outperform baseline recoding.

Theorem 3. *Algorithm 2 approximates (8) in $\mathcal{O}(|\mathcal{L}|)$ time. If $t^{\mathcal{L}}_{max} \leq \sum_{b\in\mathcal{L}} r_b$, then the algorithm solves (8).*

Algorithm 2: Equal opportunity approximation scheme.

Data: $t^{\mathcal{L}}_{\max}$; $\{r_b\}_{b\in\mathcal{L}}$
Result: An assignment $\{t_b\}_{b\in\mathcal{L}}$ approximating (8)
$\ell \leftarrow t^{\mathcal{L}}_{\max}$; $t_b \leftarrow 0, \forall b \in \mathcal{L}$; $L \leftarrow 0$;
foreach $b \in \mathcal{L}$ **do**
  if $r_b \geq \ell$ **then**
    $t_b \leftarrow \ell$;
    return The assignment $\{t_b\}_{b\in\mathcal{L}}$;
  else if $r_b > 0$ **then**
    $t_b \leftarrow r_b$; $\ell \leftarrow \ell - r_b$; $L \leftarrow L + 1$;
if $L = 0$ **then**
  return An arbitrary feasible solution $\{t_b\}_{b\in\mathcal{L}}$;
$r \leftarrow \ell \bmod L$; $\ell \leftarrow \lfloor \ell / L \rfloor$;
foreach $b \in \mathcal{L}$ *s.t.* $r_b > 0$ **do**
  $t_b \leftarrow r_b + \ell$;
for *the r elements which have the largest* $r_b, b \in \mathcal{L}$ **do**
  $t_b \leftarrow t_b + 1$;
return The assignment $\{t_b\}_{b\in\mathcal{L}}$;

Proof. It is easy to see that Algorithm 2 outputs $\{t_b\}_{b\in\mathcal{L}}$ which satisfies $\sum_{b\in\mathcal{L}} t_b = t^{\mathcal{L}}_{\max}$. That is, the output is a feasible solution of (8). Note that $|\mathfrak{L}| \leq |\mathcal{L}|$, so the assignments and the branches before the last **for** loop take $\mathcal{O}(|\mathcal{L}|)$ time in total. The variable L after the first **foreach** loop equals $|\mathfrak{L}|$. Adding one to the number of recoded packets for $r = \ell \bmod L$ batches with the highest ranks can be performed in $\mathcal{O}(|\mathcal{L}|)$ time. Therefore, the overall running time is $\mathcal{O}(|\mathcal{L}|)$.

If $\mathfrak{L} = \emptyset$, i.e., the whole block is lost, then any feasible $\{t_b\}_{b\in\mathcal{L}}$ is a solution, and the optimal objective value is 0. If $t^{\mathcal{L}}_{\max} \leq \sum_{b\in\mathcal{L}} r_b$, then the algorithm terminates with an output satisfying $t_b \leq r_b$ for all $b \in \mathcal{L}$, which is an optimal solution. □

For the step that adds 1 to the number of recoded packets for $r = \ell \bmod L$ batches with the highest ranks in the algorithm, the worst linear time case can be achieved by using introselect [85] (which is quickselect [86] with a random pivot, but changes to use the median of medians [87] pivot strategy when the complexity grows). We use the selection algorithm to find the r-th largest element, making use of its intermediate steps. During an iteration, one of the following three cases will occur. If the algorithm decides to search a part larger than the pivot, then the discarded part does not contain the largest r elements. If a part smaller than the pivot is selected, then the discarded part is part of the largest r elements. If the pivot is exactly the r-th largest element, then the part larger than the pivot together with the pivot are part of the largest r elements.

In practice, the batch size M is small. We can search these r batches with the highest ranks in $\mathcal{O}(|\mathcal{L}| + M)$ time using a counting technique as an efficient alternative. The technique is to use part of the counting algorithm [88]. We first compute a histogram of the number of times each rank occurs, taking $\mathcal{O}(M)$ time for initialization and $\mathcal{O}(|\mathcal{L}|)$ time to scan the block. Then, we can scan and count the frequencies of the histogram from the highest rank, and eliminate the part where the count exceeds $\ell \bmod |\mathcal{L}|$. This takes $\mathcal{O}(M)$ time. Lastly, we scan the ranks of the batches again in $\mathcal{O}(|\mathcal{L}|)$ time. If included in the modified histogram, we add 1 to the corresponding t_b and minus 1 to the corresponding frequency in the histogram.

Algorithm 2 is a $(1-p)$-approximation algorithm, although the relative performance guarantee factor $1-p$ is not tight in general. However, this suggests that the smaller the packet loss rate p, the more accurate the output the algorithm gives. We defer this discussion to Appendix E.

3.3. Speed-Up via Approximation

In this subsection, we discuss the implementation that corrects an approximate solution to an optimal solution for (9). Algorithm 3 is a greedy algorithm that uses any feasible solution of (8) as a starting point. The **foreach** loop removes the exceeding recoded packets, assigning the released slot to another batch following the iterations in Algorithm 1. This can be regarded as mimicking a replacement of $\beta_p(\cdot,\cdot)$ with the smallest possible value for the batch b that violates the constraint $t_b \le t^b_{\max}$. After this, the intermediate solution is a feasible solution to (9). Then, the last loop finds an increase to the objective by reassigning some slots among the batches.

Algorithm 3: Solver for BAR via approximation.

Data: $t^{\mathcal{L}}_{\max}; r_b, b \in \mathcal{L}$
Result: An assignment $\{t^*_b\}_{b\in\mathcal{L}}$ solving (9)
$t_b \leftarrow 0, \forall b \in \mathcal{L}$;
Run an approximation to get $t_b, b \in \{a \in \mathcal{L} \colon r_a > 0\}$;
$T \leftarrow \{b \in \mathcal{L} \colon t_b \ge t^b_{\max}\}$;
foreach $a \in \mathcal{L}$ *s.t.* $t_a > t^a_{max}$ **do**
  while $t_a > t^a_{max}$ **do**
    $b \leftarrow$ an element in $\arg\max_{b\in\mathcal{L}\setminus T} \beta_p(t_b, r_b)$;
    $t_b \leftarrow t_b + 1$; $t_a \leftarrow t_a - 1$; $T \leftarrow \{a \in \mathcal{L} \colon t_a \ge t^a_{\max}\}$;
while $\min_{a\in\mathcal{L}} \beta_p(t_a - 1, r_a) < \max_{b\in\mathcal{L}\setminus T} \beta_p(t_b, r_b)$ **do**
  $a \leftarrow$ an element in $\arg\min_{a\in\mathcal{L}} \beta_p(t_a - 1, r_a)$;
  $b \leftarrow$ an element in $\arg\max_{b\in\mathcal{L}\setminus T} \beta_p(t_b, r_b)$;
  $t_a \leftarrow t_a - 1$; $t_b \leftarrow t_b + 1$; $T \leftarrow \{a \in \mathcal{L} \colon t_a \ge t^a_{\max}\}$;
return The assignment $\{t_b\}_{b\in\mathcal{L}}$;

Note that the algorithm may query $\beta_p(t_a - 1, r_a)$ for $a \in \mathcal{L}$. If $t_a = 0$, then it has access to the value $\beta_p(-1, r_a)$. Recall that we defined $\beta_p(-1,\cdot) = 1$, the upper bound of $\beta_p(\cdot,\cdot)$ by (10). Therefore, these values act as barriers to prevent outputting a negative number of recoded packets.

Theorem 4. *Let $\{t_b\}_{b\in\mathcal{L}}$ be an approximate solution of (8) computed in $\mathcal{O}(T_{approx})$ time. Algorithm 3 can be run in $\mathcal{O}(T_{approx} + |\mathcal{L}| + \sum_{b\in\mathcal{L}} |t^*_b - t_b| \log |\mathcal{L}|)$ time.*

Proof. The assignments before the **foreach** loop takes $\mathcal{O}(T_{\text{approx}} + |\mathcal{L}|)$ time. There are a total of $\sum_{b\in\mathcal{L}} |t^*_b - t_b|/2$ iterations in the loops. The queries for the minimum and maximum values can be implemented using a min-heap and a max-heap, respectively. Similar to Algorithm 1, we can use binary heaps, taking $\mathcal{O}(|\mathcal{L}|)$ initialization time, $\mathcal{O}(1)$ query time, and $\mathcal{O}(\log |\mathcal{L}|)$ update time. Each iteration contains at most two heap queries and four heap updates. The update of the set T can be performed implicitly by setting $\beta_p(t_a, r_a)$ to 0 during the heap updates for $a \in T$. The overall time complexity is then $\mathcal{O}(T_{\text{approx}} + |\mathcal{L}| + \sum_{b\in\mathcal{L}} |t^*_b - t_b| \log |\mathcal{L}|)$. □

In the last **while** loop, we need to query the minimum of $\beta_p(t_a - 1, r_a)$ and the maximum of $\beta_p(t_b, r_b)$. It is clear that we need to decrease the key $\beta_p(t_b, r_b)$ to $\beta_p(t_b + 1, r_b)$ in the max-heap, and increase the key $\beta_p(t_a - 1, r_a)$ to $\beta_p(t_a - 2, r_a)$ in the min-heap. However, we can omit the updates for batches a and b in the max-heap and min-heap, respectively, i.e., reduce from four heap updates to two heap updates. We defer this discussion to Appendix G.

3.4. Construction of the Lookup Table

In the above algorithms, we assume that we have a lookup table for the function $\beta_p(\cdot,\cdot)$ so that we can query its values quickly. In this subsection, we propose an on-demand approach to construct a lookup table by dynamic programming.

Due to the fact that $\sum_{i=0}^{t} B_p(t,i) = 1$, we have

$$0 \le \beta_p(t,r) \le 1. \tag{10}$$

Furthermore, it is easy to see that

$$\beta_p(t,r) = 0 \text{ if and only if } r = 0 \text{ and } t \ge 0; \tag{11}$$
$$\beta_p(t,r) = 1 \text{ if and only if } t \le r-1. \tag{12}$$

A tabular form of β_p is illustrated in Figure 4 after introducing the boundaries 0 and 1 s.

1	1	1	1	1
0	1	1	1	1
0	$\beta_p(1,1)$	1	1	1
0	$\beta_p(2,1)$	$\beta_p(2,2)$	1	1
0	$\beta_p(3,1)$	$\beta_p(3,2)$	$\beta_p(3,3)$	1
0	$\beta_p(4,1)$	$\beta_p(4,2)$	$\beta_p(4,3)$	$\beta_p(4,4)$

Figure 4. The tabular appearance of the function $\beta_p(t,r)$ after introducing boundaries 0 and 1 s. The rows and columns correspond to $t = -1, 0, 1, \ldots$ and $r = 0, 1, 2, \ldots$, respectively. The row above the line is $\beta_p(-1, \cdot)$.

Being a dynamic programming approach, we need the following recursive relations:

$$B_p(t+1,r) = (1-p)B_p(t,r-1) + pB_p(t,r) \quad \text{for } 0 \le r \le t; \tag{13}$$
$$B_p(r,r) = (1-p)B_p(r-1,r-1) \quad \text{for } r > 0; \tag{14}$$
$$\beta_p(t,r) = \beta_p(t,r-1) + B_p(t,r-1) \quad \text{for } 1 < r \le t+1, \tag{15}$$

where (13) is stated in Lemma 1(a); and (14) and (15) are by the definitions of $B_p(t,r)$ and $\beta_p(t,r)$, respectively. The boundary conditions are $B_p(0,0) = 1$, $B_p(i,-1) = 0$, and $\beta_p(i,1) = B_p(i,0)$ for $i = 0, 1, \ldots$. The table can be built in-place in two stages. The first stage fills in $B_p(y, x-1)$ at the (y,x) position of the table. The second stage finishes the table by using (15). Figure 5 illustrates the two stages where the arrows represent the recursive relations (13)–(15). As $\beta_p(0,1) = \beta_p(1,2) = \ldots = 1$, the corresponding entries can be substituted in directly.

1	1	1	1	1
0	$B_p(0,0)$	1	1	1
0	$B_p(1,0)$	$B_p(1,1)$	1	1
0	$B_p(2,0)$	$B_p(2,1)$	$B_p(2,2)$	1
0	$B_p(3,0)$	$B_p(3,1)$	$B_p(3,2)$	$B_p(3,3)$
0	$B_p(4,0)$	$B_p(4,1)$	$B_p(4,2)$	$B_p(4,3)$

(**a**)

1	1	1	1	1
0	$\beta_p(0,1)$	1	1	1
0	$\beta_p(1,1) \to \beta_p(1,2)$		1	1
0	$\beta_p(2,1) \to \beta_p(2,2) \to \beta_p(2,3)$			1
0	$\beta_p(3,1) \to \beta_p(3,2) \to \beta_p(3,3) \to \beta_p(3,4)$			
0	$\beta_p(4,1) \to \beta_p(4,2) \to \beta_p(4,3) \to \beta_p(4,4)$			

(**b**)

Figure 5. The figures illustrate the two stages of the table generation. The indices start from $(-1, 0)$. The first row has the index $y = -1$, which is the row above the line. Compared to Figure 4, $\beta_p(0,1) = \beta_p(1,2) = \ldots = 1$ can be substituted in directly without using the relation (15). (**a**) The first stage of the table generation. The 1 and 0 s paddings are generated first. The solid and dashed arrows represent (13) and (14), respectively. (**b**) The second stage of the table generation. The 1 and 0 s paddings are kept. The arrows represent the recursive relation (15) with the B_p function in-place.

We can compute the values in the table on-demand. Suppose we have $\{t_b\}_{b \in \mathcal{L}}$ in an iteration of Algorithm 1 so that we need the values of $\beta_p(t_b, r_b)$ for $b \in \mathcal{L}$. Let b' be an element in $\arg\max_{b \in \mathcal{L}} r_b$. The table has $r_{b'} + 1$ columns. By the criteria of selecting b in

Algorithm 1 and by Lemmas 1(c) and (d),we have $\max_{b\in\mathcal{L}} t_b = t_{b'}$. From Figure 5, we know we have to calculate all rows of $\beta(t,r)$ for $t \le t_{b'}$. Furthermore, the recursive relations on a row only depend on the previous row; thus, we need to prepare the values of B_p in the next row so that we have the values to compute β_p in the next row. As an example, Figure 6 illustrates the values we have prepared when $t_{b'} = 2$.

1	1	1	1	1
0	$\beta_p(0,1)$	1	1	1
0	$\beta_p(1,1)$	$\beta_p(1,2)$	1	1
0	$\beta_p(2,1)$	$\beta_p(2,2)$	$\beta_p(2,3)$	1
0	$B_p(3,0)$	$B_p(3,1)$	$B_p(3,2)$	$B_p(3,3)$
*	*	*	*	*

Figure 6. The values prepared when $t_{b'} = 2$. The asterisks represent the values that are not yet initialized.

Each entry in the table is modified at most twice during the two stages. Each assignment takes $\mathcal{O}(1)$ time. Therefore, the time and space complexities for building the table are both $\mathcal{O}(MR)$, where R is the number of rows we want to construct. When restricted by the block size, we know that $R \le t^{\mathcal{L}}_{\max}$. The worst case is that we only receive one rank-M batch for the whole block, which is unlikely to occur. In this case, we have the worst case complexity $\mathcal{O}(Mt^{\mathcal{L}}_{\max})$.

Note that we can use fixed-point numbers instead of floating point numbers for a more efficient table construction. Furthermore, the numerical values in the table are not important as long as the orders for any pair of values in the table are the same.

3.5. Throughput Evaluations

We now evaluate the performance of BAR in a feedbackless multi-hop network. Note that baseline recoding is a special case of BAR with block size 1. Our main goal here is to show the throughput gain of BAR among different block sizes. In the evaluation, all (recoded) packets of a batch are sent before sending those of another batch.

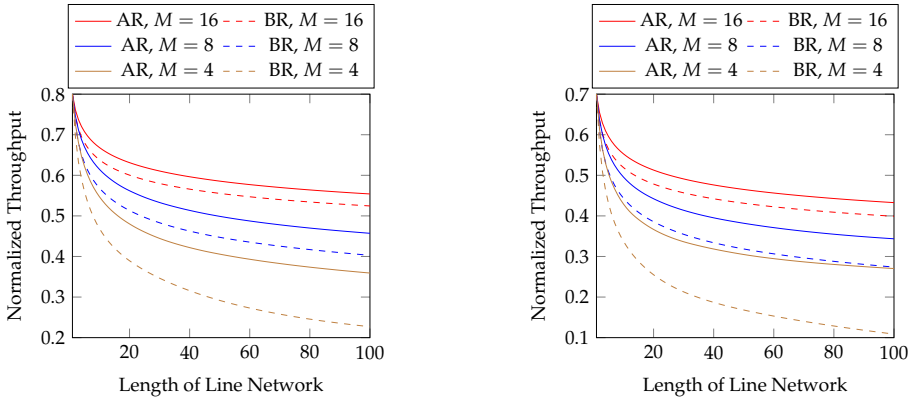
Let $(h_0, h_1, \ldots, h_M)$ be the incoming rank distribution of batches arriving at a network node. The normalized throughput at a network node is defined as the average rank of the received batches divided by the batch size, i.e., $\sum_{i=0}^{M} ih_i/M$. In our evaluations in this subsection, we set $t^{\mathcal{L}}_{\max} = M|\mathcal{L}|$ for every block $\mathcal{L}$. That is, the source node transmits M packets per batch. We assume that every link in the line network has independent packet loss with the same packet loss rate p. In this topology, we set a sufficiently large $t^b_{\max}$ for every batch, say, $t^b_{\max} = t^{\mathcal{L}}_{\max}$.

We first evaluate the normalized throughput with different batch sizes and packet loss rates. Figure 7 compares adaptive recoding (AR) and baseline recoding (BR) when we know the rank distribution of the batches arriving at each network node before the node applies BAR. In other words, Figure 7 shows the best possible throughput of AR. We compare the effect of block sizes later. We observe that

1. AR has a higher throughput than BR under the same setting;
2. the difference in throughput between AR and BR is larger when the batch size is smaller, the packet loss probability is larger, or the length of the line network is longer.

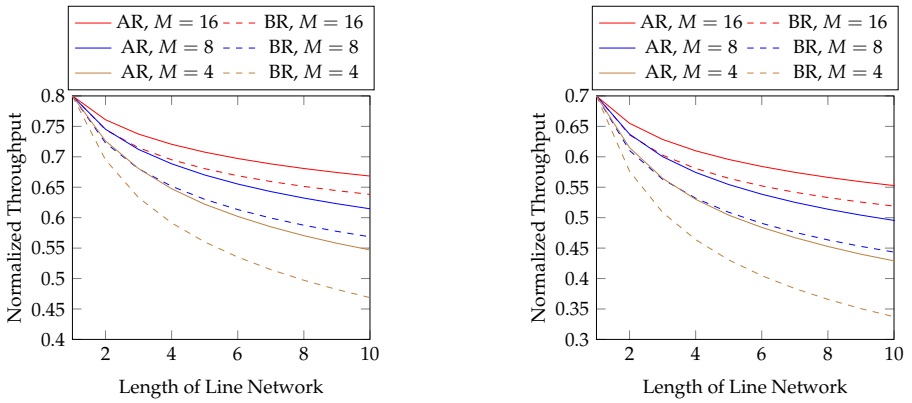
In terms of throughput, the percentage gains of AR over BR using $M = 4$ and $p = 0.2$ are 23.3 and 33.7% at the 20-th and 40-th hops, respectively. They become 43.8 and 70.3%, respectively, when $p = 0.3$.

Although the above figure shows that the throughput of BNC with AR maintains a good performance when the length of the line network is long, many applications use a much shorter line network. We zoom into the figure for the first 10 hops in Figure 8 for practical purposes.



(**a**) $p = 0.2$ (**b**) $p = 0.3$

Figure 7. Adaptive recoding (AR) vs. baseline recoding (BR) in line networks of different lengths, batch sizes and packet loss rates.

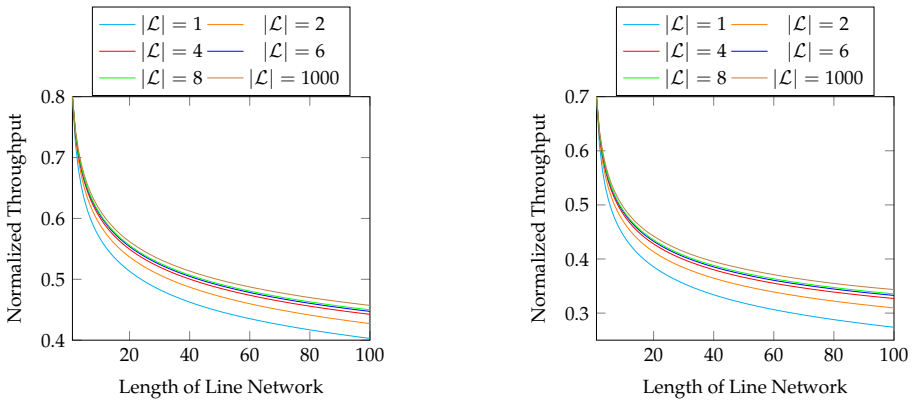


(**a**) $p = 0.2$ (**b**) $p = 0.3$

Figure 8. The first 10 hops in Figure 7.

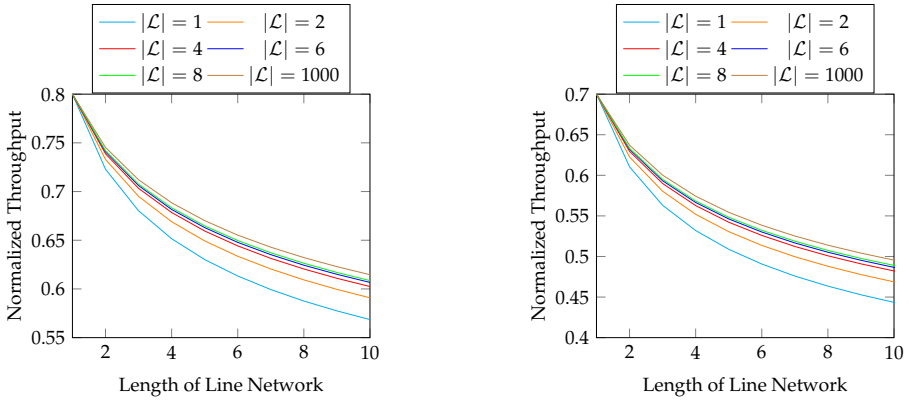
Now, we consider the effect of different block sizes. Figure 9 shows the normalized throughput of different $|\mathcal{L}|$ and p with $M = 8$. The first 10 hops in Figure 9 are zoomed in in Figure 10. We observe that

1. a larger $|\mathcal{L}|$ results a better throughput;
2. using $|\mathcal{L}| = 2$ already gives a much larger throughput than using $|\mathcal{L}| = 1$;
3. using $|\mathcal{L}| > 8$ gives little extra gain in terms of throughput.



(**a**) $p = 0.2$ (**b**) $p = 0.3$

Figure 9. The effect of different block sizes with $M = 8$.

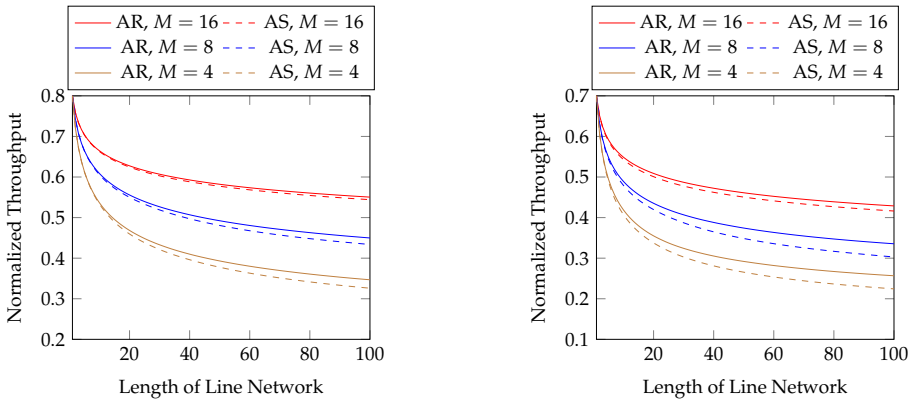


(**a**) $p = 0.2$ (**b**) $p = 0.3$

Figure 10. The first 10 hops in Figure 9.

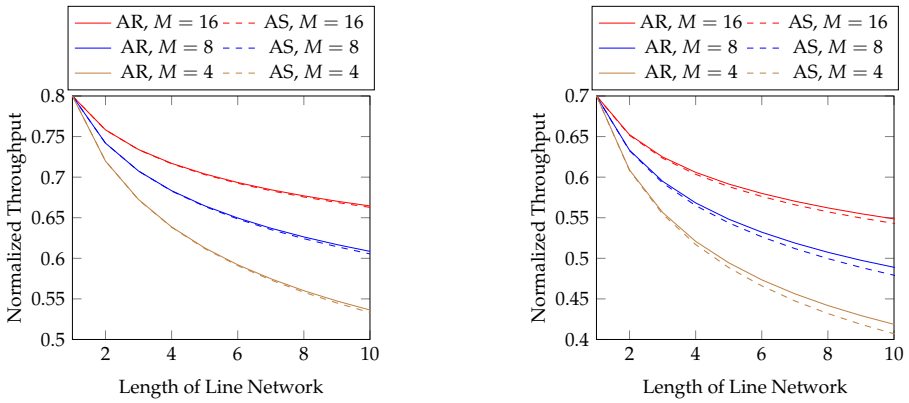
Next, we show the performance of the equal opportunity approximation scheme. Figure 11 compares the normalized throughput achieved by Algorithm 2 (AS) and the true optimal throughput (AR). We compare the best possible throughput of AR here, i.e., the same setting as in Figure 7. The first 10 hops in Figure 11 are zoomed in in Figure 12. We observe that

1. the approximation is close to the optimal solution;
2. the gap in the normalized throughput is smaller when the batch size is larger, the packet loss probability is smaller, or the length of the line network is shorter.



(**a**) $p = 0.2$ (**b**) $p = 0.3$

Figure 11. Approximation vs. optimal AR in line networks of different lengths, batch sizes and packet loss rates.



(**a**) $p = 0.2$ (**b**) $p = 0.3$

Figure 12. The first 10 hops in Figure 11.

4. Impact of Inaccurate Channel Models

In this section, we first demonstrate that the throughput of BAR is insensitive to inaccurate channel models and packet loss rates. Then, we investigate the feedback design and show that although feedback can enhance the throughput, the benefit is insignificant. In other words, BAR works very well without the need of feedback.

4.1. Sensitivity of $\beta_p(t,r)$

We can see that our algorithms only depend on the order of the values of $\beta_p(\cdot,\cdot)$; therefore, it is possible that the optimal $\{t_b\}_{b\in\mathcal{L}}$ for an incorrect p is the same for a correct p. As shown in Figure 4, the boundaries 0 and 1 s are unaffected by $p \in (0,1)$. That is, we only need to investigate the stability of $\beta_p(t,r)$ for $t \geq r > 0$. We calculate values of $\beta_p(t,r)$ corrected to four digital places in Figure 13 for $M = 4$, and $p = 0.1, 0.45$ and their 1% relative changes. We can see that the order of the values are mostly the same when we slightly change p.

0.1000	1	1	1
0.0100	0.1900	1	1
0.0010	0.0280	0.2710	1
0.0001	0.0037	0.0523	0.3439
0.0000	0.0005	0.0086	0.0815
0.0000	0.0001	0.0013	0.0159
0.0000	0.0000	0.0002	0.0027

(**a**) $p = 0.1$.

0.1010	1	1	1
0.0102	0.1918	1	1
0.0010	0.0285	0.2734	1
0.0001	0.0038	0.0533	0.3468
0.0000	0.0005	0.0088	0.0829
0.0000	0.0001	0.0013	0.0163
0.0000	0.0000	0.0002	0.0028

(**b**) $p = 0.101$.

0.0990	1	1	1
0.0098	0.1882	1	1
0.0010	0.0275	0.2686	1
0.0001	0.0036	0.0513	0.3410
0.0000	0.0004	0.0083	0.0800
0.0000	0.0001	0.0012	0.0154
0.0000	0.0000	0.0002	0.0026

(**c**) $p = 0.099$.

0.4500	1	1	1
0.2025	0.6975	1	1
0.0911	0.4252	0.8336	1
0.0410	0.2415	0.6090	0.9085
0.0185	0.1312	0.4069	0.7438
0.0083	0.0692	0.2553	0.5585
0.0037	0.0357	0.1529	0.3917

(**d**) $p = 0.45$.

0.4545	1	1	1
0.2066	0.7024	1	1
0.0939	0.4319	0.8377	1
0.0427	0.2475	0.6163	0.9115
0.0194	0.1358	0.4152	0.7505
0.0088	0.0723	0.2628	0.5676
0.0040	0.0377	0.1589	0.4013

(**e**) $p = 0.4545$.

0.4455	1	1	1
0.1985	0.6925	1	1
0.0884	0.4186	0.8295	1
0.0394	0.2355	0.6016	0.9055
0.0175	0.1268	0.3986	0.7370
0.0078	0.0662	0.2479	0.5494
0.0035	0.0338	0.1471	0.3822

(**f**) $p = 0.4455$.

Figure 13. The values of $\beta_p(t,r)$ for $r = 1,2,3,4$ and $t = 1,2,\ldots$ with different p. The coloured numbers are the largest eight values smaller than 1.

We can also check with the condition number [89] to verify the stability. Roughly speaking, the relative change in the function output is approximately equal to the condition number times the relative change in the function input. A small condition number of $\beta_p(t,r)$ means that the effect of the inaccurate p is small. As shown in Figure 13, the values of $\beta_p(t,r)$ drop quickly when t increases. In the view of the throughput, which is proportional to the sum of these values, we can tolerate a larger relative change, i.e., a larger condition number, when $\beta_p(t,r)$ is small. We calculate condition numbers of $\beta_p(t,r)$ in Figure 14 by the formula stated in Theorem 5.

1.0000	-	-	-
2.0000	0.9474	-	-
3.0000	1.9286	0.8967	-
4.0000	2.9189	1.8585	0.8479
5.0000	3.9130	2.8388	1.7898
6.0000	4.9091	3.8268	2.7596
7.0000	5.9062	4.8187	3.7412

(**a**) $p = 0.1$

1.0000	-	-	-
2.0000	0.7097	-	-
3.0000	1.5714	0.4899	-
4.0000	2.4906	1.2070	0.3296
5.0000	3.4375	2.0325	0.9059
6.0000	4.4000	2.9157	1.6288
7.0000	5.3721	3.8326	2.4384

(**b**) $p = 0.45$

Figure 14. The condition numbers of $\beta_p(t,r)$ for $r = 1,2,3,4$ and $t = 1,2,\ldots$.

Theorem 5. *Let $p \in (0,1)$ and $t \geq r > 0$. The condition number of $\beta_p(t,r)$ with respect to p is $\frac{p^{t-r+1}(1-p)^{r-1}t!}{I_p(t-r+1,r)(t-r)!(r-1)!}$, or equivalently, $\frac{\sum_{j=0}^{r-1}(-1)^j\binom{r-1}{j}p^{t-r+j+1}}{\sum_{j=0}^{r-1}(-1)^j\binom{r-1}{j}p^{t-r+j+1}/(t-r+j+1)}$.*

Proof. See Appendix F. □

4.2. Impact of Inaccurate Channel Models

To demonstrate the impact of an inaccurate channel model, we consider three different channels to present our observations.

- ch1: independent packet loss with constant loss rate $p = 0.45$.
- ch2: burst packet loss modelled by the GE model illustrated in Figure 2 with the parameters used in [70], namely $p_{\mathsf{GB}} = p_{\mathsf{BG}} = p_{\mathsf{G}} = 0.1$, $p_{\mathsf{B}} = 0.8$.
- ch3: independent packet loss with varying loss rates $p = 0.45 + 0.3\sin(2\pi c/1280)$, where c is the number of transmitted batches.

All the three channels have the same average packet loss rate of 0.45. The formula of ch3 is for demonstration purpose only.

We now demonstrate the impact of inaccurate p on the throughput. We consider a line network where all the links use the same channel (ch1, ch2, or ch3). In this topology, we set a sufficiently large $t^b_{\max}$ for every batch, say, $t^b_{\max} = t^{\mathcal{L}}_{\max}$. Similar to the previous evaluation, all (recoded) packets of a batch are sent before sending those of another batch. Furthermore, we set $t^{\mathcal{L}}_{\max} = M|\mathcal{L}|$ for every block $\mathcal{L}$.

In Figure 15 we plot the normalized throughput of the first 80 received blocks at the fourth hop where $|\mathcal{L}| = M = 4$ or 8. We use BAR with (2) for each network although ch2 is a bursty channel. The black curves with BAR are the throughput of BAR where the loss rate is known. For ch1 and ch2, this loss rate p is a constant of 0.45. The red and blue curves are the throughput of BAR when we guess $p = 0.65$ and 0.25, respectively, which is ± 0.2 from the average loss rate of 0.45. As there is no feedback, we do not change our guess on p for these curves. We can see that the throughput is actually very close to the corresponding black curves. This suggests that in the view of the throughput, BAR is not sensitive to p. Even with a wild guess on p, BAR still outperforms BR, as illustrated by the green curves. Regarding ch2, we also plot the orange curve with GE BAR, which is the throughput achieved by BAR with (3). We can see that the gap between the throughput achieved by BAR with (2) and (3) is very small. As a summary of our demonstration:

1. We can use BAR with (2) for bursty channels and the loss in throughput is insignificant.
2. BAR with an inaccurate constant p can achieve a throughput close to the one when we have the exact real-time loss rate.
3. We can see a significant throughput gain from BR by using BAR even with inaccurate channel models.

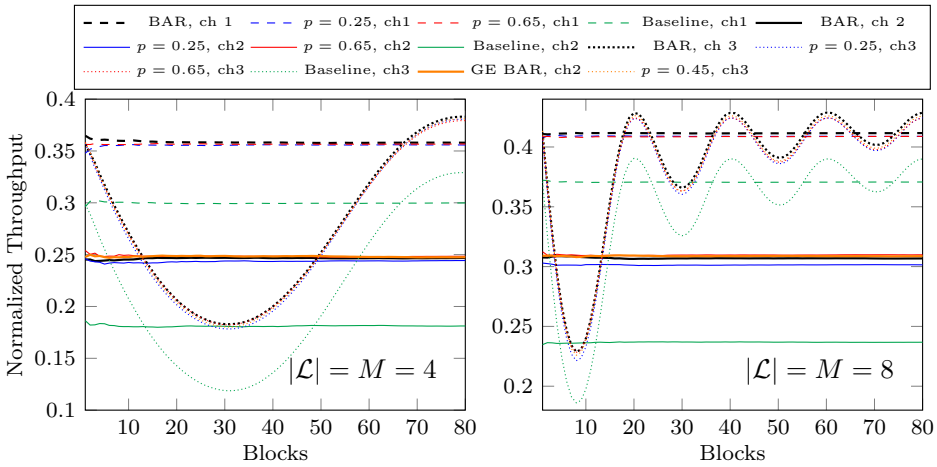


Figure 15. Throughput with inaccurate channel conditions.

4.3. Feedback Design

Although an inaccurate p can give an acceptable throughput, we can further enhance the throughput by adapting the varying p values. To achieve this goal, we need to use feedback.

We adopt a simple feedback strategy which lets the next node return the number of received packets of the batches for the current node to estimate p. Although the next node does not know the number of lost packets per batch, it knows the number of received packets per batch. Therefore, we do not need to introduce more overhead to the transmitted packets by the current node.

When we estimate p, we have to know the number of packets lost during a certain time frame. If the time frame is too small, the estimation is too sensitive so the estimated p changes rapidly and unpredictably. If the time frame is too long, we captured too much out-dated information about the channel so the estimated p changes too slowly and may not be able to adapt to the real loss rate. Recall that the block size is not large as we want to keep the delay small. We use a block as an atomic unit of the time frame. The next node gives feedback on the number of received packets per block. The current node uses the feedback of the blocks in the time frame to estimate p. We perform an estimation of p per received feedback. In this way, the estimated p is the same for each block so that we can apply BAR with (2).

If the feedback is sent via a reliable side channel, then we can assume that the current node can always receive the feedback. However, if the feedback is sent via an unreliable channel, say, the reverse direction of the same channel the data packets were sent from, then we need to consider feedback loss. Let Λ be a set of blocks in a time frame with received feedback. We handle the case of feedback loss by considering the total number of packets transmitted for the blocks in Λ as the total number of packets transmitted during the time frame. In this way, we can also start the estimation before a node sends enough blocks to fill up a time frame. Suppose no feedback is received for every block in a time frame, then we reuse the previously estimated p for BAR.

At the beginning of the transmission, we have no feedback yet so we have no information to estimate p. To outperform BR without the knowledge of p, we can use the approximation of BAR given by Algorithm 2. Once we have received at least one feedback, we can then start estimating p.

4.4. Estimators

Let x and n be the total number of packets received by the current node and the total number of packets transmitted by the previous node, respectively, in a time frame for observation. That is, the number of packets lost in the time frame is $n - x$. We introduce three types of estimators for our numerical evaluations.

(1) Maximum likelihood estimator (MLE): The MLE, denoted by $\hat{p}_{\mathsf{MLE}}$, estimates p by maximizing the likelihood function. $\hat{p}_{\mathsf{MLE}} = (n - x)/n$ is a well-known result which can be obtained via derivative tests. This form collides with the sample average, so by the law of large numbers, $\hat{p}_{\mathsf{MLE}} \to p$ when $n \to \infty$ if p does not change over time.

(2) Minimax estimator: The minimax estimator achieves the smallest maximum risk among all estimators. With the popular mean squared error (MSE) as the risk function, it is a Bayes estimator with respect to the least favourable prior distribution. As studied in [90,91], such prior distribution is a beta distribution $\mathsf{Beta}(\sqrt{n}/2, \sqrt{n}/2)$. The minimax estimator of p, denoted by $\hat{p}_{\mathsf{MM}}$, is the posterior mean, which is $\frac{\sqrt{n}}{1+\sqrt{n}} \frac{n-x}{n} + \frac{1}{1+\sqrt{n}} \frac{1}{2}$, or equivalently, $\frac{n-x+0.5\sqrt{n}}{n+\sqrt{n}}$.

(3) Weighted Bayesian update: Suppose the prior distribution is $\mathsf{Beta}(a, b)$, where the hyperparameters can be interpreted as a pseudo-observation having a successes and b failures. Given a sample of s successes and f failures from a binomial distribution, the posterior distribution is $\mathsf{Beta}(a + s, b + f)$. To fade out the old samples captured by the hyperparameters, we introduce a scaling factor $0 \leq \gamma \leq 1$ and let the posterior distribution be $\mathsf{Beta}(\gamma a + s, \gamma b + f)$. This factor can also prevent the hyperparameters from growing indefinitely. The estimation of p, denoted by $\hat{p}_{\mathsf{Bayes}}$, is the posterior mean with $s = n - x$ and $f = x$, which is $\frac{\gamma a + n - x}{\gamma(a+b)+n}$. To prevent a bias when there are insufficient samples, we

select a non-informative prior as the initial hyperparameters. Specifically, we use the Jeffreys prior, which is $\mathsf{Beta}(1/2, 1/2)$.

We first show the estimation of p by different schemes in Figure 16. We use BAR with (2) and $|\mathcal{L}| = M = 4$. The size of the time frame is W blocks. For $\hat{p}_{\mathsf{MLE}}$ and $\hat{p}_{\mathsf{MM}}$, the observations in the whole time frame have the same weight. For $\hat{p}_{\mathsf{Bayes}}$, the effect of each observation deceases exponentially faster. We consider an observation is out of the time frame when it is scaled into 10% of the original value. That is, we define the scaling factor by $\gamma = \sqrt[W]{0.1}$. In each subplot, the black curve is the real-time p. The red and blue curves are for the estimation without and with feedback loss, respectively. In each case, the two curves are the 25 and 75% percentiles from 1000 runs, respectively.

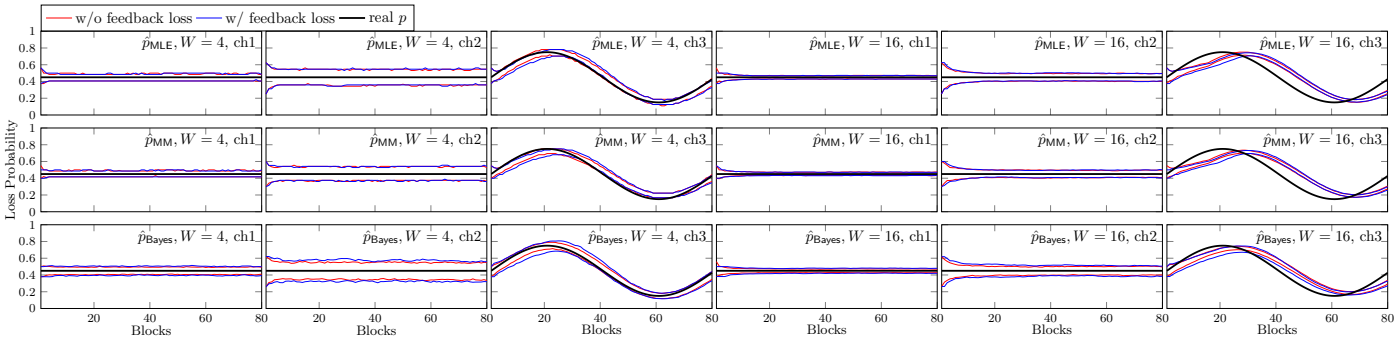


Figure 16. The 25 and 75% percentiles of the estimation of p by different schemes where $|\mathcal{L}| = M = 4$ in 1000 runs.

We can see that a larger W has a slower response to the change in p in ch3. Among the estimators, $\hat{p}_{\mathsf{Bayes}}$ has the fastest response speed as its observations in the time frame are not fairly weighted. Furthermore, although ch1 and ch2 have the same average loss rate, the estimation has a larger variance when the channel is bursty.

4.5. Throughput Evaluations

As discussed in Section 4.2, the guessed p values have an insignificant impact on the throughput. We now show the throughput achieved by the estimation schemes in Figure 17. The parameters for the networks and BAR are the same as in Section 4.2. We do not wildly guess p here so it is no surprise that we can achieve nearly the same throughput as when we know the real p for ch1 and ch2. If we look closely, we can see from Figure 15 that for ch3, there is a small gap between the throughput of BAR when we know the real-time p and the one of BAR when using a constant p. Although the estimation may not be accurate at all times, we can now adapt to the change in p to finally achieve a throughput nearly the same as when we know the real-time p. On the other hand, whether the feedback is lost or not, the plots shown in Figure 17 are basically the same.

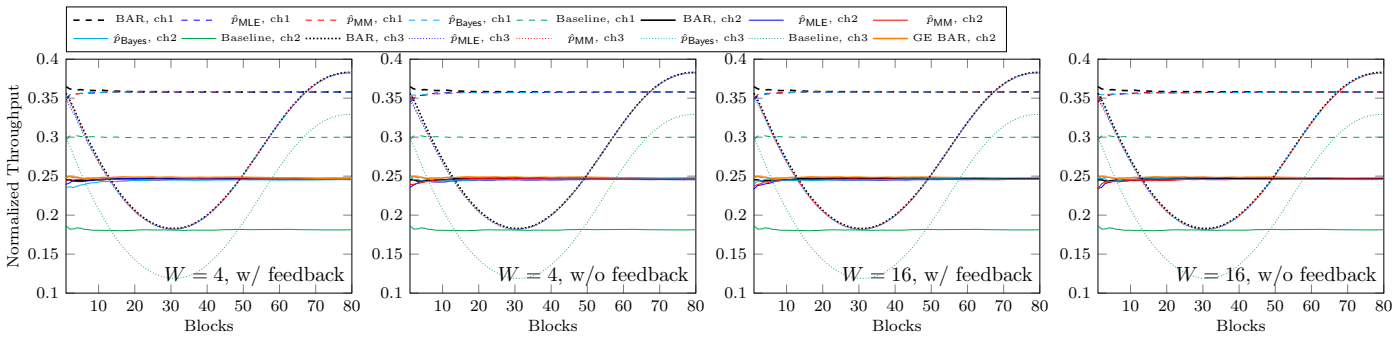


Figure 17. Throughput with estimated p via feedback where $|\mathcal{L}| = M = 4$.

5. Conclusions

We proposed BAR in this paper which can adapt to variations in the incoming channel condition. In a practical perspective, we discussed how to calculate the components of BAR and how to solve BAR efficiently. We also investigated the impact of an inaccurate channel model on the throughput achieved by BAR. Our evaluations showed that

1. BAR is insensitive to the channel model: guessing the loss rate still outperforms BR.
2. For bursty channels, the throughput achieved by BAR with an independent loss model is nearly identical to one with the real channel model. That is, we can use the independent loss model for BAR in practice and apply the techniques in this paper to reduce the computational costs of BAR.
3. Feedback can slightly enhance the throughput for channels with a dynamic loss rate. This suggests that BAR works very well without the need of feedback. On the other hand, feedback loss barely affects the throughput of BAR. Therefore, we can send the feedback through a lossy channel without the need of retransmission. Unless we need to use an accurate estimated loss rate in other applications, we can use MLE with a small time frame for BAR to reduce the computational time.

These encouraging results suggest that BAR is suitable to be deployed in real-world applications.

One drawback of our proposed scheme is that we need to change the default behaviour of some intermediate network nodes, which can be a practical problem in existing networks. In fact, this is a common issue for all network coding schemes. Some routers have hard-wired circuits to efficiently handle heavy traffic, so it is unfeasible to deploy other schemes on them without replacing the hardware. For these heavy-loaded nodes, one may consider producing a hardware to speed up the network coding operations, e.g., [92,93], inducing extra costs on the deployment. On the other hand, the protocol for BNC is not standardized yet, meaning two parties may adopt BAR with incompatible protocols, thus restricting the application of BNC in public networks. However, it is not easy to build a consensus on the protocol, because there are still many research directions to improve the performance of BNC so the protocol design is subject to change in the near future.

6. Patents

The algorithms in Section 3 are variants of those that can be found in the U.S. patent 10,425,192 granted on 24 September 2019 [94]. The linear programming-based algorithm for BAR in Appendix H can be found in the U.S. patent 11,452,003 granted on 20 September 2022 [95].

Author Contributions: Conceptualization, H.H.F.Y. and S.Y.; methodology, H.H.F.Y.; software, H.H.F.Y. and L.M.L.Y.; validation, H.H.F.Y., Q.Z. and K.H.N.; formal analysis, H.H.F.Y.; investigation, H.H.F.Y., S.Y. and Q.Z.; writing—original draft preparation, H.H.F.Y.; writing—review and editing, H.H.F.Y., S.Y. and Q.Z.; visualization, H.H.F.Y.; supervision, H.H.F.Y. and S.Y.; project administration, H.H.F.Y.; funding acquisition, S.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by NSFC under Grants 12141108 and 62171399.

Institutional Review Board Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: Part of the work of Hoover H. F. Yin, Shanghao Yang and Qiaoqiao Zhou was conducted when they were with the Institute of Network Coding, The Chinese University of Hong Kong, Shatin, New Territories, Hong Kong. The work of Lily M. L. Yung was performed when she was with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, New Territories, Hong Kong. The work of Ka Hei Ng was performed when he was with the Department of Physics, The Chinese University of Hong Kong, Shatin, New Territories, Hong Kong. Some results in this paper were included in the thesis of Hoover H. F. Yin [96].

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

BAR	Blockwise adaptive recoding
IoT	Internet of Things
BNC	Batched network coding
RLNC	Random linear network coding
LDPC	Low-density parity-check code
BATS code	Batched sparse code
GE model	Gilbert–Elliott model
MLE	Maximum likelihood estimator
MSE	Mean squared error

Appendix A. Accuracy of the Approximation $\zeta_j^{i,r} \approx \delta_{j,\min\{i,r\}}$

We demonstrate the accuracy of the approximation $\zeta_j^{i,r} \approx \delta_{j,\min\{i,r\}}$ by showing the percentage error of the expected rank function corrected to three decimal places when $q = 2^8$, $p = 0.2$ and $X_t \sim \text{Binom}(t, 1-p)$ in Table A1. That is, the table shows the values

$$\frac{100\left|\sum_{i=0}^{t}\binom{t}{i}(1-p)^i p^{t-i}(\min\{r,i\} - \sum_{j=0}^{\min\{i,r\}} j\zeta_j^{i,r})\right|}{\sum_{i=0}^{t}\binom{t}{i}(1-p)^i p^{t-i}\sum_{j=0}^{\min\{i,r\}} j\zeta_j^{i,r}}$$

for different r and t.

From the table, we can see that only three pairs of (r,t) have percentage errors larger than 0.1%, where they occur when $r, t \leq 2$. For all the other cases, the percentage errors are less than 0.1%. Therefore, such an approximation is accurate enough for practical applications.

Table A1. Percentage error when approximating expected rank functions.

t	$r=1$	$r=2$	$r=3$	$r=4$	$r=5$	$r=6$	$r=7$	$r=8$	$r=9$	$r=10$	$r=11$	$r=12$	$r=13$	$r=14$	$r=15$	$r=16$
1	0.39216	0.00153	0.00001	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2	0.13140	0.15741	0.00061	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
3	0.03841	0.08032	0.08397	0.00033	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
4	0.01025	0.03091	0.05791	0.05042	0.00020	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
5	0.00258	0.01024	0.02761	0.04398	0.03229	0.00013	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
6	0.00062	0.00308	0.01091	0.02502	0.03416	0.02155	0.00008	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
7	0.00015	0.00087	0.00382	0.01147	0.02258	0.02685	0.01479	0.00006	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
8	0.00003	0.00023	0.00122	0.00457	0.01178	0.02023	0.02123	0.01036	0.00004	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
9	0.00001	0.00006	0.00037	0.00165	0.00526	0.01182	0.01798	0.01686	0.00738	0.00003	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
10	0.00000	0.00002	0.00011	0.00055	0.00210	0.00585	0.01163	0.01585	0.01342	0.00532	0.00002	0.00000	0.00000	0.00000	0.00000	0.00000
11	0.00000	0.00000	0.00003	0.00017	0.00077	0.00257	0.00632	0.01125	0.01388	0.01070	0.00387	0.00002	0.00000	0.00000	0.00000	0.00000
12	0.00000	0.00000	0.00001	0.00005	0.00027	0.00103	0.00302	0.00665	0.01072	0.01208	0.00854	0.00284	0.00001	0.00000	0.00000	0.00000
13	0.00000	0.00000	0.00000	0.00002	0.00009	0.00038	0.00131	0.00344	0.00685	0.01009	0.01046	0.00682	0.00210	0.00001	0.00000	0.00000
14	0.00000	0.00000	0.00000	0.00000	0.00003	0.00013	0.00052	0.00160	0.00381	0.00693	0.00940	0.00901	0.00545	0.00156	0.00001	0.00000
15	0.00000	0.00000	0.00000	0.00000	0.00001	0.00004	0.00020	0.00069	0.00190	0.00412	0.00689	0.00866	0.00773	0.00436	0.00117	0.00000
16	0.00000	0.00000	0.00000	0.00000	0.00000	0.00001	0.00007	0.00028	0.00087	0.00219	0.00436	0.00677	0.00792	0.00660	0.00349	0.00088
17	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00002	0.00010	0.00037	0.00106	0.00246	0.00454	0.00656	0.00718	0.00562	0.00279
18	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00001	0.00004	0.00015	0.00048	0.00126	0.00271	0.00466	0.00629	0.00647	0.00476
19	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00001	0.00006	0.00020	0.00060	0.00147	0.00293	0.00471	0.00598	0.00579
20	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00002	0.00008	0.00027	0.00073	0.00167	0.00311	0.00470	0.00563

Appendix B. Proof of Lemma 1

We have the following properties when a and b are positive integers and $0 \leq x \leq 1$ ([79], Equations 8.17.20 and 8.17.21):

$$I_x(a,b) - I_x(a+1,b) = \binom{a+b-1}{a} x^a (1-x)^b; \tag{A1}$$

$$I_x(a,b+1) - I_x(a,b) = \binom{a+b-1}{b} x^a (1-x)^b. \tag{A2}$$

Proof of Lemma 1(a). It is trivial for $i = 0$. For $i > 0$, recall the recursive formula of binomial coefficients ([79], Equation 1.2.7):

$$\binom{t+1}{i} = \binom{t}{i-1} + \binom{t}{i}, i = 1, 2, \ldots, t.$$

Applying the formula, we have

$$\begin{aligned}
&B_p(t+1,i) \\
&= \binom{t+1}{i}(1-p)^i p^{t+1-i} \\
&= (1-p)\binom{t}{i-1}(1-p)^{i-1}p^{t-(i-1)} + p\binom{t}{i}(1-p)^i p^{t-i} \\
&= (1-p)B_p(t,i-1) + pB_p(t,i).
\end{aligned}$$ □

Proof of Lemma 1(b). Case I: $t < r$. By (10) and (12), $\beta_p(t+1,r) \leq 1 = \beta_p(t,r)$, and the equality holds if and only if $t+1 \leq r-1 < r$.

Case II: $t \geq r > 0$. By (6) and (A1),

$$\begin{aligned}
&\beta_p(t,r) - \beta_p(t+1,r) \\
&= I_p(t-r+1,r) - I_p(t-r+2,r) \\
&= \binom{t}{t-r+1} p^{t-r+1}(1-p)^r \\
&> 0.
\end{aligned}$$

Case III: $t \geq r = 0$. By (11), the equality always hold. □

Proof of Lemma 1(c). Case I: $t < r$. By (12), the equality always hold.

Case II: $t \geq r > 0$. By (6) and (A2),

$$\begin{aligned}
&\beta_p(t+1,r+1) - \beta_p(t,r) \\
&= I_p(t-r+1,r+1) - I_p(t-r+1,r) \\
&= \binom{t}{r} p^{t-r+1}(1-p)^r \\
&> 0.
\end{aligned}$$

Case III: $t \geq r = 0$. By (10) and (11), $\beta_p(t,r) = 0 < \beta_p(t+1,r+1)$. □

Proof of Lemma 1(d). Case I: $t = -1$. By definition, $\beta_p(t,r+1) = \beta_p(t,r) = 1$.

Case II: $t \geq 0$. Recall that $\beta_p(t,r)$ is the partial sum of the probability mass of the binomial distribution $\mathsf{Binom}(t, 1-p)$. By summing one more term, i.e., $\beta_p(t,r+1)$, the partial sum must be larger than or equal to $\beta_p(t,r)$. Note that $B_p(t,i) \neq 0$ when $0 \leq i \leq t$, so the equality holds if and only if $\beta_p(t,r) = 1$ and if $t < r$ by (12). □

Proof of Lemma 1(e) and (f). Inductively by Lemma 1(b), we have

$$\beta_p(t_a + u, r_a) \leq \beta_p(t_a, r_a) \leq \beta_p(t_a - v, r_a) \tag{A3}$$

for all $a \in \Lambda$ where u, v are non-negative integers such that $t_a - v \geq -1$. By (10),

$$0 \leq \min_{b\in\Lambda} \beta_p(t_b, r_b) \leq \beta_p(t_a, r_a) \leq \max_{b\in\Lambda} \beta_p(t_b, r_b) \leq 1 \tag{A4}$$

for all $a \in \Lambda$. Combining (A3) and (A4), the proof is complete. □

Appendix C. Proof of Lemma 3

By Lemma 3(a), we have $E(r, t+1) = E(r, t) + (1-p)\beta_p(t, r)$. If $t < r$, we have

$$\beta_p(t, r) = \sum_{i=0}^{r-1} B_p(t, i) = 1,$$

which proves Lemma 3(a).

For Lemma 3(b), note that we have the initial condition

$$E(r, 0) = B_p(0, 0) \min\{r, 0\} = 0 = (1-p) \sum_{j=0}^{(0)-1} \beta_p(j, r).$$

We can evaluate Lemma 2 recursively and obtain the first equality in Lemma 3(b).

By Lemma 3(a), we can show that when $t < r$, we have

$$E(r, t) = t(1-p). \tag{A5}$$

This implies that when $t \geq r$, we have

$$E(r, t) = (1-p)\left(r + \sum_{j=r}^{t-1} \beta_p(j, r)\right). \tag{A6}$$

When $t < r$, the summation term $\sum_{j=r}^{t-1} \beta_p(j, r)$ in (A6) equals 0. So, we can combine (A5) and (A6) and give

$$E(r, t) = (1-p)\left(\min\{r, t\} + \sum_{j=r}^{t-1} \beta_p(j, r)\right).$$

Appendix D. Proof of Theorem 2

Suppose $t_m > t_n$ for some $r_m < r_n$, i.e.,

$$t_m > t_n \geq r_n > r_m. \tag{A7}$$

We define

$$t'_b = \begin{cases} t_m & \text{if } b = n, \\ t_n & \text{if } b = m, \\ t_b & \text{otherwise} \end{cases}$$

for all $b \in \mathcal{L}$. We consider the difference of

$$\begin{aligned} &\sum_{b\in\mathcal{L}} E(r_b, t'_b) - \sum_{b\in\mathcal{L}} E(r_b, t_b) \\ &= [E(r_m, t_n) + E(r_n, t_m)] - [E(r_m, t_m) + E(r_n, t_n)] \\ &= [E(r_n, t_m) - E(r_n, t_n)] + [E(r_m, t_n) - E(r_m, t_m)] \\ &= (1-p)\left[\left(\sum_{j=0}^{t_m-1} \beta_p(j, r_n) - \sum_{j=0}^{t_n-1} \beta_p(j, r_n)\right) + \left(\sum_{j=0}^{t_n-1} \beta_p(j, r_m) - \sum_{j=0}^{t_m-1} \beta_p(j, r_m)\right)\right] \end{aligned} \tag{A8}$$

$$
\begin{aligned}
&= (1-p)\left[\sum_{j=t_n}^{t_m-1}\beta_p(j,r_n) - \sum_{j=t_n}^{t_m-1}\beta_p(j,r_m)\right] \\
&= (1-p)\sum_{j=t_n}^{t_m-1}\left(\beta_p(j,r_n) - \beta_p(j,r_m)\right) \\
&> 0, \qquad \text{(A9)}
\end{aligned}
$$

where

- (A8) follows Lemma 3(b);
- (A9) follows Lemma 1(d) together with (A7).

The above result contradicts that $\{t_b\}_{b\in\mathcal{L}}$ solves (8), which gives us that $t_m \le t_n$ for all $r_m < r_n$.

Next, we suppose $t_m = t_n$ for some $r_m < r_n$, i.e.,

$$t_m = t_n \ge r_n > r_m. \qquad \text{(A10)}$$

We define

$$t''_b = \begin{cases} t_n + 1 & \text{if } b = n, \\ t_m - 1 & \text{if } b = m, \\ t_b & \text{otherwise} \end{cases}$$

for all $b \in \mathcal{L}$. Moreover, we compare the difference of

$$
\begin{aligned}
&\sum_{b\in\mathcal{L}} E(r_i, t''_i) - \sum_{b\in\mathcal{L}} E(r_i, t_i) \\
&= [E(r_n, t_n+1) + E(r_m, t_m-1)] - [E(r_n,t_n) + E(r_m,t_m)] \\
&= [E(r_n, t_n+1) - E(r_n,t_n)] - [E(r_m,t_m) - E(r_m,t_m-1)] \\
&= (1-p)[\beta_p(t_n, r_n) - \beta_p(t_m-1, r_m)] \qquad \text{(A11)} \\
&\ge (1-p)[\beta_p(t_m, r_m+1) - \beta_p(t_m-1, r_m)] \qquad \text{(A12)} \\
&> 0, \qquad \text{(A13)}
\end{aligned}
$$

where

- (A11) follows Lemma 3(a)
- (A12) follows (A10) and Lemma 1(d)
- (A13) follows Lemma 1(c) together with (A10).

This contradicts $\{t_b\}_{b\in\mathcal{L}}$ and solves (8). Therefore, we have $t_m \ne t_n$ for all $r_m < r_n$. Combining the two cases, the proof is complete.

Appendix E. Performance Guarantee and Bounded Error of Algorithm 2

We start the discussion with the following theorem.

Theorem A1. *Let SOL and OPT be the solution given by Algorithm 2 and the optimal solution of (8), respectively, then*

$$
\begin{cases}
SOL \ge (1-p)OPT, \\
OPT - SOL \le (1-p)\sum_{b\in\mathfrak{L}} \sum_{j=r_b+\ell}^{r_b+|\mathfrak{L}|\ell'-1} \beta_p(j, r_b),
\end{cases}
$$

where $\ell' = (t^{\mathcal{L}}_{max} - \sum_{b\in\mathcal{L}} r_b)/|\mathfrak{L}|$ *and* $\ell = \lfloor \ell' \rfloor$.

Proof. We first show that the algorithm has a relative performance guarantee factor of $1-p$. As stated in Theorem 3, when $t^{\mathcal{L}}_{\max} \le \sum_{b\in\mathcal{L}} r_b$, the algorithm guarantees an optimal

solution. Therefore, we only consider $t^{\mathcal{L}}_{\max} > \sum_{b\in\mathcal{L}} r_b$. Let $\{t_b\}_{b\in\mathcal{L}}$ be the approximation given by the algorithm.

Note that any linear combinations of r-independent vectors cannot obtain more than r-independent vectors. Therefore, the expected rank of a batch at the next hop must be no larger than the rank of the batch at the current hop, and, be non-negative. That is,

$$0 \le E(r_b, t) \le r_b, \forall t \ge 0, b \in \mathcal{L}. \tag{A14}$$

This gives a bound of the optimal solution by

$$0 \le \mathrm{OPT} \le \sum_{b\in\mathcal{L}} r_b. \tag{A15}$$

We consider the exact formula of the approximation:

$$\begin{aligned}
\mathrm{SOL} &= (1-p)\sum_{b\in\mathcal{L}} r_b + (1-p)\sum_{b\in\mathcal{L}}\sum_{j=r_b}^{t_b-1}\beta_p(j, r_b) && \text{(A16)}\\
&\ge (1-p)\sum_{b\in\mathcal{L}} r_b && \text{(A17)}\\
&\ge (1-p)\mathrm{OPT}, && \text{(A18)}
\end{aligned}$$

where

- (A16) is stated in Lemma 3(b)
- (A17) holds as $\beta_p(j, r_b) \ge 0$ for all j, r_b, which is by (10);
- (A18) follows the inequality (A15).

Lastly, we show the bounded error. Let $\{t^*_b\}$ be a solution to (8). We write $t^*_b = r_b + \ell_b$ where $\ell_b \ge 0$ for all $b \in \mathcal{L}$. Note that the constraint of (8), i.e., $\sum_{b\in\mathcal{L}} t^*_b = t^{\mathcal{L}}_{\max}$, suggests that

$$\ell_b \le t^{\mathcal{L}}_{\max} - \sum_{b\in\mathcal{L}} r_b = |\mathfrak{L}|\ell'. \tag{A19}$$

On the other hand, it is easy to see that the approximation must either give $t_b = r_b + \ell$ or $t_b = r_b + \ell + 1$. That is, we have $t_b \ge r_b + \ell$. By Lemma 3(b),we have

$$\mathrm{SOL} \ge (1-p)\sum_{b\in\mathfrak{L}}\left[r_b + \sum_{j=r_b}^{r_b+\ell-1}\beta_p(j, r_b)\right]. \tag{A20}$$

We consider the difference between OPT and SOL:

$$\begin{aligned}
&\mathrm{OPT} - \mathrm{SOL}\\
&\le (1-p)\sum_{b\in\mathfrak{L}}\left(\sum_{j=r_b}^{r_b+\ell_b-1}\beta_p(j, r_b) - \sum_{j=r_b}^{r_b+\ell-1}\beta_p(j, r_b)\right) && \text{(A21)}\\
&= (1-p)\sum_{b\in\mathfrak{L}}\left(\sum_{\substack{j=r_b+\ell,\\ \ell_b>\ell}}^{r_b+\ell_b-1}\beta_p(j, r_b) - \sum_{\substack{j=r_b+\ell_b,\\ \ell_b<\ell}}^{r_b+\ell-1}\beta_p(j, r_b)\right)\\
&\le (1-p)\sum_{b\in\mathfrak{L}}\sum_{\substack{j=r_b+\ell,\\ \ell_b>\ell}}^{r_b+\ell_b-1}\beta_p(j, r_b) && \text{(A22)}\\
&\le (1-p)\sum_{b\in\mathfrak{L}}\sum_{j=r_b+\ell}^{r_b+|\mathfrak{L}|\ell'-1}\beta_p(j, r_b), && \text{(A23)}
\end{aligned}$$

where

- (A21) is the difference between the exact form of OPT by Lemma 3(b) after substituting the lower bound of SOL shown in (A20);
- the condition $\ell_b > \ell$ in the summation of (A22) can be removed, as we have $r_b + \ell_b - 1 < r_b + \ell$ if $\ell_b \leq \ell$;
- (A23) follows (A19) and the fact shown in (10) that the extra $\beta_p(j, r_b)$ terms are non-negative.

The proof is done. □

If the relative performance guarantee factor of $1 - p$ is tight, we need both equalities in (A17) and (A18) to hold. First, by (10), we know that $\beta_p(j, r_b)$ is always non-negative. The equality in (A17) holds if and only if $\sum_{j=r_b}^{t_b-1} \beta_p(j, r_b) = 0$ for all $b \in \mathcal{L}$. The sum equals 0 only when

- $r_b = 0$ and $t_b \geq 0$ according to (11); or
- $t_b - 1 < r_b$ which forms an empty sum.

The equality in (A18) holds if and only if $\text{OPT} = \sum_{b\in\mathcal{L}} E(r_b, t_b^*) = \sum_{b\in\mathcal{L}} r_b$. Note that (A14) shows that $E(r_b, t_b^*)$ is upper bounded by r_b. This implies that we need $E(r_b, t_b^*) = r_b$ for all $b \in \mathcal{L}$. When $t_b^* \leq r_b$, we can apply Lemma 3(a) to obtain $E(r_b, t_b^*) = (1 - p)t_b^*$, which equals r_b if and only if $r_b = 0$, as we assumed $0 < p < 1$ in this paper. By Lemma 2, $E(r_b, t)$ is a monotonic increasing function in terms of t for all $r_b \geq 0$. Therefore, when $r_b \neq 0$ we need $t_b^* > r_b$, which implies that $t_{\max}^{\mathcal{L}} > \sum_{b\in\mathcal{L}} r_b$. Then, the approximation will also give $t_b > r_b$ for some $b \in \mathcal{L}$ in this case, and the equality in (A17) does not hold.

That is, we have $\text{SOL} = (1 - p)\text{OPT}$ only when $r_b = 0$ for all $b \in \mathcal{L}$. In this case, we have $\text{SOL} = \text{OPT} = 0$. In practice, the probability of having $r_b = 0$ for all $b \in \mathcal{L}$ is very small. Therefore, we can consider that the bound is not tight in most cases but it guarantees that the approximation is good when the packet loss probability is small.

Appendix F. Proof of Theorem 5

Let $\mathbb{B}(a, b; y) := \int_0^y x^{a-1}(1 - x)^{b-1}\, dx$ be the incomplete beta function. We have the beta function $\mathbb{B}(a, b) := \mathbb{B}(a, b; 1)$.

From (6), we have $\beta_p(t, r) = I_p(t - r + 1, r) = \frac{\mathbb{B}(t-r+1,r;p)}{\mathbb{B}(t-r+1,r;1)}$. By direct calculation, the condition number is

$$\begin{aligned}
\left|\frac{p\frac{d\beta_p(t,r)}{dp}}{\beta_p(t,r)}\right| &= \left|\frac{p\frac{d}{dp}\int_0^p x^{t-r}(1-x)^{r-1}\,dx}{\mathbb{B}(t-r+1,r;1)I_p(t-r+1,r)}\right| \\
&= \frac{p^{t-r+1}(1-p)^{r-1}}{\mathbb{B}(t-r+1,r;1)I_p(t-r+1,r)} \\
&= \frac{p^{t-r+1}(1-p)^{r-1}}{\int_0^p x^{t-r}(1-x)^{r-1}\,dx} \\
&= \frac{p^{t-r+1}\sum_{j=0}^{r-1}(-1)^j\binom{r-1}{j}p^j}{\int_0^p x^{t-r}\sum_{j=0}^{r-1}(-1)^j\binom{r-1}{j}x^j\,dx} \\
&= \frac{\sum_{j=0}^{r-1}(-1)^j\binom{r-1}{j}p^{t-r+j+1}}{\sum_{j=0}^{r-1}(-1)^j\binom{r-1}{j}\int_0^p x^{t-r+j}\,dx} \\
&= \frac{\sum_{j=0}^{r-1}(-1)^j\binom{r-1}{j}p^{t-r+j+1}}{\sum_{j=0}^{r-1}(-1)^j\binom{r-1}{j}p^{t-r+j+1}/(t-r+j+1)},
\end{aligned} \tag{A24}$$

where the absolute value disappears as both numerator and denominator are non-negative. The first form of the condition number can be obtained by substituting $\mathbb{B}(t - r + 1, r; 1) = \frac{(t-r)!(r-1)!}{t!}$ into (A24).

Appendix G. Corrupted Heaps

In this appendix, we explain why we can omit two of the heap updates in Algorithm 3.

Before we start, we need some mathematical descriptions of the optimal solutions of BAR. Then, we will introduce our lazy evaluation technique on a modified heap that we called the corrupted heap.

To simplify the notations, we redefine $\beta_p(t_b, r_b)$ as

$$\beta_p(t,r) = \begin{cases} 0 & \text{if } t_b \geq t^b_{\max}, \\ 1 & \text{if } t_b \leq \min\{r_b, t^b_{\max}\} - 1, \\ \sum_{i=0}^{r_b-1} \binom{t_b}{i}(1-p)^i p^{t_b-i} & \text{otherwise.} \end{cases}$$

in this appendix. In other words, $\beta_p(\cdot,\cdot)$ is now a function of b. When $t_b \geq t^b_{\max}$, $\beta_p(t_b, r_b)$ is the smallest possible value in the image of $\beta_p(\cdot,\cdot)$. Therefore, the algorithms in this paper will not assign more recoded packets to the batch b.

Appendix G.1. Optimality Properties of BAR

First, we introduce the following theorem that states a condition for non-optimality (or optimality after taking contraposition).

Theorem A2. *Let $\{t_b\}_{b\in\mathcal{L}}$ be a feasible solution of (9). Then, $\{t_b\}_{b\in\mathcal{L}}$ is not an optimal solution of (9) if and only if there exists two distinct batches κ and ρ with $t_\rho \geq 1$ such that $(1-p)\beta_p(t_\kappa, r_\kappa) > (1-p)\beta_p(t_\rho - 1, r_\rho)$.*

Proof. We first prove the sufficient condition. If $\{t_b\}_{b\in\mathcal{L}}$ does not solve (9), then it means that there exists another configuration $\{t'_b\}_{b\in\mathcal{L}}$ which can give a higher objective value. Since $\sum_{b\in\mathcal{L}} t_b = \sum_{b\in\mathcal{L}} t'_b = t^{\mathcal{L}}_{\max}$, there exists distinct $\kappa, \rho \in \mathcal{L}$ such that $t'_\kappa > t_\kappa$ and $t'_\rho < t_\rho$. Note that $t'_\rho \geq 0$ so we must have $t_\rho \geq 1$. We define

$$\Theta = \{\kappa \colon t'_\kappa > t_\kappa\} \quad \text{and} \quad \Phi = \{\rho \colon t'_\rho < t_\rho\},$$

where

$$\sum_{\theta\in\Theta} (t'_\theta - t_\theta) = \sum_{\phi\in\Phi} (t_\phi - t'_\phi) > 0. \tag{A25}$$

Using the fact that $\{t'_b\}_{b\in\mathcal{L}}$ gives a larger objective value and by Lemma 3(b),we have

$$\sum_{\theta\in\Theta} \sum_{t=t_\theta}^{t'_\theta-1} (1-p)\beta_p(t, r_\theta) > \sum_{\phi\in\Phi} \sum_{t=t'_\phi}^{t_\phi-1} (1-p)\beta_p(t, r_\phi). \tag{A26}$$

Now, we fix κ and ρ such that

$$\kappa \in \arg\max_{\theta\in\Theta} \beta_p(t_\theta, r_\theta) \quad \text{and} \quad \rho \in \arg\min_{\phi\in\Phi} \beta_p(t_\phi - 1, r_\phi).$$

We have

$$\begin{aligned} &\sum_{\theta\in\Theta} (t'_\theta - t_\theta)(1-p)\beta_p(t_\kappa, r_\kappa) \\ &\geq \sum_{\theta\in\Theta} (t'_\theta - t_\theta)(1-p)\beta_p(t_\theta, r_\theta) \\ &\geq \sum_{\theta\in\Theta} \sum_{t=t_\theta}^{t'_\theta-1} (1-p)\beta_p(t, r_\theta) \end{aligned} \tag{A27}$$

$$> \sum_{\phi\in\Phi}\sum_{t=t'_\phi}^{t_\phi-1}(1-p)\beta_p(t,r_\phi) \tag{A28}$$
$$\geq \sum_{\phi\in\Phi}(t_\phi-t'_\phi)(1-p)\beta_p(t_\phi-1,r_\phi) \tag{A29}$$
$$\geq \sum_{\phi\in\Phi}(t_\phi-t'_\phi)(1-p)\beta_p(t_\rho-1,r_\rho),$$

where

- (A27) and (A29) follows Lemma 1(b);
- (A28) is the inequality shown in (A26).

Applying (A25), we have

$$(1-p)\beta_p(t_\kappa,r_\kappa) > (1-p)\beta_p(t_\rho-1,r_\rho),$$

which proves the sufficient condition.

Now we consider the necessary condition, where we have

$$(1-p)\beta_p(t_\kappa,r_\kappa) > (1-p)\beta_p(t_\rho-1,r_\rho) \tag{A30}$$

for some distinct κ and ρ. Let

$$t'_b = \begin{cases} t_\kappa+1 & \text{if } b=\kappa, \\ t_\rho-1 & \text{if } b=\rho, \\ t_b & \text{otherwise} \end{cases}$$

for all $b\in\mathcal{L}$. Then, we consider the following:

$$\begin{aligned}
&\sum_{b\in\mathcal{L}}E(r_b,t'_b)\\
&=\sum_{b\in\mathcal{L}\setminus\{\kappa,\rho\}}E(r_b,t_b)+E(r_\kappa,t_\kappa+1)+E(r_\rho,t_\rho-1)\\
&=\sum_{b\in\mathcal{L}\setminus\{\kappa,\rho\}}E(r_b,t_b)+E(r_\rho,t_\rho-1)+[E(r_\kappa,t_\kappa)+(1-p)\beta_p(t_\kappa,r_\kappa)] &\text{(A31)}\\
&>\sum_{b\in\mathcal{L}\setminus\{\kappa,\rho\}}E(r_b,t_b)+E(r_\kappa,t_\kappa)+[E(r_\rho,t_\rho-1)+(1-p)\beta_p(t_\rho-1,r_\rho)] &\text{(A32)}\\
&=\sum_{b\in\mathcal{L}\setminus\{\rho\}}E(r_b,t_b)+E(r_\rho,t_\rho) &\text{(A33)}\\
&=\sum_{b\in\mathcal{L}}E(r_b,t_b),
\end{aligned}$$

where

- (A31) and (A33) follow Lemma 3(a)
- (A32) follows (A30).

meaning that $\{t_b\}_{b\in\mathcal{L}}$ is not an optimal solution of (9). □

Next, we define the sub-problems (A34) of (9) for $k\in\{0,1,\ldots,t^{\mathcal{L}}_{\max}\}$, which present the optimal substructure of (9).

$$\begin{aligned}
\max_{t_b\in\{0,1,2,\ldots\},\forall b\in\mathcal{L}}\quad & \sum_{b\in\mathcal{L}}E(r_b,t_b)\\
\text{s.t.}\quad & \sum_{b\in\mathcal{L}}t_b=k \\
& t_b\leq t^b_{\max},\forall b\in\mathcal{L}.
\end{aligned} \tag{A34}$$

Note that we assume $\sum_{b\in\mathcal{L}} t^b_{\max} \geq t^{\mathcal{L}}_{\max} \geq k$, or otherwise we should include more batches in the block.

We define a multiset Ω_r that collects the value of $(1-p)\beta_p(t,r)$ for all integers $t \geq 0$, i.e.,

$$\Omega_r = \{(1-p)\beta_p(t,r)\colon t \in \{0,1,2,\ldots\}\}.$$

By Lemma 3(b),we have $E(r,t) = (1-p)\sum_{j=0}^{t-1}\beta_p(j,r)$. As $E(r,t)$ is concave with respect to t, $\beta_p(t,r)$ is a monotonic decreasing function on t (also stated in Lemma 1(b)).

This implies the following lemma.

Lemma A1. *$E(r,t)$ equals the sum of the largest t elements in Ω_r.*

Proof. By Lemma 3(b), $E(r,t) = (1-p)\sum_{j=0}^{t-1}\beta_p(j,r)$. By Lemma 1(b), $\beta_p(t,r)$ is a monotonic decreasing function on t. By (10) and (12), we have $\beta_p(0,r) = 1 \geq \beta_p(t,r)$ for all positive integers t. Therefore, $E(r,t)$ is the sum of the largest t elements in Ω_r. □

We fix a block $\mathcal{L}$ and define a multiset

$$\Omega := \biguplus_{b\in\mathcal{L}} \Omega_{r_b} = \{(1-p)\beta_p(t_b,r_b)\colon t_b \in \{0,1,2,\ldots,t^b_{\max}-1\}, b\in\mathcal{L}\}.$$

If two batches $a,b \in \mathcal{L}$ have the same rank, i.e., $r_a = r_b$, then for all t, we have $(1-p)\beta_p(t,r_a) = (1-p)\beta_p(t,r_b)$. As Ω is a multiset, the duplicated values are not eliminated. Now, we have the following lemma to connect (A34) and Ω.

Lemma A2. *The optimal value of* (A34) *is the sum of the largest k elements in Ω.*

Proof. Let $\{t_b\}_{b\in\mathcal{L}}$ solves (A34). We suppose the optimal value is not the sum of the largest k elements in Ω. However, Lemma A1 states that $E(r_b,t_b)$ equals the sum of the largest t_b elements in Ω_{r_b} for all $b \in \mathcal{L}$. This means that there exists two distinct batches $\kappa,\rho \in \mathcal{L}$ with $t_\kappa \leq t^b_{\max} - 1$ and $t_\rho \geq 1$ such that $(1-p)\beta_p(t_\kappa,r_\kappa) > (1-p)\beta_p(t_\rho-1,r_\rho)$.

By setting $t^{\mathcal{L}}_{\max} = k$, we can apply Theorem A2 which gives that $\{t_b\}_{b\in\mathcal{L}}$ is not an optimal solution of (A34). The proof is completed by contradiction. □

We define a multiset $\Omega'_{t^{\mathcal{L}}_{\max}}$ which is a collection of the largest $t^{\mathcal{L}}_{\max}$ elements in Ω. By Lemma A2, $\sum_{\omega\in\Omega'_{t^{\mathcal{L}}_{\max}}} \omega$ is the optimal value of (9). For any non-optimal solution $(t^{(k)}_b)_{b\in\mathcal{L}}$, we define a multiset

$$\mho_k := \{(1-p)\beta_p(t,r_b)\colon t \in \{0,1,\ldots,t^{(k)}_b - 1\}, b\in\mathcal{L}\},$$

where the value $k \in \{0,1,\ldots,t^{\mathcal{L}}_{\max}-1\}$ is the number of elements in $\Omega'_{t^{\mathcal{L}}_{\max}}$ which are also contained in $\mho_k$.

Appendix G.2. Lazy Evaluations

We consider an iteration in the last **while** loop in Algorithm 3. Suppose we choose to increase t_b by 1 and decrease t_a by 1.

Lemma A3. *If batch a is selected by the max-heap or batch b is selected by the min-heap in any future iteration, then the optimal solution is reached.*

Proof. Suppose batch a with key $\mathcal{A}$ is selected by the max-heap in a future iteration. Note that $\mathcal{A}$ was once the smallest element in $\mho_k$ for some k. Therefore, at the current state where $k' > k$, every element in $\mho_{k'}$ must be no smaller than $\mathcal{A}$. Equivalently, we have $(1-p)\beta_p(t_\kappa,r_\kappa) \leq (1-p)\beta_p(t_\rho-1,r_\rho)$ for all $\kappa,\rho \in \mathcal{L}$. By Theorem A2, the optimal solution is reached. The min-heap counterpart can be proved in a similar fashion. □

Suppose we omit the update for the batch ρ in the heap. We call the key of the batch ρ a corrupted key, or the key of the batch ρ is corrupted. A key which is not corrupted is called an uncorrupted key. A heap with corrupted keys is called a corrupted heap. In other words, the key of a batch is corrupted in a corrupted max-heap if and only if the same batch was once the minimum of the corresponding original min-heap, and vice versa. As a remark, we do not have a guaranteed maximum portion of corrupted keys as an input. Furthermore, we do not adopt the carpooling technique. This suggests that the heap here is not a soft heap [97].

Lemma A4. *If the root of a corrupted heap is a corrupted key, then the optimal solution is reached.*

Proof. We only consider a corrupted max-heap in the proof. We can use similar arguments to show that a corrupted min-heap also works.

In a future iteration, suppose batch a is selected by the corrupted max-heap. We consider the real maximum in the original max-heap. There are three cases.

Case I: batch a is also the root of the original max-heap. As the key of a is corrupted, it means that the batch was once selected by the corresponding min-heap. By Lemma A3, the optimal solution is reached.

Case II: the root of the original max-heap is batch a' where the key of a' is also corrupted. Similar to Case I, batch a' was once selected by the corresponding min-heap, and we can apply Lemma A3 to finish this case.

Case III: the root of the original max-heap is batch a'' where the key of a'' is not corrupted. In this case, the uncorrupted key of a'' is also in the corrupted max-heap. Note that the corrupted key of a is no larger than the actual key of a in the original max-heap. This means that the key of a, a'' and the corrupted key of a have the same value. It is equivalent to let the original max-heap select batch a, as every element in $\mho_{k'}$ must be no smaller than the key of a'', where k' represents the state of the current iteration. Then, the problem is reduced to Case I.

Combining the three cases, the proof is completed. □

Theorem A3. *The updates for batch a in the max-heap and batch b in the min-heap can be omitted.*

Proof. When we omit the updates, the heap itself becomes a corrupted heap. We have to make sure that when a batch with corrupted key is selected, the termination condition of the algorithm is also met.

We can express the key of batch π in a corrupted max-heap and min-heap by $\beta_p(t_\pi + s_\pi, r_\pi)$ and $\beta_p(t_\pi - 1 - u_\pi, r_\pi)$, respectively, where s_π, u_π are non-negative integers. When s_π or u_π is 0, the key is uncorrupted in the corresponding corrupted heap. By Lemma 1(b), we have

$$\begin{aligned}\beta_p(t_\pi + s_\pi, r_\pi) &\leq \beta_p(t_\pi, r_\pi),\\ \beta_p(t_\pi - 1, r_\pi) &\leq \beta_p(t_\pi - 1 - u_\pi, r_\pi).\end{aligned}$$

That is, the root of the corrupted max-heap is no larger than the root of the original max-heap. Similar for the min-heap. Mathematically, we have

$$\max_{\pi\in\mathcal{L}} \beta_p(t_\pi + s_\pi, r_\pi) \leq \max_{\pi\in\mathcal{L}} \beta_p(t_\pi, r_\pi), \tag{A35}$$

$$\min_{\pi\in\mathcal{L}} \beta_p(t_\pi - 1, r_\pi) \leq \min_{\pi\in\mathcal{L}} \beta_p(t_\pi - 1 - u_\pi, r_\pi). \tag{A36}$$

Suppose a corrupted key is selected. By Lemma A4, we know that the optimal solution is reached. Therefore, we can apply the contrapositive of Theorem A2 and know that

$$(1-p)\beta_p(t_\kappa, r_\kappa) \leq (1-p)\beta_p(t_\rho - 1, r_\rho) \tag{A37}$$

for all $\kappa, \rho \in \mathcal{L}$. We can omit the condition $t_\rho \geq 1$ because by (10) and (12), we have $\beta_p(-1, \cdot) = 1 \geq \beta_p(\cdot, \cdot)$. The inequality (A37) is equivalent to

$$\max_{\pi \in \mathcal{L}} \beta_p(t_\pi, r_\pi) \leq \min_{\pi \in \mathcal{L}} \beta_p(t_\pi - 1, r_\pi).$$

We can mix this inequality with (A35) and (A36) to show that when a corrupted key is selected, we have

$$\max_{\pi \in \mathcal{L}} \beta_p(t_\pi + s_\pi, r_\pi) \leq \min_{\pi \in \mathcal{L}} \beta_p(t_\pi - 1 - u_\pi, r_\pi),$$

which is the termination condition shown in Algorithm 3 after we replaced the heaps into corrupted heaps.

We just showed that once a corrupted key selected, the termination condition is reached. In the other words, before a corrupted key is selected, every previous selection must be an uncorrupted key. That is, the details inside the iterations are unaffected. If an uncorrupted key is selected where it also satisfies the termination condition, then no corrupted key is touched, and the corrupted heap still acts as a normal heap at this point.

The correctness of the algorithm when using a corrupted heap is proven. Moreover, we do not need to mark which key is corrupted. This is, we can omitted the mentioned heap updates for a normal heap. □

We do not need to mark down which key is corrupted while the algorithm still works, so we can simply omit the mentioned updates as lazy evaluations. As there are two heaps in algorithm, we can reduce from four to two heap updates.

Appendix H. Linear Programming Formulation of BAR

In [81], a distributionally robust optimization [98] for AR is formulated as a linear programming problem. It is based on an observation that when the expected rank function $E(r, t)$ is concave with respect to t, we can reformulate it by

$$E(r, t) = \min_{i \in \{0, 1, \ldots, \bar{\imath}\}} (\Delta_{r,i} t + \xi_{r,i})$$

if we fix an artificial upper bound $t \leq \bar{\imath}$, where $\Delta_{r,t} := E(r, i+1) - E(r, i)$ and $\xi_{r,i} := E(r, i) - i\Delta_{r,i}$. In (9), we implicitly have $t \leq t^{\mathcal{L}}_{\max}$, so we can make use of this expression to write (9) as

$$\begin{aligned} \max_{t_b, e_b \geq 0, \forall b \in \mathcal{L}} \quad & \sum_{b \in \mathcal{L}} e_b \\ \text{s.t.} \quad & \sum_{b \in \mathcal{L}} t_b = t^{\mathcal{L}}_{\max} \\ & t_b \leq t^b_{\max}, \forall b \in \mathcal{L} \\ & e_b \leq E(r_b, i) + (E(r_b, i+1) - E(r_b, i))(t_b - i), \forall b \in \mathcal{L}, \forall i \in \{0, 1, \ldots, t^{\mathcal{L}}_{\max}\}, \end{aligned}$$

where t_b is allowed to be a non-integer. A non-integer t_b means that we first generate $\lfloor t_b \rfloor$ recoded packets, then we generate one more recoded packet with probability $t_b - \lfloor t_b \rfloor$. Note that there are $|\mathcal{L}| t^{\mathcal{L}}_{\max}$ constraints for e_b.

To turn such a non-deterministic solution into a deterministic one, we perform the following steps:

1. Collect the batches with non-integer recoded packets into a set S.
2. Calculate $R = \sum_{b \in S} (t_b - \lfloor t_b \rfloor)$. Note that R is an integer for BAR.
3. For every $b \in S$, remove the fractional part of t_b.
4. Randomly select R batches from S and add one recoded packet to each of these batches.

We have an integer R because $\sum_{b \in \mathcal{L}} t_b = t^{\mathcal{L}}_{\max}$. Furthermore, we have $R < |S|$. Referring to the idea of Algorithm 1, we have the same value of $\Delta_{r_b, \lfloor t_b \rfloor}$ for all $b \in S$. After

removing the fractional part of t_b for all $b \in S$, it becomes the sub-problem (A34) (defined in Appendix G.1) with $k = t^{\mathcal{L}}_{\max} - R$. The last step follows Algorithm 1 such that the output is a solution to (9) where t_b for all $b \in \mathcal{L}$ are all integers.

References

1. Luby, M. LT Codes. In Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, Vancouver, BC, Canada, 19 November 2002; pp. 271–282.
2. Shokrollahi, A. Raptor Codes. *IEEE Trans. Inf. Theory* **2006**, *52*, 2551–2567. [CrossRef]
3. Maymounkov, P. *Online Codes*; Technical Report; New York University: New York, NY, USA, 2002.
4. Ho, T.; Koetter, R.; Médard, M.; Karger, D.R.; Effros, M. The Benefits of Coding over Routing in a Randomized Setting. In Proceedings of the 2003 IEEE International Symposium on Information Theory (ISIT), Yokohama, Japan, 29 June–4 July 2003; p. 442.
5. Ho, T.; Médard, M.; Koetter, R.; Karger, D.R.; Effros, M.; Shi, J.; Leong, B. A Random Linear Network Coding Approach to Multicast. *IEEE Trans. Inf. Theory* **2006**, *52*, 4413–4430. [CrossRef]
6. Ahlswede, R.; Cai, N.; Li, S.Y.R.; Yeung, R.W. Network Information Flow. *IEEE Trans. Inf. Theory* **2000**, *46*, 1204–1216.
7. Koetter, R.; Médard, M. An Algebraic Approach to Network Coding. *IEEE/ACM Trans. Netw.* **2003**, *11*, 782–795. [CrossRef]
8. Li, S.Y.R.; Yeung, R.W.; Cai, N. Linear Network Coding. *IEEE Trans. Inf. Theory* **2003**, *49*, 371–381. [CrossRef]
9. Wu, Y. A Trellis Connectivity Analysis of Random Linear Network Coding with Buffering. In Proceedings of the 2006 IEEE International Symposium on Information Theory (ISIT), Seattle, WA, USA, 9–14 July 2006; pp. 768–772.
10. Lun, D.S.; Médard, M.; Koetter, R.; Effros, M. On coding for reliable communication over packet networks. *Phys. Commun.* **2008**, *1*, 3–20. [CrossRef]
11. Chou, P.A.; Wu, Y.; Jain, K. Practical Network Coding. In Proceedings of the Annual Allerton Conference on Communication Control and Computing, Monticello, IL, USA, 1–3 October 2003; Volume 41, pp. 40–49.
12. Pandi, S.; Gabriel, F.; Cabrera, J.A.; Wunderlich, S.; Reisslein, M.; Fitzek, F.H.P. PACE: Redundancy Engineering in RLNC for Low-Latency Communication. *IEEE Access* **2017**, *5*, 20477–20493. [CrossRef]
13. Wunderlich, S.; Gabriel, F.; Pandi, S.; Fitzek, F.H.P.; Reisslein, M. Caterpillar RLNC (CRLNC): A Practical Finite Sliding Window RLNC Approach. *IEEE Access* **2017**, *5*, 20183–20197. [CrossRef]
14. Lucani, D.E.; Pedersen, M.V.; Ruano, D.; Sørensen, C.W.; Fitzek, F.H.P.; Heide, J.; Geil, O.; Nguyen, V.; Reisslein, M. Fulcrum: Flexible Network Coding for Heterogeneous Devices. *IEEE Access* **2018**, *6*, 77890–77910. [CrossRef]
15. Nguyen, V.; Tasdemir, E.; Nguyen, G.T.; Lucani, D.E.; Fitzek, F.H.P.; Reisslein, M. DSEP Fulcrum: Dynamic Sparsity and Expansion Packets for Fulcrum Network Coding. *IEEE Access* **2020**, *8*, 78293–78314. [CrossRef]
16. Tasdemir, E.; Tömösközi, M.; Cabrera, J.A.; Gabriel, F.; You, D.; Fitzek, F.H.P.; Reisslein, M. SpaRec: Sparse Systematic RLNC Recoding in Multi-Hop Networks. *IEEE Access* **2021**, *9*, 168567–168586. [CrossRef]
17. Tasdemir, E.; Nguyen, V.; Nguyen, G.T.; Fitzek, F.H.P.; Reisslein, M. FSW: Fulcrum sliding window coding for low-latency communication. *IEEE Access* **2022**, *10*, 54276–54290. [CrossRef]
18. Fu, A.; Sadeghi, P.; Médard, M. Dynamic rate adaptation for improved throughput and delay in wireless network coded broadcast. *IEEE/ACM Trans. Netw.* **2014**, *22*, 1715–1728. [CrossRef]
19. Chatzigeorgiou, I.; Tassi, A. Decoding delay performance of random linear network coding for broadcast. *IEEE Trans. Veh. Technol.* **2017**, *66*, 7050–7060. [CrossRef]
20. Yazdani, N.; Lucani, D.E. Revolving codes: Overhead and computational complexity analysis. *IEEE Commun. Lett.* **2021**, *25*, 374–378. [CrossRef]
21. Torres Compta, P.; Fitzek, F.H.P.; Lucani, D.E. Network Coding is the 5G Key Enabling Technology: Effects and Strategies to Manage Heterogeneous Packet Lengths. *Trans. Emerg. Telecommun. Technol.* **2015**, *6*, 46–55. [CrossRef]
22. Torres Compta, P.; Fitzek, F.H.P.; Lucani, D.E. On the Effects of Heterogeneous Packet Lengths on Network Coding. In Proceedings of the European Wireless 2014, Barcelona, Spain, 14–16 May 2014; pp. 385–390.
23. Taghouti, M.; Lucani, D.E.; Cabrera, J.A.; Reisslein, M.; Pedersen, M.V.; Fitzek, F.H.P. Reduction of Padding Overhead for RLNC Media Distribution with Variable Size Packets. *IEEE Trans. Broadcast.* **2019**, *65*, 558–576. [CrossRef]
24. Taghouti, M.; Tömösközi, M.; Howeler, M.; Lucani, D.E.; Fitzek, F.H.; Bouallegue, A.; Ekler, P. Implementation of Network Coding with Recoding for Unequal-Sized and Header Compressed Traffic. In Proceedings of the 2019 IEEE Wireless Communications and Networking Conference (WCNC), Marrakech, Morocco, 15–19 April 2019.
25. Schütz, B.; Aschenbruck, N. Packet-Preserving Network Coding Schemes for Padding Overhead Reduction. In Proceedings of the 2019 IEEE 44th Conference on Local Computer Networks (LCN), Osnabrueck, Germany, 14–17 October 2019; pp. 447–454.
26. de Alwis, C.; Kodikara Arachchi, H.; Fernando, A.; Kondoz, A. Towards Minimising the Coefficient Vector Overhead in Random Linear Network Coding. In Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada, 26–31 May 2013; pp. 5127–5131.
27. Silva, D. Minimum-Overhead Network Coding in the Short Packet Regime. In Proceedings of the 2012 International Symposium on Network Coding (NetCod), Boston, MA, USA, 29–30 June 2012; pp. 173–178.
28. Gligoroski, D.; Kralevska, K.; Øverby, H. Minimal Header Overhead for Random Linear Network Coding. In Proceedings of the 2015 IEEE International Conference on Communication Workshop (ICCW), London, UK, 8–12 June 2015; pp. 680–685.

29. Silva, D.; Zeng, W.; Kschischang, F.R. Sparse Network Coding with Overlapping Classes. In Proceedings of the 2009 Workshop on Network Coding, Theory, and Applications, Lausanne, Switzerland, 15–16 June 2009; pp. 74–79.
30. Heidarzadeh, A.; Banihashemi, A.H. Overlapped Chunked Network Coding. In Proceedings of the 2010 IEEE Information Theory Workshop on Information Theory (ITW), Dublin, Ireland, 30 August–3 September 2010; pp. 1–5.
31. Li, Y.; Soljanin, E.; Spasojevic, P. Effects of the Generation Size and Overlap on Throughput and Complexity in Randomized Linear Network Coding. *IEEE Trans. Inf. Theory* **2011**, *57*, 1111–1123. [CrossRef]
32. Tang, B.; Yang, S.; Yin, Y.; Ye, B.; Lu, S. Expander graph based overlapped chunked codes. In Proceedings of the 2012 IEEE International Symposium on Information Theory (ISIT), Cambridge, MA, USA, 1–6 July 2012; pp. 2451–2455.
33. Mahdaviani, K.; Ardakani, M.; Bagheri, H.; Tellambura, C. Gamma Codes: A Low-Overhead Linear-Complexity Network Coding Solution. In Proceedings of the 2012 International Symposium on Network Coding (NetCod), Cambridge, MA, USA, 29–30 June 2012; pp. 125–130.
34. Mahdaviani, K.; Yazdani, R.; Ardakani, M. Linear-Complexity Overhead-Optimized Random Linear Network Codes. *arXiv* **2013**, arXiv:1311.2123.
35. Yang, S.; Tang, B. From LDPC to chunked network codes. In Proceedings of the 2014 IEEE Information Theory Workshop on Information Theory (ITW), Hobart, TAS, Australia, 2–5 November 2014; pp. 406–410.
36. Tang, B.; Yang, S. An LDPC Approach for Chunked Network Codes. *IEEE/ACM Trans. Netw.* **2018**, *26*, 605–617. [CrossRef]
37. Yang, S.; Yeung, R.W. Coding for a network coded fountain. In Proceedings of the 2011 IEEE International Symposium on Information Theory (ISIT), St. Petersburg, Russia, 31 July–5 August 2011; pp. 2647–2651.
38. Yang, S.; Yeung, R.W. Batched Sparse Codes. *IEEE Trans. Inf. Theory* **2014**, *60*, 5322–5346. [CrossRef]
39. Yang, S.; Yeung, R.W. *BATS Codes: Theory and Practice*; Synthesis Lectures on Communication Networks; Morgan & Claypool Publishers: San Rafael, CA, USA, 2017.
40. Yang, S.; Ho, S.W.; Meng, J.; Yang, E.H. Capacity Analysis of Linear Operator Channels Over Finite Fields. *IEEE Trans. Inf. Theory* **2014**, *60*, 4880–4901. [CrossRef]
41. Zhou, Q.; Yang, S.; Yin, H.H.F.; Tang, B. On BATS Codes with Variable Batch Sizes. *IEEE Commun. Lett.* **2017**, *21*, 1917–1920. [CrossRef]
42. Huang, Q.; Sun, K.; Li, X.; Wu, D.O. Just FUN: A Joint Fountain Coding and Network Coding Approach to Loss-Tolerant Information Spreading. In Proceedings of the 15th ACM International Symposium on Mobile Ad Hoc Networking and Computing, Philadelphia, PA, USA, 11–14 August 2014; pp. 83–92.
43. Yin, H.H.F.; Ng, K.H.; Wang, X.; Cao, Q. On the Minimum Delay of Block Interleaver for Batched Network Codes. In Proceedings of the 2019 IEEE International Symposium on Information Theory (ISIT), Paris, France, 7–12 July 2019; pp. 1957–1961.
44. Yin, H.H.F.; Ng, K.H.; Wang, X.; Cao, Q.; Ng, L.K.L. On the Memory Requirements of Block Interleaver for Batched Network Codes. In Proceedings of the 2020 IEEE International Symposium on Information Theory (ISIT), Angeles, CA, USA, 21–26 June 2020; pp. 1658–1663.
45. Zhou, Z.; Li, C.; Yang, S.; Guang, X. Practical Inner Codes for BATS Codes in Multi-Hop Wireless Networks. *IEEE Trans. Veh. Technol.* **2019**, *68*, 2751–2762. [CrossRef]
46. Zhou, Z.; Kang, J.; Zhou, L. Joint BATS Code and Periodic Scheduling in Multihop Wireless Networks. *IEEE Access* **2020**, *8*, 29690–29701. [CrossRef]
47. Yang, S.; Yeung, R.W.; Cheung, J.H.F.; Yin, H.H.F. BATS: Network Coding in Action. In Proceedings of the Annual Allerton Conference on Communication Control and Computing, Monticello, IL, USA, 30 September–3 October 2014; pp. 1204–1211.
48. Tang, B.; Yang, S.; Ye, B.; Guo, S.; Lu, S. Near-Optimal One-Sided Scheduling for Coded Segmented Network Coding. *IEEE Trans. Comput.* **2016**, *65*, 929–939. [CrossRef]
49. Yin, H.H.F.; Yang, S.; Zhou, Q.; Yung, L.M.L. Adaptive Recoding for BATS Codes. In Proceedings of the 2016 IEEE International Symposium on Information Theory (ISIT), Barcelona, Spain, 10–15 July 2016; pp. 2349–2353.
50. Yin, H.H.F.; Tang, B.; Ng, K.H.; Yang, S.; Wang, X.; Zhou, Q. A Unified Adaptive Recoding Framework for Batched Network Coding. *IEEE J. Sel. Areas Inf. Theory* **2021**, *2*, 1150–1164. [CrossRef]
51. Yin, H.H.F.; Ng, K.H. Impact of Packet Loss Rate Estimation on Blockwise Adaptive Recoding for Batched Network Coding. In Proceedings of the 2021 IEEE International Symposium on Information Theory (ISIT), Melbourne, VIC, Australia, 12–20 July 2021; pp. 1415–1420.
52. Yin, H.H.F.; Ng, K.H.; Zhong, A.Z.; Yeung, R.W.; Yang, S.; Chan, I.Y.Y. Intrablock Interleaving for Batched Network Coding with Blockwise Adaptive Recoding. *IEEE J. Sel. Areas Inf. Theory* **2021**, *2*, 1135–1149. [CrossRef]
53. Breidenthal, J.C. The Merits of Multi-Hop Communication in Deep Space. In Proceedings of the 2000 IEEE Aerospace Conference, Big Sky, MT, USA, 18–25 March 2000; Volume 1, pp. 211–222.
54. Zhao, H.; Dong, G.; Li, H. Simplified BATS Codes for Deep Space Multihop Networks. In Proceedings of the 2016 IEEE Information Technology, Networking, Electronic and Automation Control Conference, Chongqing, China, 20–22 May 2016; pp. 311–314.
55. Yeung, R.W.; Dong, G.; Zhu, J.; Li, H.; Yang, S.; Chen, C. Space Communication and BATS Codes: A Marriage Made in Heaven. *J. Deep. Space Explor.* **2018**, *5*, 129–139.
56. Sozer, E.M.; Stojanovic, M.; Proakis, J.G. Underwater Acoustic Networks. *IEEE J. Ocean. Eng.* **2000**, *25*, 72–83. [CrossRef]

57. Yang, S.; Ma, J.; Huang, X. Multi-Hop Underwater Acoustic Networks Based on BATS Codes. In Proceedings of the 13th International Conference on Underwater Networks & Systems, Shenzhen, China, 3–5 December 2018; pp. 30:1–30:5.
58. Sprea, N.; Bashir, M.; Truhachev, D.; Srinivas, K.V.; Schlegel, C.; Sacchi, C. BATS Coding for Underwater Acoustic Communication Networks. In Proceedings of the OCEANS 2019, Marseille, France, 17–20 June 2019; pp. 1–10.
59. Yin, H.H.F.; Yeung, R.W.; Yang, S. A Protocol Design Paradigm for Batched Sparse Codes. *Entropy* **2020**, *22*. 790. [CrossRef] [PubMed]
60. Yin, H.H.F.; Tahernia, M. On the Design of Timeout-Based Recoders for Batched Network Codes in the MU-MIMO Regime. In Proceedings of the 2022 IEEE Region 10 Conference (TENCON), Hong Kong, China, 1–4 November 2022.
61. Qing, J.; Yin, H.H.F.; Yeung, R.W. Enhancing the Decoding Rates of BATS Codes by Learning with Guided Information. In Proceedings of the 2022 IEEE International Symposium on Information Theory (ISIT), Espoo, Finland, 26 June–1 July 2022; pp. 37–42.
62. Yang, S.; Zhou, Q. Tree Analysis of BATS Codes. *IEEE Commun. Lett.* **2016**, *20*, 37–40. [CrossRef]
63. Yang, S.; Ng, T.C.; Yeung, R.W. Finite-Length Analysis of BATS Codes. *IEEE Trans. Inf. Theory* **2018**, *64*, 322–348. [CrossRef]
64. Yang, J.; Shi, Z.; Wang, C.; Ji, J. Design of Optimized Sliding-Window BATS Codes. *IEEE Commun. Lett.* **2019**, *23*, 410–413. [CrossRef]
65. Xu, X.; Zeng, Y.; Guan, Y.L.; Yuan, L. Expanding-Window BATS Code for Scalable Video Multicasting Over Erasure Networks. *IEEE Trans. Multimed.* **2018**, *20*, 271–281. [CrossRef]
66. Yin, H.H.F.; Wong, H.W.H.; Tahernia, M.; Qing, J. Packet Size Optimization for Batched Network Coding. In Proceedings of the 2022 IEEE International Symposium on Information Theory (ISIT), Espoo, Finland, 26 June–1 July 2022; pp. 1584–1589.
67. Yang, S.; Yeung, R.W. Network Communication Protocol Design from the Perspective of Batched Network Coding. *IEEE Commun. Mag.* **2022**, *60*, 89–93. [CrossRef]
68. Shokrollahi, A.; Luby, M. Raptor Codes. In *Foundations and Trends in Communications and Information Theory*; Now Publishers Inc.: Hanover, MA, USA, 2011; Volume 6.
69. Shokrollahi, A.; Lassen, S.; Karp, R. Systems and Processes for Decoding Chain Reaction Codes through Inactivation. U.S. Patent 6,856,263, 15 February 2005.
70. Xu, X.; Guan, Y.L.; Zeng, Y. Batched Network Coding with Adaptive Recoding for Multi-Hop Erasure Channels with Memory. *IEEE Trans. Commun.* **2018**, *66*, 1042–1052. [CrossRef]
71. Yin, H.H.F.; Tahernia, M. Multi-Phase Recoding for Batched Network Coding. In Proceedings of the 2022 IEEE Information Theory Workshop on Information Theory (ITW), Mumbai, India, 6–9 November 2022; pp. 25–30.
72. Ye, F.; Roy, S.; Wang, H. Efficient Data Dissemination in Vehicular Ad Hoc Networks. *IEEE J. Sel. Areas Commun. (JSAC)* **2012**, *30*, 769–779. [CrossRef]
73. Yin, H.H.F.; Xu, X.; Ng, K.H.; Guan, Y.L.; Yeung, R.W. Packet Efficiency of BATS Coding on Wireless Relay Network with Overhearing. In Proceedings of the 2019 IEEE International Symposium on Information Theory (ISIT), Paris, France, 7–12 July 2019; pp. 1967–1971.
74. Lucani, D.E.; Médard, M.; Stojanovic, M. Random Linear Network Coding for Time-Division Duplexing: Field Size Considerations. In Proceedings of the 2009 IEEE Global Telecommunications Conference, Honolulu, HI, USA, 30 November–4 December 2009; pp. 1–6.
75. Yin, H.H.F.; Xu, X.; Ng, K.H.; Guan, Y.L.; Yeung, R.W. Analysis of Innovative Rank of Batched Network Codes for Wireless Relay Networks. In Proceedings of the 2021 IEEE Information Theory Workshop on Information Theory (ITW), Kanazawa, Japan, 17–21 October 2021.
76. Zhang, C.; Tang, B.; Ye, B.; Lu, S. An efficient chunked network code based transmission scheme in wireless networks. In Proceedings of the 2017 IEEE International Conference on Communications (ICC), Paris, France, 21–25 May 2017; pp. 1–6.
77. Gilbert, E.N. Capacity of a Burst-Noise Channel. *Bell Syst. Tech. J.* **1960**, *39*, 1253–1265. [CrossRef]
78. Elliott, E.O. Estimates of Error Rates for Codes on Burst-Noise Channels. *Bell Syst. Tech. J.* **1963**, *42*, 1977–1997. [CrossRef]
79. NIST Digital Library of Mathematical Functions. Release 1.1.0. Available online: http://dlmf.nist.gov/ (accessed on 15 December 2020).
80. Galassi, M.; Davies, J.; Theiler, J.; Gough, B.; Jungman, G.; Booth, M.; Rossi, F. *GNU Scientific Library Reference Manual*, 3rd ed.; Network Theory Ltd.: London, UK, 2002.
81. Wang, J.; Jia, Z.; Yin, H.H.F.; Yang, S. Small-Sample Inferred Adaptive Recoding for Batched Network Coding. In Proceedings of the 2021 IEEE International Symposium on Information Theory (ISIT), Melbourne, VIC, Australia, 12–20 July 2021; pp. 1427–1432.
82. Dong, Y.; Jin, S.; Yang, S.; Yin, H.H.F. Network Utility Maximization for BATS Code Enabled Multihop Wireless Networks. In Proceedings of the 2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, 7–11 June 2020.
83. Dong, Y.; Jin, S.; Chen, Y.; Yang, S.; Yin, H.H.F. Utility Maximization for Multihop Networks Employing BATS Codes with Adaptive Recoding. *IEEE J. Sel. Areas Inf. Theory* **2021**, *2*, 1120–1134. [CrossRef]
84. Fredman, M.L.; Tarjan, R.E. Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms. *J. ACM (JACM)* **1987**, *34*, 596–615. [CrossRef]
85. Musser, D. Introspective Sorting and Selection Algorithms. *Softw. Pract. Exp.* **1997**, *27*, 983–993. [CrossRef]
86. Hoare, C.A.R. Algorithm 65: Find. *Commun. ACM* **1961**, *4*, 321–322. [CrossRef]

87. Blum, M.; Floyd, R.W.; Pratt, V.; Rivest, R.L.; Tarjan, R.E. Time Bounds for Selection. *J. Comput. Syst. Sci.* **1973**, *7*, 448–461. [CrossRef]
88. Seward, H.H. Information Sorting in the Application of Electronic Digital Computers to Business Operations. Master's Thesis, MIT Digital Computer Laboratory, Cambridge, UK, 1954; Report R-232.
89. Higham, N.J. *Accuracy and Stability of Numerical Algorithms: Second Edition*; Other Titles in Applied Mathematics; Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 2002.
90. Hodges, J.L., Jr.; Lehmann, E.L. Some Problems in Minimax Point Estimation. *Ann. Math. Stat.* **1950**, *21*, 182–197. [CrossRef]
91. Steinhaus, H. The Problem of Estimation. *Ann. Math. Stat.* **1957**, *28*, 633–648. [CrossRef]
92. Qing, J.; Leong, P.H.W.; Yeung, R.W. Performance Analysis and Optimal Design of BATS Code: A Hardware Perspective. *IEEE Trans. Veh. Technol.* **2023**, 1–14. [CrossRef]
93. Yang, S.; Yeung, W.H.; Chao, T.I.; Lee, K.H.; Ho, C.I. Hardware Acceleration for Batched Sparse Codes. U.S. Patent 10,237,782, 19 March 2019.
94. Yin, H.F.H.; Yang, S.; Yeung, W.H.R. Loss-Resilient Protocols for Communication Networks. U.S. Patent 10,425,192, 24 September 2019.
95. Yin, H.F.H.; Ng, K.H.; Zhong, Z.; Yeung, R.W.H.; Yang, S. Compatible Packet Separation for Communication Networks. U.S. Patent 11,452,003, 20 September 2022.
96. Yin, H.F.H. Recoding Optimizations in Batched Sparse Codes. Ph.D. Thesis, The Chinese University of Hong Kong, Hong Kong, China, 2019.
97. Chazelle, B. The Soft Heap: An Approximate Priority Queue with Optimal Error Rate. *J. ACM (JACM)* **2000**, *47*, 16. [CrossRef]
98. Gao, R.; Kleywegt, A.J. Distributionally Robust Stochastic Optimization with Wasserstein Distance. *Math. Oper. Res.* **2023**, *48*, 603–655. [CrossRef]