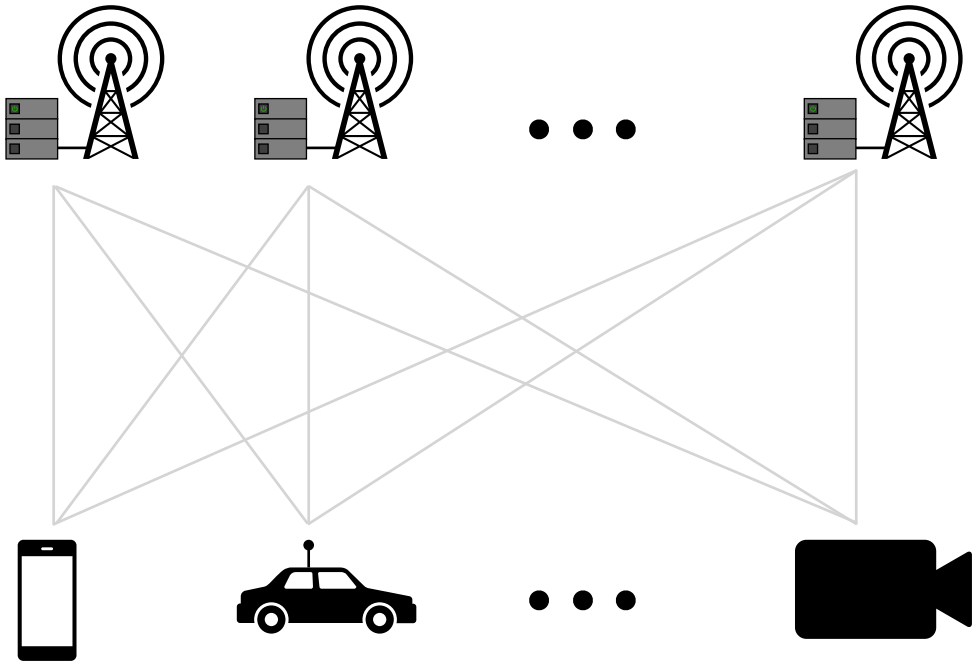
CHALMERS
UNIVERSITY OF TECHNOLOGY



Coding for Mobile Edge Computing

Using Luby-Transform codes to lower the latency in edge computing systems

Master's thesis in Communication Engineering

ANTON FRIGÅRD

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020:NN

Coding for Mobile Edge Computing

Using Luby-Transform codes to lower the latency in edge computing systems

ANTON FRIGÅRD

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Communication Systems Group
Chalmers University of Technology
Gothenburg, Sweden 2020

Coding for Mobile Edge Computing
Using Luby-Transform codes to lower the latency in edge computing systems
ANTON FRIGÅRD

© ANTON FRIGÅRD, 2020.

Supervisor: Alexandre Graell i Amat, Department of Electrical Engineering, Chalmers University of Technology, Gothenburg, Sweden
Co-supervisor: Eirik Rosnes, Simula UiB, Bergen, Norway
Examiner: Alexandre Graell i Amat, Department of Electrical Engineering, Chalmers University of Technology, Gothenburg, Sweden

Master's Thesis 2020:NN
Department of Electrical Engineering
Communication Systems Group
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Wireless edge network consisting of multiple users and edge servers.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2020

Coding for Mobile Edge Computing
Using Luby-Transform Codes to lower the latency in edge computing systems
ANTON FRIGÅRD
Department of Electrical Engineering
Chalmers University of Technology

Abstract

The idea of mobile edge computing (MEC) is to provide external computational and storage resources close to the end user, i.e., close to the network edge. Servers are placed in the vicinity of the users and can be accessed through a direct wireless connection. A typical MEC use case is computation offloading, where the users want some data processed but cannot perform it themselves due to, e.g., not having enough processing capacity or the need to save energy. The users transmit their data to the MEC servers over the wireless channel, the MEC servers process the data and then send the results back to the users.

As noted in the research on distributed computing, data servers tend to straggle, i.e., at random occasions require an unacceptable amount of time to finish their assigned tasks. The application of codes has proven to be useful in this regard, enabling recovery of the results from a subset of servers instead of having to wait for all servers to finish. The same principle can be applied in the MEC setting. There is however another dimension to MEC as compared to distributed computing; that of transmitting data over a wireless channel. In particular, it has been shown that computing the same data on multiple servers can introduce diversity in the downlink transmission. The servers that have computed the same data can then collaborate to transmit the data with lower transmission latency than would otherwise be possible.

Zhang *et al*. investigated the use of maximum distance separable (MDS) codes in the context of MEC to protect against straggling servers, and repetition codes to provide a low downlink transmission latency. Their MDS hybrid scheme shows a clear improvement to that of uncoded MEC.

In this thesis two coding schemes inspired by the work of Zhang *et al.* are proposed and evaluated by simulations. The schemes are based on Luby-Transform codes and inactivation decoding. Simulations indicate a lower total latency for one of the proposed schemes as compared to the MDS hybrid. A lower bound on the total latency is also derived and used as a benchmark for the schemes. If the computation latency dominates the total latency, all schemes perform relatively close to the bound. If instead the downlink transmission latency dominates the total latency, the schemes perform somewhat further away.

Keywords: coding theory, distributed computing, information theory, mobile edge computing,

Acknowledgements

I want to thank Alex, my supervisor, for giving me the opportunity to experience what it means to do research. You have expected me to do my best while providing me your support when needed, leading me to accomplish more than I thought I would at the start of this thesis. My sincere thanks also to the guys at Simula UiB, Bergen, Norway; Eirik Rosnes for your support as co-supervisor, and Siddharta Kumar and Reent Schlegel for a fruitful collaboration. Your support has been most appreciated.

Anton Frigård, Gothenburg, April 2020

Contents

List of Figures

List of Tables

1
Introduction

Performing heavy processing on resource-constrained devices may result in unacceptably long execution delay and high energy consumption. This is often dealt with by outsourcing computation tasks for processing at external units. Cloud computing is one technique to accomplish this by offloading computations to remote data centers. However, this strategy may also suffer from long delays, owing to the large distances between users and data centers, and congestion in the core and backhaul networks [1]. Consider for example a tourist who films objects of interest with his smart phone, and wants information about the objects to be displayed on his screen in real time. The video stream is offloaded to the cloud for image processing, but the area is dense with tourists requiring digital services, forcing contention of network resources and causing an annoying delay. For applications where latency is critical, offloading to the cloud may therefore not be an adequate solution.

Mobile edge computing (MEC) is a concept aimed at eliminating some of this delay by providing storage and processing capacity in the proximity of users [1–3]. A general MEC offloading scenario involves multiple users transmitting their data over a wireless channel to a set of servers located at the network edge, which process the data and transmit the results back to the users. Since the users are close to the servers, the propagation delay will likely be short. Furthermore, some of the load on the backhaul network is alleviated, since little or no data has to be sent to the cloud. MEC is posed to play a key role in, e.g., the realization of 5G, Internet of Things, augmented and virtual reality applications, video analytics, health care services, and connected and self-driving vehicles [1–3]. Consider for example a traffic scenario, where sensors around the road are used to detect accidents [3]. The sensor data need to be quickly analyzed, such that emergency signals can be transmitted to nearby vehicles, forcing them to slow down. The data is offloaded to MEC servers that are deployed at nearby base stations, such that the data can be processed in the close vicinity of the road and existing radio resources can be used to broadcast the signals. We will hereafter refer to the MEC servers as edge nodes (ENs).

Since the wireless channel and ENs are shared by the users, a decision must be made on how much of these resources to allocate to each user [4–12]. You *et al*. [4] considered joint optimization of radio resources, how much data should be offloaded and whether a user should offload none, part, or all of its data. The goal was to minimize the sum user energy consumption under latency constraints. In [5] a similar scenario was considered for binary offloading, i.e., when users either offload all of their data or none at all. Resource allocation involving CPU frequency scaling was consider in [6–8]. Dinh *et al*. [6] proposed algorithms for jointly optimizing task allocation and users' CPU frequency. The aim was to minimize computation

latency and users' energy consumption. A similar case was considered by Sardellitti *et al*. [7], who minimized the users' energy consumption under latency constraints by jointly optimizing the transmit pre-coding matrices of the users and the CPU cycles/second assigned to each user by the edge cloud. By jointly optimizing CPU frequency, transmit power, and offloading ratio of a user, Wang *et al*. [8] separately minimized the users' energy consumption and computation time in a single-user, single-EN setting. The authors also considered a setting with a single user and multiple ENs.

Minimization of latency under energy constraints has also been considered [9, 10]. Chen *et al*. [9] proposed an efficient binary offloading scheme for an MEC scenario with multiple users and ENs, jointly optimizing offloading decision and computational resource allocation. In [10], Liu *et al*. optimized task allocation in a single-user, single-EN scenario with partial offloading, showing that their offloading policy outperforms benchmark policies.

Joint optimization of offloading decision, and radio and computational resources was also considered in [11,12]. Zhang *et al*. [11] minimized the sum energy consumption and latency in a multi-user setting. In [12], Wang *et al*. also included caching at the edge to maximize the total revenue of the network.

By assigning data for processing at multiple ENs, diversity is introduced in the downlink and the ENs can cooperate to transmit the results back to the users in a shorter time [13,14]. This idea was leveraged by Li *et al*. [13] in a multi-user, multi-EN system. The computation task of each user is divided into subtasks, which are either executed locally, or offloaded and repeated across several ENs. An offloading strategy with optimal task partition ratios and amount of repetition was devised to minimize the total latency. While numerical evaluations show an improvement compared to not using repetition, the price paid is an increased computational load. The authors then extended this work in [14], precisely characterizing the tradeoff between latency and computational load.

Distributed computing systems employing multiple servers have been noted to randomly suffer from stragglers, i.e., servers that take a long time to finish their respective tasks [15]. Since multiple servers are employed in MEC, it is reasonable to assume the presence of stragglers here as well. By modeling stragglers in a cloud computing scenario as erasures, it has recently been shown that erasure codes can be applied to enable the completion of the overall computation from only a subset of the servers [16–23]. In fact, repeating computation tasks across servers as was done in the MEC setting in [13,14] to enable cooperative transmission is a form of coding that has been considered in the distributed computing setting as protection against stragglers [16,17]. Out of the servers that have processed the same task the system need only to wait for the quickest one.

Several works have specifically considered the use of error correcting codes for matrix multiplication in distributed computing over the cloud [18–22]. Yu *et al*. [18] proposed a coding strategy referred to as polynomial codes for large-scale matrix-matrix multiplication and showed its optimality in terms of minimum number of servers to wait for. The use of maximum distance separable (MDS) codes for distributed matrix-vector multiplication was considered by Lee *et al*. in [19], speeding up computation compared to an uncoded strategy. A more efficient use of computa-

tional resources was investigated by Mallick *et al*. [20] using Luby-Transform (LT) codes [24] for distributed matrix-vector multiplication. Contrary to previous coded schemes, this strategy does not discard computed products. MDS and LT codes were also used by Severinson *et al*. [21] for distributed matrix-vector multiplication. As opposed to previous work the encoding and decoding times were taken into account, and the MDS code was applied to matrix blocks instead of rows to lower the decoding complexity. The authors then extended this work in [22] by considering Raptor codes.

Whereas research on coding for distributed computing in data centers is extensive, few have yet considered coding in the MEC scenario. Two recent papers, one by Zhang *et al*. [23] and one by Keshtkarjahromi *et al*. [25], investigated coding for protection against straggling ENs. Keshtkarjahromi *et al*. proposed a coded strategy based on LT codes, showing its near-optimality and evaluating it in a testbed of Android devices. Zhang *et al*. considered a concatenation of repetition and MDS codes, making use of the same cooperative transmission as Li *et al*. in [13,14]. Their MDS hybrid scheme, which we will refer to as MDS-repetition scheme in this thesis, provides a lower latency than an uncoded approach. However, MDS codes have a very high decoding complexity, which makes them unsuitable for most practical applications. This fact motivates an investigation into practical coding schemes, which can provide both a reasonable decoding complexity and a low latency.

1.1 Thesis objective

In this thesis we propose two coding schemes based on LT codes [24] and repetition codes for lowering the latency in an MEC system consisting of multiple users and multiple ENs. The LT codes are decoded using inactivation decoding, a maximum likelihood decoding algorithm with moderate complexity [26]. We wish to find out how these schemes perform in terms of latency compared to the MDS-repetition scheme presented in [23]. Can we achieve better computation and communication latency? How does the performance differ when changing parameter values? Which scheme is more practical?

It is also interesting to investigate whether we can find a tight lower bound on the total latency that holds for all coding schemes. This could then be used as a reference to evaluate the performance of the proposed schemes.

1.2 Thesis structure

The thesis is organized as follows. In Chapter 2, the necessary mathematics and other preliminaries are presented. Chapter 3 provides a thorough mathematical description of the system under consideration. In Chapter 4, a lower bound on the total latency is derived. Two coding schemes based on LT codes are presented in Chapter 5, and compared with the converse bound and MDS-repetition scheme in Chapter 6. In Chapter 7, we conclude our results.

2
Preliminaries

The core of this thesis is the application of coding theory in MEC. This chapter therefore begins with an introduction to the idea of coding. A special class of random codes called LT codes is then presented, with an in-depth explanation of the encoding and decoding procedures. We then consider a simple example of wireless communication and present a form of cooperative transmission technique which is crucial to this thesis. Furthermore, we consider the application of coding in distributed computing, in some aspects a precursor to coding for mobile edge computing, which is discussed last.

2.1 Notation

Vectors will be denoted by bold, lowercase letters, and matrices by bold, uppercase letters. Random variables will be denoted by uppercase letters, and their realizations by the lowercase counterparts. Random vectors will, as matrices, be denoted by bold, uppercase letters, and their realizations will be bold and lowercase. However, their randomness will be explicitly pointed out to avoid confusion. For some integer u, we denote by $[u]$ the set $\{1, \ldots, u\}$. Sets will be denoted in calligraphic font. For some real numbers a and b, we denote by $[a, b]$ the set of all real values between a and b.

2.2 Coding theory

The field of coding theory was born shortly after the publication of Claude Shannon's landmark paper *A Mathematical Theory of Communication* in 1948, wherein Shannon laid the groundwork for a theory of information [27]. It was well known at the time that signals, which carry information, are a□ected by distortions of di□erent kinds during transmission, such as thermal noise. This distortion has the potential to render the transmitted information unintelligible to the receiving part. What Shannon showed was that to protect the information to be transmitted, one can add additional information in a structured way so as to detect and possibly correct for distortion. He called this adding of redundant information *coding*.

A very simple distortion model, or *channel*, is depicted in Figure 2.1. The left hand side of the graph constitutes the transmitter, which transmits a length-k sequence of zeros and ones, or *bits*, to the receiver at the right-hand side. During transmission a bit may be *erased*, i.e., taking an unknown value. In particular, a bit is erased with probability ε and received intact with probability $1 - \varepsilon$. This channel

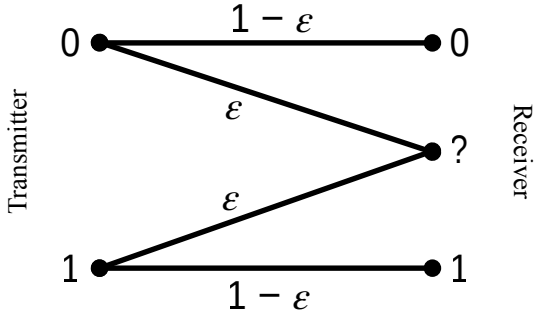


Figure 2.1: Binary erasure channel

is called the binary erasure channel (BEC). The job of the receiver is to estimate which sequence of bits was transmitted in the potential presence of erasures. Note that an erased bit is equally likely to be a 0 or 1 from the perspective of the receiver. If this bit is considered merely by itself, the only reasonable action for the receiver to take is to flip a coin, i.e., to choose between a 0 and a 1 uniformly at random.

However, to make a decision on the value of an erased bit we can consider it in a larger context; the structure of the whole sequence of transmitted bits. Consider for example the transmission of a bit sequence, or *data word*, u of length $k = 2$. For protection we *encode* u using a single parity check (SPC) code to get a *codeword* c of length $n = 3$. This entails appending a single bit to u such that there is an even number of ones in c. For example, a data word $u = [0, 1]$ would be encoded into $c = [0, 1, 1]$. The data word bits are called information bits and the codeword bits are called code bits. Ideally we would like to avoid having too many code bits since these take up space in the transmission where information bits could have been sent instead. The price we pay for the protection of coding is thus a longer time to transmit the same amount of information, or alternatively require a larger bandwidth.

Consider now transmitting $c = [0, 1, 1]$ over the BEC. Assume that the channel erases the first bit during transmission such that the received bit vector is $y = [?, 1, 1]$. At the receiver side it is known that codewords must have an even number of ones, which means that the only valid guess to which codeword was transmitted is $\hat{c} = [0, 1, 1]$. We say that the receiver *decodes* y onto $\hat{c}$, which can then be mapped back to $\hat{u} = [0, 1]$. The receiver thus managed to both *detect* and *correct* the error introduced by the channel. If instead the received word would be $y = [?, ?, 1]$, then the receiver would need to choose between the two equally likely codewords $[0, 1, 1]$ and $[1, 0, 1]$. There is therefore a risk of decoding the received word onto a codeword that was not the one transmitted, i.e., there is a probability that decoding fails.

Encoding is often expressed as the multiplication $c = uG$, where the $k \times n$ matrix G is the generator matrix of the code. In the SPC example, the generator matrix is

$$G = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

and addition is carried out modulo 2. In general, the data word and codeword need not necessarily be bit vectors. For example, in this thesis we will encode a matrix W of size $k \times l$ by a generator matrix G of size $n \times k$ as $C = GW$. In this context the rows of W are the analog of the bits of u in $c = uG$.

SPC codes are among the simplest codes constructed and can be categorized as MDS codes. An (n, k) MDS code has the special feature that the transmitted data word can be decoded from any k out of the n transmitted code symbols.

2.3 Decoding complexity

Decoding SPC codes is very simple, but most codes require more sophisticated decoding algorithms. Some algorithms provide very good error correcting capability but are heavy on processing, and vice-versa. A benchmark in terms of error correcting performance is maximum likelihood (ML) decoding. The downside of ML decoding is that it is usually prohibitively heavy on processing. We say that ML decoding has a high *decoding complexity*. The complexity of a decoding algorithm can be measured in terms of the number of operations that it performs. For example, brute force ML decoding of binary (n, k) linear block codes has a complexity of $O(2^k)$ since the decoder needs to search through all valid codewords, of which there are 2^k. Clearly, for large k there will be a huge number of codewords to search through. However, for some codes there exist ML decoding algorithms with a much lower complexity. The Viterbi algorithm for decoding convolutional codes is one such example.

2.4 Luby-Transform codes

A class of codes that perform very well on erasure channels is *fountain codes*. Fountain codes are random and can generate any desired number n of output (code) bits from an input (information) bit sequence of fixed length k to the encoder. A special class of fountain codes with a low-complexity decoding algorithm is LT codes [24], which will now be explained in more detail.

2.4.1 Encoding

We will restrict ourselves to binary LT codes. Consider the formation of output bit c_i, $i \in [n]$. The first step of encoding is drawing a degree d from a degree distribution Ω. The encoder then selects d out of the k incoming bits uniformly at random and sum them to form the output bit. Let Ω_d be the probability of generating degree d.

Let us consider an example with $k = 3$ and $n = 5$. Assume that the degrees are generated as $d_1 = d_2 = d_4 = 2$, $d_3 = 3$, and $d_5 = 1$. In this case, the output bits could be encoded as, e.g., $c_1 = x_1 + x_2$, $c_2 = x_2 + x_3$, $c_3 = x_1 + x_2 + x_3$, $c_4 = x_1 + x_3$

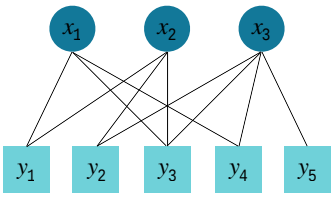


Figure 2.2: Bipartite graph representation of an LT code with $k = 3$ and $n = 5$.

and $c_5 = x_3$, where addition is over the binary field. Let y be the received bits and assume that no bit was erased. We can represent the input bits and received output bits and their connections by a *bipartite graph* as in Figure 2.2. The top circles represent input bits and the bottom squares represent output bits. An output bit is formed by the binary addition of the input bits connected to it by edges. In this graph context we will refer to the input bits as input nodes, and the output bits as output nodes.

2.4.2 Inactivation decoding

LT codes were first presented together with a suboptimal low-complexity decoding algorithm often referred to as peeling decoding [24]. However, in this thesis we will make use of an ML decoding algorithm known as inactivation decoding, which can be seen as an extension of peeling decoding [26]. Inactivation decoding is easily explained using the bipartite graph. Assume that a length-n codeword was transmitted and that out of the n code bits, m bits were non-erased. Consider the bipartite graph corresponding to the inputs bits u, which the receiver does not know and wants to estimate, and *received* output bits y. Inactivation decoding works as follows.

1. Find an output node y_i that is connected to only one input node x_j, i.e., which has degree 1. If no such output node exists, one of the input nodes is marked as *inactive*. The value of the inactive node is added to its neighbors and the inactive node along with its edges are removed from the graph. This will hopefully reduce the degree of some output node to 1, such that decoding can move on to the next step. If not, this step is repeated until there appears an output node of degree 1.

2. Since we know the value of y_i and since it is connected to only x_j, the value of x_j must be y_i. We therefore set $x_j = y_i$ and remove node y_i and its edge to x_i from the graph.

3. Add the value of x_i to the output nodes still connected to it and then remove the connecting edges from the graph. If there remain edges in the graph, go to step 1. If there remain no edges, move on to step 4.

4. Solve the linear system of equations corresponding to the inactive input nodes by Gaussian elimination. Back-substitute to find the values of the non-inactive input nodes.

A decoding example with $k = 5$ and $m = 7$ is given in Figure 2.3. Addition is carried out over the binary field.

2.4.3 Decoding failure probability

When transmitting a codeword over an erasure channel we do not know which code bits will be erased. Combining this fact with the random nature of the LT code we can not be sure that the linear system of equations involving the input and output bits has a solution, nor that it has a single solution. We model this randomness by a variable Φ, such that decoding is successful if $k + \Phi$ output bits are received intact. The distribution of Φ is of great interest, since from it we can determine the probability of decoding failure P_F. This is the probability that collecting $k + \varphi$ output bits is insufficient in order to determine a unique solution to the resulting linear system of equations, i.e., $P_F(\varphi) \triangleq \Pr(\Phi \geq \varphi)$. In [26] the decoding failure probability for finite codeword lengths was derived. The analysis results in a recursion that has to be evaluated numerically and is computationally demanding for large codeword lengths. We are instead interested in an upper bound on P_F derived in [28]. The bound is sufficiently tight for all values of φ and much less computationally demanding than the true decoding failure probability. For binary LT codes, the bound is [28]

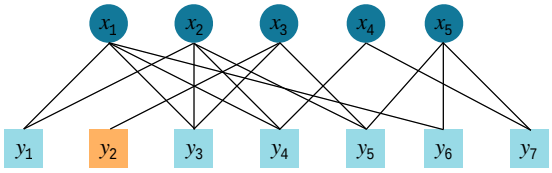
$$P_F(\varphi) \leq \sum_{i=1}^{k} \binom{k}{i} \frac{1}{2} + \frac{1}{2} \sum_{d} \Omega_d \frac{K_d(i;k)}{K_d(0;k)}^{k+\varphi} \tag{2.1}$$

where $K_\zeta(\xi;\nu)$ is the Krawtchouk polynomial

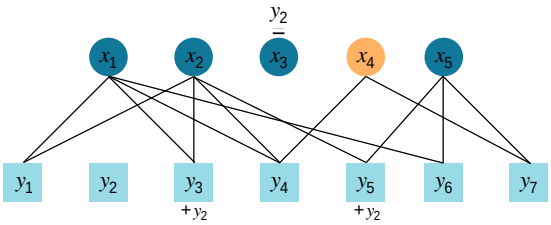
$$K_\zeta(\xi;\nu) = \sum_{i=0}^{\zeta} (-1)^i \binom{\xi}{i} \binom{\nu-\xi}{\zeta-i}.$$

2.4.4 Decoding complexity

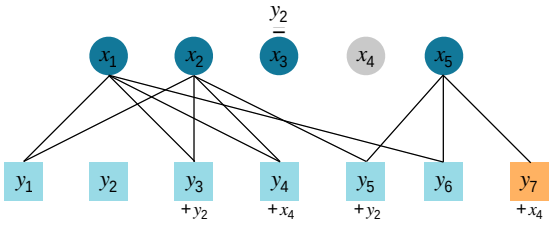
The decoding complexity of inactivation decoding is driven by step 4. Here the Gaussian elimination has a complexity of $O(\alpha^3)$, where α is the number of inactivations [26]. The degree distribution Ω has a significant impact on α and as such can be optimized to minimize decoding costs. Two degree distributions were presented in Luby's original paper on LT codes [24]; the ideal soliton distribution (ISD) and the robust soliton distribution (RSD). The ISD is designed to keep the expected number of degree-1 output nodes equal to one at every stage of the decoding process, with the aim of keeping Φ small. Unfortunately the variance around this expected value is not negligible and the lack of degree-1 output nodes may therefore be too frequent, which increases the number of inactivations and drives up decoding costs. The RSD solves this issue by increasing the expected number of degree-1 output nodes to some predetermined value. The ISD is given by



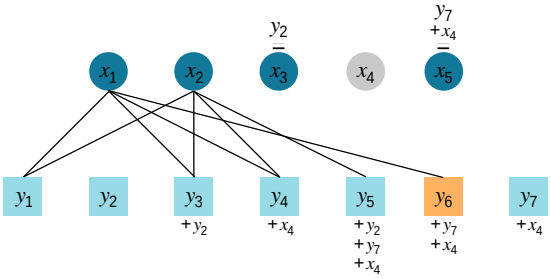
(a) The second output node is the only of degree one.



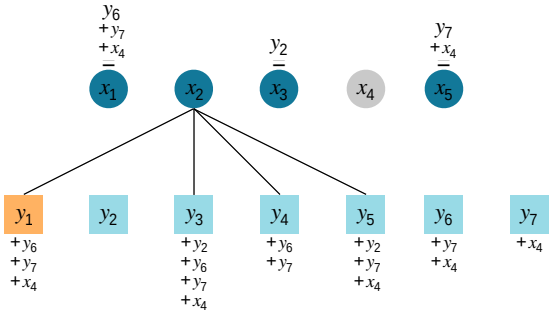
(b) We set $x_3 = y_2$, and add y_2 to y_3 and y_5. After this operation there is no degree-1 output node. We therefore randomly mark x_4 as inactive.



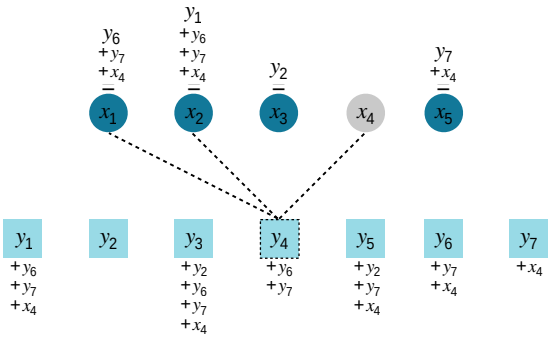
(c) x_4 is added to y_4 and y_7 and the corresponding edges are removed from the graph. After this operation the only degree-1 output node is the seventh.



(d) We set $x_5 = y_7 + x_4$ and add $y_7 + x_4$ to the current value of the fifth and sixth output node. After this there are two degree-1 output nodes. We randomly choose the sixth output node to be handled next.



(e) We set $x_1 = y_6 + y_7 + x_4$, and $y_6 + y_7 + x_4$ is added to the current value of the first, third, and fourth output node. Randomly choose the first output node to be handled next.



(f) Find the value of x_4 by solving the equation $(y_6 + y_7 + x_4) + (y_1 + y_6 + y_7 + x_4) + x_4 = y_4$. The result is $x_4 = y_1 + y_4$. We can then find the values of the other input nodes.

Figure 2.3: Example of inactivation decoding of an LT code.

$$\Omega_d^{\text{ISD}} = \begin{cases} 1/k & \text{for } d = 1 \\ 1/(d(d-1)) & \text{for } d = 2, \ldots, k \end{cases}$$

and the RSD is given by

$$\Omega_d^{\text{RSD}} = \frac{\Omega_d^{\text{ISD}} + \tau_d}{\sum_{d=1}^{k} (\Omega_d^{\text{ISD}} + \tau_d)}$$

where

$$\tau_d = \begin{cases} b/ik & \text{for } i = 1, \ldots, k/b - 1 \\ b \ln(b/\delta)/k & \text{for } d = k/b \\ 0 & \text{for } d = k/b + 1, \ldots, k \end{cases} \tag{2.2}$$

$$b = c\sqrt{k} \ln \frac{k}{\delta} \tag{2.3}$$

for some real-valued design parameters $\delta > 0$ and $c > 0$ [24].

In this thesis we do not take decoding time into account and instead aim at minimizing $\mathrm{E}[\Phi]$, i.e., the average overhead. In principle this can be done by picking different values of δ and c, computing the RSD in (2.2), computing the decoding failure probability using the bound in (2.1), and evaluating $\mathrm{E}[\Phi]$.

2.5 Distributed computing

In distributed computing, a computation task is divided into subtasks and handed out to different servers for parallel processing. The aim is to finish the task in a shorter time than would be possible with only a single server, and to leverage available computational resources more efficiently. There are however a few challenges that arise in this setting. For example, some servers may lag, or *straggle*, such that it takes an unacceptable amount of time for them to finish their respective tasks. This issue has been dealt with by the use of coding [15–17, 19–22, 29–32]. In particular, the computation can be protected by modeling a straggling server as an erasure and then apply erasure correcting codes such as MDS or LT codes.

Consider for example some length-l column vector x which we wish to multiply by some $k \times l$ matrix W. We have three servers at our disposal. A straightforward approach would be to divide W into three $\frac{k}{3} \times l$ blocks as $W = \left(W_1^T, W_2^T, W_3^T\right)^T$ and hand out one block to each server. Server s would compute $y_s = W_s x$ and the total computation would be $y = Wx = \left(y_1^T, y_2^T, y_3^T\right)^T$. However, the system is vulnerable to stragglers since it needs to wait for all servers to finish.

Consider instead partitioning W into two $\frac{k}{2} \times l$ submatrices as $W = \left(W_1^T, W_2^T\right)^T$ and then applying an MDS code. The first and second servers are handed W_1 and W_2 respectively, while the third is handed $W_1 + W_2$. The servers then compute

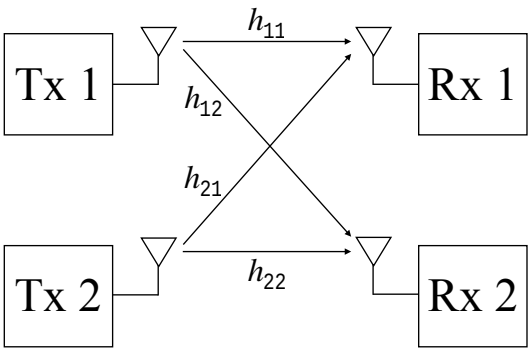


Figure 2.4: A wireless channel with 2 transmitters and 2 receivers

$y_1 = W_1x$, $y_2 = W_2x$, and $y_3 = (W_1 + W_2)x$. The desired result y can then be obtained from any set of two servers, i.e., we can tolerate one straggler without affecting the reliability of the computation.

In [19] the straggling duration of a server was modeled as an exponential random variable. Let η be the exponential distribution parameter. The expected value and variance of an exponential random variable with parameter η is $1/\eta$. This means that as η decreases the average straggling duration increases and so does its variability.

2.6 Wireless communication

This thesis involves communication over a wireless channel. In particular we will use a certain cooperative transmission technique, which is best explained by an example.

Consider the wireless channel depicted in Figure 2.4 with 2 transmitters and 2 receivers. The complex channel gain between transmitter i and receiver j is given by h_{ij}. The channel as a whole can be described by the channel matrix

$$H = \begin{bmatrix} h_{11} & h_{21} \\ h_{12} & h_{22} \end{bmatrix}.$$

Both transmitters are assumed to know the channel matrix, which is constant during transmission. Assume that the first transmitter has stored two data vectors x_1 and x_2, and that the second transmitter has stored only x_2. The first vector can be expressed as $x_1 = [x_{11}, x_{12}]$, where x_{11} and x_{12} are length-l bit vectors desired by the first and second receiver, respectively. Similarly, $x_2 = [x_{21}, x_{22}]$, where x_{21} is desired by the first receiver and x_{22} by the second.

Consider the transmission of x_1 by the first transmitter. We first map x_{11} onto a vector v of the same length, where the ith element is given by $v(i) = (-1)^{x_{11}(i)}$, and $x_{11}(i)$ is the ith element of x_{11}. The first transmitter then broadcasts the signal $u(i) = v(i)/h_{11}$, i [l], where the vector v is scaled to compensate for the gain introduced by the channel. Since the signal is destined for the first receiver, the second receiver ignores it. The signal received by the first user is $y_1(i) = h_{11}u_1(i) + z_1(i) = v(i) + z_1(i)$, i [l], where $z_1(i)$ is white Gaussian noise of unit power.

We have thus created an additive white Gaussian noise (AWGN) channel. If the signal-to-noise ratio (SNR) is high enough the transmission will succeed with high probability. We can then transmit x_{12} to the second receiver in the same way, so that x_1 in its entirety is delivered in $2l$ time units.

Consider now the transmission of x_2, which is cached by both transmitters. As in the previous case, we map x_{21} and x_{22} onto vectors v_1 and v_2. Contrary to the previous case, we may now transmit two signals at the same time, one for each transmitter. The signals can be expressed in vector form as

$$\begin{aligned}\begin{bmatrix} u_1(i) \\ u_2(i) \end{bmatrix} &= H^{-1} \begin{bmatrix} v_1(i) \\ v_2(i) \end{bmatrix} \\ &= \frac{1}{h_{11}h_{22} - h_{12}h_{21}} \begin{bmatrix} h_{22} & -h_{21} \\ -h_{12} & h_{11} \end{bmatrix} \begin{bmatrix} v_1(i) \\ v_2(i) \end{bmatrix} \\ &= \frac{1}{h_{11}h_{22} - h_{12}h_{21}} \begin{bmatrix} h_{22}v_1(i) - h_{21}v_2(i) \\ -h_{12}v_1(i) + h_{11}v_2(i) \end{bmatrix}\end{aligned}$$

for $i \in [l]$. We have now scaled, or *pre-coded*, by the channel matrix inverse. This technique is often called zero-forcing (ZF) pre-coding. Note that the transmitted signal at each transmitter involves both v_1 and v_2, and thus x_{21} and x_{22}. As such, ZF requires the data to be cached at both transmitters. The received signals are

$$\begin{bmatrix} y_1(i) \\ y_2(i) \end{bmatrix} = H \begin{bmatrix} u_1(i) \\ u_2(i) \end{bmatrix} + \begin{bmatrix} z_1(i) \\ z_2(i) \end{bmatrix} = H H^{-1} \begin{bmatrix} v_1(i) \\ v_2(i) \end{bmatrix} + \begin{bmatrix} z_1(i) \\ z_2(i) \end{bmatrix} = \begin{bmatrix} v_1(i) \\ v_2(i) \end{bmatrix} + \begin{bmatrix} z_1(i) \\ z_2(i) \end{bmatrix}$$

for $i \in [l]$. We thus created two *parallel* AWGN channels and carried out transmission in l time units. Compare this to the transmission of x_1, where we created two *serial* AWGN channels and carried out transmission over $2l$ time units. By caching the same data at multiple transmitters we can thus lower the transmission time.

3

System model

In this chapter a mathematical description of the MEC system under consideration is presented. At first the components of the system and the o□oading process are described. The o□oading process consists of three phases, each of which is explained in-depth in the subsequent sections. The most important principles and variables to remember are then summarized so that the reader can get an overview of the problem. The chapter is concluded by introducing a measure of performance, such that di□erent coding schemes can be evaluated and compared.

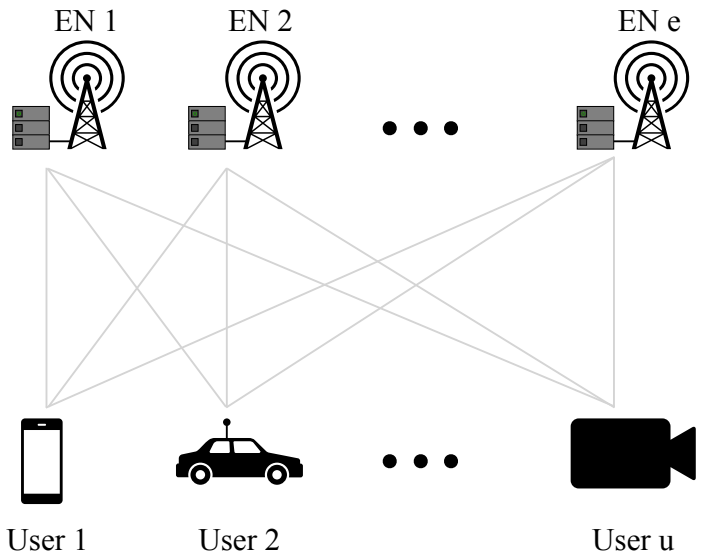


Figure 3.1: A general MEC network with e ENs and u users.

Consider a wireless network consisting of u single-antenna user devices in need of o□oading their computation tasks to e single-antenna ENs, as depicted in Figure 3.1. Consider specifically a scenario in which each user $i \in [u]$ has some length-r data vector x_i, and wants to compute the linear inference operation $y_i = W x_i$, for some $k \times r$ matrix W [23]. The elements are all from GF(q), where q is the power of a prime. The matrix is stored distributively across the ENs. The o□oading process consists of three phases: uplink, computation, and downlink. Each phase will now be explained in more detail.

Table 3.1: Scheduling table for the system in Example 1.

EN 1	EN 2	EN 3
1	2	3
4	5	6
7	8	9
10	11	12
3	1	2
6	4	5

3.1 Uplink phase

The process starts with the users transmitting their data such that each EN can construct the $r \times u$ matrix $X = [x_1, \ldots, x_u]$. We will assume that the transmission scheme is the same for all coding schemes, yielding the same transmission latency.

3.2 Computation phase

Consider now that the ENs are prone to straggling. For protection, W is encoded as $C = GW$ before the offloading process begins, where the $n \times k$ matrix G with elements from GF(q) is the generator of some code, and C is a matrix of size $n \times r$. The ENs store subsets of the rows of C of the same size. Let μ be the fraction of rows of the original matrix W that each EN can store at most, where $1/e \leq \mu \leq 1$. This constraint is enforced to make sure that each row of W can be stored at least one time in the uncoded case, and that no EN has an unrealistic storage capacity. Each EN is then responsible for computing the products between the rows that it stores and the matrix X. For some row c of C, we will refer to a product cX simply as a product. The order in which products are computed is described by a scheduling table, where the element in column i $[e]$ and row j $[\mu k]$ is the index of the jth product to be computed by EN i.

Example 1. *Consider a scenario with $e = 3$, $u = 3$, $\mu = 1$, and a matrix W with $k = 9$ rows. We encode W with a $(12, 9)$ MDS code and obtain $n = 12$ coded rows $c_1, \ldots, c_{12}$. Some of the rows are duplicated so that each EN stores 6 rows. A valid scheduling is depicted in Table 3.1. Looking at the table we can see that the first EN is responsible for computing the products $c_1X, c_4X, c_7X, c_{10}X, c_3X, c_6X$, in the given order.*

Following previous work, the straggling times of the ENs are modeled as random variables $\Lambda_1, \ldots, \Lambda_e \overset{\text{i.i.d}}{\sim} \exp(\eta)$ [15]. We define the vector of straggling times as $\Lambda = [\Lambda_1, \ldots, \Lambda_e]$. As soon as each EN has constructed X the computation phase begins, marked by time $t = 0$. At time $t = \Lambda_i$, EN i will begin computing products in the order designated by the corresponding column in the scheduling table. The

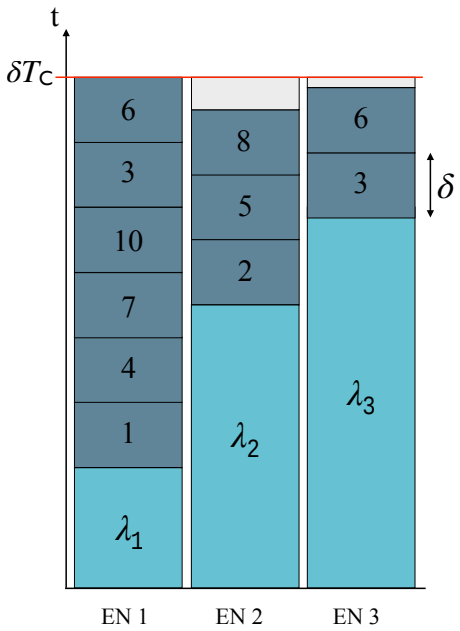


Figure 3.2: Computation phase scenario in Example 2.

ENs are assumed to have identical processing capabilities, such that the time it takes to compute a product is some deterministic value δ for each of them. The number of products completed by the ENs is represented by a length-e random process vector $D(t)$, where the ith element is the number of products completed by EN i at time t.

The computation phase is halted when a certain stopping criterion is satisfied. Assume some code and scheduling, and let d be a possible outcome of D at a given time. We say that d is a stopping vector if we can recover the desired results $y_1, \ldots, y_u$ by decoding the computed products. To control the computation phase we introduce a stopping set S of stopping vectors, such that the computation phase ends as soon as $D(t) \in S$. The stopping set is designed with the code and scheduling in mind and need not necessarily contain all possible stopping vectors. Following the description above we define the computation latency as

$$T_C \triangleq \frac{1}{\delta} \min\{t : D(t) \in S\}. \tag{3.1}$$

Assuming that the code, scheduling, and stopping set are known, T_C is a function of Λ and we will therefore sometimes write $T_C(\Lambda)$.

Example 2. *Consider the scenario in Example 1 with scheduling in Table 3.1. Since we are using a* $(12, 9)$ *MDS code, we may recover the results* y_1, y_2, *and* y_3 *from any computation containing* 9 *or more distinct products. Thus, valid stopping vectors are, e.g.,* $[3, 3, 3]$, $[6, 3, 0]$, *and* $[6, 3, 2]$. *A valid stopping set is* $S = \{[3, 3, 3], [6, 3, 2]\}$. *An example computation phase is depicted in Figure 3.2, where the straggling times are such that* $D = [6, 3, 2]$ *at the end of the computation.*

Since a row of C may be stored at several ENs, the corresponding product may have been computed more than once. We therefore introduce M_i, $i \in [n]$, as

Table 3.2: Example of transmission layout.

block	user 1	user 2	user 3	user 4
1	1	1	1	
2	2	2		1
3	3		2	2
4		3	3	3

the number of times product $c_i X$ has been computed across the ENs by time T_C. By definition M_i must satisfy $0 \leq M_i \leq e$. We will hereafter refer to M_i as the multiplicity of product $c_i X$.

We denote by P the total number of products computed by time T_C. Note that P includes duplicates of computed products, and that it can be determined by knowing the stopping set S and realization of Λ

For various reasons we might not want to transmit all computed products to the users. For example, the ENs might have computed more distinct products than are needed in order for the users to recover their data, i.e., to decode. Transmitting such redundant products would increase the communication latency but result in no benefits. Let V be the set of indices of all distinct products chosen for transmission. Note that at least k products are needed in order for the users to recover their data, which means that $|V| \geq k$. V will sometimes informally be referred to as containing products, while technically it contains the *indices* of products.

Example 3. *For the computation phase scenario considered in Example 2 and depicted in Figure 3.2 the total number of computed products is* $P = 11$. *There are* 9 *distinct products, so that* $V = \{1, \ldots, 8, 10\}$. *The multiplicities are* $M_3 = M_6 = 2$ *and* 1 *for the rest.*

3.3 Downlink phase

We define the communication latency as

$$T_D \triangleq \lim_{\sigma \to \infty} \frac{S}{u \log_2(q) / \log_2(1+\sigma)}$$

where S is the total transmission time, $u \log_2(q)$ is the data size of a product in bits, and $\log_2(1+\sigma)$ is the channel capacity of a discrete-time AWGN channel with signal-to-noise ratio σ. We adopt the transmission strategy of [23], which is based on [33]. The strategy is best explained by an example.

Example 4. *Consider a product* cX *which has been computed by* 3 *ENs. There are* 4 *users, such that* $cX = [cx_1, cx_2, cx_3, cx_4]$ *where user* j *desires the* $\log_2(q)$*-bit product* cx_j. *Let us divide* cx_j *into* 3 *packets* $cx_j^{(1)}$, $cx_j^{(2)}$, $cx_j^{(3)}$, *each of*

$\log_2(q)/3$ *bits. Transmission is carried out by sending* 3 *packets at a time, over* 4 *blocks. The transmission strategy is illustrated in Table 3.2, where the index of the* i*th user column and* j*th row represents the packet that will be sent to user* i *in block* j*.*

Consider now the transmission of packets $\boldsymbol{c}\boldsymbol{x}_1^{(1)}$, $\boldsymbol{c}\boldsymbol{x}_2^{(1)}$, $\boldsymbol{c}\boldsymbol{x}_3^{(1)}$ *in block* 1*. We map packet* $\boldsymbol{c}\boldsymbol{x}_i^{(1)}$ *onto a sequence* $s_i(t)$, $t \in [t]$*, of complex symbols, where* $t = \frac{\log_2(q)/3}{\log_2(1+\tilde{\sigma})}$ *and* $\tilde{\sigma}$ *is the transmit power. Let* $\boldsymbol{H}$ *be the channel matrix between the ENs and the first three users. Assuming* $\boldsymbol{H}$ *is invertible, the ENs transmit the signals*

$$\begin{bmatrix} u_1(t) \\ u_2(t) \\ u_3(t) \end{bmatrix} = \boldsymbol{H}^{-1} \begin{bmatrix} s_1(t) \\ s_2(t) \\ s_3(t) \end{bmatrix}$$

for $t \in [t]$*, where we have pre-coded by the channel matrix inverse. This is the ZF technique presented in Section 2.6. Note that the symbol sequences must be known to all ENs. The three first users then receive the signals*

$$\begin{bmatrix} y_1(t) \\ y_2(t) \\ y_3(t) \end{bmatrix} = \boldsymbol{H} \begin{bmatrix} u_1(t) \\ u_2(t) \\ u_3(t) \end{bmatrix} + \begin{bmatrix} z_1(t) \\ z_2(t) \\ z_3(t) \end{bmatrix} = \begin{bmatrix} s_1(t) \\ s_2(t) \\ s_3(t) \end{bmatrix} + \begin{bmatrix} z_1(t) \\ z_2(t) \\ z_3(t) \end{bmatrix}$$

for $t \in [t]$*, where* $z_i(t)$ *is complex white Gaussian noise of unit power, independent across users and time. We have thus created* 3 *parallel and independent AWGN channels, on which the users can recover the transmitted symbols with high probability if* $\tilde{\sigma}$ *is large enough. Transmitting the other packets in the same way results in a total transmission time of* $4t$*. Then the entire product* $\boldsymbol{c}\boldsymbol{X}$ *is transmitted in*

$$\lim_{\tilde{\sigma}\to\infty} \frac{4t}{u\log_2(q)/\log_2(1+\tilde{\sigma})} = \lim_{\tilde{\sigma}\to\infty} \frac{\frac{4\log_2(q)/3}{\log_2(1+\tilde{\sigma})}}{\frac{u\log_2(q)}{\log_2(1+\tilde{\sigma})}} = \frac{1}{3}$$

(normalized) time units.

Applying the results from [33] one can show that a computed product $\boldsymbol{c}_i\boldsymbol{X}$ can be transmitted in $1/\min\{M_i, u\}$ time units [23]. We will assume that $e \leq u$, such that $M_i \leq u, \quad i \in [n]$. The products are transmitted sequentially and T_{D} can therefore be expressed as

$$T_{\mathrm{D}} = \sum_{i \in V} \frac{1}{M_i}. \tag{3.2}$$

Let $G|_V$ and $C|_V$ be the matrices with rows corresponding to the indices in V, such that $C|_V = G|_V W$. User j will have received $C|_V x_j$ at the end of the downlink phase, such that it can decode onto $G|_V^{-1} C|_V x_j = G|_V^{-1} G|_V W x_j = W x_j = y_j$, where $G|_V^{-1}$ is the left inverse of $G|_V$. While this is not how decoding is carried out in practice (see, e.g., Section 2.4.2 for inactivation decoding of LT codes), it serves to illustrate the principle.

3.3.1 Performance measure

The performance of the system is measured in terms of the total latency τ, defined as

$$\tau(\gamma) \triangleq \mathrm{E}[T_{\mathrm{C}}] + \gamma \mathrm{E}[T_{\mathrm{D}}],$$

where the average is over Λ and γ is a parameter used for modeling the relative weight between the computation and communication latency. Since we have assumed that the uplink transmission latency is the same for any coding scheme, it is not included in the total latency.

3.3.2 Key concepts

Some parts from this section will be used in the rest of the paper and we therefore summarize the key concepts to remember. User j [u] wants to compute $y_j = W x_j$. The k rows of W are encoded to form the n rows of matrix C. These rows are distributed across the ENs, possibly such that a row is duplicated. The order in which the ENs compute their assigned products is determined by a scheduling table. The straggling times $\Lambda_1, \ldots, \Lambda_e$ of the ENs are i.i.d. exponentially distributed with parameter η. A stopping set S is needed in order to determine when to stop the computation phase. The computation latency is denoted by T_{C}. The total number of computed products is denoted by P. The set V contains the indices of all distinct products scheduled for transmission in the downlink, where $|V| \geq k$. The number of times product $c_i X$ has been computed across the ENs by the end of the computation phase is denoted by M_i. Lastly, we defined T_{D} as the communication latency and gave an expression for it under ZF pre-coding.

Note that our freedom of design encompasses the code, scheduling, and stopping set S. We will therefore specifically refer to this combination as the *coding scheme*. A coding scheme and realization of Λ completely specify T_{C}, P, V, T_{D}, and multiplicities $M_1, \ldots, M_n$.

4

Converse bound

Bounds are important to understand the performance limits of a system. In this chapter a lower bound on the total latency $\tau(y)$ is derived. Bounds that express limits along the lines of *one cannot do better than this* are called converse bounds. Note that such a bound reveals nothing about the possibility of achieving it. We can for example claim that $\tau(y) \geq 0$ although it is clear that no practical system could ever have a latency of zero. This chapter begins with a lemma needed to prove the subsequent theorem. The result of the first theorem is then used to prove the main theorem at the end of the chapter.

Lemma 1. *Let a and b be positive, real-valued variables satisfying $a > b$. Then the function*

$$f(a,b) = \begin{cases} \frac{b^2}{a} & \text{if } a/b \quad \mathbb{N} \\ \frac{a/b\ b-a}{a/b} + \frac{a-b\ a/b}{a/b} & \text{otherwise} \end{cases} \tag{4.1}$$

is monotonically decreasing in a and monotonically increasing in b.

Proof. Clearly, the term b^2/a is monotonically decreasing in a and monotonically increasing in b.

We now prove monotonicity in a for the second case in (4.1). Let $a_1 = cb$ and $a_2 = (c+1)b$ for some positive integer c. If we only consider values of a in the interval $a_1 < a < a_2$ the fraction a/b can not be an integer and thus $f(a,b)$ is evaluated by the second expression. Furthermore, we can conclude that $a/b = a_2/b = c+1$ and $a/b = a_1/b = c$. Then

$$\begin{aligned} f(a,b) &= \frac{(c+1)b-a}{c} + \frac{a-cb}{c+1} \\ &= \frac{b}{c}\frac{(c+1)b-a}{b} + \frac{b}{c+1}\frac{a-cb}{b} \\ &= \frac{b}{c}\left(1 - \frac{a-cb}{b}\right) + \frac{b}{c+1}\frac{a-cb}{b} \\ &= \frac{b}{c}\left(1 - \frac{a-cb}{(c+1)b-cb}\right) + \frac{b}{c+1}\frac{a-cb}{(c+1)b-cb} \\ &= \frac{b}{c}\left(1 - \frac{a-a_1}{a_2-a_1}\right) + \frac{b}{c+1}\frac{a-a_1}{a_2-a_1} \\ &= \frac{b^2}{a_1}\left(1 - \frac{a-a_1}{a_2-a_1}\right) + \frac{b^2}{a_2}\frac{a-a_1}{a_2-a_1}. \end{aligned} \tag{4.2}$$

Let $\alpha = \frac{a-a_1}{a_2-a_1}$, such that α goes from 0 to 1 as a goes from a_1 to a_2. Then (4.2) can be rewritten as

$$f(a,b) = \frac{b^2}{a_1}(1-\alpha) + \frac{b^2}{a_2}\alpha.$$

This is a convex combination of the values b^2/a_1 and b^2/a_2, where $b^2/a_1 > b^2/a_2$. The function must therefore be monotonically decreasing on the interval $a_1 < a < a_2$. Now note that at the end points (a_1, b) and (a_2, b) of this interval, the function $f(a,b)$ is evaluated by the first expression in (4.1), yielding exactly b^2/a_1 and b^2/a_2, respectively. Thus, the first expression fills in the points for which the discontinuous second expression would evaluate to zero, making $f(a,b)$ continuous in a on the interval $a_1 \le a \le a_2$. Recall that a takes values in $[b, \infty)$. This interval can be seen as a concatenation of intervals $[cb, (c+1)b]$ for $c = 1, 2, \ldots$, and so $f(a,b)$ must be continuous and monotonically decreasing in a on the entire interval $[b, \infty)$.

We now prove monotonicity in b. Let d be a positive integer and define $b_1 = a/(d+1)$ and $b_2 = a/d$. Consider b on the interval $b_1 < b < b_2$, such that $a/b = a/b_1 = d+1$ and $a/b = a/b_2 = d$. Then

$$\begin{aligned}
f(a,b) &= \frac{(d+1)b-a}{d} + \frac{a-db}{d+1} \\
&= \frac{a}{d}\frac{(d+1)b-a}{a} + \frac{a}{d+1}\frac{a-db}{a} \\
&= \frac{a}{d}\frac{(d+1)b-a}{a+da-da} + \frac{a}{d+1}\frac{a-db}{a+da-da} \\
&= \frac{d}{d}\frac{a}{d}\frac{(d+1)b-a}{(d+1)a-da} + \frac{d+1}{d+1}\frac{a}{d+1}\frac{a-db}{(d+1)a-da} \\
&= \frac{a}{d^2}\frac{(d+1)db-ad}{(d+1)a-da} + \frac{a}{(d+1)^2}\frac{(d+1)a-(d+1)db}{(d+1)a-da} \\
&= \frac{a}{d^2}\frac{(d+1)db-ad}{(d+1)a-da} + \frac{a}{(d+1)^2}\left(1 - \frac{(d+1)db-da}{(d+1)a-da}\right) \\
&= \frac{a}{d^2}\frac{b-\frac{a}{d+1}}{\frac{a}{d}-\frac{a}{d+1}} + \frac{a}{(d+1)^2}\left(1 - \frac{b-\frac{a}{d+1}}{\frac{a}{d}-\frac{a}{d+1}}\right) \\
&= \frac{b_2^2}{a}\frac{b-b_1}{b_2-b_1} + \frac{b_1^2}{a}\left(1 - \frac{b-b_1}{b_2-b_1}\right).
\end{aligned} \tag{4.3}$$

Let $\beta = \frac{b-b_1}{b_2-b_1}$, such that β goes from 0 to 1 as β goes from b_1 to b_2. Then (4.3) can be rewritten as

$$f(a,b) = \frac{b_1^2}{a}(1-\beta) + \frac{b_2^2}{a}\beta,$$

which is a convex combination of the values b_1^2/a and b_2^2/a, where $b_1^2/a < b_2^2/a$. The function must therefore be increasing on the interval $b_1 < b < b_2$. Again, note that at the end points (a, b_1) and (a, b_2) of this interval, the function $f(a,b)$ is evaluated

by the first expression in (4.1), yielding exactly b_1^2/a and b_2^2/a, respectively. As such, the first expression fills in the points for which the discontinuous second expression would evaluate to zero, making $f(a, b)$ continuous in b on the interval $b_1 \leq b \leq b_2$. Recall that b takes values in $[0, a)$. This interval can be seen as a concatenation of the intervals $\left[\frac{a}{d+1}, \frac{a}{d}\right)$, $d = 1, 2, \ldots$, and so $f(a, b)$ must be continuous and monotonically increasing in b on the entire interval $[0, a)$. □

We now use the results of the lemma to derive a lower bound on the communication latency T_D.

Theorem 1 (Lower bound on the communication latency). *For any coding scheme and Λ resulting in P computed products, T_D can be lower bounded as*

$$T_D \geq T_D(P), \tag{4.4}$$

where

$$T_D(P) = \begin{cases} \frac{k^2}{P} & \text{if } P/k \in \mathbb{N} \\ \frac{P/k\ k-P}{P/k} + \frac{P-k\ P/k}{P/k} & \text{otherwise,} \end{cases} \tag{4.5}$$

and k is the number of rows in matrix W.

Proof. Consider an arbitrary coding scheme. Assume that we have a realization λ that leads to a total number of computed products p, set V with cardinality v, and set of multiplicities $\{m_i\}$. Let α be the average multiplicity of the products in V. Then αv is the total number of products in V, plus all corresponding duplicates. We can express αv as

$$\alpha v = \sum_{i \in V} m_i. \tag{4.6}$$

Under ZF pre-coding, the communication latency is given by

$$t_D = \sum_{i \in V} \frac{1}{m_i}. \tag{4.7}$$

Looking at (4.7) we see that t_D is completely specified by the number of products v in V, and the multiplicities $\{m_i\}$ of these products. Also note that the multiplicities are constrained by (4.6), since α and v are fixed. The aim of this proof is now to find the combination of multiplicities $\{m_i\}$ that minimizes t_D for a fixed α and v, and then to optimize over α and v to provide the most beneficial constraint given by (4.6).

To this end, we define v_m as the number of products in V of multiplicity m, where $1 \leq m \leq e$. We can then express v, αv, and t_D in terms of v_m as

$$v = \sum_{m=1}^{\infty} v_m \tag{4.8}$$

$$\alpha v = \sum_{m=1}^{\infty} m v_m \tag{4.9}$$

$$t_{\mathrm{D}} = \sum_{m=1}^{\infty} \frac{v_m}{m}. \tag{4.10}$$

Consequently, determining the best distribution of multiplicities $\{m_i\}$ is equivalent to determining the best distribution of $\{v_m\}$.

We will now first derive a bound that holds when α is an integer, and then a second bound that holds when α is not an integer. The final bound is then a combination of these two bounds.

First bound. Consider the case when α is an integer. If all products in V have the same multiplicity α, i.e., $v_m = v$ for $m = \alpha$ and zero otherwise, by (4.10) the latency is v/α. Comparing this to all other choices of $\{v_m\}$ we have

$$\begin{aligned}
t_{\mathrm{D}} - \frac{v}{\alpha} &= \sum_{m=1}^{\infty} \frac{v_m}{m} - \frac{v}{\alpha} \\
&\overset{(a)}{=} \sum_{m=1}^{\infty} \frac{v_m}{m} - \frac{1}{\alpha} \sum_{m=1}^{\infty} v_m \\
&= \sum_{m=1}^{\infty} v_m \frac{\alpha - m}{m\alpha} \\
&= \sum_{m=1}^{\alpha} v_m \frac{\alpha - m}{m\alpha} + \sum_{m=\alpha}^{\infty} v_m \frac{\alpha - m}{m\alpha} \\
&\geq \frac{1}{\alpha^2} \sum_{m=1}^{\alpha} v_m(\alpha - m) + \frac{1}{\alpha^2} \sum_{m=\alpha}^{\infty} v_m(\alpha - m) \\
&= \frac{1}{\alpha^2} \sum_{m=1}^{\infty} v_m(\alpha - m) \\
&\overset{(b)}{=} \frac{1}{\alpha^2}(\alpha v - \alpha v) \\
&= 0
\end{aligned}$$

where in (a) we used (4.8), and in (b) we used (4.8) and (4.9). The latency v/α is thus the lowest possible when α is an integer.

Second bound. Consider now the case when α is not an integer. The idea of the proof in this case is to partition V into two sets and then apply the same calculations as in the previous case to each of the subsets.

Let us partition V into the disjoint sets A and B, with $|A| = (\lceil \alpha \rceil - \alpha)v$ and $|B| = (\alpha - \lfloor \alpha \rfloor)v$. Denote by α_A and α_B the average multiplicity of the products in set A and B, respectively. Note that $\alpha_A |A|$ is the number of products in A plus all

its duplicates. The same holds for $\alpha_B |B|$. Together they must make up αv products, i.e., all products in V plus their corresponding duplicates. Then

$$\alpha v = \alpha_A(\alpha - \alpha)v + \alpha_B(\alpha - \alpha)v \tag{4.11}$$

Denote by v_m^A and v_m^B the number of products of multiplicity m in set A and B, respectively. Then

$$\alpha_A |A| = \sum_{i \in A} m_i = \sum_{m=1}^{\alpha} m v_m^A \tag{4.12}$$

and the same holds for B. We may also express t_{D} as

$$\begin{aligned} t_{\mathrm{D}} &= \sum_{i \in V} \frac{1}{m_i} \\ &= \sum_{i \in A} \frac{1}{m_i} + \sum_{i \in B} \frac{1}{m_i} \\ &= \sum_{m=1}^{\alpha} \frac{v_m^A}{m} + \sum_{m=1}^{\alpha} \frac{v_m^B}{m}. \end{aligned} \tag{4.13}$$

Consider the case when all products in A have the same multiplicity α and all products in B have the same multiplicity α, i.e., when $v_m^A = |A|$ for $m = \alpha$ and $v_m^B = |B|$ for $m = \alpha$, and zero otherwise. In this case $\alpha_A = \alpha$ and $\alpha_B = \alpha$, so that constraint (4.11) is satisfied. Then by (4.13) the latency is $\frac{|A|}{\alpha} + \frac{|B|}{\alpha}$. We would like to compare this latency with the latency of any other choice of $\{v_m^A\}$ and $\{v_m^B\}$. To this end, consider first

$$\begin{aligned} \sum_{m=1}^{\alpha} \frac{v_m^A}{m} - \frac{|A|}{\alpha} &\overset{(a)}{=} \sum_{m=1}^{\alpha} \frac{v_m^A}{m} - \frac{1}{\alpha} \sum_{m=1}^{\alpha} v_m^A \\ &= \sum_{m=1}^{\alpha} v_m^A \frac{\alpha - m}{m\alpha} \\ &= \sum_{m=1}^{\alpha} v_m^A \frac{\alpha - m}{m\alpha} + \sum_{m=\alpha}^{\alpha} v_m^A \frac{\alpha - m}{m\alpha} \\ &\geq \frac{1}{\alpha\alpha} \sum_{m=1}^{\alpha} v_m^A(\alpha - m) + \frac{1}{\alpha\alpha} \sum_{m=\alpha}^{\alpha} v_m^A(\alpha - m) \\ &= \frac{1}{\alpha\alpha} \sum_{m=1}^{\alpha} v_m^A(\alpha - m) \\ &\overset{(b)}{=} \frac{\alpha |A| - \alpha_A |A|}{\alpha\alpha} \end{aligned}$$

where we used $\sum_{m=1}^{\alpha} v_m^A = |A|$ in both (a) and (b), and in (b) we also used (4.12). Similarly, for B we have

$$\begin{aligned}
\sum_{m=1}^{\alpha v} \frac{v_m^B}{m} - \frac{|B|}{\alpha} &= \sum_{m=1}^{\alpha v} \frac{v_m^B}{m} - \frac{1}{\alpha} \sum_{m=1}^{\alpha v} v_m^B \\
&= \sum_{m=1}^{\alpha v} v_m^B \frac{\alpha v - m}{m \alpha v} \\
&= \sum_{m=1}^{\alpha v'} v_m^B \frac{\alpha v - m}{m \alpha v} + \sum_{m=\alpha v'}^{\alpha v} v_m^B \frac{\alpha v - m}{m \alpha v} \\
&\geq \frac{1}{\alpha v \alpha v'} \sum_{m=1}^{\alpha v'} v_m^B (\alpha v' - m) + \frac{1}{\alpha v \alpha v'} \sum_{m=\alpha v'}^{\alpha v} v_m^B (\alpha v - m) \\
&= \frac{1}{\alpha v \alpha v'} \sum_{m=1}^{\alpha v} v_m^B (\alpha v' - m) \\
&= \frac{\alpha v' |B| - \alpha_B |B|}{\alpha v \alpha v'}.
\end{aligned}$$

We can then conclude that

$$\begin{aligned}
t_{\mathrm{D}} - \frac{|A|}{\alpha} - \frac{|B|}{\alpha} &= \\
&= \sum_{m=1}^{\alpha v} \frac{v_m^A}{m} + \sum_{m=1}^{\alpha v} \frac{v_m^B}{m} - \frac{|A|}{\alpha} - \frac{|B|}{\alpha} \\
&\geq \frac{\alpha v' |A| - \alpha_A |A|}{\alpha v \alpha v'} + \frac{\alpha v |B| - \alpha_B |B|}{\alpha v \alpha v'} \\
&= \frac{\alpha v' |A| + \alpha v |B| - (\alpha_A |A| + \alpha_B |B|)}{\alpha v \alpha v'} \\
&\overset{(a)}{=} \frac{\alpha v v' - \alpha v v'}{\alpha v \alpha v'} \\
&= 0,
\end{aligned}$$

where in (a) we used (4.11). Thus, the lowest possible latency for α not an integer is $\frac{|A|}{\alpha} + \frac{|B|}{\alpha} = \frac{\alpha v' v - \alpha v v'}{\alpha v} + \frac{\alpha v v' - \alpha v' v}{\alpha v'}$.

Final Bound. Letting $v' = \alpha v$ and combining the two bounds, we define

$$f(v, v') = \begin{cases} \frac{v^2}{v'} & \text{if } v'/v \in \mathbb{N} \\ \frac{v'/v \; v - v'}{v'/v} + \frac{v' - v \; v'/v}{v'/v} & \text{otherwise} \end{cases} \tag{4.14}$$

By Lemma 1, this function is monotonically decreasing in v and monotonically increasing in v'. Since $v' \leq p$ and $v \geq k$, we can therefore lower bound $f(v, v')$ by $t_{\mathrm{D}}(p)$, where

$$t_{\mathrm{D}}(p) = \begin{cases} \frac{k^2}{p} & \text{if } p/k \in \mathbb{N} \\ \frac{p/k \; k - p}{p/k} + \frac{p - k \; p/k}{p/k} & \text{otherwise.} \end{cases} \tag{4.15}$$

This bound holds for any coding scheme and λ resulting in p computed products. □

We are now ready to state and prove the main theorem of this thesis.

Theorem 2 (Lower bound on the total latency). *For any coding scheme, the total latency can be lower bounded as*

$$\tau(\gamma) \geq \mathrm{E}\left[\min_{p \in \{k,\ldots,e\mu k\}} T_{\mathrm{C}}(\Lambda, p) + \gamma T_{\mathrm{D}}(p)\right], \tag{4.16}$$

where k is the number of rows in matrix $\boldsymbol{W}$, $T_{\mathrm{C}}(\Lambda, p)$ is the computation latency when waiting for p products in total and $T_{\mathrm{D}}(p)$ is given by

$$T_{\mathrm{D}}(p) = \begin{cases} \frac{k^2}{p} & \text{if } p/k \in \mathbb{N} \\ \frac{p/k\ k-p}{p/k} + \frac{p-k\ p/k}{p/k} & \text{otherwise.} \end{cases}$$

Proof. The aim of this proof is to lower bound $\tau(\gamma)$ until the bound does not depend on some underlying coding scheme. Consider

$$\begin{aligned} \tau(\gamma) &= \mathrm{E}[T_{\mathrm{C}}(P) + \gamma T_{\mathrm{D}}(\Lambda)] \\ &\geq \mathrm{E}[T_{\mathrm{C}}(P) + \gamma T_{\mathrm{D}}(P)] \\ &\geq \mathrm{E}\left[\min_{p \in \{k,\ldots,e\mu k\}} T_{\mathrm{C}}(\Lambda, p) + \gamma T_{\mathrm{D}}(p)\right]. \end{aligned} \tag{4.17}$$

Recall that the computation phase results in P computed products in total. In the first inequality we thus begin by choosing the smallest possible communication latency for P computed products, which follows from Theorem 1. However, to evaluate this expression we would need to assume a stopping set, which is part of the coding scheme. This is because a stopping set and Λ completely determines T_{C} and P.

Consider then the last inequality. We here take control over P and choose the value that would minimize the total latency for the given Λ. Note that we do not know if there exists a coding scheme that for the given Λ would result in the optimal P. Neither do we know if there exists a coding scheme that would result in the optimal P *at all times*. This is why the bound is a converse.

We still need to know the value of Λ in order to determine the optimal number of products to compute and T_{C}. The dependence of T_{C} on both p and Λ is therefore specifically highlighted. On the other hand, $T_{\mathrm{D}}(P)$ depends on Λ through P and as such, $T_{\mathrm{D}}(p)$ does not depend on Λ. □

The bound can be interpreted as using the best possible coding scheme for every outcome of Λ. This is of course practically impossible since the coding scheme must be set before the system is used and not during runtime.

5
Coding schemes

Recall that the code, scheduling, and stopping set S constitute the freedom of design of the system. In this chapter we present three different coding schemes that will be evaluated by simulations in the next chapter. The first two coding schemes are based on LT codes and inactivation decoding. The third scheme was presented in [23] and is based on MDS codes. It will serve as a benchmark for the two LT schemes designed in this thesis.

As discussed in Section 2.4.3 LT codes are random and can be decoded if $k + \Phi$ out of the n transmitted output symbols are received intact. If $k + \varphi$ output symbols are collected there is a risk that decoding will not succeed, since Φ is random. We referred to the probability of this event as the decoding failure probability $P_F(\varphi) = \Pr(\Phi \geq \varphi)$. Let us therefore accept some pre-determined failure probability P_e. Note that there exists an overhead $\varphi_{\min}$ such that if $k + \varphi_{\min}$ output symbols are always collected the probability of decoding failure will be P_e. In fact, $\varphi_{\min}$ can be found by numerically solving the equation $P_e = P_F(\varphi_{\min})$, where the upper bound on $P_F(\varphi)$ presented in (2.1) is convenient to use.

5.1 LT scheme

We encode the k rows of W by a fixed-rate LT code to get $e\mu k$ rows $c_1, \ldots, c_{e\mu k}$. The scheduling is depicted in Table 5.1. The stopping set is designed as

$$S = \left\{ s : \sum_{i=1}^{e\mu k} s_i = k + \varphi_{\min},\ 0 \leq s_i \leq \mu k \right\}. \tag{5.1}$$

The computation phase is thus halted as soon as the first $k + \varphi_{\min}$ products have been computed. We are now ready to express the total latency of the LT scheme.

Proposition 1 (Total latency of the LT scheme).

$$\tau(\gamma) = \mathrm{E}[T_C] + \gamma(k + \varphi_{\min}). \tag{5.2}$$

Proof. The ENs will always compute $k + \varphi_{\min}$ products in total and each product will have multiplicity 1. Then from (3.2) we have $T_D = \sum_{i \in V} 1/M_i = k + \varphi_{\min}$. □

5.2 LT-repetition scheme

Let ρ_1 and ρ_2 be positive, rational numbers. We encode the rows of W by a fixed-rate LT code to get $\rho_1 k$ coded rows $c_1, \ldots, c_{\rho_1 k}$. These are then repeated ρ_2 times,

Table 5.1: Scheduling for a general LT scheme.

EN 1	EN 2	...	EN e
1	2	...	e
$e + 1$	$e + 2$	...	$2e$
...	...	...	...
$e(\mu k - 1) + 1$	$e(\mu k - 1) + 2$	...	$e\mu k$

Table 5.2: Scheduling for Example 5

EN 1	EN 2	EN 3	EN 4
1	2	3	4
5	6	7	8
9	10	11	12
4	1	2	3
8	5	6	7
12	9	10	11
3	4	1	2
7	8	5	6

resulting in $\rho_1\rho_2 k$ rows in total. If ρ_2 is not an integer, $(\rho_2 - \rho_2)\rho_1 k$ of the $\rho_1 k$ LT-coded rows are repeated ρ_2 times, and the remaining $(\rho_2 - \rho_2)\rho_1 k$ rows are repeated ρ_2 times. Each EN stores $\rho_1\rho_2 k/e$ rows. The scheduling is best described by an example.

Example 5. *For an LT-repetition scheme with parameters $e = 4$, $k = 6$, $\rho_1 = 2$, and $\rho_2 = 8/3$ the scheduling is depicted in Table 5.2. To begin, the $\rho_1 k = 12$ LT-coded rows are assigned to the upper $\frac{\rho_1 k}{e} \times e$ block of the scheduling table, from left to right, top to bottom. A duplicate of the block is created, circularly shifted and placed beneath the upper block. A duplicate of the second block is created, out of which the top 2 rows are singled out, circularly shifted, and placed beneath the second block.*

In general, we have

$$\rho_1 \quad \{1\} \quad \left\{ \frac{k + \varphi_{\min}}{k}, e\mu \right\}, \tag{5.3}$$

$$\rho_2 \quad [1, e\mu]. \tag{5.4}$$

where $\rho_1 = 1$ corresponds to the uncoded option, i.e., not using an LT code at all. Note that the specific pair $\rho_1 = (k + \varphi_{\min})/k$ and $\rho_2 = 1.0$ corresponds to the LT scheme in the previous section.

Since the total storage capacity of the ENs is $e\mu k$ rows and the total number of coded rows is $\rho_1\rho_2 k$, any valid combination of ρ_1 and ρ_2 must satisfy

$$\rho_1\rho_1 \leq e\mu. \tag{5.5}$$

We also enforce the constraint

$$\rho_1 \frac{k}{e} \quad \mathbb{N} \tag{5.6}$$

to make sure that no row in the scheduling table is partly filled. We design the stopping set as

$$S(p) = \left\{ s : \sum_{i=1}^{e} s_i \geq p,\ 0 \leq s_i \leq \rho_1\rho_2 k/e,\ |V| = k + \varphi_{\min} \right\}, \tag{5.7}$$

where

$$p \quad \{k, \ldots, \rho_1\rho_2 k\} \tag{5.8}$$

is the least number of products that we want to compute in total. Thus, the computation stops when $k + \varphi_{\min}$ distinct products have been computed *and* at least p products have been computed in total. The parameter p has a similar function to the p considered in the minimization of the converse bound in (4.16); it is used to find the specific LT-repetition coding scheme that provides the lowest total latency on average.

Proposition 2 (Total latency of LT-repetition scheme).

$$\tau(\gamma) = \min_{\rho_1,\rho_2,p} \mathrm{E}[T_{\mathrm{C}} + \gamma T_{\mathrm{D}}] \tag{5.9}$$

under the constraints given by (5.3), (5.4), (5.5), (5.6), *and* (5.8).

5.3 MDS-repetition scheme

Zhang *et al*. presented in [23] a coding scheme which they call the MDS hybrid scheme. It will be used as a benchmark in this thesis and we therefore state some of its details here, with some minor modifications. A thorough explanation of the scheme can be found in the original paper. For consistency, we will refer to the scheme as the MDS-repetition scheme.

Let ρ_1 be a positive, rational number. The k rows of W are encoded using a $(\rho_1 k, k)$ MDS code and then repeated ρ_2 times to get $\rho_1\rho_2 k$ coded rows in total. In contrast to the LT-repetition scheme, ρ_2 must now be an integer. The scheduling is designed by dividing the $\rho_1 k$ MDS-coded rows into batches and distribute the batches across the ENs. As with the LT-repetition scheme the storage condition

$$\rho_1\rho_2 \leq e\mu \tag{5.10}$$

must be satisfied.

The stopping set is designed such that the system waits for the quickest ξ ENs to finish their entire assigned set of products, where $0 \leq \xi \leq e$. Any products computed by ENs that are not among the ξ quickest are discarded, even if these products are duplicates of the products computed by the ξ quickest ENs, and as such could be used to lower the communication latency. This restriction is solely enforced to enable an analysis of the total latency.

In order to guarantee a successful recovery of the results, ρ_1, ρ_2, and ξ must satisfy

$$\left\lceil \frac{e}{\rho_2} \right\rceil - \left\lceil \frac{e-\xi}{\rho_2} \right\rceil \geq \frac{1}{\rho_1} \left\lceil \frac{e}{\rho_2} \right\rceil . \tag{5.11}$$

Let $d_{\min} = \max\{\rho_2 - (e - \xi)\}$ and $d_{\max} = \max\{\xi, \rho_2\}$. Define

$$H(l) \triangleq \sum_{j=1}^{l} \frac{1}{j},$$

$$B(d) \triangleq \frac{\binom{e-\xi}{d}\binom{\rho_2}{e-d}\rho_1 k}{\binom{e}{\rho_2}}$$

for some $d \in [d_{\min}, d_{\max}]$. Let also

$$d_\xi \triangleq \inf\left\{ i : \sum_{d=i}^{d_{\max}} B(d) \leq k \right\}.$$

Proposition 3 (Total latency of the MDS-repetition scheme).

$$\tau(\gamma) = \min_{\xi} \left\{ \frac{H(e) - H(e-\xi)}{\eta\tau} + \frac{\rho_1\rho_2 k}{e} + \gamma \min_{\rho_1,\rho_2} \left\{ \sum_{d=d_\xi}^{d_{\max}} \frac{B(d)}{d} + \frac{m - \sum_{d=d_\xi}^{d_{\max}} B(d)}{d_\xi - 1} \right\} \right\} \tag{5.12}$$

where $\rho_1 \in [1, e\mu]$, $\rho_2 \in \{\xi\mu, \ldots, e\mu\}$, *and* $\xi \in \{1/\mu, \ldots, e\}$. *The parameters must satisfy constraints* (5.10) *and* (5.11).

6

Numerical results

The coding schemes were evaluated for the parameter combinations $e = 6$, $k = 600$, $\mu = 0.6$, and $\delta = 0.0005$. In (5.12) we provided an analytical expression for the total latency of the MDS-repetition scheme [23], which will be used as a benchmark. The total latencies for the two LT schemes in (5.2) and (5.9) contain expectations and as such must be evaluated by Monte Carlo simulations. The total latency for each scheme is depicted in Figure 6.1 for $\eta = 0.8$ (top curves) and $\eta = 8$ (bottom curves). Also included in the figure is the converse bound in (4.16). The bound contains an expectation and must therefore also be evaluated by simulation.

For the two LT schemes, we accept a decoding failure probability of $P_e = 10^{-4}$. Using the bound in (2.1) the overhead $\varphi_{\min} \approx 15$ is found numerically, which is relatively small. By (5.3), (5.4), and (5.8) the parameters of the LT-repetition scheme takes values as $\rho_1 \quad \{1\} \quad [615/600, 3.6]$, $\rho_2 \quad [1, 3.6]$ and $p \quad \{600, \ldots, 2160\}$ under constraints $\rho_1\rho_2 \leq 3.6$ and $100\rho_1 \quad \mathbb{N}$ given by (5.5) and (5.6).

Contrary to the LT-based schemes, the MDS-repetition scheme guarantees successful decoding. From Proposition 3 we have $\xi \quad \{2, \ldots, 6\}$, $\rho_1 \quad [1, 3.6]$, and $\rho_2 \quad \{0.6\xi, \ldots, 3\}$. The parameters must also satisfy constraints (5.10) and (5.11). The resulting parameter choices for the LT-repetition and MDS-repetition scheme are presented in Table 6.1 for given values of γ.

Recall that the converse bound can be interpreted as choosing the best possible coding scheme for every Λ, i.e., choosing the scheme during runtime. In practice the coding scheme is set before the system is in use, which is of course the case with the two LT schemes and the MDS-repetition scheme. The pre-set coding scheme will most likely not be optimal for every Λ and we would therefore expect the schemes to perform worse than the converse on average. By looking at Figure 6.1 this is indeed the case. Furthermore, a smaller η will yield more severe straggling and a larger spread of the straggling times. We would therefore expect it to be less likely that a specific coding scheme is optimal for most realizations of Λ, since the system would operate in a highly varying environment. This is indeed also the case; we see in general that the schemes perform further away from the converse bound for $\eta = 0.8$ as compared to $\eta = 8$. Another consequence of severe straggling is that the computation latency will increase on average. This is why the curves for $\eta = 0.8$ are shifted higher up than the curves for $\eta = 8$.

Consider the results for $\eta = 0.8$. For small values of γ the computation latency $\mathrm{E}[T_C]$ dominates the total latency and we would therefore expect it to be advantageous to use codes with good protection against stragglers. By looking at Table 6.1 we see that both hybrid schemes result in their respective largest ρ_1 and smallest ρ_2, which corresponds to using only LT and MDS codes. These codes provide better

Table 6.1: Design parameter values of LT- and MDS-repetition schemes for simulations with $e = 6$, $k = 600$, $\mu = 0.6$, and $\delta = 0.0005$. The values are given for $\eta = 0.8$ without parentheses and for $\eta = 8$ inside parentheses.

	LT-repetition			MDS-repetition		
y	ρ_1	ρ_2	p	ρ_1	ρ_2	ξ
0	3.6 (3.4)	1.0 (1.0)	600 (600)	3.0 (2.0)	1.0 (1.0)	2 (3)
1	3.6 (3.4)	1.0 (1.0)	600 (1320)	3.0 (2.0)	1.0 (1.0)	2 (4)
2	3.6 (1.0)	1.0 (3.6)	600 (1620)	3.0 (1.0)	1.0 (3.0)	2 (5)
3	3.6 (1.0)	1.0 (3.6)	600 (1740)	3.0 (1.0)	1.0 (3.0)	2 (5)
4	1.0 (1.0)	3.6 (3.6)	1080 (1800)	3.0 (1.0)	1.0 (3.0)	2 (5)
5	1.0 (1.0)	3.6 (3.6)	1080 (1800)	3.0 (1.0)	1.0 (3.0)	2 (5)
6	1.0 (1.0)	3.6 (3.6)	1080 (1800)	1.0 (1.0)	3.0 (3.0)	4 (6)
7	1.0 (1.0)	3.6 (3.6)	1440 (1800)	1.0 (1.0)	3.0 (3.0)	4 (6)
8	1.0 (1.0)	3.6 (3.6)	1440 (2040)	1.0 (1.0)	3.0 (3.0)	4 (6)
9	1.0 (1.0)	3.6 (3.6)	1440 (2040)	1.0 (1.0)	3.0 (3.0)	4 (6)
10	1.0 (1.0)	3.6 (3.6)	1440 (2100)	1.0 (1.0)	3.0 (3.0)	4 (6)

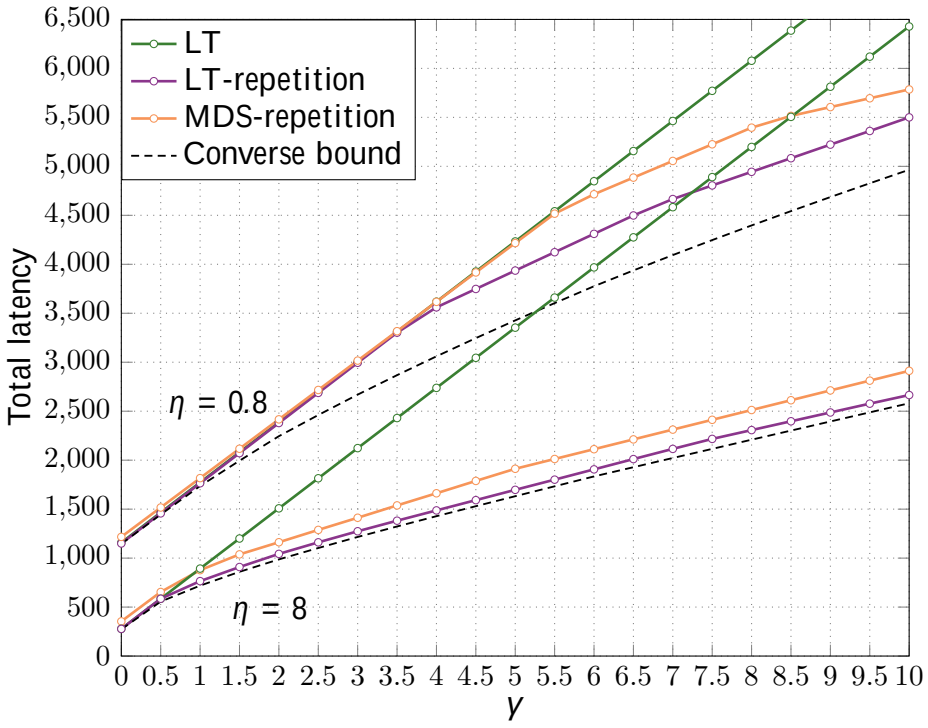


Figure 6.1: Total latency $\tau(y)$ as a function of y for parameters $e = 6$, $k = 600$, $\mu = 0.6$, $\delta = 0.0005$, and two values of η.

straggler protection than repetition codes, and we can therefore conclude that our expectation is correct. Note that with $\rho_1 = 3.6$ and $\rho_2 = 1.0$ the LT-repetition scheme is identical to the LT scheme, which is why the corresponding curves are on top of each other for small γ.

For small γ we would also expect the schemes to opt for computing as few products as possible in order to lower the computation latency. As discussed, the LT-repetition scheme is identical to the LT scheme for small γ, and the ENs will therefore compute the minimum number of products $k + \varphi_{\min}$. The MDS-repetition scheme stops the computation phase when the smallest possible number of ENs have completed their products, which is $\xi = 2$. The results thus seem to be in order. Similarly, in the minimization in (4.16) the converse bound favors small values of p for all Λ, i.e., it favors computing as few products as possible in order to obtain a low computation latency. This is why the schemes perform close to the converse for small γ.

As γ grows the communication latency $\mathrm{E}[T_{\mathrm{D}}]$ has a larger impact on the total latency and it is therefore more beneficial to use a repetition code rather than an MDS or an LT code. This is why the LT-repetition scheme tunes into $\rho_1 = 1.0$ and $\rho_2 = 3.6$ at roughly $\gamma = 4$, which corresponds to no LT code and maximum amount of repetition. The MDS-repetition scheme similarly converts to a pure repetition code at roughly $\gamma = 6$. Recall that a repetition code not only decreases the communication latency, but also protects against straggling servers. This is why it is so beneficial to go for repetition as γ grows large enough.

The LT-repetition scheme has a better performance than the MDS-repetition scheme for all considered values of γ. In general this is a consequence of the MDS-repetition scheme discarding products computed by ENs that are not among the ξ quickest, which is clearly a wasteful strategy. On the other hand, the LT-repetition scheme avoids this by accepting products computed by any EN. The difference in outcome for the two strategies is clearly seen for larger γ. Here the inclines of the curves of the MDS and LT-repetition schemes are roughly the same, which tells us that the communication latency is roughly the same. However, to achieve the same communication latency the MDS-repetition scheme needs to compute more products than the LT-repetition scheme and thus have a higher computation latency on average. Lastly, the LT scheme can not adapt to a changing γ and its curve is therefore a straight line.

For $\eta = 8$, the results follow the same principles as for $\eta = 0.8$. Note however that the two hybrid schemes convert to pure repetition at a smaller γ than in the $\eta = 0.8$ case. This is because the computation latency is lower, which means that the communication latency has a larger impact on the total latency. As such, the schemes opt for a lot of repetition and computing a lot of products since the decrease in communication latency outweighs the increase in computation latency.

7
Conclusion

We considered latency-critical distributed matrix-vector multiplication in a mobile edge computing scenario consisting of multiple users and multiple ENs. The results indicate that coding has the potential to lower the overall latency by providing protection against straggling ENs and enabling transmission cooperation in the downlink, at the cost of an increased computational load. We presented two coding schemes based on Luby-Transform (LT) codes and inactivation decoding, which has a lower decoding complexity than the MDS-coded scheme recently presented by Zhang *et al* [23]. One scheme is based solely on an LT code and is aimed at providing a low computation latency by computing as few products as possible, and avoiding stragglers. The other scheme is based on a concatenation of an LT code and a repetition code. In this case the idea is to leverage both the straggler protection provided by the LT code and the cooperative transmission enabled by the repetition code. The results show that our LT-repetition scheme outperforms the MDS-repetition scheme from [23] for most parameter combinations.

We also derived a lower bound on the total latency, which can be interpreted as choosing the optimal coding scheme for every outcome of straggling. However, in practice a coding scheme is chosen before the system is in use, and can not be changed during runtime. Furthermore, even if a coding scheme performs well on average, it is unlikely to perform well at all times. In general, this causes a discrepancy between the bound and the curves of the coding schemes. For certain choices of parameters the schemes do however perform close to the bound. This is especially the case when the computation latency is critical.

This work will lead to the submission of a journal article, in which additional results will be presented. Importantly, the time it takes for users to decode will be taken into account. This is in fact vital to investigate, since the decoding time might increase the total latency up to the point where it would be quicker for a user to process the data by itself, instead of o□oading it.

Another factor to take into account in future work is the cost of coordinating the ENs in the computation phase. Our LT-coded schemes most likely require a larger number of messages sent between the ENs than the MDS-coded scheme by Zhang *et al*, which will increase the communication load. Lastly, there are indications that the converse bound potentially can be tightened, and it would be interesting to investigate whether this is indeed plausible.

Bibliography

[1] Pavel Mach and Zdenek Becvar. Mobile Edge Computing: A Survey on Architecture and Computation O□oading. *IEEE Communications Surveys and Tutorials*, 19(3):1628–1656, 2017.

[2] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. Mobile Edge Computing: A Survey. *IEEE Internet of Things Journal*, 5(1):450–465, 2018.

[3] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. Mobile Edge Computing A key technology towards 5G. *ETSI White Paper No. 11 Mobile*, (11):1–16, 2015.

[4] Changsheng You, Kaibin Huang, Hyukjin Chae, and Byoung Hoon Kim. Energy-E□cient Resource Allocation for Mobile-Edge Computation O□oading. In *IEEE Transactions on Wireless Communications*, volume 16, pages 1397–1411. IEEE, 2017.

[5] Ke Zhang, Yuming Mao, Supeng Leng, Quanxin Zhao, Longjiang Li, Xin Peng, Li Pan, Sabita Maharjan, and Yan Zhang. Energy-E□cient O□oading for Mobile Edge Computing in 5G Heterogeneous Networks. *IEEE Access*, 4:5896–5907, 2016.

[6] Thinh Quang Dinh, Jianhua Tang, Quang Duy La, and Tony Q.S. Quek. O□oading in Mobile Edge Computing: Task Allocation and Computational Frequency Scaling. *IEEE Transactions on Communications*, 65(8):3571–3584, 2017.

[7] Stefania Sardellitti, Gesualdo Scutari, and Sergio Barbarossa. Joint optimization of radio and computational resources for multicell mobile-edge computing. *IEEE Transactions on Signal and Information Processing over Networks*, 1(2):89–103, 2015.

[8] Yanting Wang, Min Sheng, Xijun Wang, Liang Wang, and Jiandong Li. Mobile-Edge Computing: Partial Computation O□oading Using Dynamic Voltage Scaling. *IEEE Transactions on Communications*, 64(10):4268–4282, 2016.

[9] Min Chen and Yixue Hao. Task O□oading for Mobile Edge Computing in Software Defined Ultra-Dense Network. *IEEE Journal on Selected Areas in Communications*, 2018.

[10] Juan Liu, Yuyi Mao, Jun Zhang, and Khaled B. Letaief. Delay-optimal computation task scheduling for mobile-edge computing systems. In *IEEE International Symposium on Information Theory - Proceedings*, volume 2016-Augus, pages 1451–1455, Barcelona, aug 2016. Institute of Electrical and Electronics Engineers Inc.

[11] Jiao Zhang, Xiping Hu, Zhaolong Ning, Edith C.H. Ngai, Li Zhou, Jibo Wei, Jun Cheng, and Bin Hu. Energy-latency tradeo for energy-aware oloading in mobile edge computing networks. *IEEE Internet of Things Journal*, 5(4):2633–2645, 2018.

[12] Chenmeng Wang, Chengchao Liang, F. Richard Yu, Qianbin Chen, and Lun Tang. Computation Oloading and Resource Allocation in Wireless Cellular Networks with Mobile Edge Computing. *IEEE Transactions on Wireless Communications*, 16(8):4924–4938, 2017.

[13] Kuikui Li, Meixia Tao, and Zhiyong Chen. Exploiting Computation Replication in Multi-User Multi-Server Mobile Edge Computing Networks. *2018 IEEE Global Communications Conference, GLOBECOM 2018 - Proceedings*, pages 1–7, 2018.

[14] Kuikui Li, Meixia Tao, and Zhiyong Chen. Exploiting Computation Replication for Mobile Edge Computing: A Fundamental Computation-Communication Tradeo Study. pages 1–23, 2019.

[15] Jerey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

[16] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. Effective straggler mitigation: Attack of the Clones. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2013*, pages 185–198, 2019.

[17] Da Wang, Gauri Joshi, and Gregory Wornell. Using straggler replication to reduce latency in large-scale parallel computing. *Performance Evaluation Review*, 43(3):7–11, 2015.

[18] Qian Yu, Mohammad Ali Maddah-Ali, and A. Salman Avestimehr. Polynomial codes: An optimal design for high-dimensional coded matrix multiplication. *Advances in Neural Information Processing Systems*, 2017-Decem(Nips):4404–4414, 2017.

[19] Kangwook Lee, Maximilian Lam, Ramtin Pedarsani, Dimitris Papailiopoulos, and Kannan Ramchandran. Speeding Up Distributed Machine Learning Using Codes. *IEEE Transactions on Information Theory*, 64(3):1514–1529, 2018.

[20] Ankur Mallick, Malhar Chaudhari, Utsav Sheth, Ganesh Palanikumar, and Gauri Joshi. Rateless Codes for Near-Perfect Load Balancing in Distributed Matrix-Vector Multiplication. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(3):1–40, dec 2019.

[21] Albin Severinson, Alexandre Graell I. Amat, and Eirik Rosnes. Block-Diagonal and LT Codes for Distributed Computing with Straggling Servers. *IEEE Transactions on Communications*, 67(3):1739–1753, 2019.

[22] Albin Severinson, Alexandre Graell I. Amat, Eirik Rosnes, Francisco Lázaro, and Gianluigi Liva. A Droplet Approach Based on Raptor Codes for Distributed Computing with Straggling Servers. In *International Symposium on Turbo Codes and Iterative Information Processing, ISTC*, volume 2018-Decem, 2019.

[23] Jingjing Zhang and Osvaldo Simeone. On Model Coding for Distributed Inference and Transmission in Mobile Edge Computing Systems. *IEEE Communications Letters*, 23(6):1065–1068, 2019.

[24] Michael Luby. LT Codes. In *43 rd Annual IEEE Symposium on Foundations of Computer Science*, page 10. IEEE, 2002.

[25] Yasaman Keshtkarjahromi, Yuxuan Xing, and Hulya Seferoglu. Dynamic Heterogeneity-Aware Coded Cooperative Computation at the Edge. In *Proceedings - International Conference on Network Protocols, ICNP*, pages 23–33, 2018.

[26] Francisco Lazaro, Gianluigi Liva, and Gerhard Bauch. Inactivation Decoding of LT and Raptor Codes: Analysis and Code Design. *IEEE Transactions on Communications*, 65(10):4114–4127, 2017.

[27] C. E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 1948.

[28] Birgit Schotsch, Giuliano Garrammone, and Peter Vary. Analysis of LT codes over finite fields under optimal erasure decoding. *IEEE Communications Letters*, 17(9):1826–1829, 2013.

[29] Songze Li, Mohammad Ali Maddah-Ali, and A. Salman Avestimehr. A unified coding framework for distributed computing with straggling servers. In *2016 IEEE Globecom Workshops, GC Wkshps 2016 - Proceedings*, pages 1–6. IEEE, 2016.

[30] Songze Li, Mohammad Ali Maddah-Ali, Qian Yu, and A. Salman Avestimehr. A fundamental tradeo between computation and communication in distributed computing. *IEEE Transactions on Information Theory*, 64(1):109–128, 2018.

[31] Albin Severinson, Alexandre Graell I. Amat, and Eirik Rosnes. Block-Diagonal Coding for Distributed Computing With Straggling Servers. *IEEE Transactions on Communications*, 67(3):1739–1753, 2019.

[32] Sanghamitra Dutta, Viveck Cadambe, and Pulkit Grover. 'Short-Dot': Computing Large Linear Transforms Distributedly Using Coded Short Dot Products. In *IEEE Transactions on Information Theory*, volume 65, pages 6171–6193, 2019.

[33] Jingjing Zhang and Osvaldo Simeone. Fundamental Limits of Cloud and Cache-Aided Interference Management with Multi-Antenna Base Stations. *IEEE International Symposium on Information Theory - Proceedings*, 2018-June:1425–1429, 2018.