

ON THE FEASIBILITY OF USING
NETWORK CODING WITH THE
INTER-PLANETARY FILE SYSTEM

BY

OMAR ABDULLAH SALEH LAJAM

A Thesis Presented to the
COLLEGE OF GRADUATE AND INTERDISCIPLINARY STUDIES
KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE
In
COMPUTER SCIENCE

JANUARY 2024

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **OMAR ABDULLAH SALEH LAJAM** under the direction of his thesis adviser and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER SCIENCE**.

Thesis Committee


Dr. Salahadin Adem Mohammed (Adviser)

 12/5/2024
Prof. El-Sayed M. El-Alfy (Member)


Dr. Wasfi G. Al-Khatib (Member)


Dr. Motaz A. Alfarraj
Department Chairman


Prof. Suliman S. Al-Homidan
Dean of Graduate Studies



Date

©Omar Lajam
2024

To Gaza!

ACKNOWLEDGMENTS

This thesis journey wouldn't have been possible without the unwavering support and trust of my supervisor, Dr. Salahadin Adem Mohammed. I am also deeply indebted to my committee members, Prof. El-Sayed M. El-Alfy, whose insightful feedback and guidance significantly enhanced the quality of this work, and Dr. Wasfi G. Al-Khatib, whose belief in my research direction fueled my determination. Finally, I would like to extend my heartfelt gratitude to my friend Dr. Anas Abu Daqa, whose initial spark of inspiration ignited my interest in this thesis topic.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	vi
LIST OF TABLES	xi
LIST OF FIGURES	xii
LIST OF ABBREVIATIONS	xiv
ABSTRACT (ENGLISH)	xv
ABSTRACT (ARABIC)	xvii
CHAPTER 1 INTRODUCTION	1
1.1 Research Problem and Importance	2
1.2 Research Scope	2
1.3 Research Objectives	3
1.4 Thesis Contribution	4
1.5 Thesis Organization	5
CHAPTER 2 BACKGROUND	6
2.1 IPFS & BitSwap	6
2.1.1 Identity Management	7
2.1.2 Storage Management	7
2.1.3 BitSwap Data Exchange	10
2.1.4 BitSwap Data Discovery & Routing	17
2.1.5 Network Connections	18

2.2	Network Coding	19
2.2.1	Coding	19
2.2.2	Galois Field	20
2.2.3	Deterministic Network Coding	22
2.2.4	Random Linear Network Coding	26
2.2.5	The Encoder and The Decoder	30
2.3	Why is Network Coding for P2P Content Distribution?	32
CHAPTER 3 LITERATURE REVIEW		36
3.1	IPFS & BitSwap Advancements	36
3.2	Network Coding	40
3.2.1	Review Methodology	40
3.2.2	Coding Data Structures	42
3.2.3	Network Coding Schemes	47
3.2.4	Implementation details	61
3.2.5	Evaluation metrics	66
3.2.6	Practical benefits	69
CHAPTER 4 PROTOCOL DESIGN		79
4.1	Problem Statement	79
4.2	CodedBitSwap Design	81
4.2.1	Aligning IPFS Blocks with RLNC Packets	81
4.2.2	CodedBitSwap Message	82
4.2.3	Client Request	84
4.2.4	Server Response	85
4.2.5	Client Handling of Generation Responses	87
4.2.6	An Example	90
4.3	RLNC Potential Advantages for BitSwap	91
CHAPTER 5 PROTOCOL EVALUATION		94
5.1	Trial Experiments: Optimal Generation and Packet Sizes	94

5.1.1	Objectives	95
5.1.2	Variables and Parameters	95
5.1.3	Experimental Setup	96
5.1.4	Operations & Metrics	97
5.1.5	Experimental Procedure	98
5.1.6	Results & Discussion	99
5.1.7	Recommendations	103
5.2	CodedBitSwap Evaluation Methodology	105
5.2.1	Experiment Testbed	105
5.2.2	Experiment Materials	106
5.2.3	Operations & Procedure	106
5.2.4	Data Exchange Scenarios	107
5.2.5	Measurements	108
5.3	Results & Discussion	109
5.3.1	Blocks Duplication	109
5.3.2	Download Time	111
5.3.3	Coding Time	114
CHAPTER 6 CONCLUSION		117
6.1	Lessons Learned	117
6.2	On Network Coding Feasibility	118
6.3	CodedBitSwap: A Promising Solution	119
6.4	Future Directions	120
APPENDIX A ENCODING DATA PACKETS CALCULATIONS		123
A.1	Assumptions	123
A.2	Encoding Calculation	123
A.2.1	First Encoded Packet	124
A.2.2	Second Encoded Packet	125
APPENDIX B PUBLICATIONS DERIVED FROM THESIS WORK		127

REFERENCES	131
VITAE	152

LIST OF TABLES

Table 3.1	Classification of network coding data structures and schemes . .	43
Table 3.2	Look up table for multiplication operations over $GF(2^2)$ using the irreducible polynomial $x^2 + x + 1$	63
Table 3.3	Metrics used in evaluating network coding-based Peer-to-Peer (P2P) content distribution systems	68
Table 3.4	Main findings and implementation details for content distribution network coding-based P2P systems	69
Table 3.5	Main findings and implementation details for multimedia stream- ing network coding-based P2P systems	75
Table 5.1	Amounts of blocks transmitted among peers in the network . . .	110

LIST OF FIGURES

Figure 2.1	An example of a merkle DAG structure in IPFS	10
Figure 2.2	The internal modular architecture of BitSwap	11
Figure 2.3	The flow of multicasted packets using routing, where the flow of both packets between t_0 and t_2 is shown in (a), the flow of the yellow packet between t_2 and t_4 in (b), and the flow of the blue packets between t_3 and t_5 in (c).	23
Figure 2.4	The flow of multicasted packets using network coding, where the flow between t_0 and t_2 is shown in (a) and between t_2 and t_4 in (b).	24
Figure 2.5	An example for a network topology of four nodes	26
Figure 2.6	An illustration for the encoding and decoding outputs and packets transmission among network members	27
Figure 2.7	Illustration for the input and output for the (a) encoder and (b) decoder	31
Figure 3.1	Thirty-six generation sizes used by 23 studies related to P2P content distribution	64
Figure 3.2	Eighteen packet sizes used by 26 studies related to P2P content distribution	65
Figure 3.3	Twenty-two file sizes used by 18 studies related to P2P content distribution	66
Figure 4.1	Sequence diagram of an example of blocks transmission using CodedBitSwap	90

Figure 5.1	Encoding times for ten file sizes each experimented with four generation sizes	100
Figure 5.2	Encoding speeds for ten file sizes each experimented with four generation sizes	101
Figure 5.3	Decoding times for ten file sizes each experimented with four generation sizes	101
Figure 5.4	Decoding speeds for ten file sizes each experimented with four generation sizes	102
Figure 5.5	Experimental network topology of 11 nodes labeled from 0-10 .	105
Figure 5.6	Average download times of three files for peers 1-10	112
Figure 5.7	Average of encoding and decoding times by nodes 0-10	115

LIST OF ABBREVIATIONS

CDNs	Content Delivery Networks
IPFS	Inter-Planetary File System
P2P	Peer-to-Peer
RLNC	Random Linear Network Coding
TCP	Transmission Control Protocol
GF	Galois Field
CIDs	Content Identifiers
DNS	Domain Name Server
DHT	Distributed Hash Table
RTT	Round-Trip Time
DAG	Directed Acycle Graph
WoS	Web of Science

THESIS ABSTRACT

NAME: Omar Abdullah Saleh Lajam

TITLE OF STUDY: On the Feasibility of Using Network Coding with the Inter-Planetary File System

MAJOR FIELD: Computer Science

DATE OF DEGREE: January 2024

The InterPlanetary File System (IPFS) has emerged in 2015 as a promising peer-to-peer (P2P) distributed file-sharing system poised to become the backbone of Web3. However, its BitSwap protocol, responsible for block exchange, encounters redundancy issues when multiple peers respond with duplicate blocks. To address this limitation, we propose CodedBitSwap, an innovative network coding-based data exchange protocol that integrates Random Linear Network Coding (RLNC) into BitSwap. Considering that RLNC operations incur additional computational overhead, the RLNC-based protocol is designed with careful attention to its computational complexity that is investigated through trial experiments guiding the selection of coding parameters and structures. To assess the feasibility and performance of CodedBitSwap, an experimental evaluation that compares it with BitSwap was conducted in different scenarios

using a controlled testbed environment consisting of 11 nodes exchanging three files of different sizes. During file exchange, the amount of data transmitted, download time, and encoding and decoding times were measured for each node. The evaluation results demonstrate that CodedBitSwap effectively eliminates redundancy at a relatively low cost of increased download time. The introduced RLNC computational complexity was optimized by the generation-based design strategy that minimizes it, ensuring that the cost of the reduced redundancy remains relatively low. The undertaken design methodology of CodedBitSwap offers a practical approach for future systems, which balances the overhead of RLNC coding with the benefits it brings. This work contributes to the advancement of network coding in P2P networks and demonstrates its potential to improve the efficiency of IPFS, opening up avenues for future research.

ملخص الرسالة

الاسم:

عمر عبدالله صالح لعجم

عنوان الدراسة:

حول جدوى استخدام ترميز الشبكة مع نظام الملفات العابر للكواكب

التخصص:

علوم الحاسب

تاريخ الدرجة العلمية: جمادى الثاني 1445

في العام 2015، برز ما يسمى نظام الملفات العابر للكواكب (IPFS) باعتباره نظامًا واعدًا لمشاركة الملفات الموزعة من نظير إلى نظير (P2P) والذي يستعد ليصبح العمود الفقري للويب 3. ومع ذلك، فإن بروتوكول BitSwap الخاص به والمسؤول عن تبادل البيانات، يواجه مشكلات تعدد النسخ عندما يقوم أكثر من عضو في الشبكة بالرد ببيانات مكررة تم استلامها سابقاً. لمعالجة هذه المشكلة، نصمم في هذه الأطروحة CodedBitSwap، وهو بروتوكول تبادل بيانات مبتكر يدمج ترميز الشبكة الخطية العشوائية (RLNC) في BitSwap. وبالنظر إلى أن عمليات RLNC تتطلب إجراء معالجات حسابية إضافية، فقد تم تصميم البروتوكول مع الاهتمام الدقيق بالتعقيد الحسابي الذي تم الاعتناء به من خلال مجموعة من تجارب الأداء التي وجهت اختيار بنية التشفير الأكثر ملاءمة. لتقييم جدوى وأداء بروتوكول CodedBitSwap، تم إجراء تقييم تجريبي ضمن سيناريوهات مختلفة لمقارنته مع BitSwap باستخدام بيئة اختبار محكمة تتكون من 11 عضو شبكي تنتقل بينهم ثلاثة ملفات بأحجام مختلفة. أثناء عملية نقل الملفات، تم قياس كمية البيانات المرسلة ووقت التنزيل وأوقات التشفير وفك التشفير لكل عضو. توضح نتائج التقييم أن CodedBitSwap يزيل التكرار بشكل فعال بتكلفة منخفضة نسبيًا تمثل في زيادة بسيطة في وقت تحميل البيانات. يمكن أن تكون

منهجية التصميم المتبعة في CodedBitSwap بمثابة نهج عملي للأنظمة المستقبلية، والتي توازن تكاليف ترميز RLNC مع فوائدها المكتسبة. يساهم هذا العمل في تطوير ترميز الشبكة في شبكات النظير للنظير ويوضح قدرته على تحسين كفاءة IPFS، وفتح آفاق جديدة للبحث العلمي مستقبلاً.

CHAPTER 1

INTRODUCTION

P2P networks have become increasingly popular in recent years due to their ability to share data without relying on central servers. One of the most promising P2P networks is the Inter-Planetary File System (IPFS) [1], a content-addressable distributed file system that connects computer devices with a common file system [2], [3]. IPFS is well-positioned to become the underlying infrastructure of a new, decentralized web, known as web 3.0 [4]. The IPFS network has been designed to be more efficient than traditional client-server models by allowing users to access data from multiple sources simultaneously instead of relying on central servers [4]. It is an open-source community-driven project, with implementations in Go ¹ and Javascript ², and a global community of millions of users [3]. IPFS has been used to build various applications, including systems for e-commerce, secure distribution of scientific data sets, mirroring Wikipedia, creating new social networks, sharing cancer data, blockchain applications, secure and encrypted personal-file storage and sharing, developer tools, and data

¹<https://github.com/ipfs/kubo> (accessed: 2024-04-27)

²<https://github.com/ipfs/js-ipfs> (accessed: 2024-04-27)

analytics [5].

1.1 Research Problem and Importance

While IPFS addresses some of the problems associated with centralized systems, such as the high load on servers, it has notable performance drawbacks [6], [7]. One obvious issue with IPFS is the redundancy incurred when a peer receives duplicate data blocks from multiple peers. Random Linear Network Coding (RLNC) has the potential to improve the performance of P2P networks by reducing the amount of redundant data that is exchanged between peers [8]. IPFS can benefit from RLNC by applying coding operations at its data exchange protocol, BitSwap, that make data blocks equally valuable and, as a result, reduce the number of duplicate blocks and improve the overall network performance, as suggested by Lajam [7], Zimmermann et al. [9], and Rocha et al. [10].

However, the introduction of RLNC operations inevitably introduces additional complexity overhead. The encoding and decoding operations, even if assumed to be computationally efficient, add an extra layer of processing to the block exchange process. The key question to address is whether the benefits of reduced redundancy and improved efficiency outweigh the additional costs incurred by RLNC.

1.2 Research Scope

This study evaluates the feasibility of applying RLNC concepts into BitSwap protocol and proposes a new protocol, called CodedBitSwap, that extends the original BitSwap

protocol with the added RLNC technology. To thoroughly assess the feasibility of applying RLNC into BitSwap, a comprehensive performance evaluation is conducted. This evaluation considered various network scenarios, including different node densities and file sizes. By measuring performance metrics such as the amount of data transmitted in the network, download time, and coding time, a clear understanding of the trade-off between efficiency and complexity is established.

1.3 Research Objectives

This research investigates the potential of integrating RLNC into the BitSwap protocol for peer-to-peer content distribution. Our primary focus is to assess the feasibility and effectiveness of this approach in reducing data redundancy and improving content delivery efficiency. The objectives of this research can be summarized as follows:

- Conduct a comprehensive review of existing literature on how network coding has been applied in peer-to-peer (P2P) content distribution systems.
- Assess the feasibility and potential benefits of using network coding for efficient data dissemination in P2P networks.
- Develop a novel P2P content distribution protocol called CodedBitSwap that leverages network coding within the existing BitSwap protocol. The proposed protocol should enable peers to retrieve data from multiple connected peers in a single request using network coding, minimizing redundancy when popular data is widely available within the network.

- Identify the most suitable network coding techniques and parameters for optimal performance within CodedBitSwap.
- Conduct a thorough evaluation of the advantages and potential drawbacks associated with using network coding for P2P content distribution.
- Analyze the trade-off between coding overhead introduced by network coding and the benefits of reduced data redundancy in CodedBitSwap.

1.4 Thesis Contribution

Our study focuses mainly on developing optimized RLNC protocol specifically tailored for BitSwap. However, the approach taken to design CodedBitSwap can inspire the implementation of RLNC into other P2P applications such as blockchain technology.

The contribution of this work can be summarized as follows:

- A comprehensive review of nearly two decades of network coding literature that described the concepts and classified network coding structures and schemes.
- A novel data exchange protocol for IPFS that leverages RLNC technology, named CodedBitSwap, has been designed and implemented.
- A comprehensive exploration has been undertaken to understand the potential advantages and the additional overhead associated with the application of RLNC to BitSwap.
- The experimental results demonstrate that the CodedBitSwap protocol effectively addresses the redundancy problem while maintaining an acceptable level

of complexity overhead.

1.5 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 describes the background of IPFS, BitSwap, and network coding. Chapter 3 presents a literature review on the topic. Chapter 4 presents the design of the proposed CodedBitSwap protocol. Chapter 5 outlines the experimental evaluations of the proposed protocol. Lastly, Chapter 6 concludes the thesis and gives future research directions.

CHAPTER 2

BACKGROUND

This chapter provides an important background. The first section presents a background on IPFS and BitSwap protocol and how they work. The second section discusses the main concepts and principles of network coding with practical examples. The last section of this chapter discusses why network coding can be useful for P2P content distribution.

2.1 IPFS & BitSwap

In this section, we present the internal components of IPFS and BitSwap protocol [11]. Generally, IPFS operations can be classified into five modules that deliver functionalities for each other to process client requests. These modules manages peers identity, storage architecture, data exchange, peers and data discovery, and network connections. Each module is critical to support the distributed file system operations and further discussed in the following subsections.

2.1.1 Identity Management

Each node in a P2P file system needs to be recognized by a unique identifier that is guaranteed to be collision-free in order to communicate in the network; two distinct nodes can never map to the same identifier. In IPFS, each node is identified by the encrypted hash (in multi-hash format) of a public key, called *NodeId*. The multi-hash identifier is formatted in the following format: *<hash function code, hash digest length, hash digest bytes>*. Nodes periodically exchange and verify their NodeIds and public keys when establishing connections.

2.1.2 Storage Management

Merkle Trees

A binary tree called Merkle tree [12] is formed using a cryptographic hash algorithm for each file. A Merkle tree contains hash values for each leaf that is computed using one or more data elements. Each parent node computes its hash value from its children, which in turn compute theirs recursively from all the values in their subtree.

Merkle trees are frequently utilized in the blockchain space for rapid validations (e.g., block validation in Bitcoin [13]) and integrity checks (e.g., light peers in Ethereum [14]). Integrity checking can be performed by only saving the root of a merkle tree since even the smallest modification can have a significant impact on the tree's root. Only a small number of node hashes, instead of the complete tree, are required to verify if a node is in a merkle tree.

Content Addressing

In traditional file sharing systems, such as the World Wide Web, files are located using URLs (Uniform Resource Locators) such as `www.example.com`. URLs provide a hierarchical path to a file's location on a specific server, essentially translating into an IP address that specifies the file's physical location in the network. This approach, known as Location-Based Addressing, relies on the underlying network infrastructure to maintain the mapping between URLs and IP addresses, utilizing services like the Domain Name Server (DNS) to resolve domain names to actual network locations.

In contrast, the IPFS employs a Content-Based Addressing scheme, where files are addressed by their unique Content Identifiers (CIDs) (e.g., `/ipfs/QmXoy..../`). CIDs are cryptographic hashes of the file's content, ensuring that any file with the same content will have the same CID, regardless of its physical location in the network. This makes IPFS more resilient to network disruptions; if a particular node hosting a file becomes unavailable, the file can still be retrieved from other nodes based on its CID. This addressing mechanism eliminates the reliance on centralized servers and DNS resolution, enabling a decentralized and resilient file-sharing ecosystem.

Merkle DAG

Merkle Directed Acycle Graph (DAG) [15], [13], which is used in IPFS as a data organization model, shares the same idea of the Merkle tree. Data blocks are therefore addressed using their hashes. Each file is splitted into blocks of size 256KB and encrypted with a hash function. This hash, called Content Identifier (CID), is appended into the block. If a file consists of more than one block, then its blocks are linked with

a parent block consists of a hash computed using the file block hashes in a Markle DAG. The name by which the file can be located in the IPFS (the CID) is determined by the hash of this parent block.

When storing a file into IPFS, its data is first supplied to a splitter, which splits it into blocks. Then, they are stored in a local *blockstore*. One of the main challenges with versioning and distributing large files is finding the right way to split them into independent blocks. IPFS uses a variety of methods to split the file into separate blocks, including the rsync rolling-checksum algorithm [16] and Rabin Fingerprints [17].

Figure 2.1 illustrates how the IPFS Merkle DAG arranges the structure of files and directories (or folders). In this figure, there are two files (*example.js* and *hello.txt*) and one folder path *dir* in the root path of this file directory. *example.js* is divided into three different data pieces. The file path *dir* has two files: *other.txt* and *example.txt* (here file content of *example.txt* in *dir* and *hello.txt* in root path are exactly the same therefore they are linked to the same object). Each object derives its value though computing its children's value and the content of data.

Merkle DAG provides several important properties for IPFS, including:

1. **Content Addressing:** all content is uniquely identified by its multihash checksum, including links.
2. **Tamper resistance:** all content is verified with its checksum. If data is tampered with or corrupted, IPFS detects it.
3. **Deduplication:** all objects that hold the exact same content are equal, and

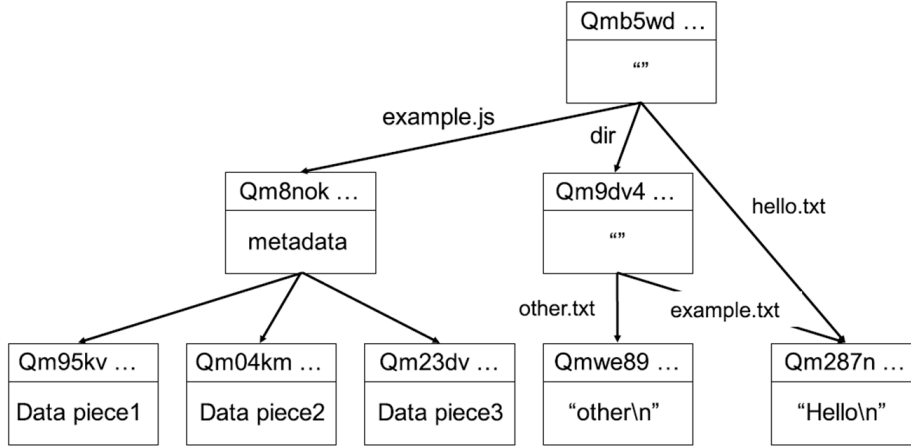


Figure 2.1: An example of a merkle DAG structure in IPFS

only stored once. This is particularly useful with index objects, such as git trees and commits, or common portions of popular data.

2.1.3 BitSwap Data Exchange

BitSwap¹ is a core module of IPFS responsible for managing block transmissions among peers in the network. It is a message-based protocol where all messages between peers have a common structure (called BitSwapMessage) that contains wantlists and/or data blocks. BitSwap has three main jobs: 1) managing peers connections, 2) discovering blocks locations, and 3) exchanging blocks among interested peers [10]. In this section, the focus is on the data exchange part of the protocol.

When new IPFS node joins the network, it establishes connections with a group of peers, typically around 20. These *connected peers* form the immediate peer neighborhood and play a crucial role as a starting point for data discovery and exchange.

A node can be either a *client* or a *server*. The client is a node which is interested

¹<https://docs.ipfs.tech/concepts/bitswap/> (accessed: 2024-04-27)

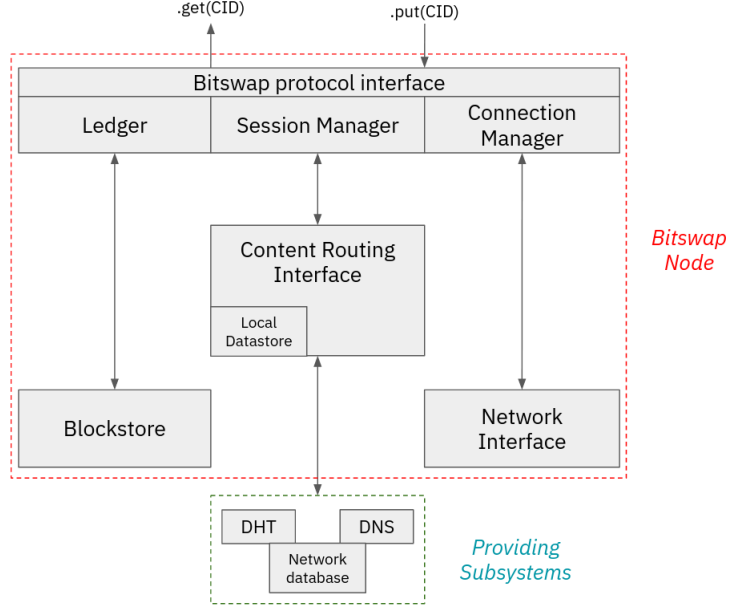


Figure 2.2: The internal modular architecture of BitSwap

in a block that does not exist in its local store and send requests to other peers to get it. The server is a node that responds to the client requests. Note that, since it is a P2P network, a node can play the roles of a client and a server at the same time. This means that a BitSwap message can contain a list of block requests (called want-list) and, at the same time, raw data blocks sent as a response to previous request.

In the following subsections, we will discuss BitSwap messages structures and how BitSwap protocol manages the client and the server to exchange data between them. Figure 2.2 shows the internal architecture of BitSwap protocol.

Block Requests & Responses

There are three types of block requests and three types of block responses in which clients and servers nodes can exchange in the BitSwap protocol [10]. The client's requests and their corresponding meanings are as follows:

- *WANT-HAVE*: This request says: “I want the block (CID...), do you have it?”. It is sent by the client when it wants to know if the block exists at a server node.
- *WANT-BLOCK*: This request says: “I want a block (CID...), please send it”. It is sent by the client when it wants the block from a server that potentially has the block.
- *CANCEL*: This request says: “I do not want the block (CID...) anymore”. It is sent by the client who has previously sent a *WANT-BLOCK* request to a server informing it to cancel the request.

The server’s responses and their corresponding meanings are as follows:

- *HAVE*: This response says: “I have the block (CID...) you are looking for”. It is sent by the server as a response to *WANT-HAVE* requests.
- *DONT-HAVE*: This response says: “I do not have the block (CID...) you are looking for”. It is sent by the server as a response to *WANT-HAVE* or *WANT-BLOCK* requests.
- *BLOCK*: This response contains the raw data of a block. It is sent by the server as a response to *WANT-BLOCK* requests.

BitSwap Message

The BitSwap message serves as a standard data structure in the protocol, enabling nodes to communicate their block requirements and availability. The BitSwap message comprises three distinct maps:

- **Wantlist:** This map maintains a mapping between content identifiers (CIDs) and corresponding block requests. Each CID key is associated with one request type, indicating whether the requesting node is seeking the block (WANT-HAVE), want the data of the block (WANT-BLOCK), or is canceling a previous request (CANCEL).
- **Block Presence:** This map mirrors the Wantlist, but instead of client requests, it stores server responses. Each block CID key is paired with a server response type, indicating whether the node has the requested block (HAVE) or not (DONT-HAVE).
- **Blocks:** This map serves as a response for the client WANT-BLOCK request for actual data blocks, where each CID key is mapped to the corresponding raw data block.

The standardized structure of the BitSwap message ensures consistency and interoperability across the P2P network. All nodes use the same message format, allowing them to interpret and respond to requests from each other.

Client Request

When an IPFS peer wants a specific block identified by a CID, the BitSwap protocol controls the client as following:

- **Broadcasting WANT-HAVE Requests:** The client initiates the process by broadcasting a WANT-HAVE request to all connected peers. This request signals the client's interest in the block, allowing other peers with the block to

identify themselves as potential providers. The client maintains a local wantlist to track all requested blocks, ensuring that only blocks that are still needed are accepted upon reception.

- **Optimistic WANT-BLOCK Request:** The client sends an *optimistic* WANT-BLOCK request to at most two randomly-selected peers. It requests the entire block from the chosen peer(s), even though the client has not yet received confirmation of its availability. This can reduce the block retrieval Round-Trip Time (RTT) by one but it has a drawback of potentially retrieving duplicate blocks if the request is sent to more than one peer [10].

Server Response

Upon receiving a WANT-HAVE, a WANT-BLOCK, or a CANCEL request, the server peer engages in a sequence of actions to respond:

- **Block Availability Check:** The server starts by verifying whether the requested block exists in its local blockstore. This initial check ensures that the peer can fulfill the request if it has the desired block.
- **Block Present:** If the requested block is indeed present in the local blockstore, the server initiates a task-scheduling process to respond to the requesting peer. This process differentiates between WANT-HAVE and WANT-BLOCK requests:
 - **WANT-HAVE Request:** For a WANT-HAVE request, the server peer schedules a task to transmit a HAVE response to the requesting peer.

This response informs the requesting peer that the server peer owns the requested block, enabling the requesting peer to initiate a block exchange process.

- **WANT-BLOCK Request:** For a WANT-HAVE request, the server schedules a task to send the requested block to the requesting peer.
- **Block Absence:** In the event that the requested block is not found in the local blockstore, the server sends a DONT-HAVE response to the requesting peer. This response informs the requesting peer that the block is not available at the server, allowing the requesting peer to explore alternative options for block retrieval.
- **CANCEL Message:** When a CANCEL request is received from the requesting peer, the server immediately identifies the corresponding block request and cancels any scheduled block transfer task for that particular block. The server does not respond after this cancellation.

Receiving Responses

When a client receives a BitSwap message in response to its previously sent WANT-HAVE or WANT-BLOCK requests, it engages in a series of steps to process the received data and manage its block requirements:

- **Wantlist Verification:** The client first verifies whether it still needs the block included in the received BitSwap message. This involves calculating the hash of

the CID of the received block and checking its own wantlist to confirm that the corresponding block was indeed requested.

- **Valid Block Processing:** Assuming the wantlist verification is successful, indicating that the block is still needed, the client proceeds with the following steps:
 - **Blockstore Storage:** The client stores the received block in its local storage.
 - **CANCEL Message Broadcast:** To ensure that other peers are informed about the block reception, the client broadcasts a CANCEL message for the block to all peers from which it had previously requested the block.
- **Invalid Block:** If the wantlist verification fails, indicating that the block is no longer needed or tampered with, the client discards the received block. This action prevents the storage of unneeded data blocks.
- **HAVE or DONT-HAVE Response:** If the client gets a HAVE or a DONT-HAVE, it stores this information and repeat requesting the block again after a timeout of not getting a response for a sent WANT-BLOCK to other peer(s).
- **DHT Query for Block Discovery:** If none of the connected peers has the requested block, the client resorts to querying the Distributed Hash Table (DHT) (discussed next) to discover a peer that has the block. The DHT serves as a decentralized index, enabling the client to locate peers with the desired data.

2.1.4 BitSwap Data Discovery & Routing

When a node requires a specific block, it first seeks assistance from its immediate neighbors. If the block is not found, the query is relayed further through the global IPFS network. Only in the event that the peers' neighbors does not own the block, the DHT [18] step in, providing a fallback mechanism for BitSwap block discovery. It is important to note that IPFS is highly modular and the DHT of BitSwap can be replaced by another discovery mechanism such as DNS.

The DHT, a fundamental component of IPFS routing module inspired by S/Kademlia [19] and Coral [20], plays a crucial role in managing the network's topology and facilitating data discovery. It functions as a distributed hash table, where each file's unique content identifier (CID) is mapped to the nodes that store its corresponding data blocks. Each node in the global network stores a portion of this hash table.

When a peer seeks a particular file, it initiates a search through the DHT, using the file's CID as a key. The DHT traverses the network, querying neighboring nodes, and relaying the search until it locates peers that hold the desired data blocks.

IPFS's reliance on the DHT extends beyond data discovery. The DHT also plays a crucial role in routing, enabling nodes to locate and communicate with each other. The DHT does not only stores the mappings from CID to NodeId, but it also stores the mapping from NodeId to node physical address (e.g. IP address). After a node finds the NodeId of the node who owns the block, it queries the DHT again to get the node physical address. As nodes join and leave the network, the DHT dynamically updates

its mappings, ensuring that nodes can always locate and connect with each other. This dynamic nature is crucial for maintaining a robust and resilient peer-to-peer network [21].

2.1.5 Network Connections

libP2P² is a powerful generic P2P solution that is used as the network layer within the IPFS architecture. It is based on the DHT mechanism used by BitTorrent and can transport data using any network protocol. Depending on whether the underlying network is stable, IPFS may use either SCTP [22] or UTP [23]. IPFS uses the multiaddr formatting approach, which mixes addresses and accompanying protocols, to primarily enable free switching between different options.

One of the key benefits of using libP2P is that it provides a pluggable transport layer, which means that IPFS can be easily extended to support new network protocols. This is important for IPFS, as it allows it to operate in a wide variety of environments, including the Internet, local networks, and mobile networks.

Another benefit of using libP2P is that it provides a secure and reliable transport layer. IPFS uses a variety of security mechanisms to protect data in transit, including encryption and authentication. Additionally, libP2P provides a number of features that make it reliable, such as congestion control and re-transmission mechanisms.

²<https://github.com/libp2p> (accessed: 2024-04-27)

2.2 Network Coding

Since most of the network coding-related studies assume prior knowledge on the topic and provide high-level descriptions, we give a primer knowledge on important related topics [24]. First, we explain the meaning of coding in general, then we discuss mathematical foundations on Galois fields algebraic structures. After that, we then explain network coding concepts with some practical examples.

2.2.1 Coding

Encoding is the process of converting a string of symbols (e.g., bytes) into a new format, and decoding is the process of converting an encoded string back to its original format [25]. These two processes are referred to as coding. The difference between coding and encryption is that coding does not require a secret key, whereas in encryption, there must be a secret key that is used to encrypt and decrypt the data [26], [27]. In general, coding is used to maintain data usability, whereas encryption is used to maintain data confidentiality.

One of the simplest coding schemes is repetition coding, where the encoding process simply repeats the symbols of a string. For example, to encode the string ‘1011’ using 3 repetitions, the encoded string will be ‘111000111111’, where each bit is repeated three times. This form of coding is used to increase the usability of the string in case errors occurred during transmission. To decode the string, each 3 bits are replaced with the majority bit. For example, the string ‘101000110111’ will be decoded to ‘1011’. These kinds of codes are called error correction codes [28], [29].

2.2.2 Galois Field

In mathematics, a *field* is a set of elements on which four arithmetic operations are defined: Addition, subtraction, multiplication, and division. Examples of fields are the set of real numbers ($\mathbb{R}$) and integer numbers ($\mathbb{Z}$). A Galois Field (GF)³ or a finite field is a field that contains a finite number of elements [30]. The *size* of a finite field is the number of the elements it contains. If the size of a finite field is $q = p^k$, where p is a prime or a power of a prime and k is a positive integer, then p is called the characteristic and the field can be denoted by $\mathbb{F}_q$, F_q , or $GF(q)$, where the latter notation stands for Galois Field.

The elements of a finite field of size q can be represented by integers in the range $\{0, \dots, q-1\}$, by binary elements, or by polynomials in the form $a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x^1 + a_0$, where x is an indeterminate (a variable that doesn't represent a specific value) and a_i is either 0 or 1, n is the degree of the polynomial (highest power of x). For example, $GF(2^2)$ elements can be represented in integers as $\{0,1,2,3\}$, in binary as $\{00,01,10,11\}$, and in polynomials as $\{0, 1, x + 0, x + 1\}$.

Addition, subtraction, multiplication, and division of polynomials are performed by following the rules of polynomial arithmetic. However, these operations are performed modulo a chosen irreducible polynomial. An irreducible polynomial is one that cannot be factored into non-trivial polynomials over the prime field. The modulo operation ensures that the resulting polynomial remains within the finite field by wrapping around when reaching the degree of the irreducible polynomial. Let a and

³Named after the French mathematician Evariste Galois

a and b be two elements of some finite field $GF(p^k)$. Addition, subtraction, multiplication, and division are defined respectively by the following formulas:

$$a + b \equiv a + b \pmod{p}$$

$$a - b \equiv a - b \pmod{p}$$

$$a \cdot b \equiv a \cdot b \pmod{P(x)}$$

$$a/b \equiv a \cdot b^{-1} \pmod{P(x)}$$

where $P(x)$ is an irreducible polynomial [31] of a degree higher than $k - 1$ used to define the finite field, and b^{-1} is the multiplicative inverse of b defined by:

$$b^{-1} \cdot b \equiv 1 \pmod{P(x)}$$

For elements defined over the finite field $GF(2^k)$ for any positive integer k , addition, subtraction, and bit-wise XOR are three equivalent operations, making the choice of $GF(2^k)$ popular in computer science applications [32]. The multiplication and division are similar to the polynomial operations. For example, the multiplication of the two integers 7 and 5 (111 and 101 in binary, respectively) over $GF(2^3)$ with the irreducible polynomial $P(x) = x^3 + x + 1$ is:

$$7 \times 5 = 111 \times 101 = (x^2 + x + 1) \times (x^2 + 1) = x^4 + x^3 + x + 1$$

Because $(x^4 + x^3 + x + 1) \notin GF(2^3)$, the irreducible polynomial $P(x)$ is used:

$$(x^4 + x^3 + x + 1) \bmod (x^3 + x + 1) = x^2 + x = 110 = 6$$

For simplicity, hereafter, elements of finite fields are represented by integers.

2.2.3 Deterministic Network Coding

The network coding [33] technique is introduced as a more efficient approach for data transmission over a network compared to routing [34], where intermediate nodes do not just forward data, but they encode it using a pre-defined scheme before forwarding. The term 'network' implies the participation of the network members in the coding operations. The concept of network coding is illustrated using the following butterfly example [33], in which a single source node wants to multicast two packets to two destination nodes through intermediate relay nodes in the network.

Consider the network topology consisting of 7 nodes shown in Figure 2.3. Node S wants to send two data packets 1011 and 1100 (denoted by blue and yellow colors) to both D_1 and D_2 . Since it has no direct connections with them, it will send the data packets through the relay nodes, denoted by R_1 to R_4 . The assumptions are that the capacity of each link is one packet per time unit (t) and each link is error-free. In this network, there is a bottleneck link in the middle (between R_3 and R_4), where one of the two packets can pass in one-time unit.

Consider the case where normal routing is used shown in Figure 2.3. Let t_0 be time zero at which the packet transmission will start. At t_1 , as shown in Figure 2.3

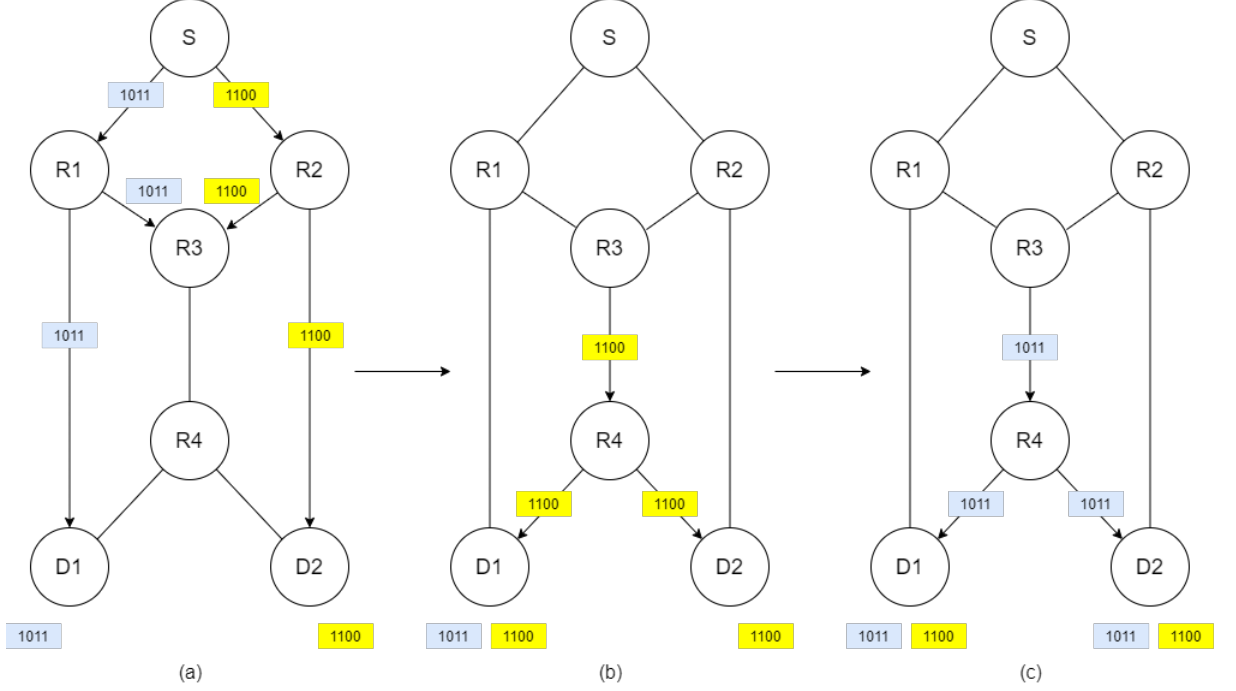


Figure 2.3: The flow of multicasted packets using routing, where the flow of both packets between t_0 and t_2 is shown in (a), the flow of the yellow packet between t_2 and t_4 in (b), and the flow of the blue packets between t_3 and t_5 in (c).

(a), R_1 and R_2 will have a copy of the blue and yellow packets, respectively. At t_2 , the blue packet will arrive at D_1 and R_3 coming from R_1 , and the yellow packet will arrive at D_2 and R_3 coming from R_2 . At this time (t_2), R_3 has received both the blue and yellow packets while R_4 has not received any packets yet. Between t_2 and t_3 , as shown in Figure 2.3 (b), R_3 will send only the yellow packet to R_4 and will keep the blue packet in its buffer waiting for the link to be freed. At t_4 , the yellow packet will arrive at D_1 and D_2 coming from R_4 . D_2 has already a copy of the yellow packet, therefore, it will discard it. At this time (t_4), the transmission of the data packets has completed at D_1 , while the blue packet has arrived at R_4 coming from R_3 , as shown in Figure 2.3 (c). At t_5 , D_1 will receive a redundant blue packet coming from R_4 , which will be discarded, while D_2 will receive a new blue packet. At this time (t_5), packet

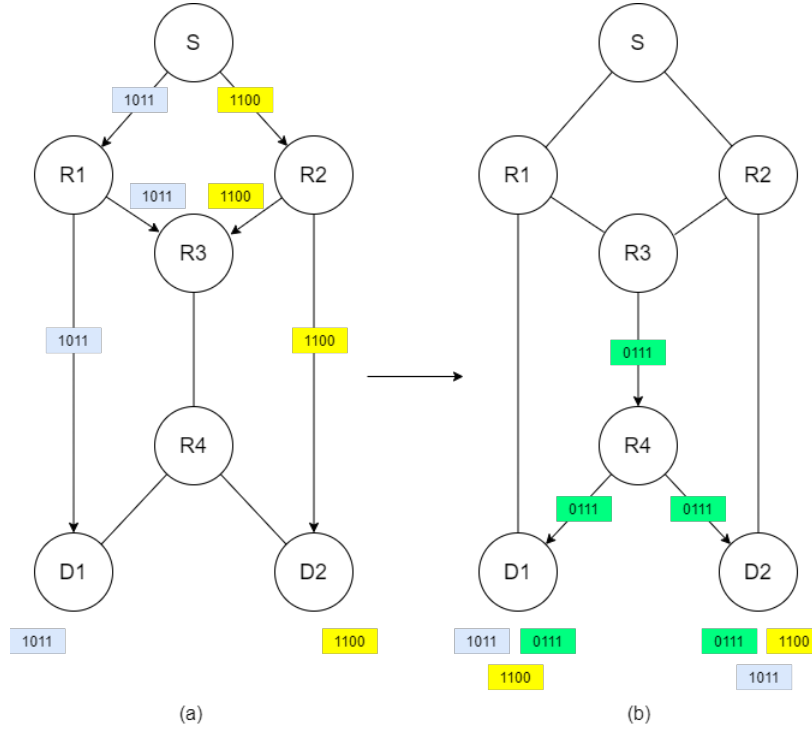


Figure 2.4: The flow of multicasted packets using network coding, where the flow between t_0 and t_2 is shown in (a) and between t_2 and t_4 in (b).

transmission has been completed for D_2 .

From this example, we can observe several problems. First, both D_1 and D_2 have received redundant data packets. Second, the network bandwidth was not optimally utilized due to the delay caused by the bottleneck link between R_3 and R_4 . Third, R_3 had to store one packet and wait until the link is freed. Forth, the transmission time for D_2 was longer than D_1 , which caused inconsistent data loading for them.

These problems can be solved by using network coding, as shown in Figure 2.4. Simply, instead of sending plain packets by R_3 , it will XOR both packets and send the encoded packet over the link to R_4 only once, as shown in Figure 2.4 (b), where the green packets are a combination of the blue and the yellow packets. At time t_4 , both D_1 and D_2 will have one plain packet and one encoded packet. To decode the

encoded packet, they need simply to XOR it with a blue or a yellow packet, the one they already have received.

In this scenario, the transmission was completed optimally. However, this approach required some extra cost. First, R_3 had to do additional encoding processing, while it was only doing forwarding with the routing scenario. Second, D_1 and D_2 had to do additional decoding processing, while they were only arranging the packets based on their sequence. Third, to decode successfully, the receiver nodes D_1 and D_2 had to know that the green packet is a combination of the blue and the yellow packets, which required additional information about the received packets.

This type of network coding described in the example is called *deterministic network coding*, in which the method of how data is transmitted is specified based on the network condition, i.e., it exactly specifies which nodes will encode which packets and whether nodes will forward original or encoded packets.

Deterministic network coding is less practical because it assumes a static network condition. Considering the discussed example, what if the source node wants to send more than two packets, or what if the topology of the network is more complex and has a large number of nodes. Deterministic network coding will be more practical if the network topology is fixed and each node has advanced knowledge of its role, or if the encoding and decoding algorithms require to input the full network topology. For the latter case, frequent updates about the network topology should be circulated among network members, especially in dynamic networks, where nodes are frequently joining and leaving. To address these problems, a more complex network coding approach, called RLNC, can be used, and it will be discussed next.

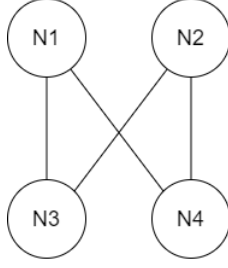


Figure 2.5: An example for a network topology of four nodes

2.2.4 Random Linear Network Coding

Instead of carefully designing the deterministic network coding scheme, a better, yet more complex approach, called RLNC, can be used. In RLNC, the sender does not send plain packets. Instead, it sends a linear combination of the packets using random coefficients. This approach is first proposed in [35], which was built on top of the work done in [36] and [37] that is lacking the randomness factor.

RLNC is explained using the following example. Consider two data packets P_1 and P_2 to be transmitted by two source nodes N_1 and N_2 to two destination nodes N_3 and N_4 , in the network shown in Figure 2.5. The encoding, decoding, and packets transmission between network members are illustrated in Figure 2.6. Let the coding operations be over the Galois field $GF(2^8)$ with the irreducible polynomial $P(x) = x^8 + x^4 + x^3 + x^2 + 1$. Since this Galois field element length is 8 bits, the coding operations on the packets will be applied to every 8 bits. Each 8 bits of a packet is called a *symbol*. To keep the example simple, packet symbols are represented by a group of comma-separated 8-bits decimal integers⁴.

⁴To follow with the example, it is recommended to use a finite field calculator, e.g. <https://asecuritysite.com/principles/gf>

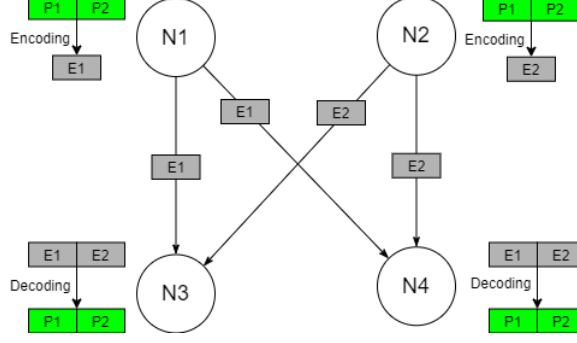


Figure 2.6: An illustration for the encoding and decoding outputs and packets transmission among network members

Assume that the first packet $P_1 = (145, 167)$, the second packet $P_2 = (120, 228)$, and both P_1 and P_2 are available at N_1 and N_2 . The encoding algorithm will construct encoded packets by combining the two packets P_1 and P_2 . To construct an encoded packet E_1 , N_1 will follow these steps (Detailed calculation are presented in Appendix A):

1. Randomly choose two coefficients $\alpha_{1,1}$ and $\alpha_{1,2}$ from $GF(2^8)$, one for each packet (P_1 and P_2), where for $(\alpha_{i,j})$, i is the encoded packet number and j is the original packet number. Assume that the values of the randomly chosen coefficients for E_1 are $\alpha_{1,1} = 17$ and $\alpha_{1,2} = 136$. These two coefficients represent an *encoding vector* for the encoded packet E_1 .
2. Each packet symbol will be then multiplied by its packet coefficient. Multiplying $\alpha_{1,1}$ by each symbol of P_1 will give us $M_1 = (116, 5)$, and multiplying $\alpha_{1,2}$ by each symbol of P_2 will give us $M_2 = (49, 16)$.
3. The encoded packet E_1 is constructed by bit-wisely XORing M_1 and M_2 , which results in $E_1 = (69, 21)$. E_1 now is a complete encoded packet that can be

transmitted to the decoder with its encoding vector.

Since E_1 is a combination of two packets, it cannot be decoded alone, more encoded packets are needed to restore the original packets P_1 and P_2 . In this example, the other encoded packet E_2 will be constructed by node N_2 by repeating exactly the same explained process. Since the coefficients are randomly selected, the newly constructed encoded packet will most likely be different from the first one. If the newly constructed encoded packet is linearly dependent⁵ on the coded packet of the other encoder, then it will be considered redundant and useless by the decoder. Assuming that the randomly selected coefficients for E_2 are $\alpha_{2,1} = 72$ and $\alpha_{2,2} = 28$, the resulting encoded packet will be $E_2 = (134, 193)$. Using the same process, each of the nodes N_1 and N_2 can construct an endless amount of encoded packets for whoever is requesting them.

The encoding vectors, original packets, and encoded packets are represented algebraically as matrices, where C is the encoding vectors matrix, P is the original packets matrix, and E is the encoded packets matrix. These three matrices are a linear system that has the following relationship:

$$C \times P = E$$

For n packets $(P_1, \dots, P_n)$ encoded into $(E_1, \dots, E_n)$, $p_{i,j}$ is the j^{th} symbol of packet P_i , $e_{i,j}$ is the j^{th} symbol of the encoded packet E_i , and $\alpha_{i,j}$ is the j^{th} packet coefficient of the encoded packet E_i . Let the number of symbols s be $\lceil \frac{l}{k} \rceil$, where l is

⁵Two or more encoded packets are linearly dependent if one of their encoding vectors can be represented by the other in the vector space

the packet length in bits, and k is the power of p in $GF(p^k)$. The general algebraic relationship of the matrices C , P , and E is given by:

$$\begin{bmatrix} \alpha_{1,1} & \dots & \alpha_{1,n} \\ \vdots & \ddots & \vdots \\ \alpha_{n,1} & \dots & \alpha_{n,n} \end{bmatrix} \times \begin{bmatrix} p_{1,1} & \dots & p_{1,s} \\ \vdots & \ddots & \vdots \\ p_{n,1} & \dots & p_{n,s} \end{bmatrix} = \begin{bmatrix} e_{1,1} & \dots & e_{1,s} \\ \vdots & \ddots & \vdots \\ e_{n,1} & \dots & e_{n,s} \end{bmatrix}$$

By substituting for the example, we get:

$$\begin{bmatrix} \alpha_{1,1} & \alpha_{1,2} \\ \alpha_{2,1} & \alpha_{2,1} \end{bmatrix} \begin{bmatrix} p_{1,1} & p_{1,2} \\ p_{2,1} & p_{2,2} \end{bmatrix} = \begin{bmatrix} e_{1,1} & e_{1,2} \\ e_{2,1} & e_{2,2} \end{bmatrix}$$

$$\begin{bmatrix} 17 & 136 \\ 72 & 28 \end{bmatrix} \begin{bmatrix} 145 & 167 \\ 120 & 228 \end{bmatrix} = \begin{bmatrix} 69 & 21 \\ 134 & 193 \end{bmatrix}$$

In our example, each of nodes N_3 and N_4 should at least get two linearly independent packets to be able to obtain the original packets P_1 and P_2 . They can request encoded packets from each of N_1 and N_2 . Upon getting the two linearly independent packets E_1 and E_2 and their encoding vectors, the decoder will be able to obtain the original packets P_1 and P_2 by solving the linear system:

$$\begin{bmatrix} 17 & 136 \\ 72 & 28 \end{bmatrix} \begin{bmatrix} p_{1,1} & p_{1,2} \\ p_{2,1} & p_{2,2} \end{bmatrix} = \begin{bmatrix} 69 & 21 \\ 134 & 193 \end{bmatrix}$$

To solve this linear system, Gaussian Elimination [38] can be used, which requires $An^3 + Bn^2 + Cn = O(n^3)$ operations for $n \times n$ matrix. In our example, the decoder

will obtain $P_1 = (145, 167)$ and $P_2 = (120, 228)$.

Note that for a successful decoding, the decoder must receive a sufficient number of linearly independent packets. Linearly dependent packets provide no new information and are useless for the decoder. In other words, if the encoding coefficient matrix does not have a multiplicative inverse due to linear dependence among its rows or columns, the system may become unsolvable. This issue can be mitigated by selecting a large Galois Field, such as $GF(256)$. A larger field offers a greater number of elements, making it statistically less likely for randomly chosen coefficients to create linear dependencies [39].

2.2.5 The Encoder and The Decoder

The encoding process of RLNC combines multiple data packets to construct encoded packets. The set of packets encoded together is called a *generation* [40]. The generation is so named because it is an independent decodable data unit. Upon successfully decoding it, a *generation* of the originally encoded packets will be released at once. The *generation size* refers to the number of packets the generation consists of, while the *packet size* refers to the size of the packet in term of bytes. A *packet* is a portion of an original data object or file, while an *encoded packet* is the output of the encoder. Note that the generation is also called segment, chunk, group, class, block, or cluster in the literature. Similarly, the packet is called a symbol, block, or piece [41].

Figure 2.7 shows the general inputs and outputs of the RLNC encoder and decoder, where p_i is a plain packet and (e_i, c_i) is an encoded packet and its encoding vector. The encoder takes n packets to construct one encoded packet at a time. Each encoded

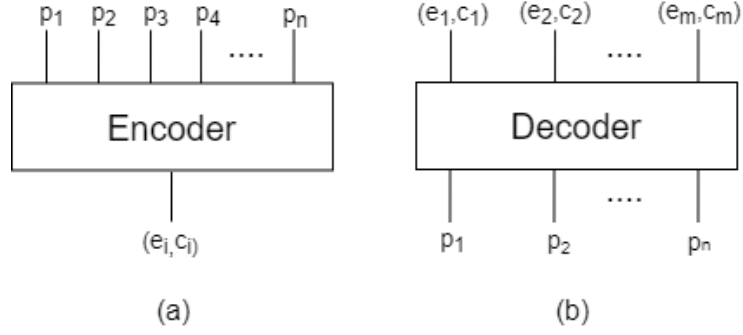


Figure 2.7: Illustration for the input and output for the (a) encoder and (b) decoder

packet is a transmittable unit, i.e. it can be directly shared with the decoder after its construction. However, an encoded packet is useless by itself and is only beneficial when decoded with other encoded packets of the same generation. Therefore, the decoding process cannot succeed unless a sufficient amount of linearly independent encoded packets are available. The decoder takes a set of those encoded packets (with their encoding vectors) and outputs a set of original packets at once. Note that m in Figure 2.7 can be larger than n due to the potential existence of linearly dependent packets.

The standard job of an RLNC encoder is matrix multiplication, with algorithm complexity ranging from $O(n^{2.376})$ to $O(n^3)$ for multiplying two $n \times n$ matrices [42]. To construct encoded packets, the encoder multiplies a coefficient matrix of size $n \times i$, where n is the generation size and i is the number of the constructed encoded packets (typically 1), by the packets matrix of size $n \times s$, where n is generation size and s is the number of symbols in a packet. The Galois field determines the number of symbols per packets depending on the field element size in term of bits. For example, for $GF(2^8)$, a packet of size 1KB will be composed of 1024 symbols (8-bits each), while

for $GF(2^4)$, it will be composed of 2048 symbols (4-bits each).

For the decoder, the process of solving linear equations can be reduced to a matrix-inversion problem solved by Gaussian elimination, which has a time complexity of $O(n^3)$ for $n \times n$ matrices. This implies that the time complexity of the decoder can be greater than the time complexity of the encoder due to the additional steps involved and the potential for numerical instability. Again, the number of rows and columns in the decoding matrices depends on the generation size and the chosen Galois field. The coefficients matrix size is at least $n \times n$, where n is the generation size, while the encoded packets matrix size is at least $n \times s$, where n is the generation size and s is the number of symbols per packet. These matrices can be larger when there are linear dependencies among encoded packets or if the decoder received more encoded packets than needed.

It's important to note that these complexities are theoretical upper bounds, and actual performance can vary depending on several factors, including the specific characteristics of the input matrices and the hardware on which the algorithm is run.

2.3 Why is Network Coding for P2P Content Distribution?

The main goals of using RLNC for P2P content distribution are to maximize throughput utilization, minimize redundancy (possibly caused by re-transmissions or sending duplicate packets by multiple senders), and minimize transmission control overhead (possibly increased by feedback messages sent as acknowledgments by receivers).

In computer networks, there are several schemes in which data can be transmitted among network members. The simplest scheme is point-to-point data transmission, where a single node sends data packets to another. However, to truly leverage the P2P topology benefits, in many scenarios, it is more desirable to enable multiple senders to send data to single or multiple receivers without redundancy. That may require coordination overhead and some scheduling techniques. In BitTorrent, there is a coordination server, called a *tracker*, that decides which data blocks should be downloaded from which peers [43]. However, since any peer can join or leave the swarm at any moment, this coordination may not be optimal [44]. Furthermore, the tracker server may introduce some system failure threats because peers will always be dependent on it [45].

With a typical data transmission protocol, packets may be lost/dropped causing the sender to re-send them. Those re-transmissions increase network traffic and communication latency. In addition, for reliable communications, the receiver of the data packets should send feedback in the form of acknowledgments constantly during transmission, which increases the overhead. The feedback messages also decrease the scalability in multicast scenarios, where the sender is overwhelmed with the feedback of many receivers.

For those issues, network coding, especially RLNC, promises to be the optimal option for efficient and reliable data transmission for point-to-point, multi-casting, multi-hops, multi-paths, and multi-sources to multi-destinations data transmission schemes in a completely distributed manner [46]. In RLNC, data receivers can decode without a coordination server; without knowing the network topology; and even if

senders and receivers are joining and leaving the network arbitrarily or part of the network is disconnected; or even when there are data losses and without the need to know where the losses are. Moreover, they can decode at the same time consistently if they have the same decoding speed [40].

RLNC is not the only rateless block codes that can generate an endless amount of encoded packets for a given finite plain packets without restarting the coding process from scratch. There are other block codes, called Fountain codes [47], [48], [49], [50], that have a much lower coding complexity than RLNC, including LT [51], [52], [53] and Raptor [54], [55] codes. Fountain codes are block-based forward-error-correction (FEC) schemes that offer rateless coding (i.e. they can generate encoded packets endlessly from a given finite number of original packets). They are end-to-end (E2E) codings that reduce the required feedback and make point-to-point and multicast communications more efficient. Several techniques have been proposed in the literature that employ these Fountain codes for P2P content distribution [56], [57], [58], [59], [60].

However, these Fountain codes are efficient only in end-to-end communications for classical single-source communications (point-to-point and multicast), but may not be efficient for multi-source communications [46]. Moreover, they incur a delay in multi-hop transmission because each hop should decode the entire packets, encode them again, then send them to the next hop. That may also require a larger buffer size at each hop [61]. Some proposed techniques try to overcome some of these limitations by combining Fountain codes with RLNC [62], [63].

While RLNC can behave as some other block codes (e.g., Reed-Solomon [64] and

Fountain codes), there are two more unique features of RLNC: 1) The re-coding property and 2) the sliding window encoding [46]. Recording enables middle nodes to make coded packets out of a subset of coded packets that are fully or partially coded by original senders. This is an important property because it enables multi-source data transmission efficiently since middle nodes can send different coded packets that are not redundant (or linearly dependent) for the receiver node(s) [46], [65]. The sliding window encoding property enables reliable real-time streaming with low latency for point-to-point data transmission [66].

CHAPTER 3

LITERATURE REVIEW

In this chapter, a rigorous review of the literature on IPFS, BitSwap, and network coding is presented. Firstly, IPFS and BitSwap advancements are discussed and the existing literature gaps are identified. Secondly, network coding data structures and coding schemes employed in the literature are classified and discussed.

3.1 IPFS & BitSwap Advancements

In the early 2000s, P2P networking was proposed as an alternative network architecture paradigm and extensively studied [67]. The potential benefit of P2P networking is that it can offload both central network servers and backhaul links by handling data transfers in a P2P manner [68]. This architectural paradigm was not only promising for general file transfers [69], but also for CDN architectures [70], VoD platforms [71], [72], and real-time VoIP communication.

The research by Nygaard [73] tackles inefficiency in decentralized data storage with three proposed solutions. Snarl, paired with entangled Merkle trees, repairs missing

data chunks efficiently, boosting storage utilization and data accessibility. SNIPS, another contribution, tackles data synchronization across devices. It uses cryptographic proofs to significantly reduce the data required for this process. Lastly, SUP, the third innovation, checks for existing copies of data before uploading, potentially slashing data transfers by up to 94%. These improvements combined can enhance decentralized data storage availability.

One of the most popular P2P file sharing systems is BitTorrent, a peer-to-peer file sharing protocol that allows users to share large files over the internet and uses a tit-for-tat protocol to encourage users to share files with each other [74]. *Trackers* in BitTorrent are servers that help connecting peers who are downloading and uploading the same file. They act as a central hub for communication between peers and provide information about other peers in the swarm (group of users sharing the same file). The dependency on those trackers in BitTorrent network make it a decentralized P2P network that is not fully distributed.

Studies on improving the performance of P2P systems, such as BitTorrent, have primarily focused on enhancing the content routing system, with Distributed Hash Tables [75], pubsub protocols [76], and tracker operation improvements being the main areas of focus for optimization [44]. Other approaches, such as collaborative downloads to speed up downloads [77] and application-specific content discovery based on user interests and social networking links [78], [79], have also been explored.

IPFS [1], as a more advanced and fully distributed file system, has recently gained traction among developers and enthusiasts who are interested in building decentralized applications or creating more resilient web infrastructure. One of the key features of

IPFS is its content-addressed system. Instead of relying on trackers to locate and exchange data, IPFS uses a novel BitSwap protocol for data exchange. BitSwap is a message-based, peer-to-peer network protocol used for both content routing and data transfer in IPFS [80]. It allows IPFS nodes to request and send data blocks between each other in the P2P content-addressable network [10].

In the context of IPFS advancements, previous research [81], [82], [83] has proposed several approaches to enhance IPFS performance. Le et al. [81] improved IPFS performance by implementing key-value memory caching, which reduces the time required to find a data provider and speeds up data retrieval from IPFS. They utilized blockchain technology to enhance the security and performance of IPFS.

Chen et al. [82] proposed a storage strategy for IPFS that combines triple replication schemes and erasure codes to solve the problem of large data storage for service providers. The authors suggest a better approach to the P2P file system using IPFS and Blockchain to tackle the issue of high-throughput for individual users in IPFS. They introduce content service providers to address data reliability, availability, storage overhead, and other concerns. To enhance the block storage model provided by IPFS, they propose a new zigzag-based storage model. Additionally, they integrate blockchain with this storage model. Based on their evaluation, the proposed model can effectively resolve the aforementioned problems.

Song et al. [83] introduced an efficient data transmission scheme for IPFS called IPFSz, which enhances I/O performance by enabling real-time compression while maintaining decentralization and P2P communication features. In IPFSz, data is compressed and decompressed during I/O operations using a compression algorithm.

Experimental results demonstrate that IPFSz outperforms existing IPFS by up to $7.9\times$ in terms of performance and saves storage space by up to $6.3\times$.

To the best of our knowledge, the only attempt to use RLNC technology with IPFS was made by Zimmermann et. al. [9]. With the aim to reduce the servers load in Content Delivery Networks (CDNs), they introduced Multi Access Recoding System (MARS), a protocol that combines IPFS and RLNC. Simulations have demonstrated that MARS can cut server load and download time by up to 50% compared to a single server transmission and up to 45% compared to the same setup without RLNC. Furthermore, MARS reduces download time by at least 15% even for relatively few peers in the network or peers with limited storage.

However, their proposed approach is primarily designed to alleviate server load in CDNs and is not intended for implementing RLNC in the BitSwap protocol for standard P2P file sharing. In fact, we do not perceive any advantage in utilizing RLNC for reducing server load, as IPFS is inherently distributed, and RLNC does not offer any additional benefits in this context. This poses a significant threat to the construct validity of the study. Furthermore, the evaluation of this approach overlooked decoding time, a crucial metric that should be monitored to assess the introduced coding complexity imposed on network members. Additionally, the study fails to provide a detailed protocol design and does not employ generation-based RLNC, making the proposed approach inefficient for large file sizes. Another critical issue with their approach is the modification of the file addition operation to incorporate encoding operations at file addition, leading to the creation of different CIDs for the same file, which disables file sharing with other peers and significantly disrupts the normal op-

eration of IPFS. Moreover, the evaluation was conducted using a single file size and network setting, limiting the generalizability and reliability of the results.

Insufficient attention has been given to the BitSwap protocol’s data exchange mechanism. We identified only one study [10] that focuses on enhancing BitSwap content discovery rates by employing information gathered from the content routing subsystem to make more informed decisions about content retrieval sources. This study introduces various BitSwap enhancement strategies to enhance its effectiveness, efficiency, and reduce its bandwidth consumption.

In this study, we focus on enhancing the data exchange mechanism of BitSwap by designing a comprehensive protocol that leverages RLNC, while carefully evaluating the advantages and the introduced complexity associated with coding operations.

3.2 Network Coding

This section presents a comprehensive review of nearly two decades of network coding literature. It explores the practical applications of network coding-based P2P systems in the literature and classify coding data structures and schemes.

3.2.1 Review Methodology

Search Databases

The search databases that will be searched to find related studies are IEEEExplore, ACM Digital Library, and Web of Science (WoS). These databases are selected because they are known for indexing computer science-related studies and they should capture

most related studies that answer the research questions.

Time frame

The search of this review will be restricted to the maximum possible date range supported by the selected search databases, which is from 2005-2023. The time frame of this review is chosen to be large to answer the research questions as comprehensively as possible. Additionally, RLNC was proposed in 2006, making the selected time frame cover almost all related publications.

Search Strings

The search string should be identified carefully to capture all related studies that can provide answers to the research questions. After several trials, the identified search string is:

(p2p OR "peer to peer") AND (file OR content OR data) AND (system
OR distribution OR shar* OR app* OR stor*) AND "network coding"

Note that the asterisk (*) at the end of partial words tells the search engine to find all possible endings for the word. For example, the term (shar*) will include share, sharing, shared, ...etc.

Inclusion & Exclusion Criteria

In conducting the review on network coding, the search databases results will be filtered by their relevance to the topic. After that, the relevant studies will be filtered again using the inclusion and exclusion criteria outlined as follows:

The selection of relevant studies in this review will be based on the following inclusion criteria:

- The study is related to network coding.
- The study is related to P2P content distribution.
- The study is written in English.
- The study is either published in a journal or a conference.

The exclusion of the relevant studies will be based on the following exclusion criteria:

- The study is related to client-server storage.
- The study is related to network security or proposing security measure.
- The study is related to Wireless Ad-hoc Networks (WANET), Mobile Ad-hoc Networks (MANET), or Vehicular Ad-hoc Networks (VANET) due to their distinct characteristics and challenges.
- The study is related to data error detection, repair, and recovery.
- The study is a review or a survey.

3.2.2 Coding Data Structures

Before starting the coding operations on a data object, it should be partitioned and re-structured to be processed by the encoder. In this section, we present and discuss the

Table 3.1: Classification of network coding data structures and schemes

Category	Count	Proportion	Papers
Coding data structure			
Multi-generations structure	32	78.0%	[84], [85], [86], [87], [88], [89], [90], [91], [92], [93], [94], [95], [96], [97], [98], [99], [100], [101], [102], [103], [104], [105], [106], [107], [108], [109], [110], [111], [112], [113], [114], [115]
Single-generation structure	8	19.6%	[116], [117], [118], [9], [119], [120], [121], [122]
Multi-generations with overlapping structure	1	2.4%	[123]
Coding scheme			
Complete coding	29	70.8%	[116], [84], [118], [85], [86], [88], [9], [92], [93], [94], [115], [95], [96], [98], [99], [100], [103], [104], [121], [105], [106], [107], [122], [108], [109], [110], [111], [112], [123]
Sparse coding	8	19.5%	[117], [87], [89], [90], [91], [119], [97], [120]
Deterministic coding	3	7.3%	[102], [113], [114]
Sliding window coding	1	2.4%	[101]

three literature approaches in structuring the data prior to applying coding algorithms. These structures are the single-generation structure, the multi-generation structure, and the multi-generation with overlapping structure (recall that a generation is an independent decodable data unit as discussed in Section 2.2.5). The studies that used each of them are presented in Table 3.1.

Without loss of generality, we focus our discussion on files as they are the main transmittable data objects in content distribution networks. Consider the case where encoding is to be applied on a file F by an encoder. The very first and necessary step

is to partition it into a set of n packets $P_n = [p_1, p_2, \dots, p_n]$. The number of packets n can be either a fixed number determined by the partitioning algorithm, or it can be a variable dependent on a fixed packet size in term of bytes.

Single-Generation Structure

The first and simplest structure is called *single-generation* structure, where the entire set of packets P belongs to one generation G , i.e., $G = P$. This structure is used by about 20% of the studies. It is easy to implement. Design-wise, it only involves the determination of the appropriate packet size in terms of bytes and the number of packets within a generation (i.e., the generation size). However, having a large number of packets (or a large generation size) will have an impact on the coding complexity, which can grow exponentially as the file size increases. Therefore, this structure is uncommon and may not be a good design decision for large file sizes due to its high coding complexity [39].

Multi-Generation Structure

This structure is used by the majority of the studies (shown in Table 3.1) because it lowers the coding complexity. This structure is also called generation-based RLNC [124], in which the file is partitioned into multiple smaller independent generations. Let file F consists of n packets p . The set of packets P_n are separated into multiple generations $[g_1, g_2, \dots, g_k]$ so that $g_i = [p_s, \dots, p_e] \subset P_n$, and $1 \leq s < e \leq n$.

The multi-generations structure offers an important advantage by significantly reducing the computational complexity associated with the coding process. This struc-

tured approach scales down the complexity by a factor of k , where k represents the number of file generations. Consequently, the complexity exhibits linear growth in relation to the file size, in contrast to the exponential growth of the single-generation structure, where complexity increases exponentially with the increase in the file size. In addition, since this structure provide some control over the complexity, the generation size can serve as a user-defined constraint for the coding delay [125].

This structure introduces minor implantation issues that should be handled. Since the file is splitted further into generations, there is extra information that should be communicated to restore the file after successfully decoding its generations. For example, each coded packet should contain the generation number it belongs to, and each generation should contain the generation sequence number and the total number of generations of the original file.

Moreover, this structure may increase the transmission control overhead by introducing a new problem: For a file consisting of k generations, which generation should be sent next? This problem is called the generation scheduling problem [126], [127], [40], and it can be handled using various approaches depending on the transmission scheme (point-to-point, multicast, multi-hops,...etc.) and the distribution of the generations among the peers (whether each of them own all or part of the file generations). Generation scheduling can be made by control messages from the requester, where it requests specific generations instead of a file, by round-robin algorithm, where generations are selected sequentially, or by random selection, where any generation is selected by random [40], [127].

The round-robin scheduling is not scalable in multicast scenarios because receivers

may not have the same decoding speed, and senders are forced to wait until all sent generations are successfully decoded by the receivers before moving to the next generation. Random scheduling introduces the coupon collector problem [128], [126], where the receiver receives redundant packets for decoded generations while waiting for the last generation to become decodable. On the other hand, random scheduling provides the “rateless” property which reduces the need for receiver feedback [126].

Nevertheless, this issue can be resolved in scenarios where the scheduling control can be made by the requester. In this case, the requester can simply request generations by their unique identifying addresses instead of requesting the file itself.

Multi-Generation with Overlapping Structure

A special structure that is proposed to mitigate the multi-generations random scheduling problems is called *multi-generations with overlapping* [129], [130], [131], [132]. Out of the reviewed studies, only one study has used this structure [123]. In this structure, file packets can belong to more than one generation. Let g_i and g_j be two generations of a file. The intersection of the sets of packets of the two generations is not empty ($g_i \cap g_j \neq \phi$). The redundancy in packets across generations enables the decoder upon successfully obtaining the original packets of a generation to use some of them in decoding other generations. In other words, the overlapping enables the decoder to alternate between Gaussian elimination and back substitution to solve the linear system. This can also decrease the time in which the decoder waits for linearly independent packets for a generation, especially if some encoded packets are missed or are rare due to P2P network dynamics. Consequently, this structure enhances content

availability and system robustness. However, this comes at the cost of an increase in computational complexity due to the increase in generation size [126].

3.2.3 Network Coding Schemes

The network coding schemes determine how the encoding and decoding algorithms of each network member work. In other words, the network coding scheme, or coding scheme for short, answers the question: Given a set of packets that belong to a generation, how the encoder encodes them, and how the decoder decodes them? According to the reviewed studies, four coding schemes are used in P2P content distribution applications. These schemes are: Deterministic coding, complete coding, sparse coding, and sliding window coding. In this section, each of these coding schemes is described and its most important related studies are discussed.

Deterministic Coding

The deterministic network coding scheme specifies exactly how packets are encoded or decoded for each of the network members. This coding scheme is static in the sense that it tells how coding is performed by each connected peer in the network. In case of peers' departure or arrival, the coding scheme should be updated. A simple form of deterministic network coding is explained in the butterfly example discussed in Section 2.2.3, where its advantages and disadvantages are also discussed. The role of each encoder and decoder in the network is known, depending on how the network is configured. This scheme is highly dependent on network topology and requires global awareness about it. Therefore, it is best used for static network topologies, and not

practical for dynamic networks.

This coding scheme is used by three related studies. The study in [114] proposed a network coding technique developed primarily to improve the throughput and reliability of communication networks. This technique is very similar to the one explained in the butterfly example in Section 2.2.3, in which the intermediate nodes simply XORing data packets before transmitting them to the end nodes. In order to obtain the original data, the received encoded packets are decoded in the end nodes by XORing them with the uncoded received packets. The performance of this deterministic network coding scheme in collaborative multi-source settings is studied with diverse multimedia resources (e.g., images and PDF files). The proposed multi-source strategy is helpful when different components of multimedia data objects are generated or kept in two or more locations. The study evaluated the performance of P2P network coding compared to that of client-server typical networking. The results suggested that utilizing a network coding scheme in a collaborative multi-source scheme has significant advantages.

Another study [113] presented a dynamic network coding strategy for a collaborative multi-source system. Peers are synchronized by a coordinator server, which is in charge of assigning dynamic roles to the system's nodes during the network coding process. In addition, the coordinator server must ensure that the network coding procedure has finished. This study has shown benefits in reducing content distribution time in the collaborative system.

Min Yang and Yuanyuan Yang [102] proposed a technique for applying network coding for file sharing that makes use of a P2P network to distribute files stored on a

server. This technique makes use of a form of network topology known as a combination network. The proposed technique divides peers into several groups and encodes a file into multiple messages, with each group responsible for relaying one of the messages. The encoding scheme is intended to satisfy the property that any subset of the messages can be used to decode the original file as long as the subset size is large enough. This study developed a deterministic linear network coding scheme that satisfies the desired property, in which peers are connected in the same group to flood the corresponding message, and peers in different groups are connected to distribute messages for decoding. Furthermore, the technique is extensible to accommodate connection heterogeneity and topology awareness, which may be used to increase system performance in terms of throughput, reliability, and link stress. The simulation results demonstrated that the proposed technique can achieve 15%-20% greater throughput than Narada (an another P2P multicast system that does not use network coding [133]), and it can achieve high reliability and robustness in the face of link failure or peer churn.

Complete Coding

The complete coding (also called full coding and dense coding) scheme combines the entire set of data packets to construct encoded packets, as we have explained earlier in the example in Section 2.2.4. Let $P = [p_1, p_2, \dots, p_n]$ be a set of original packets (of a generation). Each encoded packet e_i is constructed by the encoding algorithm using the formula:

$$e_i = \sum_{j=1}^n \alpha_{i,j} \times p_j \quad (3.1)$$

where $\alpha_{i,j}$ is a randomly selected coefficient from a finite field $GF(q)$. To obtain the original packets, the decoder has to obtain at least n linearly independent encoded packets associated with their encoding vectors (coefficients). Then, Gaussian elimination algorithm [38] can be used to restore the original set of packets P .

Two important parameters determine the level of computational complexity involved in this scheme: 1) Galois field size, and 2) generation size n (the number of packets combined to construct encoded packets). As the value of these two parameters increases, the computational complexity increases [87], [134].

Most of the reviewed studies have used the complete coding scheme, as shown in Table 3.1. Some of them have used the single-generation structure while others have used the multi-generations structure. The most important studies of each will be discussed next.

Single-generation Structure. The study by Wang and Ansari [116] presented a mathematical analysis based on modern P2P file-sharing protocols to examine the efficiency of network coding in P2P file-sharing networks. Considering the unlucky combination problem (related to the linear dependency problem) that RLNC algorithm encounters, they presented a Downloader-Initiated Random Linear Network Coding technique that allows the downloader to choose the random linear coding to minimize the unlucky combination problem and thereby enhance the performance. A similar approach is given by Leu et al. [122], where they proposed a network coding-

based file sharing system for unstructured P2P networks that allows selecting coding coefficients at session start to reduce the probability of receiving redundant linearly dependent coded packets.

Due to the higher likelihood of node failures, communication channel losses, and limited device resources in P2P storage networks when compared to centralized storage solutions, it is critical to minimize data retrieval time while also retaining high system robustness. To approach that, Marandi et al. [118] suggested a solution based on RLNC for data storage and retrieval, as well as the usage of Kademlia. They compared the theoretical performance of the new RLNC-based technique to the standard Kademlia in terms of data retrieval time and resilience to node failures and channel losses. The theoretical analysis and simulation findings revealed that the proposed RLNC-based technique greatly outperforms standard Kademlia.

With the aim to reduce the servers load in CDNs, Zimmermann et al. [9] introduced Multi Access Recoding System (MARS), a protocol that combines IPFS and RLNC. Simulations have demonstrated that MARS can cut server load and download time by up to 50% compared to a single server transmission and up to 45% compared to the same setup without RLNC. Furthermore, MARS reduces download time by at least 15% even for relatively few peers in the network or peers with limited storage.

One significant problem for RLNC is the substantial transmission cost caused by transmitting enormous coefficients vectors as headers. Furthermore, by using the Gauss-Jordan elimination algorithm [135], significant computational cost can be imposed on peers when decoding encoded packets and checking linear dependency among coefficient vectors. To solve these issues, Barekatin et al. [115] introduced MATIN,

an RLNC-based architecture for efficient P2P video streaming. MATIN features a novel coefficients matrix creation mechanism, which ensures that the resulting coefficients matrix has no linear dependency. Using the suggested architecture, each peer encapsulates one coefficient entry instead of n into the resultant encoded packet, resulting in very reduced transmission overhead. The simulation results showed better performance compared to typical coding in RLNC.

Multi-generations Structure. Gkantsidis and Rodriguez [84] were the first who proposed an RLNC-based file system for large-scale content distribution, called Avalanche. They investigated network coding performance in heterogeneous networks with dynamic peers joining and leaving patterns, clustering topologies, and incentive systems to prevent free-riding. The simulations on some network scenarios indicated that network coding improves the estimated file download speed by more than 20-30% compared to source-only coding, and by more than 2-3 times compared to distributing unencoded data. An analysis study on Avalanche [136] presented methods for system analysis and showed that RLNC offers the system with maximum bandwidth efficiency and robustness.

Wang and Li [93] attempted to answer the question: How helpful is network coding in P2P streaming? They used Gauss-Jordan elimination [135] to implement the decoding operations, which allows them to be performed as coded packets are received progressively. The study presented architectural design issues in implementing network coding for multimedia streaming, as well as a pull-based P2P live streaming protocol. The evaluation results showed that network coding allows for finer-grained streaming, which decreases redundancy in bandwidth utilization, enhances resilience

to network dynamics, and is especially useful when bandwidth supply barely meets streaming demand. The same authors revisited the problem and designed R^2 [98], a novel streaming technique that combines RLNC with a randomized push algorithm. It is intended to improve live streaming performance in terms of initial buffering delays, peer dynamics robustness, and bandwidth cost on master streaming servers.

Chung et al. [95] proposed a P2P Network Coding-based Streaming system (PNECOS) for providing an overlay multimedia streaming service. PNECOS utilizes network coding to provide stateless and independent encoded data, to efficiently utilize peer node buffer capacity, and to balance network traffic load and server load. The simulation results suggested that network coding can increase the effective buffer length of peers by increasing storage spaces for coded data, resulting in a significant reduction in server bandwidth consumption.

Nguyen et al. [111] proposed a network coding framework for efficient video streaming in CDNss and P2P networks in which several servers and peers are used to stream a video to a single receiver simultaneously. The study demonstrated how network coding techniques can 1) eliminate the need for strict synchronization between senders, 2) be easily integrated with Transmission Control Protocol (TCP), and 3) reduce server storage in CDNs environments. Additionally, it suggested using the Hierarchical Network Coding (HNC) approach with scalable video bit streams to mitigate Internet bandwidth fluctuation problem. The evaluation showed that, for some scenarios, the suggested network coding strategies can save up to 60% more bandwidth compared with traditional schemes.

Cui and Lu [103] proposed SocialStreaming, a P2P-assisted video streaming system

for social networks. They proposed a storage technique based on network coding that can increase the utilization of each peer’s limited storage capacity. This technique may increase the diversity of stored data with the same storage capacity, which can improve the efficiency of data sharing. The study presented a social network-based distributed streaming pre-delivery algorithm, derived from the Metropolis-Hastings algorithm [137]. The proposed algorithm can effectively deliver video streaming to some peers, who will most likely watch the video later. In addition, The study presented a social network-based neighbor selection algorithm, which combines the network performance factor and the social factor as a trade-off to select appropriate neighbors for data exchange.

Braun et al. [92] proposed two WebRTC-based protocols, called WebPeer and CodedWebPeer, to establish a P2P-assisted video-on-demand streaming system with network coding. WebRTC [138] is a technology that offers an API for establishing direct P2P connections between web browsers. The study presented two metrics: network health, which assesses total data saturation, and network stability, which determines whether peers in a network can serve each other without the assistance of the server. The study demonstrated that by using network coding, network health may be boosted by up to 100% without increasing the cache size or the number of peers. Furthermore, network stability is accomplished by employing up to half the cache in comparison to the uncoded scenario, without increasing the load on the server. The study validated the proposed system’s scalability by testing the protocols in a network of more than 100 tablets that delivers smooth video streaming.

From a practical standpoint, Wang et al. [94] investigated how network coding

works in a BitTorrent-like protocol. To examine the advantages of network coding in BitTorrent, they created NCTorrent, a BitTorrent client that supports network coding. The user behaviors of roughly 10,000 real-world BitTorrent sessions are tracked and used for evaluation. Based on the findings, a pessimistic conclusion on the advantages of network coding was reached: Network coding does not aid much in real-world environments as claimed by prior studies. The logged data, in particular, showed that the last block problem, where network coding is thought to be useful, only occurs rarely.

Multi-generations with Overlapping. One study [123] has used the multi-generations with overlapping structure for content distribution in P2P networks. In this study, a network coding method based on the concept of generation crossing is presented. The proposed approach allows a peer to decode the encoded packets even when just a portion of the generation packets is received. The evaluation findings indicated that with a small number of overlapped packets between succeeding generations, the success rate for delivering the entire file to peers increases dramatically. The findings also confirmed that generation crossing (or overlapping) can enhance resilience to peer dynamics in P2P networks.

Sparse Coding

To reduce the computational complexity incurred in complete coding, sparse coding is proposed [139]. In this scheme, the encoder selects a subset of packets of the generation. Let $P = [p_1, p_2, \dots, p_n]$ be packets of a generation. An encoded packet e_i is constructed by encoding m packets randomly selected from P , where $m < n$.

The value of m determines the coding *density*, which is defined by m/n . In complete coding, the density is 1, because $m = n$, while in sparse coding, it is always < 1 . The value of m determines the number of combined packets, where it can be a fixed value for all encoded packets or a tunable value based on the decoder behavior [140], [141].

The first step the encoder does is to select a coefficient for each original packet, as explained in Section 2.2.4. To partially select from the entire set of packets, some packets are assigned the value of zero for their coefficients. The encoder then multiplies the packets by their coefficients as in formula (3.1). This means that the packets in which their corresponding coefficient is zero will become equal to zero. In other words, when the encoder selects a zero coefficient for a packet, it decides to exclude that packet from the constructed encoded packet. In complete coding, all of the coefficients are non-zeros, leading to the inclusion of the entire set of packets into the encoded packet.

For example, consider the following coefficients matrix for four encoded packets, where $\alpha_{i,j}$ is the coefficient of the encoded packet i and j is the original packet number:

$$\begin{bmatrix} \alpha_{1,1} & \alpha_{1,2} & \alpha_{1,3} & \alpha_{1,4} \\ \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} & \alpha_{2,4} \\ \alpha_{3,1} & \alpha_{3,2} & \alpha_{3,3} & \alpha_{3,4} \\ \alpha_{4,1} & \alpha_{4,2} & \alpha_{4,3} & \alpha_{4,4} \end{bmatrix}$$

When constructing four encoded packets, the sparse coding encoder will randomly assign zeros to a number of coefficients according to the designated coding density. After the four encoded packets have been constructed, the shape of coefficients matrix

can be:

$$\begin{bmatrix} \alpha_{1,1} & 0 & \alpha_{1,3} & 0 \\ 0 & \alpha_{2,2} & \alpha_{2,3} & \alpha_{2,4} \\ \alpha_{3,1} & 0 & 0 & \alpha_{3,4} \\ 0 & \alpha_{4,2} & 0 & \alpha_{4,4} \end{bmatrix}$$

In this matrix, the first encoded packet encoded the original packets 1 and 3, the second encoded packet encoded 2 and 4, the third encoded packet encoded 1 and 2, and the forth encoded packet encoded 2,3, and 4.

The sparse coding scheme makes the decoding process faster due to the existence of zeros in the coefficient matrix [142]. Another benefit of the sparse coding scheme is the ability to perform partial decoding, which enables decoding part of the original packets before collecting all required encoded packets [143], [144]. However, sparse coding benefits come at the cost of a higher probability of having linearly dependent packets [145], which may increase the communication overhead due to re-transmitting encoded packets and may consume higher bandwidth [146].

The second most used coding scheme in the reviewed studies is the sparse coding scheme, where some of them used the single-generation structure while others used the multi-generation structure. The most important studies of each will be discussed next.

Single-generation Structure. Cai and Lo [117] investigated the viability of employing network coding to improve content availability of BitTorrent swarms [147]. In comparison to prior network coding schemes, they proposed a sparse network coding scheme with lower coding complexity and disk operation overhead. The proposed

coding scheme is implemented in BitTorrent, and a new block scheduling algorithm is developed based on BitTorrent’s original rarest-first block scheduling algorithm. The performance evaluation revealed that the proposed coding scheme is effective in terms of increasing the content availability of BitTorrent swarms.

SmartCode, a P2P content distribution system with sparse network coding, was introduced by Ma et al. [119]. SmartCode employs pre-checking to prevent the transmission of linearly dependent packets. Additionally, two systems, called Ecode and LRF, were implemented under the SmartCode architecture. Ecode system uses sparse network coding as well but without pre-checking. LRF system is similar to BitTorrent in that it uses local-rarest-first for block scheduling. According to the experimental results, SmartCode reduces distributing time by 11% on average and up to 19% when compared to LRF, and by 7% on average and up to 16% when compared to Ecode. SmartCode outperforms Ecode and LRF in downloading time and peer completion times in dynamical cases, where peers arrive to and depart from the content distribution sessions frequently and the seed leaves the session after it has distributed a specific proportion of blocks over the total blocks of the content.

Multi-generations Structure. Xy et al. [91] presented DirectedPush, a P2P video streaming system based on sparse network coding. In this system, the video is encoded into a number of coded sub-streams with enhanced content diversity. To distribute the coded sub-streams, peers are arranged into a two-layer overlay and use an efficient push method. The gain of using network coding in this system is a reduction in both packet arrival delay and coding overhead. Furthermore, the system became more resistant to peer dynamics. The performance evaluation demonstrated

that the system outperforms earlier techniques, particularly in challenging flash crowd scenarios [148], [149].

Ortolf et al. [90] proposed paircoding, a sparse coding approach for P2P file sharing. Paircoding encodes and distributes only a linear combination of two data packets, which alleviates BitTorrent’s coupon collector problem [150] without the high computational complexity introduced by complete network coding. It performs at least as well as BitTorrent and has approximately the same disk operations complexity. Paircoding outperforms BitTorrent in terms of network flow and performs as well as complete network coding in some scenarios.

Kaiqian et al. [89] proposed Dasher, a P2P multi-generations sparse network coding-based content distribution system. Additionally, they developed three comparative systems: a BitTorrent-like system called Mybt, a system that uses single-generation sparse coding called Sparser, and a system that uses multi-generations complete coding called Chunker. According to the comparison results, Dasher with particular generation sizes can cut average downloading time by up to 15% compared to Mybt, and up to 43% compared to Chunker. Using proper generation sizes, Dasher’s downloading time is nearly the same as Sparser’s. Dasher has the same average decoding rate as Chunker and is almost k times faster than Sparser, where k is the number of generations. Furthermore, in terms of robustness, Dasher performs worse than Sparser, nearly as good as Chunker, and better than Mybt.

Sliding Window Coding

In the sliding window coding scheme [151], the set of original packets (of a generation) is associated with a virtual sliding window that determines the next set of packets that will be encoded together to construct new encoded packets [152]. Let G be a generation consisting of a set of n packets $[p_1, p_2, \dots, p_n]$. A window with length w is associated with G , where $1 \leq w \leq n$, and it has a start index s and end index e , where $1 \leq s \leq e \leq n$. The state of the window includes the values of its length, start index, and end index. When a decoder requests encoded packets, the encoder will input the packets based on the current window state. In dynamic window size [153], the start length of the window is 1, then it is opened when the decoder acknowledges the encoder about successfully decoding the packet. To decrease the coding complexity during the transmission, the window closes to reduce the number of input packets. The window size can also be fixed during the transmission session [154]. The main benefit of this scheme is to have control over the coding computational complexity based on the network situation, by changing the window state accordingly. However, due to the required decoder feedback, this scheme may be used only for point-to-point data transmission.

Out of the reviewed studies, only one study, conducted by Su et al. [101], has used concepts of sliding window coding for content distribution in P2P networks. The study presented the concept of a size-adaptive encoding window based on resource distribution to reduce needless traffic overhead. Furthermore, encoding costs are decreased to some extent by employing the upper triangular encoding matrix rather than

the sparse encoding matrix employed by Kaiqian et al. [89]. Instead of addition and multiplication, the XOR operation on the Galois field is utilized, which can lower the coding computational cost. The evaluation results revealed that the proposed design outperforms BitTorrent and sparse coding in terms of both download time and coding degree.

3.2.4 Implementation details

The technical implementation details of network coding can have a significant impact on performance. To compare fairly between the performance of two network coding implementations, many dependent variables (parameters) should be consistent. In this review, we focused on how the literature reported the values of five parameters: programming library, Galois field, packet size, generation size, and file size.

Programming library

Despite its well-crafted implementation, Kodo¹ [155] library was only used by a few studies. Kodo is an open-source software library that provides an efficient implementation for network coding functions using several coding schemes and different Galois fields, including $GF(2)$, $GF(2^8)$, and $GF(2^{16})$. It is fully tested, built with C++ programming language, and works well with many platforms, including Windows, iOS, Linux, and Raspberry Pi.

Another network coding library is called `libgf2`, designed by the authors of [156]. `libgf2` is implemented with C++ programming language and supports binary Galois

¹<https://kodo.steinwurf.com/> (Accessed: 2024-04-15)

field $GF(2)$. This library was used by only one of the reviewed studies.

Various libraries, such as `kodr` [157] and `rnc-lib`², provide support for RLNC operation over $GF(256)$, but none of the reviewed studies used them. Most reviewed studies have used their curated implementation for network coding functions. The difference in the algorithmic details of each implementation can have an impact on the coding performance [62]. In addition, these libraries are not publicly available and have not been extensively tested. Therefore, the reliability of the results obtained from studies that utilize those curated libraries cannot be definitively established.

Galois field

The Galois field has three main impacts on coding performance [39], [134]. The first impact is on the linear dependency, where small field sizes have a higher probability to construct linearly dependent packets because the random selection of the coding coefficient is limited to few options [158]. The second impact of the Galois field is on the sizes of the encoding vector added as a header to encoded packets. As the field size increases, each of its elements is represented by a larger number of bits, which requires more space and increases the communication overhead. The third impact of the Galois field is on the computational complexity required for arithmetic operations. If the Galois field size is small, then the products of its elements can be stored in a look-up table in memory so that no computation is required for field element multiplications. On the other hand, if the Galois field size is large, then storing all elements' products will require a large space. For example, consider the field $GF(2^{16})$.

²<https://github.com/avisegradi/rnc-lib/> (Accessed: 2024-04-15)

Table 3.2: Look up table for multiplication operations over $GF(2^2)$ using the irreducible polynomial $x^2 + x + 1$

$\times$	00	01	10	11
00	00	00	00	00
01	00	01	10	11
10	00	10	11	01
11	00	11	01	10

Storing the combinations of all elements products requires $2^{16} \times 2^{16} \times 16\text{bits} = 8.6 \text{ GB}$.

Therefore, online computation for arithmetic operations (discussed in Section 2.2.2) is more practical. For $GF(2^2)$, the size of look-up table is $2^2 \times 2^2 \times 2 \text{ bits} = 32 \text{ bits} = 4 \text{ bytes}$. An example of a look-up table for the multiplication operations over $GF(2^2)$ is shown in Table 3.2.

The dominant used Galois field in the literature is $GF(2^8)$, where 17 studies (41.8%) have used it. Despite its significant impact on the performance, the Galois field size is not reported by 10 reviewed studies (23.5%). Missing such important information prohibits the reproduction of the results obtained in the evaluation experiment and decreases the reliability of the study. Three other Galois fields are also found: $GF(2)$, used by 8 studies, $GF(2^{16})$, used by 6 studies, and $GF(2^3)$, used by 1 study.

Generation size

The generation size is the number of packets inputted into the encoder to construct encoded packets. The value of the generation size is an absolute positive number without a unit. As the generation size increases, the coding computational complexity and the encoding vector size increase. In addition, the generation size impacts the

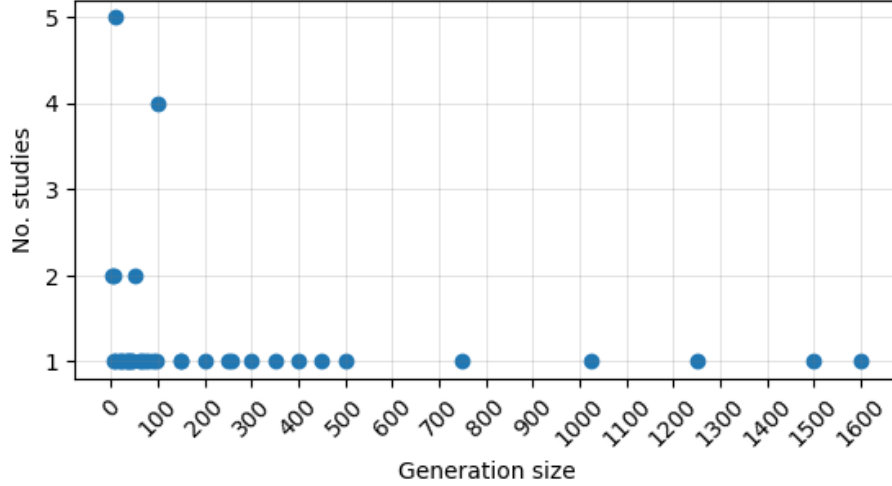


Figure 3.1: Thirty-six generation sizes used by 23 studies related to P2P content distribution

period in which the decoder has to wait until it gets a sufficient amount of decodable packets. However, as the generation size increases, the probability of having linear dependency decreases [134]. Therefore, the selection of the generation size is a trade-off between decodability and efficiency.

For 23 studies that have reported the generation size, there are 36 generation sizes used in the evaluation experiments. Most studies (19) have used only one generation size, while other studies (4) have tried several sizes. The minimum generation size is 2 and the maximum is 1600. Most generation sizes (21 of 36) are 100 or less. The generation sizes and the number of studies used each of them are shown in Figure 3.1.

Packet size

The packet is the smallest unit inputted into the encoder and is part of an original larger data object (e.g., file or generation). The packet size is measured in bytes or equivalent units. As the packet size increases, the coding computational complexity

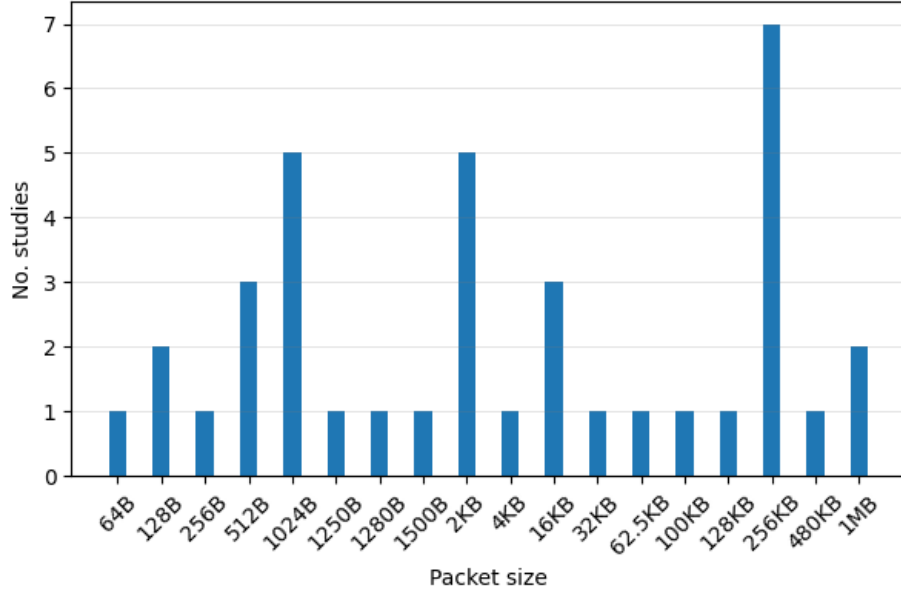


Figure 3.2: Eighteen packet sizes used by 26 studies related to P2P content distribution

increases. However, increasing the packet size does not increase the encoding vector size and it does not have an impact on the linear dependency problem [158].

For 26 studies that have reported packet sizes, 18 distinct packet sizes are used in the evaluation experiment. The minimum packet size is 64B and the maximum packet size is 1MB. The most used packet size is 256KB, followed by 1KB and 2KB. Figure 3.2 shows the packet sizes used in the literature and the number of studies used each of them.

File size

The file is a container that stores data, information, or commands used by computer applications. The increase in the file size will lead to an increase in the packet size, generation size, or both of them.

For 18 studies that have reported file sizes, there are 22 distinct file sizes that are

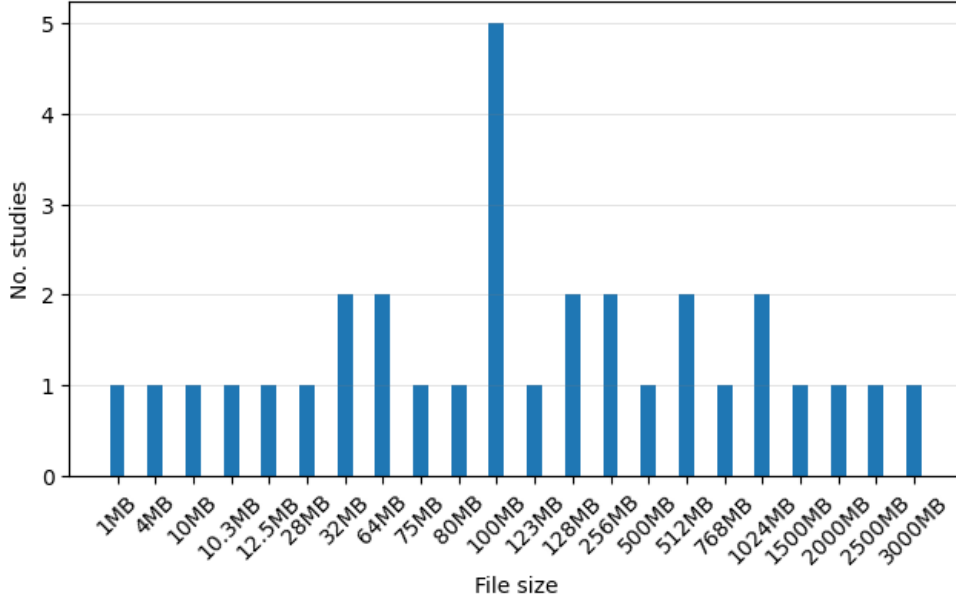


Figure 3.3: Twenty-two file sizes used by 18 studies related to P2P content distribution

used in the evaluation experiment. The smallest file size is 1MB and the largest file size is 3000MB. The most used file size is 100MB. Figure 3.3 shows the file sizes used in the literature and the number of studies used each of them. The diverse range of file sizes employed in network coding evaluations highlights the need for comprehensive testing and analysis across a wide spectrum of data volumes to ensure the effectiveness of network coding in real-world applications.

3.2.5 Evaluation metrics

For the 41 reviewed studies, there are 44 metrics used to evaluate the implementation of network coding. These metrics are analyzed and categorized into five categories: 1) Availability metrics, which assess the availability of the content in the P2P network, 2) Communication overhead metrics, which assess the amount of data transmitted be-

tween network members, 3) Computational complexity metrics, which assess the computational processing required for coding operations, 4) Effectiveness metrics, which assess the usefulness of network coding for content distribution, and 5) Streaming quality metrics, which assess the quality of network coding-based multimedia streaming. These categories, metrics, number of studies used them (Count column), and the studies reference are shown in Table 3.3.

Unsurprisingly, about half of the studies have used the download time metric to evaluate data downloading performance, since it is the most notable feature by the user. Most metrics have not been used widely, which indicates the lack of a standard framework for evaluating network coding-based P2P content distribution systems. The heterogeneity of these metrics makes the comparison between different approaches infeasible, which hardens the judgment on the practicability of a proposal due to the lack of a prior study that used the same metrics. In addition, some studies may use the metrics that are believed to demonstrate the validity of the proposed network coding-based system, while neglecting important metrics that may have been negatively impacted. For example, the study of Zimmermann et al. [9] has stated explicitly that "assembling and storing the file is not part of the measurement", where the download time is measured as the time from initiating a block request until it's received. How about the decoding time and the time it takes to assemble the entire original file? Ignoring such measurements may report biased results that favor some measurements over other important ones.

While network coding promises to improve network communications, it also introduces non-negligible computational complexity that is considered to be a main

Table 3.3: Metrics used in evaluating network coding-based P2P content distribution systems

Category	Metric	Count	Ref.
Availability	Success rate of downloading	3	[117], [86], [100]
	Percentage of peers who successfully decoded the entire content	2	[120], [88]
	Percentage of decoded blocks during transmission time	1	[90]
	Download failure rate	1	[110]
	Content loss rate	1	[100]
	Content lifetime	1	[100]
	Percentage of peer-assisted downloadings	1	[103]
	Network health	1	[92]
	Network stability	1	[92]
	Number of streaming requests served	1	[106]
Communication overhead	Number of control messages	3	[117],[122], [104]
	Server upload data amount	3	[9],[86], [105]
	Number of re-transmissions for same data	3	[102], [119], [121]
	Percentage of redundant data	3	[93], [98], [85]
	Network load	2	[9], [84]
	Uplink bandwidth consumption on the server	2	[98], [95]
	Link stress	2	[102], [110]
	Server bandwidth consumption	1	[103]
	Percentage of extra re-transmissions	1	[86]
	Drop rate	1	[101]
Computational complexity	Decoding speed	2	[89], [85]
	Percentage of CPU utilization for coding	1	[85]
	Encoding speed	1	[85]
	Number of valid decodable blocks	1	[101]
	Encoding time	1	[91]
	Decodability	1	[111]
Efficiency	Download time	20	[116], [118], [89], [9], [123], [119], [105], [110], [85], [100], [113], [114], [87], [84], [94], [120], [121],[122], [101], [86]
	Throughput	3	[102], [107], [110]
	Bandwidth utilization	2	[117], [85]
	Data storage time	1	[118]
	Average start-up time	1	[101]
	Transmission time	1	[85]
	Average streaming rate	1	[106]
	Start up latency	6	[91], [96], [104], [115], [97], [111]
	Continuity index	5	[91], [112], [96], [99], [104]
	Percentage of skipped playbacks	2	[93], [98]
Streaming quality	Percentage of completely received segments in playback buffer	2	[93], [98]
	Playback lag	1	[104]
	Video switch lag	1	[104]
	Streaming quality	1	[106]
	Fraction of late frames	1	[106]
	Video distortion	1	[115]
	Frame dependency distortion	1	[115]
	Playback error rate	1	[108]

drawback. Despite that, only five studies, as can be seen in Table 3.3, have measured the computational complexity as part of their evaluation. Understanding how much complexity is incurred in coding operations is important to study the trade-off between the cost paid and the value gained.

3.2.6 Practical benefits

Tables 3.4 and 3.5 contain a summary of each reviewed study including the implementation details, main findings, and baseline protocol used in the evaluation for content distribution and multimedia streaming, respectively.

Table 3.4: Main findings and implementation details for content distribution network coding-based P2P systems

ID	Ref.	GF	Coding scheme	Coding struc- ture	Main findings
1	[84]	$GF(2^{16})$	complete	multi- generations	The performance benefits provided by network coding in terms of throughput are more than 20-30% compared to coding at the server, and can be more than 2-3 times better compared to transmitting unencoded blocks.
2	[120]	$GF(2^8)$	sparse	single- generation	Coded P2P session offers poorer performance in terms of downloading times.

3	[87]	no info	sparse	multi- generations	If the network coding parameters are chosen properly, the system performance can be promoted by at most 20% to 30% in terms of download time.
4	[105]	$GF(2^8)$	complete	multi- generations	Network coding has advantages in content distribution over a dynamic network. Swifter has high performance with respect to short download time and low computational cost.
5	[89]	$GF(2^8)$	sparse	multi- generations	Dasher, with certain generation sizes, has less average downloading time compared to both Mybt and Chunker, and close to Sparser.
6	[90]	no info	sparse	multi- generations	Paircoding outperforms BitTorrent while complete RLNC outperforms both of them. In the tradeoff situation of disk operations and file sharing effectiveness, Paircoding ranks between the other schemes as an excellent compromise.
7	[123]	$GF(2^8)$	complete	multi- generations with overlap- ping	The benefit is reduction in download time and resultant resilience to peer dynamics in a P2P system.; no overlapping

8	[119]	$GF(2^8)$ sparse	single-generation	newEcode outperforms the noncoding system (LRF) and the coding system without prechecking (Ecode), with less downloading time and higher robustness.
9	[109]	$GF(2^8)$; complete $GF(2^{16})$	multi-generations	The proposed scheme reduces the amount of data flowing in the network, thus obtaining more free disk space on storage nodes, at the price of a very small reduction in the resource availability.
10	[94]	$GF(2^8)$ complete	multi-generations	Results on the benefits of network coding in BitTorrent systems are somewhat pessimistic. The last block problem is rare and BitTorrent itself works well in most cases without network coding.
11	[116]	$GF(2)$ complete	single-generation	Network coding can help a P2P file sharing network distribute data more effectively

12	[100]	$GF(2^8)$	complete	multi- generations	The true benefit of network coding lies in its resilience to peer churn. Small generation sizes, less than 20, suffice to realize these benefits in terms of reducing download times and enhancing content availability.
13	[121]	$GF(2)$	complete	single- generation	Linear coding only at the sources is not viable, while the one based on network coding allows an efficient data dissemination inside the swarm, where entire swarm network is positively influenced in such cases.
14	[117]	$GF(2^8)$	sparse	single- generation	With the proposed network coding scheme, the duration of period during which no seed is present but content is available is greatly prolonged, and only a few peers are enough to render the content available.
15	[102]	no	deterministic	multi- generations	Simulation results show that the new scheme can achieve 15%–20% higher throughput and increase reliability and robustness to link failure or churn.

16	[85]	$GF(2)$	complete	multi- generations	The purposed scheme effectively reduces link usage of the server and decreases overall transfer time by utilizing bandwidth resources available in the peers.
17	[88]	no info	complete	multi- generations	The proposed implementation outperforms the baseline BitTorrent implementation by up to 20% in terms of download time in certain scenarios.
18	[122]	$GF(2)$	complete	single- generation	Network coding has benefits if certain conditions were met, including using downloader-initiated methods to avoid unlucky combinations; carefully selecting the generation and packet sizes to avoid the coding speed being slower than the transmission speed; and utilizing early decoding to resolve the problem of long delay time.
19	[86]	no info	complete	multi- generations	CodedWebPeer protocol can take over the server load and serve 80% of the data. It can also be beneficial for clients because they can download data faster.

20	[114]	$GF(2)$	deterministic	multi-generations	The performance of a P2P networking with network coding against a client-server communications shows great benefits in collaborative multi-source scheme.
21	[101]	$GF(2)$	window	multi-generations	The average download time of the proposed system is 21.0% and 3.17% lower than other schemes respectively, which effectively improves the distribution efficiency.
22	[9]	no	complete	single-generation	In comparison to HTTP and IPFS, MARS was able to achieve large gains in terms of server load and download time.
23	[110]	no	complete	multi-generations	Dynamic Smart Network Coding technique enhances the performance of network coding in P2P content distribution with respect to scalability, reliability and resilience against peer's churn.
24	[118]	$GF(2^8)$	complete	single-generation	The proposed method achieves lower data retrieval time and higher resiliency to peer churn and channel losses.

Table 3.5: Main findings and implementation details for multimedia streaming network coding-based P2P systems

ID	Ref.	GF	Coding scheme	Coding structure	Main findings
1	[99]	$GF(2^{16})$	complete	multi- generations	The performance advantages of using network coding over that not using network coding have been demonstrated.
2	[93]	$GF(2^8)$	complete	multi- generations	Network coding has demonstrated its advantages in P2P live streaming when peers are volatile and dynamic with respect to their arrivals and departures.
3	[97]	no info	sparse	multi- generations	The proposed scheme can result in bandwidth saving up to 40% for many cases over the traditional schemes.
4	[98]	$GF(2^8)$	complete	multi- generations	The proposed scheme enjoys clear streaming advantages that should not be overlooked or underestimated.
5	[107]	$GF(2^{16})$	complete	multi- generations	Providing high quality VoD using P2P is feasible using a combination of techniques (a) network coding, (b) optimized resource allocation across different parts of the video, and (c) overlay topology management algorithms.

6	[104]	$GF(2^8)$	complete	multi- generations	The system enjoys great advantage compared to R^2 .
7	[108]	no info	complete	multi- generations	A robust resilience is maintained in networks with (1) limited bandwidth, (2) actual packet drop rate, and (3) rapid churning of peers.
8	[95]	$GF(2^{16})$	complete	multi- generations	With network coding, the effective buffer length of peers can be extended by reducing storage spaces for coded content, and the server bandwidth consumption can be reduced.
9	[96]	$GF(2^8)$	complete	multi- generations	Simulations for overlays with up to 3000 peers showed the ability to withstand heavy peers dynamics with low start-up delay and high streaming rate.
10	[106]	$GF(2^3)$	complete	multi- generations	The evaluation confirms the significant potential performance gain, in terms of visual quality perceived by peers, average streaming rates, streaming capacity, and adaptation to high peer dynamics.
11	[111]	$GF(2^8)$	complete	multi- generations	Under certain scenarios, the proposed schemes can result in bandwidth saving up to 60% over the traditional schemes.

12	[91]	$GF(2^8)$	sparse	multi- generations	DirectedPush has achieved a much lower packet delay and a smaller computational overhead. Meanwhile, the system also demonstrates improved robustness against both peer dynamics and flash crowd.
13	[112]	$GF(2)$	complete	multi- generations	The proposed protocol enables to support hundreds of users with a small initial buffering and network coding-based P2P video streaming is suitable for low-delay video communications.
14	[103]	$GF(2^{16})$	complete	multi- generations	The effectiveness and efficiency of SocialStreaming have been demonstrated.
15	[115]	$GF(2^8)$	complete	multi- generations	The proposed scheme provides better video quality on peers in terms of the four important performance metrics: video distortion, dependency distortion, end-to-end delay and initial startup delay.
16	[92]	no info	complete	multi- generations	With network coding, network health can be increased without increasing the amount of data stored on each peer or the number of peers.

17	[113]	$GF(2)$	deterministic multi-	Results report benefits with respect to
			generations	reduction of distribution times of con-
				tents in a collaborative system.

Due to the complexities introduced by network coding, there was a debate on whether it has practical benefits or not. Some studies [94], [120] have shown that using network coding in P2P networks does not provide additional benefits in the best cases, compared with no coding scenarios. Even though the majority of the reviewed studies (39 of 41) have shown optimistic results regarding using network coding in P2P networks for content distribution, we cannot draw a general conclusion about the practical advantages gained by using network coding.

That can be attributed to one or more of the following in the reviewed studies: a) The potential biased introduced by the used metrics, b) the lack of comparison with prior studies that have used the same evaluation settings, c) the selection of specific parameters (e.g., packet size, generation size,...etc.) that may give desirable results but cannot be generalized, d) the uncertainty on the quality of the implementation of the network coding algorithms that are not publicly available, e) the evaluations might be carried on specific transmission schemes or scenarios that are not generalizable, and lastly f) the use of simulations that may give inaccurate results.

Therefore, the best we can conclude is that network coding has demonstrated practical advantages for P2P content distribution in some scenarios, and more rigorous research is needed to leverage the potential advantages of network coding for P2P content distribution.

CHAPTER 4

PROTOCOL DESIGN

This chapter consists of three sections. The first section states the problem statement, laying the groundwork for the subsequent discussions. The second section outlines the design of CodedBitSwap protocol and how it extends and changes BitSwap protocol to integrate it with RLNC. Lastly, the third section discusses RLNC potential advantage for BitSwap.

4.1 Problem Statement

BitSwap's initial implementation employed a simplistic message exchange mechanism involving only a WANT request and a BLOCK response. When a peer wants a block, it broadcasts a WANT request to its connected peers. Upon receiving a WANT request, any peer that has the requested block would respond with BLOCK response containing the corresponding data block. This straightforward approach, however, exhibited a significant drawback: if more than one peer have the block, the requester would be overwhelmed with duplicate blocks. This redundancy results in inefficient bandwidth utilization, increases network congestion, and requires more processing by the requester due to the need to compute the hash of each received block.

To address this issue, BitSwap developers introduced an enhanced message exchange mechanism that replaces the WANT request with two requests: WANT-HAVE and WANT-BLOCK, and adds two additional responses: HAVE and DONT-HAVE. With this improved scheme, the requester first broadcasts a WANT-HAVE request, prompting connected peers to respond with HAVE or DONT-HAVE responses indicating whether they have the requested block [10]. The requester then sends targeted WANT-BLOCK request to a peer who has the block, eliminating the redundant transmission of duplicate blocks. While this enhancement introduces an additional RTT, the trade-off of reduced redundancy and improved network efficiency is deemed worthwhile.

The efficiency of each of the two versions of BitSwap protocol depends on the block ownership likelihood for the connected peers. If the probability of the block existence on the connected peers is low, the first (old) version with its minimal RTT may be sufficient. However, in scenarios where multiple peers are more likely to have the block, the enhanced version with its targeted request delivery could be more efficient. To address this trade-off, BitSwap developers introduced the concept of *optimistic* WANT-BLOCK request. In this approach, in its first block request, the requester randomly selects a maximum of two peers to receive WANT-BLOCK requests, while the remaining peers receive WANT-HAVE requests. This compromise can reduce redundancy with possibly fewer RTTs, but it does not solve the duplication problem and can still exhibit significant redundancy in scenarios where a peer requests many blocks that are available on most of the connected peers. Note that the distance between two peers in the network can be very large (arguably in two planets, hence the name Inter-Planetary), making the cost of a single RTT significant.

The fundamental question at hand is whether there exists a more efficient mechanism for requesting blocks from multiple peers and receiving them without unnecessary duplication

and within minimal RTTs. One promising approach involves leveraging RLNC, a network coding technique that makes data blocks equally valuable, enabling multiple peers to respond with encoded blocks that can be decoded by the requester without duplication [41]. The main idea behind CodedBitSwap is to integrate RLNC into BitSwap to reduce redundancy, as will be described in the following section.

4.2 CodedBitSwap Design

The BitSwap protocol will be extended to include RLNC-enabling messages and structures, and will control clients and servers differently. The modified version of BitSwap is called CodedBitSwap and is described in this section.

4.2.1 Aligning IPFS Blocks with RLNC Packets

IPFS utilizes a distributed file storage mechanism that divides files into fixed-sized blocks of 256KB. These blocks form the fundamental infrastructure of data storage and retrieval. When integrating RLNC with IPFS, it is natural to align the decodable unit (a generation) as a set of the IPFS blocks. This alignment simplifies the encoding and decoding operations and ensures compatibility between the two technologies.

In RLNC, a packet refers to a portion of file, which is mapped to an IPFS block in CodedBitSwap. Therefore, hereafter, the packet in RLNC terminology will be referred to as block. And according to the recommendations presented in section 5.1.7, each four IPFS blocks will compose a generation in CodedBitSwap.

4.2.2 CodedBitSwap Message

To manage and utilize RLNC-based data exchange within the CodedBitSwap protocol, BitSwap message will be extended to include two new structures, GenerationRequest and GenerationResponse. These structures facilitate communication between peers regarding the generation-level organization of coded blocks.

The GenerationRequest structure serves as a request for a specific generation of coded blocks belonging to a particular file. It comprises three essential attributes:

- **ID:** This attribute serves as a unique identifier for the generation within the context of the file. It distinguishes this generation from other generations belonging to the same file, ensuring that coded blocks are mapped to their correct generation.
- **CIDs:** This attribute represents a list of blocks' CIDs that belongs to the generation. These blocks should be encoded together to construct coded blocks. The length of this list serves as the generation size.
- **NumberOfCodedBlocks:** This attribute specifies the number of coded blocks required. It should not be less than one or larger than the generation size.

The GenerationResponse structure serves as a response to a GenerationRequest. It encapsulates three key pieces of information:

- **ID:** This attribute is the unique identifier of the generation, which is specified in the corresponding GenerationRequest.
- **CIDs:** This attribute represents a list of blocks' CIDs that are coded together to construct the coded blocks. This list could be identical to that of the generation request or a subset of it.

- **CodedBlocks:** This attribute represents a list of coded blocks belonging to the requested generation. These coded blocks are generated from the original data blocks identified by their CIDs presents in the GenerationRequest.

These structures can be represented as following in go language:

```
type BitSwapMessage struct {  
  
    ....  
  
    GenerationRequests    []GenerationRequest  
    GenerationResponses   []GenerationResponse  
}
```

```
type GenerationRequest struct {  
  
    // The CIDs of the generation  
    CIDs          []Cid  
  
    // The generation ID  
    ID            int  
  
    // Number of coded blocks needed  
    NoCodedBlocks int  
}
```

```
type GenerationResponse struct {  
  
    // The generation ID  
    ID            int  
  
    // List of coded blocks  
    CodedBlocks   []CodedBlock
```

}

4.2.3 Client Request

The CodedBitSwap protocol will change how the client requests blocks. Let's say the client want to get a file F that consists of n blocks ($CID_1, CID_2, \dots, CID_n$), the number of connected peers is k , and the generation size is s . The client will request the file F according to the following:

- **Generation Requests Construction:** The client disjointedly groups the wanted n blocks into generation requests, ensuring that each generation consists of at maximum s blocks. A unique index, or generation ID, is assigned to each generation of blocks, facilitating its identification. If a generation contains only a single block because there are no remaining blocks to group it with them, then the block will be requested as a plain (uncoded) block, following the standard BitSwap protocol.
- **Generation Requests Distribution Among Peers:** To optimize load distribution, the client systematically distributes generation requests among connected peers, where the generation coded blocks are requested from a maximum of s peers. The client randomly selects those peers from the set of k connected peers. The load on the selected peers is equally distributed, thus each of them is requested at least one coded block. For example, if $k = 3$ and $s = 4$, then two peers will be asked to send one coded block, while one will be asked to send two coded blocks.
- **Channel Creation for Each Generation:** To handle incoming generation responses, the client establishes a dedicated channel for each generation ID. This channel actively listens for and processes responses related to the corresponding generation,

ensuring timely and accurate block assembly.

- **WANT-HAVE Requests to Other Peers:** For peers that are not selected to participate in sending coded blocks, the client will follow the standard BitSwap protocol, sending WANT-HAVE requests to indicate its interest in acquiring the corresponding blocks. This approach ensures seamless integration with the BitSwap protocol, allowing for normal block retrieval process if selected peers failed to send coded blocks.

The algorithm pseudo-code of how the client constructs and distributes generation requests among peers is presented in Algorithm 1.

4.2.4 Server Response

Upon receiving a generation request, the server will handle the request as following:

- **All Blocks Available:** If all generation blocks are available, the server constructs the requested number of coded blocks within a generation response and sends it.
- **Partial Block Availability:** If a subset of the generation blocks is available, the server encodes them together constructing the requested number of coded blocks and sends the resulting coded blocks within a generation response. In case that only one of the generation blocks is available, the server sends it as a plain block within a generation response. Using what is called sparse coding [144], the RLNC decoder has the ability to successfully decode data even if it receives a plain block or a partially coded block, given that different received coded blocks have a diverse mix of the generation original blocks.
- **No Block Availability:** If none of the requested generation blocks are available, the server acknowledges the client by sending a DONT-HAVE response for each generation

Algorithm 1 CodedBitSwap client request construction algorithm

```
1: function CONSTRUCT_CLIENT_REQUESTS(file_CID, block_CIDs, num_blocks,
   num_peers, generation_size)
2:   generation_requests  $\leftarrow$  []
3:   remaining_blocks  $\leftarrow$  block_CIDs.copy()
4:   while remaining_blocks do
5:     generation_blocks  $\leftarrow$  []
6:     for  $i$  in range(min(generation_size, len(remaining_blocks))) do
7:       block_CID  $\leftarrow$  remaining_blocks.pop(0)
8:       Append block_CID to generation_blocks
9:     end for
10:    if len(generation_blocks) == 1 then
11:      generation_request  $\leftarrow$  "type": "PLAIN_BLOCK", "block_CID": generation_blocks[0]
12:    else
13:      generation_ID  $\leftarrow$  generate_unique_ID()
14:      generation_request  $\leftarrow$  "type": "CODED_BLOCKS", "generation_ID": generation_ID, "block_CIDs": generation_blocks
15:    end if
16:    Append generation_request to generation_requests
17:  end while
18:  for generation_request in generation_requests do
19:    if generation_request["type"] == "CODED_BLOCKS" then
20:      selected_peers  $\leftarrow$  random.sample(range(num_peers), min(generation_size, num_peers))
21:      blocks_per_peer  $\leftarrow$  math.ceil(len(generation_request["block_CIDs"]) / len(selected_peers))
22:      peer_requests  $\leftarrow$  []
23:      for peer_index in selected_peers do
24:        peer_blocks  $\leftarrow$  generation_request["block_CIDs"][peer_index * blocks_per_peer : (peer_index + 1) * blocks_per_peer]
25:        Append "peer_ID": peer_index, "block_CIDs": peer_blocks to peer_requests
26:      end for
27:      generation_request["peer_requests"]  $\leftarrow$  peer_requests
28:    end if
29:  end for
30:  want_have_requests  $\leftarrow$  []
31:  for block_CID in remaining_blocks do
32:    Append "type": "WANT_HAVE", "block_CID": block_CID to want_have_requests
33:  end for
34:  for generation_request in generation_requests do
35:    create_channel(generation_request["generation_ID"])
36:  end for
37:  return generation_requests, want_have_requests
end function
```

block CID, indicating that it is currently unavailable.

The algorithm pseudo-code of how the server responses to generation requests is presented in Algorithm 2.

Algorithm 2 CodedBitSwap server response algorithm

```
1: function HANDLE_GENERATION_REQUEST(request)
2:   generation_blocks  $\leftarrow$  request["block_CIDs"]
3:   available_blocks  $\leftarrow$  []
4:   for block_CID in generation_blocks do
5:     if ( thenblock_exists(block_CID))
6:       Append block_CID to available_blocks
7:     end if
8:   end for
9:   if len(available_blocks) == len(generation_blocks) then
10:    coded_blocks  $\leftarrow$  construct_coded_blocks(available_blocks)
11:    send_generation_response(request["generation_ID"], coded_blocks)
12:   else if len(available_blocks) > 0 then
13:    coded_blocks  $\leftarrow$  construct_coded_blocks(available_blocks)
14:    send_generation_response(request["generation_ID"], coded_blocks)
15:   else
16:     for block_CID in generation_blocks do
17:       send_DONT_HAVE_response(block_CID)
18:     end for
19:   end if
20: end function
```

4.2.5 Client Handling of Generation Responses

Upon receiving a generation response, the client handles it according to the following:

- **Generation Processing:** The client extracts the generation ID from the response and pass it to the channel that is actively listening for it. The peer that sent the coded blocks is marked as owner for those blocks, as if it had sent (HAVE) response.
- **Successful Decoding:** If the decoder successfully decodes the generation after passing the received coded blocks, their CIDs will be verified and a cancellation message

is sent to all peers that were previously got generation requests or WANT-BLOCK requests for any of the decoded blocks. Additionally, the client stops listening for further generation responses for the decoded generation ID.

- **Incomplete Decoding:** If the decoder remains unable to decode after receiving some coded blocks, the client checks its queue of pending generation requests. If there are pending requests, the client awaits for their responses. Conversely, if there are no outstanding generation requests and the client has got some coded blocks from some peers, it sends targeted generation requests for additional coded blocks sufficient for decoding the entire generation from those peers who own the blocks.
- **Blocks Unavailable:** In the event that no peer shows ownership of the requested blocks or if the client receives DONT-HAVE responses from all peers to whom it sent generation requests, control rolls back to the standard BitSwap protocol. Note that the CodedBitSwap protocol uses WANT-HAVE requests as part of its first requests to the connected peer who do not get generation requests, thus BitSwap will be able to use the knowledge of the blocks availability and move forward from this point.

The algorithm pseudo-code of how the client handles generation responses is presented in Algorithm 3.

Algorithm 3 CodedBitSwap client response handling algorithm

```
1: function HANDLE_GENERATION_RESPONSE(response)
2:   generation_ID  $\leftarrow$  response["generation_ID"]
3:   coded_blocks  $\leftarrow$  response["coded_blocks"]
4:   channel  $\leftarrow$  get_channel(generation_ID)
5:   channel.process_generation(coded_blocks)
6:   if decoder.decode(coded_blocks) then
7:     verified_block_CIDs  $\leftarrow$  verify_block_CIDs(coded_blocks) // Implemen-
      tation specific
8:     for (doblock_CID in verified_block_CIDs)
9:       cancel_requests(block_CID)
10:    end for
11:    close_channel(generation_ID)
12:  else
13:    pending_requests  $\leftarrow$  get_pending_generation_requests()
14:    if pending_requests  $\neq$  [] then
15:      await_responses(pending_requests)
16:    else
17:      block_owners  $\leftarrow$  get_block_owners(coded_blocks)
18:      if block_owners  $\neq$  [] then
19:        request_additional_blocks(block_owners, generation_ID)
20:      else
21:        fallback_to_BitSwap()
22:      end if
23:    end if
24:  end if
25: end function
```

4.2.6 An Example

Figure 4.1 shows an example for blocks transmission using CodedBitSwap. The scenario is that Peer A wants to download a file that consists of 3 blocks from three connected peers B, C, D. Each of these peers has the file at its storage.

Peer A constructs generation requests by grouping the 3 file blocks into a generation and creating a generation request message. Therefore, the generation size (k) is set to 3, and each connected is expected to send one coded block.

After Peer A selects the connected peers (B, C, and D) to participate in sending coded blocks, it distributes the requests among them so that each peer send one coded blocks that encodes the three original file blocks. It then sends a generation request message to each of them. The message specifies a generation ID, the requested block CIDs (all 3 blocks in this

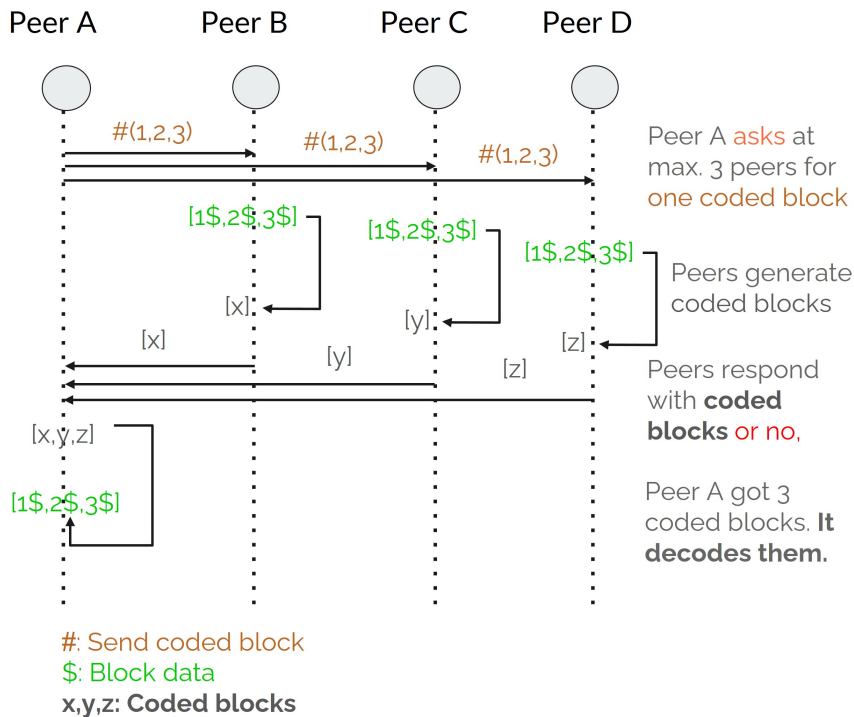


Figure 4.1: Sequence diagram of an example of blocks transmission using CodedBitSwap

case), and the number of coded blocks needed.

When peers B, C, and D receive the generation request message from A, each of them checks if it has all the requested block CIDs. If a peer has all the requested blocks, it constructs a coded block using those blocks and sends it back to A in a generation response message. In this example, all peers send one coded block to Peer A. If a peer doesn't have any of the requested blocks, it sends a DONT-HAVE response message to A, indicating it cannot contribute to this generation.

When Peer A receives the generation response messages from B, C, and D, it feeds the received coded blocks to its decoder. If the decoder is successful, it recovers the original 3 data blocks of the file.

4.3 RLNC Potential Advantages for BitSwap

In this section, we briefly explain the concepts of RLNC and discuss its potential advantages for P2P data exchange protocol such as BitSwap.

The primary motivation behind incorporating RLNC into the BitSwap protocol lies in the pursuit of an optimal data exchange protocol capable of transmitting block requests to a number of peers, and receiving data without duplicates, regardless of the number of participating peers. The main potential advantage of RLNC is to minimize the occurrence of duplicate blocks by exchanging coded blocks that are linear combinations of the original blocks.

It's worth noting that IPFS's versatility extends beyond public file sharing, including applications in private networks, shared storage solutions, and database replication systems. In these environments, the advantage of redundancy reduction becomes particularly important due to the higher likelihood of encountering a large number of owners for the same data.

By increasing the number of participating peers up to the generation size (because coded blocks within a generation have equal value to the requester), the probability of retrieving all blocks within minimal RTTs significantly increases. Considering that the geographical distances among peers can be very large, the cost of even a single RTT can be substantial.

In addition to solving the the duplication problem, RLNC has the potential to unlock other notable advantages, including:

- **Reduced Network Overhead:** Network coding can reduce network overhead by eliminating the need for redundant transmissions of the same data packets. This can free up bandwidth and improve overall network performance.
- **Improved Network Throughput:** Network coding enables intermediate nodes in the network to combine and transmit encoded data packets, leading to a more efficient use of network bandwidth. This can significantly improve the overall throughput of the P2P network and reduce the bandwidth requirements of the protocol [159].
- **Enabling Multiple Streams of Transmission:** Network coding can be used to combine multiple streams of transmission for the same content into a single stream. This can improve the overall bandwidth utilization and reduce the latency of content delivery [33], [10].
- **Improved Download Performance:** RLNC can improve download performance for requester peers because it increases the number of peers who can participate in responding to the coded blocks requests without knowing which of them have the wanted blocks. This approach can lead to fewer RTTs and shorter download times, particularly when peers are geographically dispersed, making one RTT substantial.
- **Enhanced Network Robustness:** Network coding makes the P2P network more

resilient to node failures and network congestion. By generating multiple encoded packets from a single source file, the system can recover lost data from different sources even if some nodes drop packets or the network becomes congested, without caring which part of the file is lost.

What makes RLNC applicable for BitSwap protocol is its scalability to large networks and adaptability to heterogeneous network conditions. Network coding can scale well to large P2P networks, as it does not require centralized control or coordination. This makes it suitable for large-scale content distribution systems [41]. Additionally, network coding can adapt to heterogeneous network conditions, where nodes may have different processing capabilities and network bandwidth. This makes it suitable for a wide range of P2P applications [160].

CHAPTER 5

PROTOCOL EVALUATION

This chapter consists of three sections. The first section details the execution of controlled experiments aimed at identifying the most effective coding parameters for CodedBitSwap that will ensure its optimal performance. The second section describes CodedBitSwap protocol evaluation methodology. Lastly, the third section presents and discusses the results of the protocol experimental evaluation.

5.1 Trial Experiments: Optimal Generation and Packet Sizes

In this section, an experimental study is presented that aims to gain insights into which combination of generation size and packet size values yield optimal performance in terms of coding time for the RLNC-based block exchange protocol, CodedBitSwap.

5.1.1 Objectives

The impact of generation size and packet size on coding performance is substantial. To determine the optimal sizes and structures for the CodedBitSwap protocol, experimental evaluations will be performed on a set of files of varying sizes. These evaluations will lead to a conclusion on the appropriate coding parameters and structures for the proposed protocol. The criteria for determining optimal parameters are the encoding time required to make a minimum number of packets sequentially and the decoding time required to restore the original data.

5.1.2 Variables and Parameters

- **Galois Field.** The Galois field of size 256 (2^8) is a preferred choice for RLNC operations due to its reduced chances of generating linearly dependent pieces, which can lead to redundant coded packets [134]. GF(256) strikes a good balance between performance and memory consumption, making it an optimal choice for RLNC operations. Additionally, GF(256) is compatible with practical applications of RLNC and offers a good trade-off between worst-case delays and generation size. Limiting the generation size to less than 256 packets helps manage worst-case delays effectively. Furthermore, the selection of GF(256) simplifies implementation by leveraging existing libraries and tools. Various libraries, such as kodr ¹, kodo [155], and rnc-lib ², provide support for RLNC over GF(256).

- **File Size.** The selection of file sizes as parameters is an important step. The file size should be large enough to ensure that the experiment is meaningful and representative

¹<https://github.com/itzmeanjan/kodr> (accessed: 2024-04-27)

²<https://github.com/avisegradi/rnc-lib> (accessed: 2024-04-27)

of real-world scenarios. At the same time, it should not be too large, as this can lead to excessive overhead and increased computational complexity. Ten file sizes will be selected as parameters for this experiment, with each size being a multiple of 2 starting from 32KB up to 16MB. This range of file sizes is sufficient to cover a wide range of applications and use cases while keeping the computational complexity manageable. The selection of these file sizes will enable to evaluate the performance of RLNC under different conditions and provide insights into its effectiveness in various scenarios.

- **Generation Size.** Four generation sizes, namely 4, 8, 16, and 32, are chosen as parameters for the experiment. The choice of these generation sizes is based on the fact that they are powers of two. This is important because RLNC is based on Galois Field arithmetic over a finite field $GF(2^8)$, where 8 is the number of bits used to represent each symbol. The use of powers of two as generation sizes ensures that the number of symbols in each generation is a power of two, which simplifies the arithmetic operations involved in the encoding and decoding process. Additionally, using powers of two as generation sizes allows for easy scalability of the experiment to larger network topologies.

5.1.3 Experimental Setup

- **Software:** The experiment will utilize a code script written in Go programming language with the Kodr library. The Kodr library provides functionalities for RLNC operations.
- **Hardware:** A Windows machine with moderate processing power (Intel CPU i3-3120M CPU @2.50GHz) and 16GB RAM memory will be used to run the experiment.

The relatively low processing power of this machine is demanded because it will determine how quickly the code script can perform the operations on commodity hardware.

- **Files:** The experiment materials consist of ten files of random bytes. These files will be used as input data for the encoding algorithm, which will perform RLNC operations on them.

5.1.4 Operations & Metrics

Two operations will be performed in the experiment. The first operation is the encoding operation performed on a single data file to construct encoded packets. The second operation is the decoding operation on a set of encoded packets that belongs to one data file.

The independent variables for the coding operations are the file size and the generation size in term of number of packets.

For these operations, the following dependent variables will be measured:

- **Encoding Time:** The time of the encoding operation that generates the specified number of coded packets.
- **Decoding Time:** The time of the decoding operation for a number of generated coded packets.
- **Encoding Speed:** The ratio of the file size in bytes over the encoding time in microseconds.
- **Decoding Speed:** The ratio of the file size in bytes over the decoding time in microseconds.

5.1.5 Experimental Procedure

In this experiment, the procedure will go in following steps:

1. **File Creation:** The first step involves creating a file with a specific size. This can be done using a script that generates random bytes and fills the file content with them.
2. **Packet Split:** Once the file is created, it is divided into a number of packets equal to the generation size. Each packet represents a smaller portion of the original file and will be used as input to the encoder to generate coded packets.
3. **Encoding Operations:** Encoding operation is performed to generate one coded packet using the packets from the previous step. The minimal number of packets required for successful decoding, which is equal to the generation size, are generated sequentially one after another.
4. **Encoding Time Recording:** During the encoding process, time is measured. The time of each generated coded packet is recorded, and then the times are summed as the total encoding time of the generation.
5. **Decoding Process:** After the encoding operations, the generated coded packets are passed through the decoder that aims to restore the original file from these encoded packets. The decoding process should accurately reconstruct the original file without any loss or corruption that might happen because of linearly dependent packets. Therefore, once decoding is completed, it is verified that the decoded file matches exactly with the original file.
6. **Decoding Time Recording:** Similar to encoding time monitoring, the decoding time is measured during the decoding operation.

7. Repetition for Different Generation Sizes and File Sizes: To gather comprehensive data, this entire process is repeated multiple times with different generation sizes and file sizes.

5.1.6 Results & Discussion

Encoding Operations

The experimental results of the encoding times are shown in Figure 5.1, which shows the encoding times in milliseconds for different file sizes and four generation sizes. The findings confirm our expectations, showing that as the file size increases, the encoding time also increases. This happens because the time required to generate one coded packet of size 1MB, for example, would be more than the time required to generate one of size 256KB. Since the file sizes are multiples of two, the encoding time is almost doubled as we go right in the x-axes. Interestingly, when doubling the generation size for the same file size, the encoding time doubles as well, despite the fact that when keeping the file size constant and increasing the generation size, the packet size in terms of bytes is halved. Therefore, the reduction in packet size does not result in a decrease in encoding time for a given file size. This suggests that smaller generation sizes are more favorable.

Figure 5.2 illustrates the encoding speeds (in Byte per microsecond) for various file sizes and four generation sizes. It is evident that for small file sizes, the encoding speeds are relatively low. However, after surpassing a 1MB file size, the speeds become nearly stable. Additionally, decreasing the generation size for a given file size improves the encoding speed by a factor of 2 or more. This is because the encoder generates the coded packets one after another, thus taking double the time when the generation size (number of packets) is doubled. It is obvious that the decrease in packet size in term of bytes does not increase

the speed on generating coded packets. Furthermore, within the same generation size category, as the file size increases, the encoding speed initially increases; however, it eventually decreases and stabilizes at an almost similar level.

Decoding Operations

The experimental results of the decoding times are presented in Figure 5.3, which shows the decoding times in millisecond for ten file sizes and four generation sizes. The results confirm the expectations regarding performance, as the decoding time increases with larger file sizes. It is obvious that, for files of the same size, doubling the generation size leads to a doubling of the decoding time. This is possibly because the coded packets are passed to the decoder more frequently one after another. Additionally, for files with the same generation size, the decoding time increases almost exponentially or at least non-linearly. This suggests that decoding smaller chunks of a file is faster than decoding the entire file at once. To

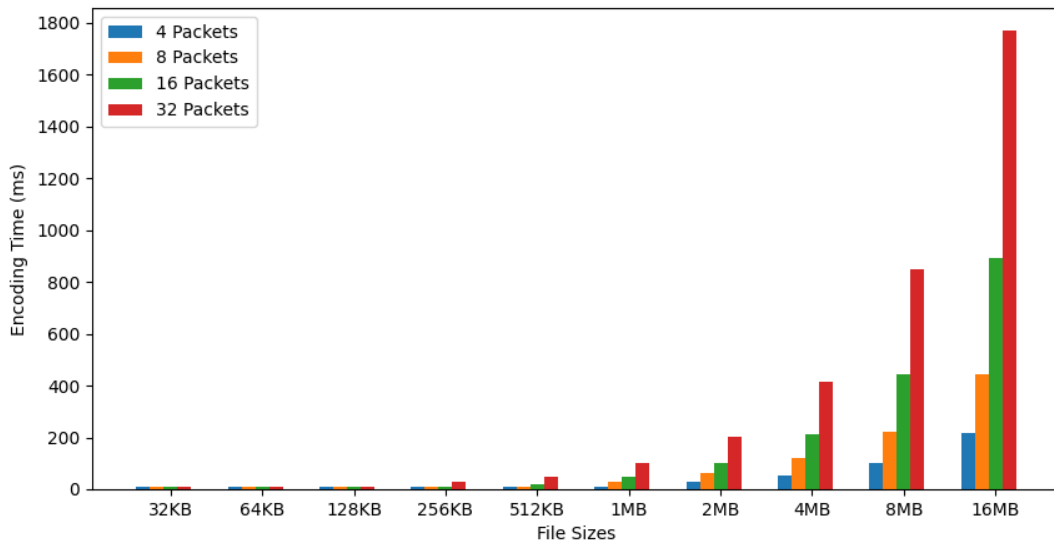


Figure 5.1: Encoding times for ten file sizes each experimented with four generation sizes

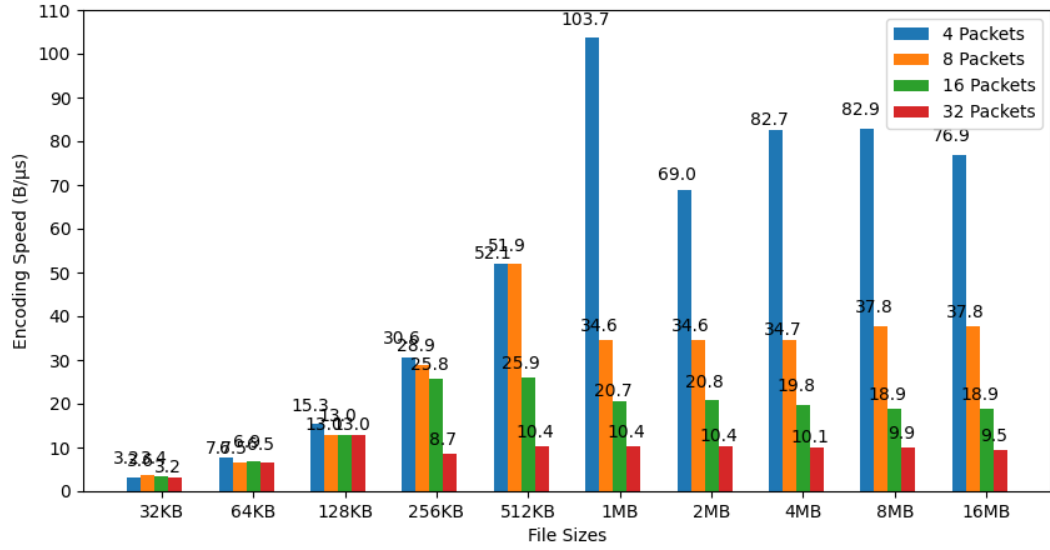


Figure 5.2: Encoding speeds for ten file sizes each experimented with four generation sizes

illustrate this, a 1MB file with a generation size of 4 packets would take approximately 10ms to decode. On the other hand, decoding a 16MB file with the same generation size would take around 218ms. It would be faster to decode 16 files of size 1MB sequentially (which

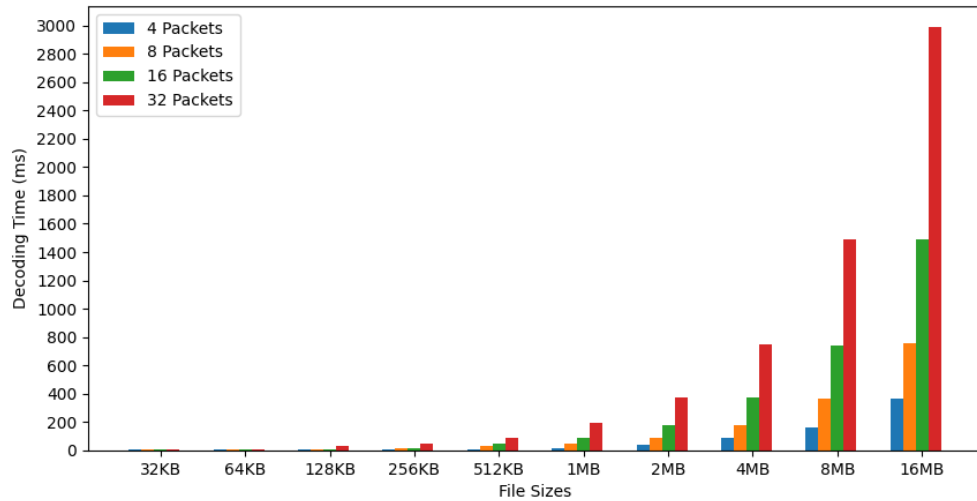


Figure 5.3: Decoding times for ten file sizes each experimented with four generation sizes

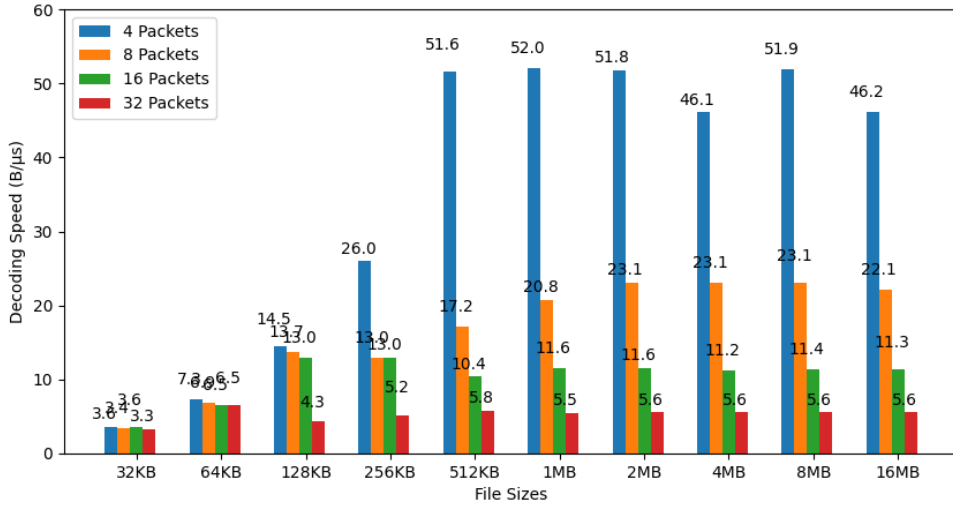


Figure 5.4: Decoding speeds for ten file sizes each experimented with four generation sizes

would take approximately 160ms). This highlights the advantage of working with smaller portions of a file in terms of decoding time and speed, or with what is known in network coding terminology as *generations* [124].

Figure 5.4 presents the decoding speeds (in Byte per microsecond) for ten file sizes and four generation size. Similar to what was observed for encoding speeds, it can be seen that decoding speeds are initially low for small file sizes but become more stable after surpassing a 1MB file size. Furthermore, for files of the same size, decreasing the generation size improves the decoding speed by a factor of at least 2.

It is also noticed that, for files with the same generation size, the decoding speed initially increases as the file size increases but then starts to decrease before stabilizing at an almost similar level. Overall, it is not surprising that both decoding times and speeds are worse than their respective encoding counterparts, because the decoding complexity is upper bounded by $O(n^3)$ [127].

In summery, these results demonstrate that larger file sizes with larger generation sizes lead to longer decoding times and slower speeds. Conversely, smaller file sizes with smaller generation sizes result in faster and more efficient decoding processes.

5.1.7 Recommendations

Based on the experiment results, the CodedBitSwap protocol should be designed with a generation size of 4 packets, each of size 256KB (the default IPFS block size). This would result in a decodable data unit (a generation) of size 1MB. This selection has been prioritized for the following reasons:

- **Minimizing encoding time:** The waiting time before decoding can start is a crucial factor that can influence the performance of the CodedBitSwap protocol. The decoding process can be initiated as soon as the first coded packet arrives. With a generation size of 4 packets, the encoder will be able to construct coded packets faster, leading to minimal waiting by the decoder. Additionally, larger generation sizes may necessitate the arrival of additional packets, potentially increasing the waiting time and extending the overall data transfer process.
- **Minimizing decoding time:** As demonstrated by the experimental results, decoding time exhibits a clear inverse relationship with generation size. This means that smaller generation size lead to reduced decoding times, which would yield better performance.
- **Balance between complexity and RTT:** The generation size sets an upper limit for the number of data providers who can participate in generating coded packets for a requester without receiving unnecessary extra coded packets. For example, for a

file of 4 packets, the minimal required coded packets to decode is 4. To avoid getting more than needed coded packets, the requester should not send requests for coded packets to more than 4 data providers. At the same time, if the requester does not have information about the data location, it would prefer to ask all providers for coded packets to increase the probability of retrieving data within a single RTT. For this reason, larger generation size would be better. However, as seen from the experimental results, increasing the generation size will increase the coding overhead. To provide a balance between coding complexity and RTTs, it might be appropriate to select generation size between 4 and 8,

- **Minimal changes to IPFS block structure:** To preserve the simplicity of the CodedBitSwap protocol, it is recommended to keep the packet size same as the IPFS default block size (256KB) to minimize alterations to the original IPFS block structure. Additionally, merging IPFS blocks should be avoided to maintain a straightforward design that is easy to implement and manage.
- **Higher encoding speed:** The decodable unit size (1MB) has been selected because the experimental results have shown that the 1MB file has slightly higher encoding speed compared to other sizes.

Overall, these recommendations aim to optimize the performance of the CodedBitSwap protocol by minimizing waiting times and reducing network traffic while keeping the design simple and easy to implement. In addition, this approach should ensure efficiency and practicality for real-world applications.

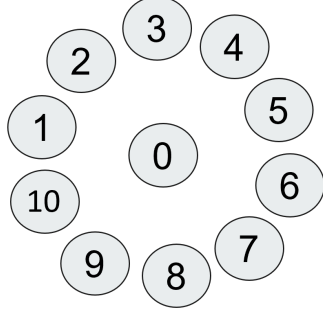


Figure 5.5: Experimental network topology of 11 nodes labeled from 0-10

5.2 CodedBitSwap Evaluation Methodology

In this section we present the methodology that is undertaken to evaluate the implementation of the CodedBitSwap protocol, compared to the baseline protocol BitSwap.

5.2.1 Experiment Testbed

The experiments will be conducted on a commodity computers since IPFS normal users would be using similar computers. The baseline measurements will be for normal BitSwap protocol that is running on nodes that are exchanging data blocks. The simulations of IPFS will be carried on using the InterPlanetary TestBed (IPTB) ³. IPTB is an simulation and testing application used to create and manage a cluster of multiple sandboxed nodes locally on a computer. The used machine is equipped with a 4-cores Intel CPU (i3-3120M @2.50GHz) processor and 16GB RAM.

The experimental network topology will consist of 11 nodes labeled from 0 to 10 (shown in Figure 5.5). Three data files will be transmitted among them. Node 0 will store the data files, then other nodes will get the data from it one by one. This mimics the scenario in which data blocks are becoming more popular in the networks.

³<https://github.com/ipfs/iptb> (accessed: 2024-04-27)

Kodr library will be used to perform the coding operations for the CodedBitSwap protocol. Kodr is an RLNC library written in Golang. It uses finite field $GF(2^8)$ and can be used for full RLNC, on-the-fly RLNC, sparse RLNC, and generational RLNC. Kodr is an easy-to-use library that can be used by developers of all levels of experience. It is a powerful tool for improving the performance of data transmission over unreliable networks.

5.2.2 Experiment Materials

The experiment material are three data files of sizes 4MB, 8MB, and 16MB. These files represent realistic data such as images, videos, or documents. The largest file is of size 16MB because default BitSwap message maximum size is 16MB. Therefore, larger file sizes would yield a multiple of its results. The default block size of IPFS is 256KB thus the experiment files will be splitted into blocks (resulting in 16, 32, and 64 blocks for each file) upon adding them to IPFS local storage. The data blocks consists of random alphabetical characters created using BASH scripting language⁴. Those blocks will be transmitted among network members and the encoding and decoding operations will be performed on them.

5.2.3 Operations & Procedure

The experimental nodes will carry out two operations. The first operation is the ADD operation, which adds a file into IPFS local storage. After a file is added, it will be available to be accessed by other IPFS nodes in the network. This operation will be executed by Node 0.

The other operation is the GET operation. This operation downloads a file from the IPFS network, leveraging nodes that have the file. Once the file is downloaded to the

⁴<https://www.gnu.org/software/bash/> (accessed: 2024-04-27)

IPFS local storage of the requesting node, it becomes accessible to other IPFS nodes in the network, expanding the network's contribution pool as the file replicates across the network. In our experiment, Nodes 1 to 10 will execute GET operation.

Each file will undergo a dedicated round of addition and download operations, with each round having its own independent measurements. Since three files will be used in the experiments, there will be a total of three rounds for each protocol, BitSwap and CodedBitSwap. Each round will follow a specific sequence:

1. Node 0 will add the designated file to the IPFS storage.
2. Sequentially one after another, nodes 1 to 10 will join the network, get connected to all existing nodes, then download the file from the network.
3. Once all nodes have completed the download, the measurements for that round will be collected then reset.
4. The process will repeat with the next file, starting from step 1.

This approach ensures that each file's addition and download operations are isolated and measured independently, providing a clear picture of the performance for each file with each of the two protocol.

5.2.4 Data Exchange Scenarios

The exchange of data among peers will undergo in two scenarios.

1. **Scenario 1: Widespread Data Availability - Best Case**

This scenario represents the core focus of the research. It assumes all peers within the network have the requested data. In this scenario, CodedBitSwap leverages network

coding to enable a single request retrieval of the data from connected peers, minimizing redundant data blocks from multiple peers.

2. Scenario 2: Limited Data Distribution - Worst Case

This scenario acknowledges that not all data may be uniformly distributed across the network. Here, the requesting peer would still utilize CodedBitSwap, but it would need to target specific peers known to have the encoded data. The efficiency might be reduced in this scenario compared to Scenario 1. The number peers that own the data blocks will randomly ranges from 1 peer up to the generation size (the maximum number of requested peers, which is four as per the protocol).

5.2.5 Measurements

During each experimental round, the measurement will be taken for the following:

1. **Download time:** The time it takes to send file request, processes the request by the responder node, and receives a response from it. For the CodedBitSwap, this includes the encoding time by the responder, but does not include the decoding time by the requester. The decoding should immediately start after the message is received.
2. **Encoding time:** The time it takes to generate the requested number of coded blocks.
3. **Decoding time:** The cumulative time required to decode coded blocks of a generation.
4. **Blocks sent:** The number of blocks sent by the node during the round of transmitting a file among all nodes.
5. **Blocks received:** The total number of blocks received by the node during the round of transmitting a file among all nodes.

6. **Data sent:** The total about of data (in bytes) sent by the node during the round of transmitting a file among all nodes.

7. **Data received:** The total about of data (in bytes) received by the node during the round of transmitting a file among all nodes.

To measure those values, appropriate modifications will be made on the code of the protocol. In addition, the IPFS command `STATS BW` will be used, which shows the total data sent and received to and from other IPFS nodes.

Since these experiments will be simulated on a single machine, ideal connections are assumed and network latency is neglected. Therefore, there are no packet losses during transmission. These assumptions allow us to concentrate on the causes of the delays and to isolate the measurement environment.

5.3 Results & Discussion

This section presents and discusses the results of the evaluation of CodedBitSwap protocol.

5.3.1 Blocks Duplication

The primary challenge this study addresses is the duplication problem that arises from requesting a block from multiple peers. To evaluate CodedBitSwap’s effectiveness in mitigating duplication, the study compares the amount of data received by peers using both BitSwap and CodedBitSwap protocols. The results, presented in Table 5.1, demonstrate a significant reduction in duplicated data when employing CodedBitSwap. As evident from the table, CodedBitSwap consistently eliminates the redundancy, while BitSwap nodes have a high percentage of duplicated blocks.

The BitSwap protocol employs an optimistic approach for block retrieval. For each block, it sends WANT-BLOCK requests to at most two randomly selected peers without prior knowledge of their block ownership. This approach aims to minimize RTT and fetch the block from the first contact. In terms of data redundancy, in the best-case scenario, the WANT-BLOCK request will be delivered to one peer that holds the desired block, resulting in a direct block transfer and minimal RTT. However, the worst-case scenario involves sending WANT-BLOCK requests to multiple peers that own the requested block, leading to unnecessary duplication.

The key advantage of CodedBitSwap lies in its ability to significantly decrease data duplication. This stems from its innovative approach of leveraging RLNC and treating portions of a file, known as generations, as discrete data units composed of equally valuable coded blocks. In contrast to the BitSwap protocol that requests specific blocks, CodedBitSwap sends requests for the equally-valuable coded blocks that belong to a specific generation from multiple peers. Upon receiving them, these coded blocks are passed to the decoder that is capable of restoring the original blocks. In both the best-case and worst-case scenarios, duplication will be minimal.

Table 5.1: Amounts of blocks transmitted among peers in the network

Protocol	File Size	Blocks Requested	Blocks Received	Duplicate Blocks	Duplication Percentage
BitSwap	4MB	160	168	8	5%
BitSwap	8MB	320	451	131	41%
BitSwap	16MB	640	828	188	29%
CodedBitSwap	4MB	160	160	0	0%
CodedBitSwap	8MB	320	320	0	0%
CodedBitSwap	16MB	640	640	0	0%

Reduction or elimination of redundancy translates to a multitude of benefits, including reduced network congestion, efficient bandwidth utilization, and enhanced resource utilization for the requester. By minimizing the transmission of redundant data, CodedBitSwap alleviates the burden on the network infrastructure, allowing for smoother data transfers and improved overall network performance. Additionally, efficient bandwidth utilization is achieved by ensuring that only the necessary data is transmitted, preventing the wastage of valuable network resources. Furthermore, the requester node resource utilization is enhanced as it processes a smaller amount of blocks, reducing the computational load of calculating and verifying the CIDs hashes and improving overall efficiency.

The work done by Zeng et al. [161] proposes IBIHA, a system that improves how IPFS finds data. It uses what they call enhanced nodes and improved name resolution to reduce the time it takes to find data and the amount of unnecessary traffic. IBIHA combines IPFS with an Information-Centric Network (ICN) for faster data location lookup. It introduces enhanced nodes and a new way to find data names that reduces search time and traffic. Their experiments showed a decrease in the number of duplicate packets sent but did not eliminate it.

5.3.2 Download Time

While CodedBitSwap offers significant advantages in terms of duplication reduction, it is crucial to consider its impact on download time. Unlike BitSwap, which directly transfers original data blocks, CodedBitSwap involves coding operations, introducing additional layer that increases overhead. It is essential to evaluate whether the overhead incurred by CodedBitSwap justifies the benefits it offers.

Figure 5.6 presents the average download times (in milliseconds) for three files using

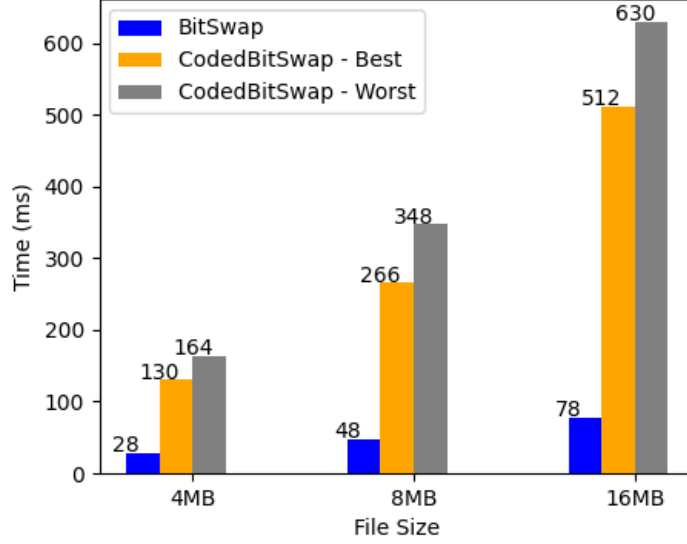


Figure 5.6: Average download times of three files for peers 1-10

BitSwap and CodedBitSwap in best and worst scenarios. The download time includes the encoding operations at the responder side but not the decoding time at the requester. The results indicate that the increase in download time with CodedBitSwap is relatively modest, averaging around 500ms for a 16MB file (or 32ms per 1MB) in best cases, and 630ms in the worst (39ms per 1MB). The worst case scenario is expected to have longer time because the requesting peer is sending another request for more coded blocks from the peers that have sent coded blocks before. The increase in time is mainly due to the responder that needs to construct additional coded blocks for the requester in another message.

Despite the download times of CodedBitSwap are much larger than BitSwap, the time difference is hardly noticeable by the end user. Therefore, we believe that this overhead might be considered acceptable given the substantial reduction in duplication and improved network traffic achieved by CodedBitSwap.

CodedBitSwap performance might be better than BitSwap in scenarios where Coded-

BitSwap can reduce the number of RTTs for data retrieval. The introduction of RLNC enhances the likelihood of encountering a peer owning the desired coded blocks from first contact, which would reduce the RTT by one. This probability is directly proportional to the generation size (which determines the number of participants getting generation requests) and data popularity. In the IPFS network, peers can be geographically dispersed. In such cases, reducing one RTT is highly advantageous due to its high network latency cost. However, our experiments does not consider network latency and does not evaluate such scenarios, opening a promising avenue for future research.

In BitSwap, requests that can potentially obtain the block from the first contact are very limited. This is because the lookup space for a given block cannot exceed two peers, while CodedBitSwap lookup space is at least a number of peers that is equal to the generation size. For example, if the generation size is four and there are ten connected peers that own the requested coded blocks, CodedBitSwap can send four coded block requests instead of at most two optimistic WANT-BLOCK requests per block as in the BitSwap protocol. This increased lookup space enhances the likelihood of encountering a peer owning the wanted blocks from first contact and within minimal RTTs.

Rocha et al. [10] have proposed a new feature in Bitswap, called the relay manager, that helps nodes find content faster by forwarding WANT messages to other nodes. The results showed a reduction in fetch time by 33% compared to the original BitSwap. The downside in their proposal is that nodes helping with the search might keep looking for blocks even if the original requester already has them, which increases network traffic. Finding the ideal balance between efficiency and traffic depends on the specific network and application.

An another research by Ascigil et al. [162] tests using IPFS for local VoD delivery within an ISP network. They found IPFS adds overhead and latency, hurting quality. The

solution is to use a different network layer (ICN) that the ISP controls. This allows for better routing and balancing of user traffic, especially for small to medium VoD content. The key is designing efficient routing that considers user bandwidth limitations. Their approach avoids playback interruptions for all buffer sizes through having full knowledge of the network load.

5.3.3 Coding Time

The computational complexity introduced by RLNC has established a debate concerning its practicality in real-world applications, as evidenced by works such as those by Wang et al. [94], [120]. Therefore, careful consideration should be given to the overhead associated with RLNC coding operations of CodedBitSwap.

Figure 5.7 show the average encoding and decoding times (in milliseconds) for three file sizes. The average encoding time for a file is the average of the time taken by each node to construct the requested number of coded blocks (node 10 is not included because it is the last node that gets the file and does not share it). Similarly, the average decoding time for a file is the average of the time taken by each node to decode the received coded blocks (node 0 is not included because it is the first node that starts sharing the file and does not download it).

Contrary to the expectation of the exponential complexity growth with the increase in the file size (as seen in Section 5.1), both encoding and decoding times exhibit linear growth despite the increase in the number of blocks of the files. This observation is evident from the consistency of the average encoding and decoding times presented in Figure 5.7 with those presented in Figures 5.1 and 5.3, respectively, for the 4-blocks generation size for the same file sizes. Recall that CodedBitSwap uses a generation size of four blocks, each of size 256KB. Consequently, an 8MB file, for instance, would consist of 32 blocks. However, the

times in Figure 5.7 deviate from those of 8MB file with 32-blocks generation size in Figures 5.1 and 5.3 to align with those of 8MB file with 4-blocks generation size. This means that an 8MB file would take around 50ms for encoding and 250ms for decoding instead of 800ms for encoding and 1400ms for decoding. This is thanks to the generation-based RLNC approach used in CodedBitSwap, which effectively scales down the complexity to a multiple of the generation instead of an exponential function of the number of file blocks. In essence, the complexity grows linearly with the number of file generations rather than exponentially with the number of files blocks, as would be the case for complete-coding approach that does not split the file into generations [41].

The average decoding time is around 29ms per 1MB. The average encoding time is around 18ms per 1MB. This means that the average increase in the complete download time (that includes the encoding at the server and the encoding at the client) is around 47ms per 1MB. For small downloads (e.g., a few MB), a 47ms delay per MB might be significant and

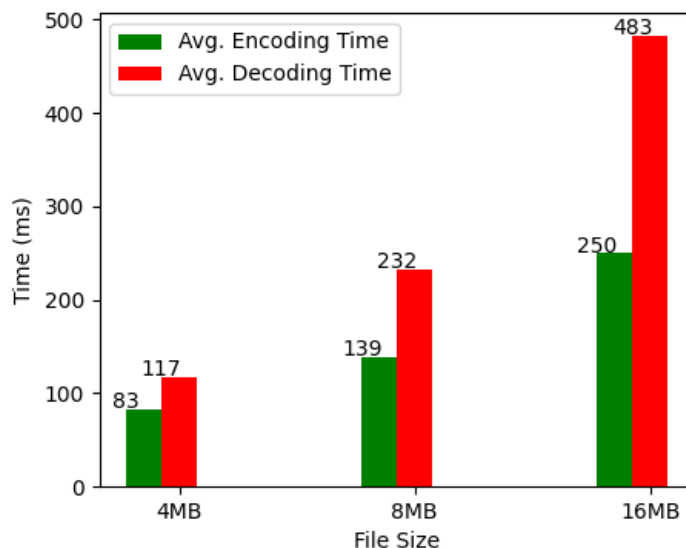


Figure 5.7: Average of encoding and decoding times by nodes 0-10

negatively impact user experience. However, for larger downloads (e.g., hundreds of MB or more), the delay becomes less noticeable as a percentage of the total download time. In high-bandwidth networks, the 47ms delay might be less acceptable because the time saved by redundancy elimination might be less significant compared to the overall download speed. In low-bandwidth networks, the delay might be more tolerable if the benefit of reduced data transfer due to redundancy elimination outweighs the additional 47ms per MB. For real-time applications (e.g., video conferencing), even a small delay can be disruptive. In such cases, a 47ms delay per MB might be unacceptable. For non-real-time applications (e.g., file downloads), the delay might be more acceptable, especially for larger downloads where the overall time savings from redundancy elimination become more prominent.

Overall, the results demonstrate remarkable efficiency in both encoding and decoding operations with average times that remain within acceptable ranges. This efficiency comes from the employed optimization techniques that minimize computational overhead. While the exact perception of efficiency may vary depending on the end user's specific requirements and resource constraints, CodedBitSwap delivers performance that is well within practical limits.

CHAPTER 6

CONCLUSION

This chapter concludes the thesis and consists of four sections. The first section discusses the lessons learned during the conduction of this research. The second section briefly discusses the feasibility of applying network coding to P2P networks. The third section summarizes the proposed solution of CodedBitSwap. Lastly, the fourth section outlines the future directions for this research.

6.1 Lessons Learned

In the course of my thesis research, I have gained profound insights into network coding, its underlying principles, and the challenges associated with its practical implementation. Network coding is a multifaceted field that demands a solid foundation in computer science. To comprehend its intricacies fully, one must delve into topics such as graph theory, linear algebra, and information theory. Understanding the nuances of network coding is essential for designing efficient protocols that harness its benefits effectively.

The practical implementation of network coding presents challenges. Finding suitable libraries and programming tools is challenging. Implementing the protocol presented a

challenge due to the programming complexities in peers management and network coding operations.

This research project significantly expanded my skillset in network application programming and exposed me to advanced practices in open-source development and to the inner workings of advanced open source projects. Learning from existing projects provides valuable insights into best practices, code quality, and collaborative development. Additionally, my research skills were considerably enhanced throughout my work on the thesis.

On the positive side, the detailed explanation and background provided on IPFS's internal components and the intricacies of network coding will serve as a valuable resource for future developers and researchers interested in network coding-based solutions. This will help them avoid the difficulties I encountered during the research process.

6.2 On Network Coding Feasibility

The debate surrounding the applicability of network coding into P2P content distribution raises a valid point because validating the findings of network coding-based P2P content distribution studies is challenging. While IPFS presents a strong case and an ideal candidate for network coding applicability, network coding benefits may not universally translate to other P2P content distribution networks. This is due to the inherent design strategies and complexities involved in each system. Furthermore, the successful application of network coding requires in-depth knowledge and meticulous protocol design. While its potential for efficiency optimization is undeniable, it shouldn't be viewed as a universally applicable solution for all P2P systems.

Several factors can influence the acceptability of using network coding into P2P file systems including the sizes of the files distributed among network members, application

type, network conditions, clients' hardware capabilities, and user tolerance.

For small downloads (e.g., a few MB), a small increase in delay (due to the introduced coding operations) might be significant and negatively impact user experience. However, for larger downloads (e.g., hundreds of MB or more), the delay becomes less noticeable as a percentage of the total download time. For real-time applications (e.g., video conferencing), even a small delay can be disruptive. In such cases, any additional delay might be unacceptable. For non-real-time applications (e.g., file downloads), the delay might be more acceptable, especially for larger downloads where the overall time savings from redundancy elimination become more prominent. In high-bandwidth, low-latency environments, the coding overhead may be more noticeable because it occupies a larger proportion of the overall data retrieval time. However, in low-bandwidth or high-latency networks, the overhead can become less noticeable because the coding delay would be insignificant compared to the overall delay. The hardware capabilities of the network members is another factor to consider. The computational power of participating nodes affects the encoding and decoding processing times. Nodes with more powerful CPUs can handle coding operations more efficiently.

6.3 CodedBitSwap: A Promising Solution

In the domain of distributed file storage systems, IPFS has emerged as a prominent file system, offering a decentralized and resilient platform for data sharing. However, the BitSwap protocol, a core component of IPFS responsible for data exchange, suffers from redundancy issues when the data is popular in the network. This study addresses this challenge by proposing, implementing, and evaluating CodedBitSwap, a novel data exchange protocol that extends the functionality of BitSwap and uses RLNC technology.

The study meticulously explores the optimal coding parameters for IPFS blocks, ensuring that the chosen parameters strike a balance between redundancy reduction and computational complexity. By carefully evaluating different coding rates and block sizes, the study identifies the most suitable parameters for maximizing the performance benefits of CodedBitSwap. Our innovative approach to designing the protocol and identifying optimal network coding parameters for maintaining a low level of coding overhead establishes a valuable framework for future research in network coding-based protocols. This pioneering approach strengthens the credibility and reliability of our findings, as we are the first to utilize this specific methodology.

The experimental evaluation results of CodedBitSwap protocol has demonstrated reduced redundancy, offering a promising solution for efficiency gains. Additionally, CodedBitSwap has the potential to reduce the number of RTTs for data retrieval due to its ability to start retrieving coded blocks from the first contact with multiple sources. However, the acceptability of the associated costs can be negotiable. The decision regarding its core integration within IPFS remains complex. By carefully considering the benefits and drawbacks, the study equips readers with a comprehensive understanding of the implications of implementing CodedBitSwap.

6.4 Future Directions

While this study takes a step forward in advancing IPFS performance and efficiency, the author envision further evolution of IPFS. CodedBitSwap serves as an inspiration trigger for future research which could potentially investigate the following avenues:

- **Adaptive CodedBitSwap:** Adaptive CodedBitSwap enables the RLNC approach

in cases that are most suitable, such as when distant peers cause RTT values to be large and where the cost of coding time is justifiable (e.g, with large files), and works as standard BitSwap protocol otherwise.

- **Adaptive Generation Size:** Dynamically adjusting the generation size based on network conditions and file characteristics could help balancing coding overhead with download efficiency.
- **Exploring Different Network Scenarios:** RLNC can be highly efficient in multi-source and multi-destination transmissions. Exploring these scenarios could reveal potential advantages.
- **Further Investigation of RLNC Potential Advantages:** Evaluating the potential advantages of RLNC that were not part of the evaluation of this study, such as improved network throughput, data encryption, tamper detection, and enhanced network robustness.
- **Pre-Generated Coded Blocks:** It is noteworthy that download time overhead can be further minimized by pre-generating coded blocks upon block addition to IPFS local storage. This approach may eliminate the need for on-demand encoding, reducing encoding latency and potentially improving overall download times. However, this strategy requires additional storage space to accommodate the pre-generated coded blocks.
- **Geographically Distant Nodes:** Developing simulations that consider network latency and geographical distances between peers would provide a better visualization of the possible level of RTTs reduction and the acceptability of the coding costs.

- **Efficient Decoding Algorithms:** Developing more efficient decoding algorithms could further minimize decoding latency and improve overall performance.

APPENDIX A

ENCODING DATA PACKETS

CALCULATIONS

This appendix demonstrates the calculations involved in encoding two data packets using RLNC as part of the example described in Section 2.2.4.

A.1 Assumptions

We are working with a finite field $GF(2^8)$. Two data packets P_1 and P_2 , each containing 16 bits, are to be encoded. The symbol length is 8 bits, making each packet containing two symbols. The online calculator available in (<https://asecuritysite.com/principles/gf>) will be used for performing the operations.

A.2 Encoding Calculation

Each of the packet consists of two symbols $P_1 = (145, 167)$, $P_2 = (120, 228)$. In this section we show how two decoded packets, E_1 and E_2 , are constructed.

A.2.1 First Encoded Packet

The selected coefficients by the first encoder are $\alpha_{1,1} = 17$ and $\alpha_{1,2} = 136$.

Multiplying $\alpha_{1,1}$ by each symbol of P_1 :

$$17 \times 145 = 116$$

$$17 \times 167 = 5$$

Therefore, the result is $M_1 = (116, 5) = (01110100, 00000101)$.

Multiplying $\alpha_{1,2}$ by each symbol of P_2 :

$$136 \times 120 = 49$$

$$136 \times 228 = 16$$

Therefore, the result is $M_2 = (49, 16) = (00110001, 00010000)$.

E_1 is constructed by bit-wisely XORing M_1 and M_2 :

$$0111010000000101$$

$$\oplus$$

$$0011000100010000$$

$$=$$

$$0100010100010101$$

Hence, the first encoded packet (after converting to decimal) is $E_1 = (69, 21)$

A.2.2 Second Encoded Packet

The selected coefficients by the second encoder are $\alpha_{2,1} = 72$ and $\alpha_{2,2} = 28$.

Multiplying $\alpha_{2,1}$ by each symbol of P_1 :

$$72 \times 145 = 79$$

$$72 \times 167 = 227$$

Therefore, the result is $M_1 = (79, 227) = (01001111, 11100011)$.

multiplying $\alpha_{2,2}$ by each symbol of P_2

$$28 \times 120 = 201$$

$$28 \times 228 = 34$$

Therefore, the result is $M_2 = (201, 34) = (11001001, 00100010)$.

E_2 is constructed by bit-wisely XORing M_1 and M_2 ,

$$010011111100011$$

$$\oplus$$

$$1100100100100010$$

$$=$$

1000011011000001

Hence, the second encoded packet is $E_2 = (134, 193)$

APPENDIX B

PUBLICATIONS DERIVED FROM THESIS WORK

This appendix presents the publications made based on the research presented in this thesis. Two publications are identified, one has already published, while the other is ready for submission.

- **Paper 1:** Optimizing Efficiency of P2P Content Distribution with Network Coding: Principles, Challenges, and Future Directions in the *Journal of Network and Computer Applications*.
- **Abstract:** Network coding technology has opened new avenues for enhancing networking applications. Peer-to-peer (P2P) content distribution stands as a particularly promising application domain where network coding can effectively reduce network load and transmission time. However, the additional complexity introduced by coding operations raises questions about the trade-offs and overall cost-benefit analysis of applying network coding technology in real-world scenarios. Moreover, the inconsistent presentation of network coding concepts and terminologies in the literature

necessitates a comprehensive examination and a consolidated summary on the topic. This study explores nearly two decades of the network coding-based applications in P2P content distribution literature by conducting a systematic review that extracts and analyzes data from 41 carefully selected studies with the aim to address specific research questions that will reveal the potential practical benefits of network coding and highlight its associated challenges. Given the diverse nature of how encoders and decoders operate, this review identifies three distinct data preparation structures for coding operations: single-generation, multi-generation, and multi-generations with overlapping. Additionally, this study provides a comprehensive categorization and in-depth discussion of the four identified network coding schemes: deterministic coding, complete coding, sparse coding, and sliding-window coding. Considering the vast array of network coding implementation options, this study presents and discusses which programming libraries, Galois fields, packet sizes, generation sizes, and file sizes are employed in the literature. Furthermore, the metrics used for evaluating network coding-based systems are enumerated and categorized, and the reported evaluation results are presented. This study has identified various literature gaps and unresolved issues, highlighting future research directions that may help overcome current literature limitations.

- **Contribution:** This paper offers a comprehensive overview of network coding’s potential for P2P efficiency optimization. It combines a theoretical foundation with practical considerations, highlighting the challenges and opportunities for further research. This work aims to stimulate further exploration in this promising area for improving the performance and scalability of P2P systems.

- **Paper 2:** Optimizing IPFS Data Exchange with CodedBitSwap: A Network Coding Approach (In journal review).
- **Abstract:** The InterPlanetary File System (IPFS) has emerged as a promising peer-to-peer (P2P) distributed file-sharing system poised to become the backbone of Web3. However, its BitSwap protocol, responsible for block exchange, encounters redundancy issues when multiple peers respond with duplicate blocks. To address this limitation, we propose CodedBitSwap, an innovative network coding-based data exchange protocol that integrates Random Linear Network Coding (RLNC) into BitSwap. Considering that RLNC operations incur additional computational overhead, the RLNC-based protocol is designed with careful attention to its computational complexity that is investigated through trial experiments that guided the selection of coding parameters and structures. To assess the feasibility and performance of CodedBitSwap, an experimental evaluation that compares it with BitSwap was conducted using a controlled testbed environment consisting of 11 nodes exchanging three files of different sizes. During file exchange, amount of data transmitted, download time, and encoding and decoding times were measured for each node. The evaluation results demonstrate that CodedBitSwap effectively eliminates redundancy at a relatively low cost of increased download time. The introduced RLNC computational complexity was optimized by the generation-based design strategy that minimizes it, ensuring that the cost of the reduced redundancy remains within a low range (below 50ms per 1MB). The design methodology of CodedBitSwap can serve as a practical approach for balancing RLNC coding overhead with its gained benefits. Our work contributes to the advancement of network coding in P2P networks and demonstrates its potential to improve the efficiency of IPFS, opening up avenues for future research.

- **Contribution:** This paper introduces CodedBitSwap, a novel network coding protocol designed specifically for IPFS. We present the design rationale, implementation details, and experimental evaluation demonstrating its potential for optimizing data exchange within IPFS. The work contributes to the development of efficient P2P protocols by showcasing a practical application of network coding in a real-world content distribution network.

By delving deeper into these areas, these publications offer valuable contributions to the advancement of both theoretical and practical applications of network coding within the P2P content distribution landscape in general, and in IPFS specifically.

REFERENCES

- [1] J. Benet, “IPFS-content addressed, versioned, P2P file system,” *arXiv preprint arXiv:1407.3561*, 2014.
- [2] E. Daniel and F. Tschorsch, “IPFS and friends: A qualitative comparison of next generation peer-to-peer data networks,” *IEEE Communications Surveys & Tutorials*, vol. 24, no. 1, pp. 31–52, 2022.
- [3] Y. Psaras and D. Dias, “The interplanetary file system and the filecoin network,” in *the 50th Annual International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*. IEEE, 2020, pp. 80–80.
- [4] D. McKay, “What is the interplanetary file system (IPFS) and how do you use it?” Feb 2022. [Online]. Available: <https://www.howtogeek.com/784295/what-is-the-interplanetary-file-system-ipfs/>
- [5] J. M. S. Yiannis Psaras and D. Dias, “Why the internet needs the interplanetary file system,” 2022. [Online]. Available: <https://spectrum.ieee.org/peer-to-peer-network>
- [6] J. Shen, Y. Li, Y. Zhou, and X. Wang, “Understanding I/O performance of IPFS storage: a client’s perspective,” in *Proceedings of the International Symposium on Quality of Service*, 2019, pp. 1–10.

- [7] O. Abdullah Lajam and T. Ahmed Helmy, “Performance evaluation of IPFS in private networks,” in *the 4th International Conference on Data Storage and Data Engineering*, 2021, pp. 77–84.
- [8] X. Chu and Y. Jiang, “Random linear network coding for peer-to-peer applications,” *IEEE Network*, vol. 24, no. 4, pp. 35–39, 2010.
- [9] S. Zimmermann, J. Rischke, J. A. Cabrera, and F. H. Fitzek, “Journey to MARS: Interplanetary coding for relieving CDNs,” in *Global Communications Conference*. IEEE, 2020, pp. 1–6.
- [10] A. De la Rocha, D. Dias, and Y. Psaras, “Accelerating content routing with Bitswap: A multi-path file transfer protocol in IPFS and filecoin,” 2021.
- [11] “IPFS official documentation,” <https://ipfs.tech/>, accessed: 2024-04-27.
- [12] M. Szydło, “Merkle tree traversal in log space and time,” in *Advances in Cryptology-EUROCRYPT: International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2004, pp. 541–554.
- [13] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” *Decentralized Business Review*, p. 21260, 2008. [Online]. Available: <https://bitcoin.org/bitcoin.pdf>
- [14] G. Wood *et al.*, “Ethereum: A secure decentralised generalised transaction ledger,” *Ethereum Project Yellow Paper*, vol. 151, no. 2014, pp. 1–32, 2014.
- [15] D. Bleichenbacher and U. M. Maurer, “Directed acyclic graphs, one-way functions and digital signatures,” in *the Annual International Cryptology Conference*. Springer, 1994, pp. 75–82.
- [16] A. Tridgell, P. Mackerras *et al.*, “The rsync algorithm,” 1996.

- [17] A. Z. Broder, “Some applications of rabin’s fingerprinting method,” in *Sequences II: Methods in Communication, Security, and Computer Science*. Springer, 1993, pp. 143–152.
- [18] M. J. Freedman, E. Freudenthal, and D. Mazieres, “Democratizing content publication with coral,” in *NSDI*, vol. 4, 2004, pp. 18–18.
- [19] I. Baumgart and S. Mies, “S/kademlia: A practicable approach towards secure key-based routing,” in *the International Conference on Parallel and Distributed Systems*. IEEE, 2007, pp. 1–8.
- [20] M. J. Freedman and D. Mazieres, “Sloppy hashing and self-organizing clusters,” in *Peer-to-Peer Systems II: Second International Workshop, IPTPS, Berkeley, CA, USA*. Springer, 2003, pp. 45–55.
- [21] IPFS. (2023) IPFS Documentation: Distributed Hash Table. [Online]. Available: <https://docs.ipfs.tech/concepts/dht/>
- [22] R. Stewart, “Stream control transmission protocol,” Tech. Rep., 2007.
- [23] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind, “Low extra delay background transport (ledbat),” Tech. Rep., 2012.
- [24] O. Lajam and S. Mohammed, “Optimizing efficiency of p2p content distribution with network coding: Principles, challenges, and future directions,” *Journal of Network and Computer Applications*, p. 103825, 2024.
- [25] H. Jürgensen and S. Konstantinidis, “Codes,” in *Handbook of Formal Languages*. Springer, 1997, pp. 511–607.

- [26] C. Easttom, “Modern cryptography,” *Applied Mathematics for Encryption and Information Security*. McGraw-Hill Publishing, 2015.
- [27] M. F. Mushtaq, S. Jamel, A. H. Disina, Z. A. Pindar, N. S. A. Shakir, and M. M. Deris, “A survey on the cryptographic encryption algorithms,” *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 11, pp. 333–344, 2017.
- [28] Z. Zhang, “Linear network error correction codes in packet networks,” *IEEE Transactions on Information Theory*, vol. 54, no. 1, pp. 209–218, 2008.
- [29] T. K. Moon, *Error correction coding: mathematical methods and algorithms*. John Wiley & Sons, 2020.
- [30] G. L. Mullen and D. Panario, *Handbook of finite fields*. CRC Press Boca Raton, 2013, vol. 17.
- [31] M. R. Murty, “Prime numbers and irreducible polynomials,” *The American Mathematical Monthly*, vol. 109, no. 5, pp. 452–458, 2002.
- [32] J. Guajardo, T. Güneysu, S. S. Kumar, C. Paar, and J. Pelzl, “Efficient hardware implementation of finite fields with applications to cryptography,” *Acta Applicandae Mathematica*, vol. 93, no. 1, pp. 75–118, 2006.
- [33] R. Ahlswede, N. Cai, S.-Y. Li, and R. W. Yeung, “Network information flow,” *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.
- [34] D. Medhi and K. Ramasamy, *Network routing: algorithms, protocols, and architectures*. Morgan Kaufmann, 2017.

- [35] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, “A random linear network coding approach to multicast,” *IEEE Transactions on Information Theory*, vol. 52, no. 10, pp. 4413–4430, 2006.
- [36] S.-Y. Li, R. W. Yeung, and N. Cai, “Linear network coding,” *IEEE Transactions on Information Theory*, vol. 49, no. 2, pp. 371–381, 2003.
- [37] R. Koetter and M. Médard, “An algebraic approach to network coding,” *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 782–795, 2003.
- [38] N. J. Higham, “Gaussian elimination,” *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 3, no. 3, pp. 230–238, 2011.
- [39] F. Fitzek, “Intra-flow coding - practical implementation of network coding,” University lecture, 2019. [Online]. Available: <https://cn.ifn.et.tu-dresden.de/wp-content/uploads/2019/04/2019-NetworkCoding-IntraFlow.pdf>(accessed:2024-04-27)
- [40] P. A. Chou, Y. Wu, and K. Jain, “Practical network coding,” in *Proceedings of the Annual Allerton Conference on Communication Control and Computing*, vol. 41, no. 1. The University; 1998, 2003, pp. 40–49.
- [41] A. A. AbuDaqa, A. Mahmoud, M. Abu-Amara, and T. Sheltami, “Survey of network coding based P2P file sharing in large scale networks,” *Applied Sciences*, vol. 10, no. 7, p. 2206, 2020.
- [42] Y. Li, S.-L. Hu, J. Wang, and Z.-H. Huang, “An introduction to the computational complexity of matrix multiplication,” *Journal of the Operations Research Society of China*, vol. 8, pp. 29–43, 2020.

- [43] C. Zhang, P. Dhungel, D. Wu, and K. W. Ross, “Unraveling the BitTorrent ecosystem,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 7, pp. 1164–1177, 2010.
- [44] R. L. Xia and J. K. Muppala, “A survey of BitTorrent performance,” *IEEE Communications Surveys & Tutorials*, vol. 12, no. 2, pp. 140–158, 2010.
- [45] B. Zhang, A. Iosup, J. Pouwelse, and D. Epema, “Identifying, analyzing, and modeling flashcrowds in BitTorrent,” in *the International Conference on Peer-to-Peer Computing*. IEEE, 2011, pp. 240–249.
- [46] F. Fitzek, “Random linear coding,” YouTube - ComNets Lectures, 2019, (accessed: 2024-04-27). [Online]. Available: <https://youtu.be/D0SM50m87CA>
- [47] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, “A digital fountain approach to reliable distribution of bulk data,” *ACM SIGCOMM Computer Communication Review*, vol. 28, no. 4, pp. 56–67, 1998.
- [48] J. W. Byers, M. Luby, and M. Mitzenmacher, “A digital fountain approach to asynchronous reliable multicast,” *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1528–1540, 2002.
- [49] D. J. MacKay, “Fountain codes,” *IEE Proceedings-Communications*, vol. 152, no. 6, pp. 1062–1068, 2005.
- [50] J. W. Byers, M. Luby, and M. Mitzenmacher, “A digital fountain retrospective,” *ACM SIGCOMM Computer Communication Review*, vol. 49, no. 5, pp. 82–85, 2019.
- [51] M. Luby, “Lt codes,” in *the 43rd Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 2002, pp. 271–271.

- [52] S. Puducheri, J. Klierer, and T. E. Fuja, “The design and performance of distributed lt codes,” *IEEE Transactions on Information Theory*, vol. 53, no. 10, pp. 3740–3754, 2007.
- [53] J. He, I. Hussain, Y. Li, M. Juntti, and T. Matsumoto, “Distributed lt codes with improved error floor performance,” *IEEE Access*, vol. 7, pp. 8102–8110, 2019.
- [54] A. Shokrollahi, “Raptor codes,” *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2551–2567, 2006.
- [55] M. Luby, A. Shokrollahi, M. Watson, and T. Stockhammer, “Raptor forward error correction scheme for object delivery,” Tech. Rep., 2007.
- [56] H. R. Oh, D. O. Wu, and H. Song, “An effective mesh-pull-based P2P video streaming system using fountain codes with variable symbol sizes,” *Computer Networks*, vol. 55, no. 12, pp. 2746–2759, 2011.
- [57] C. Westphal, “A stable fountain code mechanism for peer-to-peer content distribution,” in *Conference on Computer Communications*. IEEE, 2014, pp. 2571–2579.
- [58] P. M. Eittenberger, T. Mladenov, and U. R. Krieger, “Raptor codes for P2P streaming,” in *the 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing*. IEEE, 2012, pp. 327–332.
- [59] Z. Su, F. Wang, J. Daigle, H. Wang, and T. Shan, “RaptorQP2P: maximize the performance of P2P file distribution with raptorq coding,” in *the International Conference on Communications (ICC)*. IEEE, 2015, pp. 5331–5336.

- [60] H. R. Oh and H. Song, “Mesh-pull-based P2P video streaming system using fountain codes,” in *Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2011, pp. 1–6.
- [61] R. W. Yeung, “Coding for a network coded fountain,” YouTube - Microsoft Research, 2012, accessed: 2024-04-27. [Online]. Available: <https://youtu.be/VUwaHzvcTo8>
- [62] J. Heide, M. V. Pedersen, and F. H. Fitzek, “Decoding algorithms for random linear network codes,” in *the International Conference on Research in Networking*. Springer, 2011, pp. 129–136.
- [63] V. Nguyen, E. Tasdemir, G. T. Nguyen, D. E. Lucani, F. H. Fitzek, and M. Reisslein, “Dsep fulcrum: Dynamic sparsity and expansion packets for fulcrum network coding,” *IEEE Access*, vol. 8, pp. 78 293–78 314, 2020.
- [64] S. B. Wicker and V. K. Bhargava, *Reed-Solomon codes and their applications*. John Wiley & Sons, 1999.
- [65] P. Garrido, A. Fernandez, and R. Agueero, “To recode or not to recode: Optimizing rlnc recoding and performance evaluation over a cots platform,” in *European Wireless; 24th European Wireless Conference*. VDE, 2018, pp. 1–7.
- [66] S. Wunderlich, F. Gabriel, S. Pandi, F. H. Fitzek, and M. Reisslein, “Caterpillar RLNC (CRLNC): A practical finite sliding window rlnc approach,” *IEEE Access*, vol. 5, pp. 20 183–20 197, 2017.
- [67] A. Passarella, “A survey on content-centric technologies for the current internet: CDN and P2P solutions,” *Computer Communications*, vol. 35, no. 1, pp. 1–32, 2012.

- [68] A. Mondal, I. Trestian, Z. Qin, and A. Kuzmanovic, “P2P as a CDN: A new service model for file sharing,” *Computer Networks*, vol. 56, no. 14, pp. 3233–3246, 2012.
- [69] Y. Jia, T. Moataz, S. Tople, and P. Saxena, “OblivP2P: An oblivious peer-to-peer content sharing system.” in *USENIX Security Symposium*, 2016, pp. 945–962.
- [70] M. Zhao, P. Aditya, A. Chen, Y. Lin, A. Haeberlen, P. Druschel, B. Maggs, B. Wishon, and M. Ponc, “Peer-assisted content distribution in akamai netsession,” in *Proceedings of the Internet Measurement Conference*, 2013, pp. 31–42.
- [71] Y. Huang, T. Z. Fu, D.-M. Chiu, J. C. Lui, and C. Huang, “Challenges, design and analysis of a large-scale P2P-vod system,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 375–388, 2008.
- [72] B. Fan, D. G. Andersen, M. Kaminsky, and K. Papagiannaki, “Balancing throughput, robustness, and in-order delivery in P2P vod,” in *Proceedings of the 6th International Conference*, 2010, pp. 1–12.
- [73] R. Nygaard, “Improving data availability in decentralized storage systems,” 2023.
- [74] K. Xu, “Performance modeling of BitTorrent peer-to-peer file sharing networks,” *arXiv preprint arXiv:1311.1195*, 2013.
- [75] G. Urdaneta, G. Pierre, and M. V. Steen, “A survey of DHT security techniques,” *ACM Computing Surveys (CSUR)*, vol. 43, no. 2, pp. 1–49, 2011.
- [76] T. R. Mayer, L. Brunie, D. Coquil, and H. Kosch, “On reliability in publish/subscribe systems: a survey,” *International Journal of Parallel, Emergent and Distributed Systems*, vol. 27, no. 5, pp. 369–386, 2012.

- [77] P. Garbacki, A. Iosup, D. Epema, and M. Van Steen, “2fast: Collaborative downloads in P2P networks,” in *the 6th International Conference on Peer-to-Peer Computing (P2P)*. IEEE, 2006, pp. 23–30.
- [78] M. Khambatti, K. D. Ryu, and P. Dasgupta, “Structuring peer-to-peer networks using interest-based communities,” in *Databases, Information Systems, and Peer-to-Peer Computing: First International Workshop*. Springer, 2004, pp. 48–63.
- [79] R. Farahbakhsh, N. Crespi, Á. Cuevas, S. Adhikari, M. Mani, and T. Sangunkotchakorn, “socP2P: P2P content discovery enhancement by considering social networks characteristics,” in *IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2012, pp. 530–533.
- [80] “How IPFS works.” [Online]. Available: <https://docs.ipfs.tech/concepts/how-ipfs-works/>
- [81] V. Le, R. Moazeni, and M. Moh, “Improving security and performance of distributed IPFS-based web applications with blockchain,” in *Advances in Cyber Security: Third International Conference*. Springer, 2021, pp. 114–127.
- [82] Y. Chen, H. Li, K. Li, and J. Zhang, “An improved P2P file system scheme based on IPFS and blockchain,” in *the International Conference on Big Data*. IEEE, 2017, pp. 2652–2657.
- [83] M. Song, J. Han, H. Eom, and Y. Son, “IPFSz: An efficient data compression scheme in interplanetary file system,” *IEEE Access*, vol. 10, pp. 601–611, 2022.

- [84] C. Gkantsidis and P. R. Rodriguez, “Network coding for large scale content distribution,” in *the 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 4. IEEE, 2005, pp. 2235–2245.
- [85] M. Hundeboll, J. Ledet-Pedersen, G. Sluiterman, T. K. Madsen, and F. H. Fitzek, “Peer-assisted content distribution with random linear network coding,” in *the 79th Vehicular Technology Conference (VTC Spring)*. IEEE, 2014, pp. 1–6.
- [86] P. J. Braun, P. Ekler, and F. Fitzek, “Network coding enhanced browser based peer-to-peer streaming,” in *the International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2016, pp. 002 104–002 109.
- [87] H. Zeng, J. Huang, S. Tao, and W. Cheng, “A simulation study on network coding parameters in P2P content distribution system,” in *the 3rd International Conference on Communications and Networking in China*. IEEE, 2008, pp. 197–201.
- [88] P. J. Braun, M. Sipos, P. Ekler, and H. Charaf, “Increasing data distribution in BitTorrent networks by using network coding techniques,” in *Proceedings of European Wireless; 21th European Wireless Conference*. VDE, 2015, pp. 1–6.
- [89] O. Kaiqian, X. Yinlong, M. Guanjuan, and Z. Yulin, “Dasher: a peer-to-peer content distribution system based on combined network coding,” in *the 2nd International Conference on Broadband Network & Multimedia Technology*. IEEE, 2009, pp. 687–692.
- [90] C. Ortolfo, C. Schindelhauer, and A. Vater, “Paircoding: Improving file sharing using sparse network codes,” in *the 4th International Conference on Internet and Web Applications and Services*. IEEE, 2009, pp. 49–57.

- [91] G. Xu, Y. Hu, Y. Shen, and M. Guo, “Directedpush-a high performance peer-to-peer live streaming system using network coding,” in *the 18th International Conference on Parallel and Distributed Systems*. IEEE, 2012, pp. 292–298.
- [92] P. J. Braun, M. Sipos, P. Ekler, and F. H. Fitzek, “On the performance boost for peer to peer webrtc-based video streaming with network coding,” in *the International Conference on Communications (ICC)*. IEEE, 2017, pp. 1–6.
- [93] M. Wang and B. Li, “Network coding in live peer-to-peer streaming,” *IEEE Transactions on Multimedia*, vol. 9, no. 8, pp. 1554–1567, 2007.
- [94] S. Wang, J. Zhao, and X. Wang, “Is network coding helpful for BitTorrent: from a practitioner’s perspective,” in *the 19th International Conference on Computer Communications and Networks*. IEEE, 2010, pp. 1–6.
- [95] T.-Y. Chung, C.-C. Wang, Y.-M. Chen, and Y.-H. Chang, “Pnecos: a peer-to-peer network coding streaming system,” in *the International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC)*. IEEE, 2008, pp. 379–384.
- [96] M. Toldo and E. Magli, “A resilient and low-delay P2P streaming system based on network coding with random multicast trees,” in *the International Workshop on Multimedia Signal Processing*. IEEE, 2010, pp. 400–405.
- [97] K. Nguyen, T. Nguyen, and S.-c. Cheung, “Peer-to-peer streaming with hierarchical network coding,” in *the International Conference on Multimedia and Expo*. IEEE, 2007, pp. 396–399.

- [98] M. Wang and B. Li, “R2: Random push with random network coding in live peer-to-peer streaming,” *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 9, pp. 1655–1666, 2007.
- [99] Y. Liu, Y. Peng, W. Dou, and B. Guo, “Network coding for peer-to-peer live media streaming,” in *the 5th International Conference on Grid and Cooperative Computing (GCC)*. IEEE, 2006, pp. 149–155.
- [100] D. Niu and B. Li, “Analyzing the resilience-complexity tradeoff of network coding in dynamic P2P networks,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 11, pp. 1842–1850, 2011.
- [101] J. Su, Q. Deng, and D. Long, “Pclnc: A low-cost intra-generation network coding strategy for P2P content distribution,” *Peer-to-Peer Networking and Applications*, vol. 12, no. 1, pp. 177–188, 2019.
- [102] M. Yang and Y. Yang, “Applying network coding to peer-to-peer file sharing,” *IEEE Transactions on Computers*, vol. 63, no. 8, pp. 1938–1950, 2013.
- [103] L. Cui and N. Lu, “Socialstreaming: P2P-assisted streaming in social networks,” in *IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2013, pp. 502–507.
- [104] X. Chen, N. Ren, X. Zhang, X. Wang, and J. Zhao, “Sonicstream: A network coding based live P2P media streaming system with rich user experiences,” *Journal of Communications and Networks*, vol. 10, no. 4, pp. 430–436, 2008.

- [105] J. Xu, J. Zhao, X. Wang, and X. Xue, “Swifter: Chunked network coding for peer-to-peer content distribution,” in *the International Conference on Communications*. IEEE, 2008, pp. 5603–5608.
- [106] S. Mirshokraie and M. Hefeeda, “Live peer-to-peer streaming with scalable video coding and networking coding,” in *Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems*, 2010, pp. 123–132.
- [107] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P. R. Rodriguez, “Is high-quality vod feasible using P2P swarming?” in *Proceedings of the 16th International Conference on World Wide Web*, 2007, pp. 903–912.
- [108] L. Yu, L. Gao, J. Zhao, and X. Wang, “Sonicvod: a vcr-supported P2P-vod system with network coding,” *IEEE Transactions on Consumer Electronics*, vol. 55, no. 2, pp. 576–582, 2009.
- [109] M. Martalo, M. Picone, R. Bussandri, and M. Amoretti, “A practical network coding approach for peer-to-peer distributed storage,” in *the International Symposium on Network Coding (NetCod)*. IEEE, 2010, pp. 1–6.
- [110] M. Anandaraj, P. Ganeshkumar, K. Selvaraj, and K. Vijayakumar, “An efficient framework for network code based multimedia content distribution in hybrid P2P network,” *Journal of Circuits, Systems and Computers*, vol. 29, no. 10, p. 2050157, 2020.
- [111] K. Nguyen, T. Nguyen, and S.-C. Cheung, “Video streaming with network coding,” *Journal of Signal Processing Systems*, vol. 59, no. 3, pp. 319–333, 2010.

- [112] A. Fiandrotti, A. M. Sheikh, and E. Magli, “Towards a P2P videoconferencing system based on low-delay network coding,” in *Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*. IEEE, 2012, pp. 1529–1533.
- [113] F. de Asis López-Fuentes and J. Mendoza-Almanza, “Dynamic network coding for collaborative multisource system,” in *the 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE, 2018, pp. 378–382.
- [114] J. Mendoza-Almanza, F. d. A. López-Fuentes, and R. Hasimoto-Beltran, “Practical network coding for multi-source scenarios,” in *Smart Technology*. Springer, 2018, pp. 141–148.
- [115] B. Barekatain, D. Khezrimotlagh, M. Aizaini Maarof, H. R. Ghaeini, S. Salleh, A. A. Quintana, B. Akbari, and A. T. Cabrera, “Matin: A random network coding based framework for high quality peer-to-peer live video streaming,” *Plos One*, vol. 8, no. 8, p. e69844, 2013.
- [116] N. Wang and N. Ansari, “Downloader-initiated random linear network coding for peer-to-peer file sharing,” *IEEE Systems Journal*, vol. 5, no. 1, pp. 61–69, 2010.
- [117] Q.-C. Cai and K.-T. Lo, “Two blocks are enough: on the feasibility of using network coding to ameliorate the content availability of BitTorrent swarms,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 8, pp. 1682–1694, 2012.
- [118] A. Marandi, H. Sehat, D. E. Lucani, S. Mousavifar, and R. H. Jacobsen, “Network coding-based data storage and retrieval for kademlia,” in *the 93rd Vehicular Technology Conference (VTC2021-Spring)*. IEEE, 2021, pp. 1–7.

- [119] G. Ma, Y. Xu, K. Ou, and W. Luo, “How can network coding help P2P content distribution?” in *the International Conference on Communications*. IEEE, 2009, pp. 1–5.
- [120] M. Wang and B. Li, “How practical is network coding?” in *the 14th International Workshop on Quality of Service*. IEEE, 2006, pp. 274–278.
- [121] A. Botoş, V. Bota, and M. P. Ştef, “Study of a linear coding enhanced swarm network,” in *the 7th International Conference on Intelligent Computer Communication and Processing*. IEEE, 2011, pp. 415–421.
- [122] J.-S. Leu, M.-C. Yu, and H.-C. Yueh, “Improving network coding based file sharing for unstructured peer-to-peer networks,” *Journal of Network and Systems Management*, vol. 23, no. 4, pp. 803–829, 2015.
- [123] D. Hu, M. Wang, F. Lau, and Q. Peng, “Network coding for resilient peer-to-peer networks,” in *the 7th International Workshop on Design of Reliable Communication Networks*. IEEE, 2009, pp. 352–358.
- [124] J.-P. Thibault, S. Yousefi, and W.-Y. Chan, “Throughput performance of generation-based network coding,” in *the 10th Canadian Workshop on Information Theory (CWIT)*. IEEE, 2007, pp. 89–92.
- [125] E. Skevakis and I. Lambadaris, “Decoding and file transfer delay balancing in network coding broadcast,” in *IEEE International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–7.

- [126] Y. Li, E. Soljanin, and P. Spasojevic, “Effects of the generation size and overlap on throughput and complexity in randomized linear network coding,” *IEEE Transactions on Information Theory*, vol. 57, no. 2, pp. 1111–1123, 2011.
- [127] P. Maymounkov, N. J. Harvey, and D. S. Lun, “Methods for efficient network coding,” in *Proceedings of the 44th Annual Allerton Conference on Communication, Control, and Computing*, 2006, pp. 482–491.
- [128] W. Xu and A. K. Tang, “A generalized coupon collector problem,” *Journal of Applied Probability*, vol. 48, no. 4, pp. 1081–1094, 2011.
- [129] A. Heidarzadeh and A. H. Banihashemi, “Overlapped chunked network coding,” in *the Information Theory Workshop on Information Theory*. IEEE, 2010, pp. 1–5.
- [130] B. Tang and S. Yang, “An improved design of overlapped chunked codes,” in *the International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–6.
- [131] D. Silva, W. Zeng, and F. R. Kschischang, “Sparse network coding with overlapping classes,” in *the Workshop on Network Coding, Theory, and Applications*. IEEE, 2009, pp. 74–79.
- [132] Y. Li, W.-Y. Chan, and S. D. Blostein, “Network coding with unequal size overlapping generations,” in *the International Symposium on Network Coding (NetCod)*. IEEE, 2012, pp. 161–166.
- [133] Y.-h. Chu, S. G. Rao, S. Seshan, and H. Zhang, “A case for end system multicast,” *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1456–1471, 2002.

- [134] J. Heide, M. V. Pedersen, F. H. Fitzek, and M. Médard, “On code parameters and coding vector representation for practical rlnc,” in *the International Conference on Communications (ICC)*. IEEE, 2011, pp. 1–5.
- [135] S. C. Althoen and R. Mclaughlin, “Gauss-jordan reduction: A brief history,” *The American Mathematical Monthly*, vol. 94, no. 2, pp. 130–142, 1987.
- [136] R. W. Yeung, “Avalanche: A network coding analysis,” *Communications in Information & Systems*, vol. 7, no. 4, pp. 353–358, 2007.
- [137] S. Chib and E. Greenberg, “Understanding the metropolis-hastings algorithm,” *The American Statistician*, vol. 49, no. 4, pp. 327–335, 1995.
- [138] B. Sredojev, D. Samardzija, and D. Posarac, “WebRTC technology overview and signaling solution design and implementation,” in *the 38th International Conference on Information and Communication Technology, Electronics and Microelectronics*. IEEE, 2015, pp. 1006–1009.
- [139] G. Ma, Y. Xu, M. Lin, and Y. Xuan, “A content distribution system based on sparse linear network coding,” in *the 3rd Workshop on Network Coding (Netcod)*. Citeseer, 2007.
- [140] S. Feizi, D. E. Lucani, and M. Médard, “Tunable sparse network coding,” in *the 22th International Zurich Seminar on Communications (IZS)*. Eidgenössische Technische Hochschule Zürich, 2012.
- [141] C. W. Sorensen, A. S. Badr, J. A. Cabrera, D. E. Lucani, J. Heide, and F. H. Fitzek, “A practical view on tunable sparse network coding,” in *Proceedings of European Wireless; 21th European Wireless Conference*. VDE, 2015, pp. 1–6.

- [142] M. V. Pedersen, D. E. Lucani, F. H. Fitzek, C. W. Sørensen, and A. S. Badr, “Network coding designs suited for the real world: What works, what doesn’t, what’s promising,” in *Information Theory Workshop (ITW)*. IEEE, 2013, pp. 1–5.
- [143] A. Zarei, P. Pahlavani, and D. E. Lucani, “An analytical model for sparse network codes: Field size considerations,” *IEEE Communications Letters*, vol. 24, no. 4, pp. 729–733, 2020.
- [144] A. Zarei, P. Pahlavani, and M. Davoodi, “On the partial decoding delay of sparse network coding,” *IEEE Communications Letters*, vol. 22, no. 8, pp. 1668–1671, 2018.
- [145] P. Garrido, D. E. Lucani, and R. Agüero, “Markov chain model for the decoding probability of sparse network coding,” *IEEE Transactions on Communications*, vol. 65, no. 4, pp. 1675–1685, 2017.
- [146] X. Li, W. H. Mow, and F.-L. Tsang, “Rank distribution analysis for sparse random linear network coding,” in *the International Symposium on Networking Coding*. IEEE, 2011, pp. 1–6.
- [147] T. Hoßfeld, F. Lehrieder, D. Hock, S. Oechsner, Z. Despotovic, W. Kellerer, and M. Michel, “Characterization of BitTorrent swarms and their distribution in the internet,” *Computer Networks*, vol. 55, no. 5, pp. 1197–1215, 2011.
- [148] B. Li, G. Y. Keung, S. Xie, F. Liu, Y. Sun, and H. Yin, “An empirical study of flash crowd dynamics in a P2P-based live video streaming system,” in *Global Telecommunications Conference*. IEEE, 2008, pp. 1–5.

- [149] F. Liu, B. Li, L. Zhong, B. Li, H. Jin, and X. Liao, “Flash crowd in P2P live streaming systems: Fundamental characteristics and design implications,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 7, pp. 1227–1239, 2011.
- [150] A. Lareida, T. Hoßfeld, and B. Stiller, “The BitTorrent peer collector problem,” in *the IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2017, pp. 449–455.
- [151] E. Tasdemir, J. A. Cabrera, F. Gabriel, D. You, and F. H. Fitzek, “Sliding window rlnc on multi-hop communication for low latency,” in *the 93rd Vehicular Technology Conference (VTC)*. IEEE, 2021, pp. 1–6.
- [152] P. Garrido, D. Gómez, J. Lanza, and R. Agüero, “Exploiting sparse coding: A sliding window enhancement of a random linear network coding scheme,” in *the International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–6.
- [153] M. Karzand and D. J. Leith, “Low delay random linear coding over a stream,” in *the 52nd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2014, pp. 521–528.
- [154] S. Wunderlich, F. Gabriel, S. Pandi, and F. H. Fitzek, “We don’t need no generation - a practical approach to sliding window rlnc,” in *Wireless Days*. IEEE, 2017, pp. 218–223.
- [155] M. V. Pedersen, J. Heide, and F. H. Fitzek, “Kodo: An open and research oriented network coding library,” in *the International Conference on Research in Networking*. Springer, 2011, pp. 145–152.

- [156] J. Heide, M. V. Pedersen, F. H. Fitzek, and T. Larsen, “Network coding for mobile devices-systematic binary random rateless codes,” in *the International Conference on Communications Workshops*. IEEE, 2009, pp. 1–6.
- [157] itzmeanjan, “Kodr - random linear network coding,” <https://github.com/itzmeanjan/kodr>, 2021, accessed: 2024-04-27.
- [158] F. Fitzek, “Field size analysis,” YouTube - ComNets Lectures, 2019. [Online]. Available: <https://youtu.be/po8Dl8FSmnE>(accessed:2024-04-27)
- [159] N. J. H. Marcano, J. Heide, D. E. Lucani, and F. H. Fitzek, “On the throughput and energy benefits of network coded cooperation,” in *the 3rd International Conference on Cloud Networking (CloudNet)*. IEEE, 2014, pp. 138–142.
- [160] B. Li and D. Niu, “Random network coding in peer-to-peer networks: From theory to practice,” *Proceedings of the IEEE*, vol. 99, no. 3, pp. 513–523, 2011.
- [161] R. Zeng, J. You, Y. Li, and R. Han, “An icn-based ipfs high-availability architecture,” *Future Internet*, vol. 14, no. 5, p. 122, 2022.
- [162] O. Ascigil, S. Reñé, M. Król, G. Pavlou, L. Zhang, T. Hasegawa, Y. Koizumi, and K. Kita, “Towards peer-to-peer content retrieval markets: Enhancing ipfs with icn,” in *Proceedings of the 6th ACM Conference on Information-Centric Networking*, 2019, pp. 78–88.

VITAE

- Name: Omar Abdullah Saleh Lajam
- Nationality: Yemen
- Date of Birth: 1995-11-24
- Email: *oasl.2009@hotmail.com*
- Permenant Address: Noaman Bin Sinan Street, North Doha, Dhahran 34463, Saudi Arabia
- Bachelor Degree in Software Engineering from King Fahd University of Petroleum and Minerals (January 2020).
- Research Interests: Computer Networks, File Systems, Web Systems, Database Systems, and Software Engineering.
- Has worked on several research projects and published more than four papers in reputable journals.
- The work of this thesis has been published in the Journal of Network and Computer Applications.