Network Coder Optimization for Peer-to-Peer Content Distribution

P2P コンテンツ配信のための ネットワークコーダの最適化

September 2013

Quoc Dinh	NGUYEN

Network Coder Optimization for Peer-to-Peer Content Distribution

P2P コンテンツ配信のための ネットワークコーダの最適化

September 2013

Graduate School of Global Information and Telecommunication Studies
Waseda University

Distributed Computing Systems II

Quoc Dinh	NGUYEN

Abstract

We study the use of *network coding* to speed up content distribution in peer-to-peer (P2P) networks. Our goal is to get the underlying reason for network coding's improved performance in P2P content distribution and to optimize resource consumption of network coding.

In contrast with the current *store-and-forward* routing model, network coding allows network nodes to code, i.e. generating new information from what they have received, and after that, forward the coded information into the network. Each time a node wants to send data, it has to, for example, linearly combine currently available data by a series of numerical multiplications and additions, consuming a certain amount of computational resources.

Network coding, though having been shown to achieve maximum multicast throughput, incurs an expensive cost: practically every node in the network has to code and they code excessively whenever there are incoming requests. A large portion of that huge consumed computational resource, as we find out, can be saved with almost no impact on the optimal performance of network coding. We optimize network coding's resource consumption in the two following aspects: (1) we eliminate unnecessary coding by allowing only selected *important* network nodes to encode; and (2) we further save computational resources at each network encoder by figuring out exactly how much it should encode.

Short distribution time, i.e. the time required to distribute content to all receivers, can be achieved by placing coders at just a subset of carefully chosen peers. Peer-to-peer systems, in addition, tend to be heterogeneous in which some peers, such as hand-held devices, would not have the required capacity to encode. We therefore envision a *hybrid* network coding P2P system where some peers encode to improve distribution time and other peers, due to limited computational capacity or due to some system-wide optimization, do not encode. We begin the dissertation by devising the protocols and data selection algorithm needed to effi-

ciently realize such a hybrid network coding system. Our protocols greatly simplify network operations. They allow peers, being network coders or not, to communicate seamlessly using the same protocol to talk with each other. Our proposed data selection algorithm let peers choose the most updated data to download first which results in noticeable improvement in distribution time. The proposed protocols and data selection algorithm boost the effectiveness of network coding in shortening distribution time which we evaluate by simulations.

To optimize network coder placement, first of all, we observe analytically that in pure P2P content distribution networks without network coding, a considerable amount of data is sent multiple times from one peer to another when there are multiple paths connecting those two particular peers. The duplicated data consume bandwidth on the paths, and therefore, result in sub-optimal delivery throughput to downstream peers on those paths. Network coding, on the other hand, when applied at upstream peers, eliminates information duplication on paths to downstream peers, which results in more efficient content distribution.

Based on the insight obtained from our analysis, we then propose network coder placement algorithms to deploy network coders at selective locations in a given network topology. Our algorithms achieve comparable distribution time as network coding, yet substantially reduces the number of encoders compared to a full network coding solution in which all peers have to encode. Our placement methods put encoders at critical network positions to eliminate information duplication the most, thus, effectively shorten distribution time with just a portion of encoders. In other words, we optimize resource consumption by removing unnecessary or less important network coders.

We propose in total three algorithms to place network coders inside a P2P content distribution network. The first algorithm elaborately figures the delay in finish time a given upstream node causes to its downstream nodes due to duplication, and then, places network coders at nodes which cause the most delay in

finish time. By doing so, we can effectively accelerate content delivery from nodes which create the most duplication, thus shorten distribution time. Our first placement algorithm, which we call minimal delay placement, although can determine accurately how much distribution time can be shortened by placing a coder at a given network node, has the trade-off of rather high computation complexity. With a target to reduce the complexity, the two latter algorithms are based on *network centrality*, a concept borrowed from social network studies, to quickly pinpoint network nodes which stand on more paths and wider paths to other nodes, the characteristics which we show to correlate with the level of duplication, for network coder placement.

Performance evaluation in wide varieties of network topologies by simulations confirms the effectiveness of our proposed network coder placements which could achieve comparable distribution time as full network coding with just a portion of network coders. Evidently, a significant number of network coders can be saved while still realizing short distribution time almost the same as a full network coding solution.

Having optimized network coder placement over the whole network, we then direct our attention to each individual network coder itself. Each time a network coder generates new information from available information, it puts a piece of redundant information into the network which helps accelerate content distribution towards downstream nodes. However, excessive redundancy is not necessary to achieve short distribution time. Since the encoding process consumes computational resources of the coder it is important to figure the right level of redundancy, just enough to eliminate duplication, each network coder should generate. Assuming only a constraint number of selected peers can encode, i.e. become network coders, we next optimize the redundancy network coding generates, i.e. how much a particular encoder should encode. Given the network topology, we analytically figure out the redundancy ratio at each encoder to achieve shortest distribution

time and verify the result by simulations.

We believe our studies, which offer insights into the way network coding improves content distribution and optimize its resource consumption and implementation, will contribute to the understanding of the subject and promote a wider deployment of network coding as a method to speed up content distribution.

We conclude the dissertation with a promising outlook for extending our results to facilitate content distribution in information-centric networking, an active research field which is anticipated to build our future networks.

Acknowledgements

First of all, I would like to thank professors, staff, and fellow students in GITS, Waseda University for their support during my doctorate study.

I would like to express my thanks to my dissertation committee, Prof. Nakazato, Prof. Tanaka, Prof. Tsuda, and Prof. Sato, whose comments and advice have helped improve this dissertation.

I also very much appreciate Dr. Gkantsidis's discussion about one of his work on network coding for peer-to-peer content distribution which has motivated the research herein.

I gratefully acknowledge financial support from MEXT Scholarship throughout the course of my PhD research and support from JSPS KAKENHI Grant Number (24500098).

I am deeply indebted to my adviser, Prof. Nakazato, for his guidance. His encouragement of new ideas has excited me to do my research. And I am grateful to my adviser's proactive support on several occasions as well as the friendly, yet earnest, research atmosphere he created in our lab.

To my parents.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Network coding [1] has recently drawn much research attention owing to its ability to achieve theoretically maximum throughput in multicasting data. By allowing content to be combined at intermediate nodes while being forwarded in the network, the multicast throughput is shown to approach that of the individual maximum throughput to each receiver as if it can utilize the whole network resources [2]. The benefit of network coding, however, is unclear in practical settings such as peer-to-peer content distribution, where non-coding solutions perform reasonably well [3, 4].

We study the use of network coding in peer-to-peer (P2P) content distribution to shorten distribution time. Although peers in P2P networks can readily be turned into network coders[1], questions remain about how we can effectively deploy them. Requiring all peers to code, though may achieve shortest distribution time, is inefficient in terms of computational resources since coding consumes the peer's resources. Our motivation is to get insights into network coding and answer the questions (1) what conditions make network coding good performance, and (2) do we need to code everywhere and all the time to achieve that performance.

That issue is especially important in practical scenarios when a large content

[1]The terms *coders* and *encoders* are used interchangeably.

is distributed or when peers get involved in distributing multiple files as in those cases encoding process consumes huge resources. Reducing computational resource consumption at each peer helps speed up that particular peer's download progress, which also likely accelerates other peers who are downloading from it.

1.1 Contribution

Our contributions are as follows.

1. We identify the underlying condition for network coding to be effective compared with no coding. When there are multiple delivery paths from an upstream peer to a downstream peer, coding at the upstream peer will eliminate *content duplication* to the downstream peer and accelerate its downloading speed. We make an analysis of the duplication incurred in ordinary non-coding P2P content distribution, which serves as the foundation for our proposed network coder optimization.

2. We propose novel coder placement algorithms to reduce the number of network coders. Based on the insight about how network coding improves performance, given a constraint number of network coders, we propose algorithms to locate key network nodes to place the given number of coders in order to shorten distribution time the most.

3. We figure how much an encoder should optimally encode to achieve short distribution time. The redundancy each network coder generates helps accelerate content distribution to its downstream peers. Too high redundancy, however, consumes the encoder's resource unnecessarily. Too low redundancy, on the other hand, will be ineffective. We analyze the exact level of redundancy at each network coder to achieve shortest distribution time.

4. We devise the protocols and data selection algorithm to support P2P content distribution systems where network coding is partially enabled at selected nodes. Our protocols and algorithm simplify operation and improve performance of the P2P content distribution network.

1.2 Network Coding for Peer-to-Peer Content Distribution

Peer-to-peer content distribution is a scalable solution to distribute content, i.e. a file, from a source to all receivers which, unlike the server-client model, also contribute their available bandwidth to help delivery the file. One of the most popular P2P systems, BitTorrent [5], uses parallel downloads to accelerate download speed. The file is divided into equal-size *blocks*, i.e. chunks, pieces, which peers send and receive in parallel, utilizing both available upload and download bandwidth. Each newly joining peer connects to a set of random existing peers, such that to construct a mesh overlay network with random topologies. Furthermore, *rarest* blocks are chosen first by receiving peers to quickly disseminate the whole file into the system. To encourage peers to contribute uploading bandwidth to the system, a peer uploads to, i.e. *unchokes*, a certain number of neighboring peers at a time, which provide the uploading peer the best downloading rates. Rarest first block selection and unchoking are shown to be the reasons underlying BitTorrent excellent performance [3].

Network coding, due to Ahlswede et al. [1], is a method to maximize multicast throughput. Using linear network coding [2, 6], a network coding scheme where encoders generate new data by linearly combining the data they currently have, the multicast throughput to each receiver is shown to asymptotically approach its maximum to that individual receiver as if it can utilize all available network resources by itself. Network coding has been adopted in several research contexts

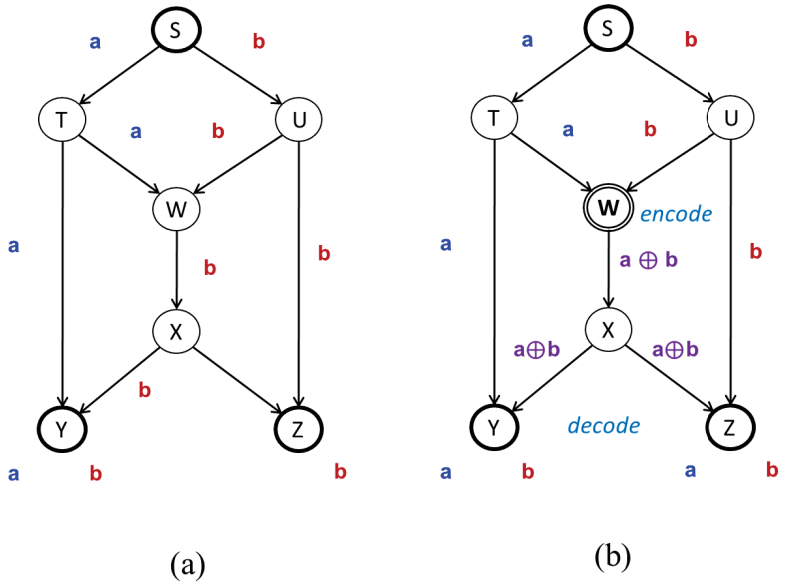

Figure 1.1: Assuming the capacity of every link is 1 bit/s, when note W codes, the throughput to each of receivers Y and Z is maximum at 2 bit/s.

such as network coding in wireless networks [7, 8, 9], physical layer network coding [10], transport-layer network coding [11], network coding for distributed storage systems [12, 13], network coding for P2P content distribution [14, 15, 16] and P2P streaming [17, 18, 19], and network coding in information-centric networks [20]. Interested readers are referred to [21, 22, 23, 24, 25] for comprehensive discussions on network coding.

Figure 1.1 illustrates the benefit of network coding in a *butterfly network* where two bit a and b are multicasted from the source S to two receivers Y and Z. The capacity of each link is 1 bit/s. Without network coding a total of only 3 bits can be sent to the two receivers in one unit time, i.e. the average throughput to each receiver is 1.5 bit/s (Figure 1.1(a)). When node W is allowed to encode (Figure 1.1(b)), W combines the two bits a and b it has received using, for example, XOR to produce a new coded bit $a \oplus b$, and after that, sends the coded bit to node X. The result is that, after decoding, each receiver can retrieve 2 bits per one unit time which is the maximum throughput from the source to it.

Widely deployed in practical systems, random linear network coding (RLNC)

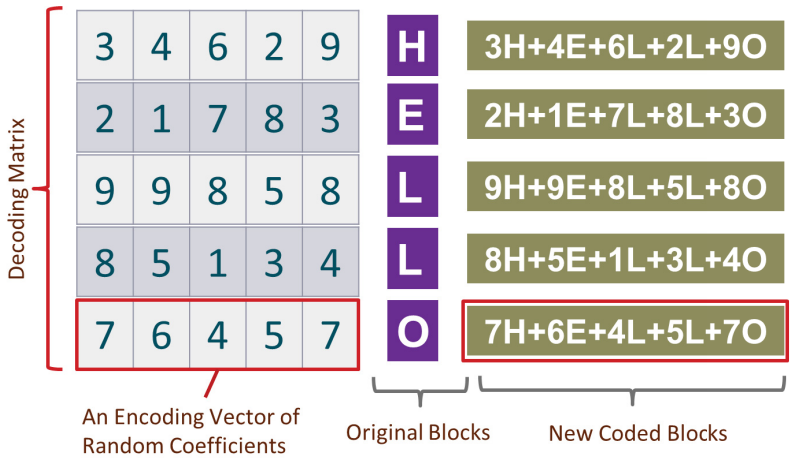

Figure 1.2: Random linear network coding coder creates new encoded blocks from the original blocks using random coefficients. The multiplication and addition are taken place in a finite field, e.g. $GF(2^8)$.

[26, 27] works in a distributed manner under which encoders, independently and randomly, make a linear combination of available data using random coefficients to generate new coded data. RLNC has been deployed with BitTorrent to speed up content distribution in Avalanche system[14, 15, 16, 28].

Avalanche allows all peers to generate new encoded blocks from what they have received, i.e. become network coders, before sending to other peers.[2] If the file consists of K blocks, using RLNC, an *encoding vector* of K coefficients is attached to each data block to specify how that coded block is generated from the K original blocks. Suppose we have a coded block C_0 with encoding vector $(c_{01}, c_{02}, ..., c_{0K})$, and K original blocks, B_1, B_2, ...,B_K. That means $C_0 = c_{01}B_1 + c_{02}B_2 + ... + c_{0K}B_K$. The coefficients, multiplications, and additions are taken place in a finite field, e.g. $GF(2^8)$. Figure 1.2 illustrates 5 new coded blocks are created from 5 original blocks of the file, each original block contains a character from the word "HELLO".

Now suppose encoder i, having received 2 blocks C_1 and C_2, wants to make a new encoded block to send to a neighboring peer (Figure 1.3). The RLNC

[2]We call the peers which are allowed to encode *network coder* or *encoder* to distinguish them from original non-encoding peers.

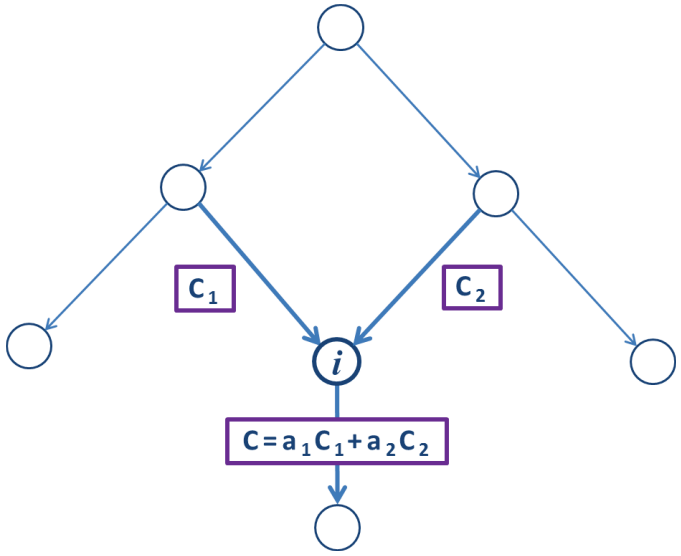

Figure 1.3: Avalanche [14] requires all peers encode. In this figure, peer i is combining the two blocks C_1 and C_2 it has downloaded to make new coded blocks C using random coefficients a_1 and a_2.

encoder i will pick up two random coefficients a_1 and a_2 and generate a new coded block C: $C = a_1C_1 + a_2C_2$, which results in an encoded block with encoding vector $(a_1c_{11} + a_2c_{21}, a_1c_{12} + a_2c_{22}, ..., a_1c_{1K} + a_2c_{2K})$. The coded block C together with K coefficients above is sent to the requesting peer. At the receiving peer, all encoding vectors are stored in a *decoding matrix* with corresponding coded data blocks. After a peer collects K independent coded blocks, i.e. the K associated encoding vectors form a full-rank matrix, it can decode to get the K original blocks by solving the set of K linear equations.

Experimental evidence in [14] confirms that Avalanche has remarkably improved performance compared with ordinary BitTorrent file distribution, especially in clustered topologies where there is limited bandwidth between sets of peers. One interesting observation is that substantial performance gain is evident even when only the source is allowed to code, i.e. *source coding*. Nevertheless, the paper omits concrete explanation for what underlies network codings good performance, and, more interestingly what we can expect if a constrained number of peers are allowed to encode. In [29, 30], source coding is also applied to improve BitTorrent without

incurring encoding at intermediate peers. Those results suggest full-scale network coding where all nodes are required to code might be more than what we need to achieve such performance.

There is, however, a disparity in the understandings of network coding benefit in practical P2P systems. Chiu et al. in [4] give an analysis on star network topologies and find no advantage in applying network coding in the P2P content distribution system. With real experiment results, Legout et al. [3] show that the rarest first algorithm used in BitTorrent guarantees close to ideal diversity of blocks among peers and that replacing rarest first with source coding and network coding cannot be justified. Experimental evidence in [14], on the other hand, confirms that network coding can significantly improve BitTorrent file distribution. Standing in the middle, results in [29, 30] support source coding [31, 32], i.e. coding only at the source, as a method to improve BitTorrent performance.

We go forward to fill the gap by quantitatively identifying conditions which justify the use of network coding in P2P content distribution. Given that insight, we furthermore propose methods to optimize network coding deployment in such environment.

Before going to the main parts of this dissertation, we review related work on optimization for network coding.

1.3 Related Work on Network Coding Optimization

Recently, there are many research efforts to optimize network coding in multicast settings.

Closely related to our work, Kim et al. [33] use an evolutionary approach to determine a minimal set of nodes where coding is required to achieve the maximum multicast rate. Their method, though can manipulate a large space of solutions,

is based on a genetic algorithm, which barely offers any insight into how network coding improves performance. Bhattad et al. [34] decomposed a multicast solution into flows to subsets of receivers and construct a linear programming problem for minimal network coding. Their approach is applicable only in multicast networks with a small number of receivers since the complexity grows exponentially with the number of receivers.

Lun et al. [35] present methods for computing subgraphs over which network coding is deployed. Their primary concern is to minimize the cost associated with bandwidth utilization on network links. Moreover, the model assumes full network coding deployment to achieve maximum multicast rate which is not suitable in case only a subset of network nodes are allowed to code. Recently, Martalo et al. [36] figure network coding complexity, i.e. the minimal number of coding nodes, and its relation to the multicast capacity and the number of receivers in random network topologies. Their evaluation, however, is limited to the case of acyclic networks which is not applicable in P2P overlay networks where circles and loops prevail.

Fragouli et al. [37, 38] partition the network into subgraphs where the same information flows and propose to place encoders at nodes on the borders of the subgraphs where multiple flows merge. The method is hard to applied in P2P content distribution where virtually all peers are intermediate nodes who receive from multiple neighboring peers. Their model is also limited to the case of 2 sources [38] which cannot be applied in P2P content distribution when the file size is larger than 2 blocks.

P2P networks usually consist of a large number of peers and the network topologies inherently contain cycles. Current approaches minimizing the number of coders whose complexity grows exponentially with the number of nodes[34] and which assume direct acyclic graphs [38] are thus impractical in such networks.

Langberg et al. in [39] give upper bounds of the number of encoders needed to

achieve maximum multicast throughput. They, in addition, prove that determining the minimum number of encoders required for maximum multicast throughput is an NP-hard problem.

In another direction, Small and Li [40], Niu and Li [41], and Crisostomo et al. [42] demonstrate that network coding efficiency largely depends on the P2P network topologies. More recently, Maheshwar et al. [43] likewise study network coding in a combination network topology and show that the coding advantage, i.e. improving multicast throughput, and the cost advantage, i.e. reducing multicast cost, are upper-bounded by a constant. Their goal, unlike ours, is to identify network coding performance compared with non-coding in various topology configurations, e.g. by changing the randomness and sparsity of the network [40]. Justifying the benefit of network coding over the whole topology is in itself a rough estimate. Within a topology, there might be some areas which benefit from coding and others which do not. In our study, we instead take a closer look into a given topology to pinpoint locations which need network coding, and then, make use of that knowledge to place coders inside the network. To the best of our knowledge, this is the first work which explicitly identifies conditions under which network coding can speed up data distribution in a detailed scale.

Cleju et al. in [44] propose coder placement algorithms to minimize streaming delay in a push-based, sender-driven overlay network. The intermediate nodes in their system, however, are not interested in the content and only act as helpers to the system and their problem is limited to direct acyclic network topologies. In our system, all peers are receivers who actively select which parts of the content they want to download. We also do not impose any constraint on the topologies which practically are random meshes where one peer connects to others at random.

Maymounkov et al. [31, 45], Champel et al. [46], and Silva et al. [47] devise network coding methods to achieve better computational efficiency. Our solution, nevertheless, will further save computational resources by reducing the number of

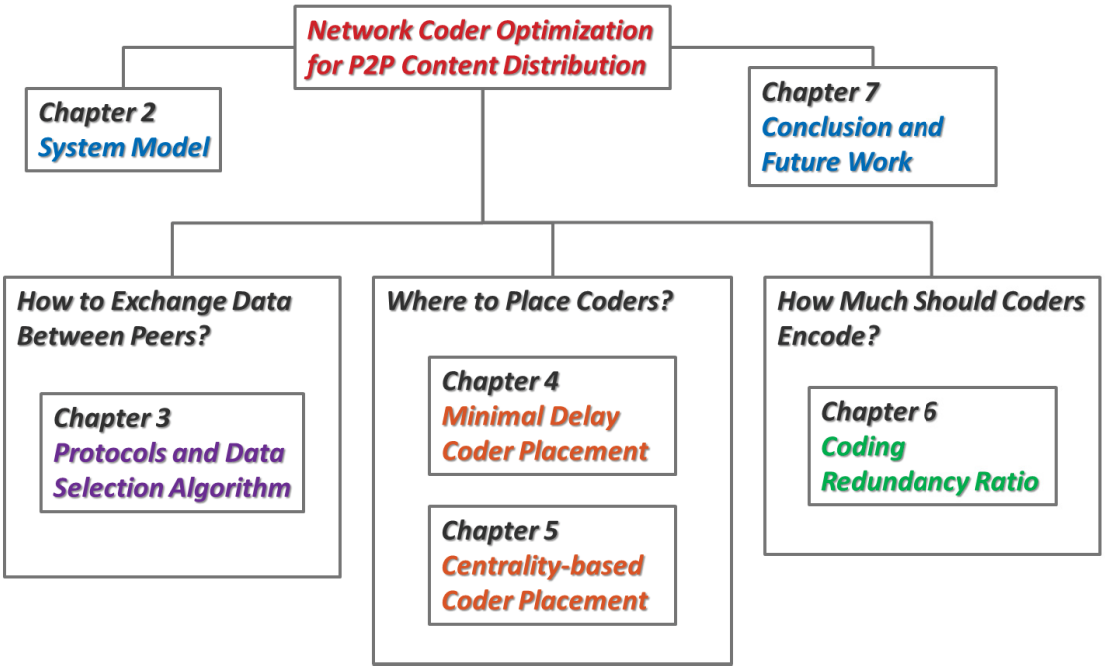

Figure 1.4: Dissertation organization. The main parts of the dissertation are centered around three questions: (1) how to exchange data between peers (Chapter 3), (2) where to place coders (Chapter 4 and Chapter 5), and (3) how much should coders encode (Chapter 6).

encoders and the number of encoding operations at each encoder.

Interesting enough, both the first work on network coding [1] and the first work where network coding was applied to P2P content distribution [14] have briefly illustrated the problem we solve in this dissertation: the multi-path duplication problem inherent to content distribution. We differ from them, yet motivated by them, in that we make a full analysis of where the duplication happens and degrades performance the most in order to place network coders there. In addition, we figure how much a network coder should encode to eliminate such duplication.

1.4 Dissertation Organization

The remaining parts of this dissertation are organized in 6 following chapters (Figure 1.4).

- **Chapter 2** describes our system model with the assumptions we have made

about the system.

- **Chapter 3** proposes the communication protocols and data selection algorithm required for a partly network coding-enabled P2P content distribution system to operate efficiently. We envision a P2P system where some peers encode to improve content distribution time and other peers, due to some system-wide optimization or resource limitation, do not encode. Such a system gives rise to a design problem which has never happened in both pure non-coding and full network coding-enabled P2P systems. We identify the problem and propose our protocols and algorithm to address it.

- **Chapter 4** begins with our network coder placement problem statement which is the main focus of the dissertation. Given a network topology, the question we would like to answer is where to place a given number of network coders inside the network to shorten distribution time the most. We then makes an elaborate analysis of data duplication in ordinary non-coding P2P content distribution in an effort to understand how network coding improves performance. The analysis forms the basis for our network coder placement to reduce number of network coders. We first analyze the duplicated data on a simple 2-receiver graph based on a probability model. The result is then extended to general topologies with multiple receivers. Taking advantages of the analysis result, we propose a novel network coder placement algorithm named *minimal delay* placement. Given a network topology, *minimal delay* placement algorithm places coders at nodes which cause the most *delay* to other nodes due to duplication. Our algorithm effectively eliminates unnecessary network coders, thus, reducing the number of coders with almost no impact on the performance of the system.

- **Chapter 5** presents our centrality-based coder placements. We target placement algorithms with lower complexity and good performance. By observing

the correlation of data duplication in a content distribution network with the characteristics of the delivery paths, we proposed two placement methods. The first exploits *betweenness centrality* [48] characteristics of each network nodes: encoders are placed at high betweenness centrality nodes, i.e. nodes which stand on more shortest paths from the source to other nodes in the network. The second algorithm uses *flow centrality* [49], a variant of betweenness centrality which takes network flows into account, as an indicator to place network coders.

- **Chapter 6** optimizes *redundancy ratio* at each network coder, i.e. how much the coder should encode. We figure the right redundancy level an encoder should generate in order to achieve shortest distribution time based on the network topology. The result is a saving in encoder's computational resources while still realizing good performance. Redundancy ratio optimization, together with our network coder placements, further reduces resource consumption of network coding.

- **Chapter 7** finally concludes the dissertation with an outlook for future work to continue the research we have presented. We discuss the improvements needed to strengthen results of this dissertation and point in one promising direction to extend the results of our study to the recent information-centric networking research trend.

Chapter 2

System Model

2.1 Network Coding Peer-to-Peer Content Distribution

We consider a peer-to-peer content distribution problem from one source to many peers where each peer maintains overlay links to some other peers at random, i.e. its *neighbors*, over which data are transferred (Figure 2.1).

A file exists at a single source and is distributed to all peers which, at the beginning, do not have any part of the file. The file is divided into K equal *blocks*[1], the same as in [5], which are exchanged between neighboring peers over the overlay links connecting them. Peers may download several blocks in parallel from many neighbors at the same time. Parallel download accelerates the downloading speed of each peer. A peer not only downloads blocks from its neighbors but also uploads blocks it has to them, contributing its own upload bandwidth resources. As soon as a peer finishes downloading a block, it can immediately act as a *server* of that block and upload the block to its neighbors if there are requests for it. A peer finishes when it has collected all blocks of the file.

[1]Other BitTorrent-like systems might use the terms *chunk* or *piece*. In this dissertation, we use *blocks* to mean equal parts of the file.

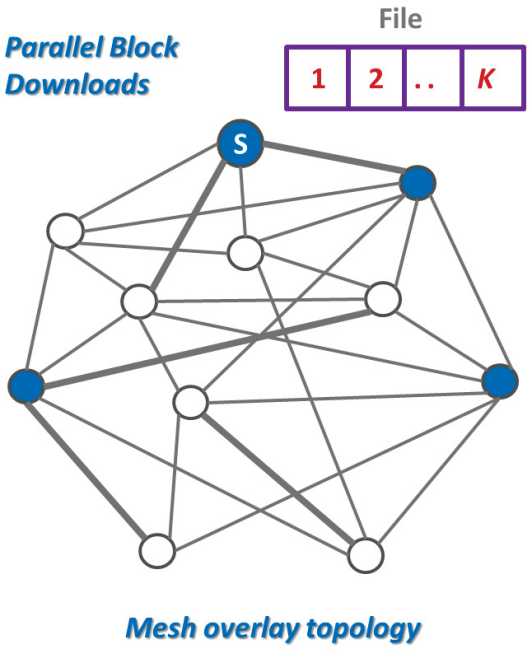

Figure 2.1: A file is distributed from the source S to all peers over a mesh overlay network. The file is divided into K equal blocks which are downloaded in parallel by peers. Network coders are placed at solid nodes to accelerate content delivery.

As in BitTorrent systems [5, 14], block exchange between peers complies with two rules:

1. rarest-first block selection at the receiving peer: a receiving peer chooses rarest blocks within its *neighborhood* to download first, and

2. an incentive scheme at the sending peer: a sending peer uploads blocks reciprocally to the neighboring peers who are also sending data to it.

Rarest-first block selection at the receiving peers works as follows. Peers keep collecting information about which blocks have been downloaded by its neighbors and how many neighbors possess a given block. Based on that information, a peer can decide which blocks are the rarest in the neighborhood. Whenever a peer has available bandwidth to download, it chooses a number of rarest blocks within its neighborhood and requests each block from the corresponding neighbor.[2] Depend-

[2]At the beginning of a content distribution session, since there are not many blocks available

ing on its available upload bandwidth and the incentive scheme, the neighbor will permit the download or not. If a peer fails to request a block from a neighbor, it tries with other neighbors who also have the block it interests in. If that also fails, the peer will pick up the next rarest block as a substitute.

The incentive scheme at the sending peer is to ensure fairness in the system: a peer not only downloads, i.e. consumes resource of other peers, but also uploads, i.e. contributes it own resources. In our system, a sending peer prefers to upload to neighboring peers who are also sending data to it. If the peer still has available bandwidth resources, it will also upload to other neighbors who are not sending data to it. That kind of *mutual exchange* incentive scheme has previously used in [14].

We assume an altruistic system where peers stay and forward blocks even after they have finished downloading. The source does also stay in the system until all peers finish.

When network coding is enabled at a peer, the peer, which we then call *network coder* or *encoder*, generates new encoded blocks from what it has received before sending to other peers. Encoders in our system uses random linear network coding [26] described in Section 1.2 to generate new coded blocks to send to its neighboring peers. A peer finishes when it collects enough coded and/or original blocks for decoding. When there are network coders deployed in the system, all peers which have received coded blocks, however, are required to decode to recover the original file.

among peers, in stead of using rarest-first selection, a peer can choose random blocks in the neighborhood to download.

2.2 Network Topology

Peers in the P2P network form a directed overlay topology, i.e. directed graph $G = \{V, E\}$ where V is the set of peers, or nodes, and E is the set of directed overlay links between peers. A path, without circles or loops, from node i to node j is a sequence of nodes starting from i and terminating at j in which two adjacent nodes are connected by a link.

A flow on a path from node i to node j is a mapping: $E \rightarrow \mathbf{R}^+$ which conforms to the two following constraints.

1. Capacity constraint: the flow along a link is not greater than the link capacity.

2. Flow conservation: the total flow coming to a node is equal to the total flow going out of a node except for the source (node i) and the sink (node j).

The value of a flow represents the total amount of flow passing from the source to the sink. A maximum flow or *maxflow* is a flow with maximum value.

Since our main target is to isolate the feature of network coding that makes good performance, we make two following assumptions.

1. We assume complete knowledge of the overlay topology and bandwidth capacity of each overlay link.

2. We assume a static scenario, i.e. there is no change in both the physical topology and the overlay topology during a content distribution session.

Those assumptions allow us to capture the essence of network coding for shortening distribution time. The insight obtained from this static, centralized case is critically important for future work which investigates the dynamic and distributed scenarios.

Chapter 3

Protocols and Data Selection Algorithm

Network coding [1, 2], which allows content to be coded at intermediate nodes while being forwarded in the network, has been shown to achieve significantly shorter distribution time in peer-to-peer (P2P) content distribution [14, 15, 16]. It is, however, too expensive and in many cases impossible to require encoding at every peer. Recent work has demonstrated that encoding is only needed at a subset of carefully chosen peers [44, 50, 51], and in some particular instances, only at the source [29, 30], to achieve comparable performance to network coding. Many other studies have focused on minimizing the number of required network coders to achieve optimal multicast throughput [33, 34, 39].

P2P networks in reality, on the other hand, usually consist of heterogeneous peers with quite different capabilities. More powerful peers can be ready for network coding-enabled operations, yet such jobs are beyond the capacity of resource-limited peers like hand-held and mobile devices. A successful network coding solution to optimize P2P network performance, therefore, cannot impose encoding at every network node.

Interested in using network coding to shorten distribution time in P2P network,

we envision a P2P system where encoding is applied at some peers while other peers, due to resource limitation or due to optimization reasons, might not code. The system, which we call a *hybrid network coding* P2P system, gives rise to a design problem which has never happened before. In pure BitTorrent P2P system [5], the source and all peers exchange pieces, i.e. blocks, of the file using rarest-block selection to quickly disseminate the file into the system. A peer chooses the rarest blocks in the neighborhood to download first. In full network coding-enabled P2P, all peers code. Before downloading from a neighbor, a peer communicates with the neighbor to determine if it can provide with new data. In the hybrid network coding system, when some peers encode and others do not, there are mixtures of coded and non-coded blocks in the neighborhood for each peer to choose from.

The questions are how to communicate in an environment where coders and non-coding peers coexist, and which data should a peer select from such a mixture of coded and non-coded data given that we would like to preserve the efficiency and simplicity of BitTorrent P2P system.

In this chapter, we design our hybrid network coding P2P system.

1. We devise information exchange protocols which allow peers in a hybrid network coding system to communicate seamlessly with its neighboring peers whether they are coding-enabled or non-encoding ones. Our design, backward-compatible to BitTorrent, requires only an addition of one field in the meta-exchange messages.

2. We propose a block-selection algorithm for the partly network encoding-enabled system to operate efficiently. Our block-selection algorithm, an extension from BitTorrent's rarest-first selection, is derived from extensive observations of the way network coded data benefit content distribution.

Our design and algorithm noticeably improve system performance in terms of distribution time compared with current network coding P2P systems.

3.1 Ordinary Network Coding Peer-to-Peer System

Network coding [1, 2, 26], which allows intermediate nodes to encode, have been applied to BitTorrent in order to shorten distribution time [14, 16]. Whenever there is an opportunity to transmit, a peer combines all blocks it has to make new coded blocks and sends to the requesting peer.

For full-scale network coding P2P where all peers encode, [14, 52] proposes a mechanism by which before downloading from a neighbor, a peer checks if the neighbor can provide it with meaningful blocks, i.e. blocks which are linearly independent from the set of blocks it has received. We call that a *try-and-download* approach which, compared to BitTorrent, requires a major update in the way peers exchange metadata (Figure 3.1):

1. a peer sends a request message to its neighbor,
2. the neighbor replies either with a newly generated *encoding vector* or with its *decoding matrix*,[1] and
3. requesting peer downloads a newly coded block from the neighbor if the encoding vector or the neighbors decoding matrix is independent from its own decoding matrix.

Try-and-download is synchronous in the sense that a peer has to be in synch with its neighbors by continuously checking if they can provide it with new data. Moreover, a receiving peer cannot know in advance exactly which and how many blocks it is going to receive from each neighbor to make a better choice. Such knowledge will help the receiving peers to decide which blocks are most valuable to it. In full-scale network coding systems where peers are somehow homogeneous

[1] Please refer to Section 1.2 for an explanation of *encoding vector* and *decoding matrix*.

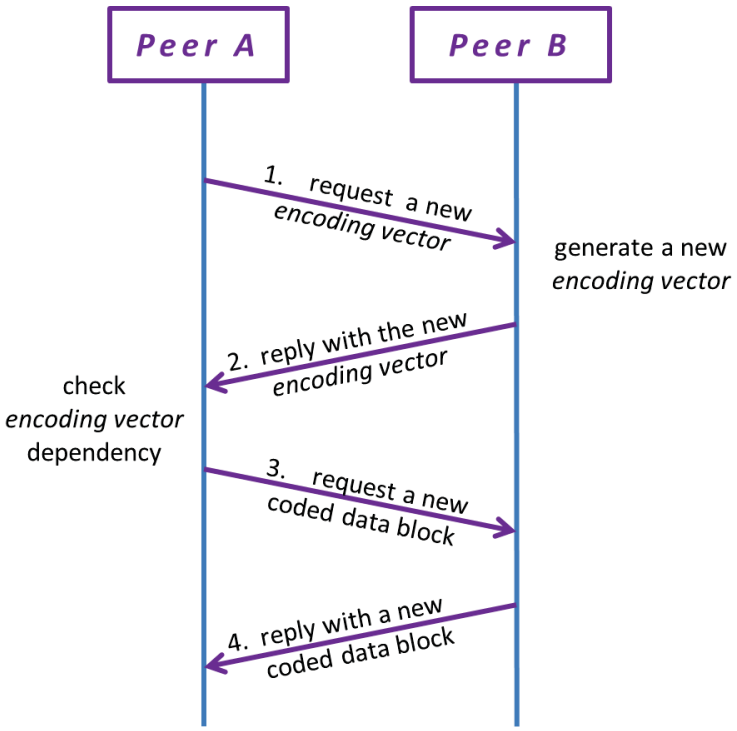

Figure 3.1: In current network coding P2P systems [14, 52] requests and replies are synchronous. The requesting peer regularly checks with each of its neighbors if it can download new independent blocks from them. If there are such blocks, the peer then requests and downloads them from the corresponding neighbors.

in terms of computational resources to encode, *try-and-download* is feasible, yet with a protocol overhead. Within a hybrid network coding P2P system, however, it is not necessarily that all peers can code. In such a scenario, requiring a resource-limited peer to frequently compare its own decoding matrix with decoding matrices of its neighbors is beyond its capacity. We need a simple, yet effective, way to do that which every peer, encoding-enabled or not, can do. To facilitate hybrid network coding P2P systems where encoding and non-encoding nodes mix together, we depart from *try-and-download* approach to introduce an extension to BitTorrent metadata exchange. We furthermore propose a block selection algorithm to improve distribution time. Our proposed solution is backward-compatible with BitTorrent and virtually requires no more protocol overhead than pure BitTorrent, yet the performance improvement is noticeable compared with original network coding P2P systems.

3.2 Proposed Information Exchange Protocol

In pure BitTorrent without network coding, there are two phases to distribute blocks. These two phases interlace and take place asynchronously.

- *Notification phase*: after downloading a block, the downloading peer notifies its neighbors about the block it has just downloaded.

- *Selection phase*: whenever bandwidth is available for downloading, a peer, based on the information it has about which blocks are available in the neighborhood, chooses one block to download using a block selection algorithm. The download, then, can proceed if the downloading peer is currently *unchocked* by its neighbor who has the chosen block and the neighbor has enough bandwidth to sustain such download. If that fails, the peer can repeat this process to choose another block. This phase stops when the peer runs out of bandwidth or has no more blocks to choose from.

In the following subsections, we concentrate on the protocols used to communicate between peers and the format of the exchanged metadata. We discuss the block selection algorithm in detail in the next section. BitTorrent unchocking algorithm is one topic in itself to handle fairness and free-rider issues and is not discussed in this section. We instead assume peers in our system are altruistic and willing to contribute their bandwidth.

3.2.1 Block Format

To identify data blocks, each block is associated with one unique *block-id*. However, one extension is needed to support network coding. Unlike non-coding systems in which the assignment is done only by the source where all the blocks originate, in network coding P2P systems, that assignment is done where the block is created or originated: both at the source and at all the encoders. To assist our block selection algorithm, in one content distribution session, the *block-id* is generated

in increasing order: a new *block-id* generated by a particular encoder is greater than all previous *block-ids* generated by that encoder.

With network coding, an *encoding vector* is attached to each coded block as described in Section 1.2. We propose an additional *encoder-id* field (Figure 3.2) which stores the identification of the encoder who generated the coded block. *Encoder-id* will be used in our block selection algorithm later on.

For each block, the metadata exchanged between neighbors in a notification message, thus, consists of three fields: *block-id*, its *encoder-id*, and its *encoding vector* (Figure 3.2(a)). The data block consists of *block-id*, *encoder-id*, and the data payload (Figure 3.2(b)). If the notification or data block is a non-coded one, its *encoding vector* and *encoder-id* can be omitted.

Having defined the block formats, we next present details of two communication protocols, either of which can be used in the hybrid network coding system.

- *Pre-code protocol*: encoding vectors of coded blocks are generated in the notification phase when encoders notify their neighbor about newly coded blocks.

- *Post-code protocol*: encoding vector for a given coded block is generated in the selection phase, just before the block is downloaded.

We discuss the pros and cons of those two protocols subsequently.

3.2.2 Pre-code Protocol

Without the assumption that every peer can code, we propose a simple adaptation to BitTorrent metadata exchange mechanism. To facilitate coding, in our system, if a peer is an encoder, for each newly downloaded block, the peer notifies each of its neighbors with metadata of one newly encoded block. The newly encoded block is different from one neighbor to another neighbor. We note that to save computational resources, only the metadata, i.e. *encoder-id*, *block-id*, and the newly

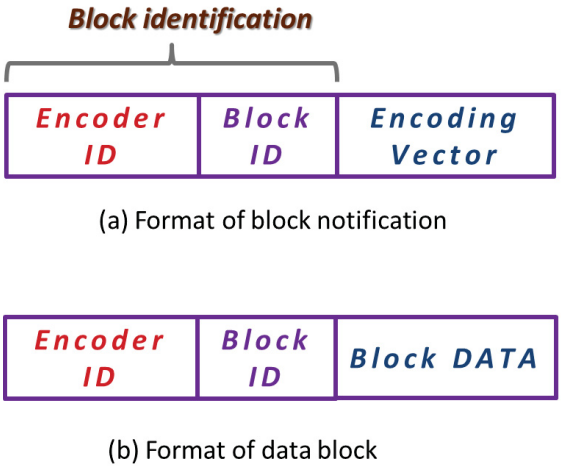

Figure 3.2: Notification and data block formats with the newly proposed *encoder-id* field.

generated *encoding vector* of the encoded block (Figure 3.2(a)), are notified to the neighbors in a notification message. Only when a neighbor decides to choose and request the notified coded block is the actual data of that block encoded. For an ordinary non-encoding peer, the metadata exchange is the same as in BitTorrent: the peer notifies its neighbors of the block it has just received. The communication protocol is illustrated in Figure 3.3. Since the system is a hybrid network coding, notifications (*message 1*) and data blocks (*message 3*) transferred between peers can be either encoded or original ones.

One might argue to use *try-and-download* here, but that will make the operation more complicated because each peer has to implement two protocols: one for encoding-enabled neighbors, one for ordinary neighbors. With our approach, all a peer has to do is to choose from candidate blocks one particular block to download based on the metadata it received in notification phase, which is the same as what happens in a pure BitTorrent system.

When a peer receives notification of a newly encoded block by a neighbor, i.e. *message 1* in Figure 3.3, the peer stores that block in a candidate list if the block is independent from all blocks it has downloaded. Otherwise, it ignores the notification. Unlike encoding-enabled peers, non-encoding peers do not encode but

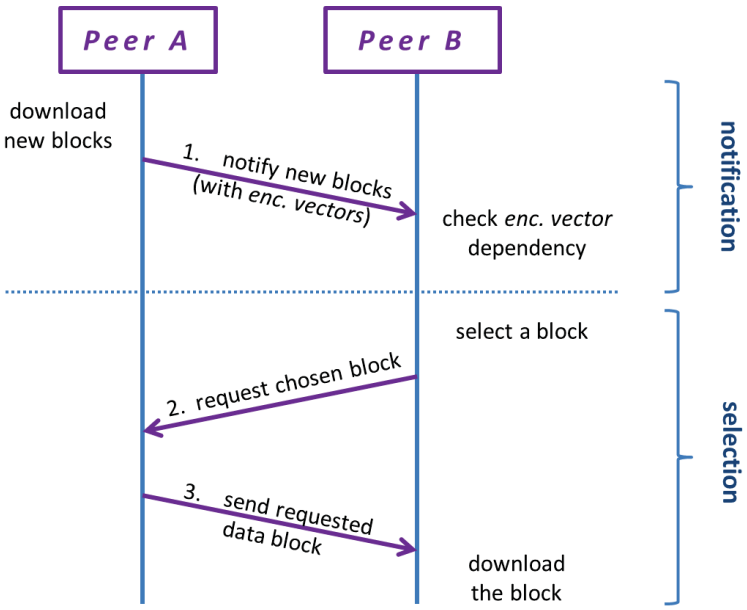

Figure 3.3: *Pre-code* protocol peers used to communicate. There are two asynchronous phases: notification phase and selection phase. This protocol is an extension from BitTorrent: the notification messages and data blocks have an additional *encoder-id*. Encoding vectors are also attached to the notification messages as described in Section 1.2.

forward what they have received: a mixture of coded and non-coded blocks. As in BitTorrent, when receiving notification from a non-encoding neighbor, a peer will update the count of that block, i.e. at how many neighbors the block exists.

When a peer can download, it selects a block using a selection algorithm and sends a request for the chosen block to the corresponding neighbor (*message 2*). If the request is accepted, the neighbor will upload the data block to the requesting peer (*message 3*).

Coding generates a large number of coded blocks, usually larger than the number of original blocks, of which many blocks are redundant. As a peer continuously downloads new blocks, some blocks in its candidate list might become dependent on what it has downloaded. Each peer is therefore required to check and discard candidate blocks which are dependent on what has been downloaded.

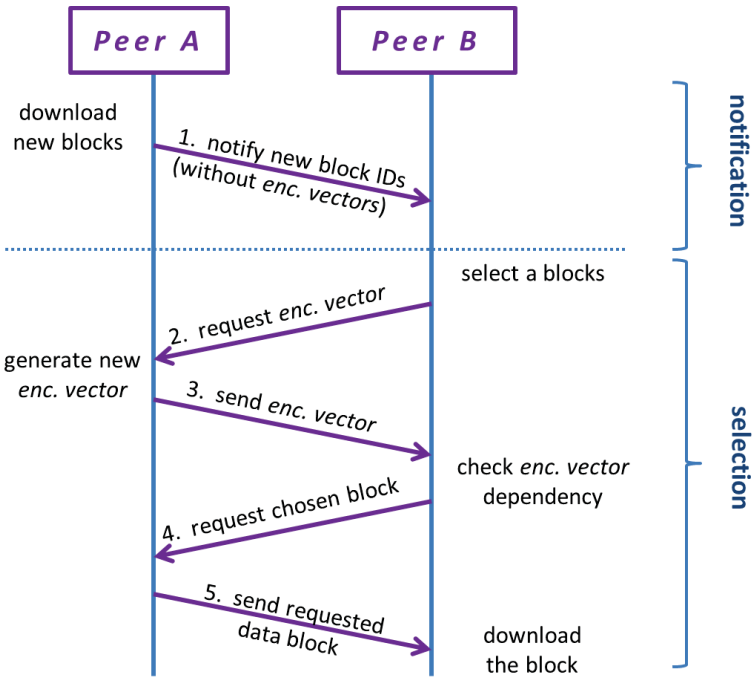

Figure 3.4: Using *post-code* protocol, the encoding vector is generated not in the notification phase but just before the requested block is sent to the receiving peer.

3.2.3 Post-code Protocol

As we mentioned before, notification phase and selection phase are asynchronous. That is, after peer A notifies an encoded block in *message 1*, some amount of time passes before peer B requests that encoded block in *message 2*. The elapsed time can arbitrarily be long if, for example, peer B decides to download several blocks from other neighbors before choosing the encoded block from peer A. In the meantime, peer A might receive some new blocks. Using pre-code protocol, that new information is not included in the encoded block since the way the block is generated, i.e. its encoding vector, was fixed at the notification time.

Encoders combine the blocks they currently have to make new coded blocks. If we can delay the act of encoding just before the coded blocks are downloaded, we can provide the receiving peers with the most updated information. Based on the above observation, we proposed an alternative protocol, namely *post-code* protocol, which is illustrated in Figure 3.4.

The differences of the post-code protocol from pre-code protocol are as follows.

- *Encoding vector* is not included in the notification message, i.e. *message 1* in Figure 3.4. Only *encoder-id* and *block-id* are notified to the neighbor (peer B) each time peer A downloads a new block. As stated before, *encoder-id* is the ID of peer A and *block-id* is an increasing number generated by peer A.

- The encoder (peer A) actually generates the *encoding vector* and sends to the receiving peer in the selection phase (*message 3*) just before the actual coded data (*message 5*). The receiving peer (peer B) needs to check if that *encoding vector* is independent from its own *decoding matrix* before requesting the encoded block (*message 4*).

Post-code protocol has the advantage of producing fresher coded blocks which expectedly accelerate content distribution. The limitation, however, is that it requires more protocol overhead: in total 5 messages for each downloaded block compared with 3 messages in case of pre-code protocol.

3.3 Block Selection Problem

In this section, we describe in detail the block selection problem associated with hybrid network coding systems and propose our solution for it. The proposed block selection, which can be used with either of the two protocols we present in the last section, completes our proposal for an efficient, high-performance P2P content distribution with network coding. We begin by describing the duplication problem in such a system using the original rarest-first block selection.

3.3.1 Duplication Problem with Current Rarest-first Block Selection

The block selection algorithm used by BitTorrent is rarest-first by which peers choose the rarest block in the neighborhood to download first.[2] If there are several rarest blocks, a random one is selected from those rarest blocks. Rarest-first selection is not enough because of two reasons.

1. Encoders combine more information in the neighborhood. When there is limited available bandwidth, for example when a bottleneck exists, non-coded blocks and coded blocks cannot be given the same attention. Coded blocks from the encoders should be preferred because they contain, in a sense, more information and can accelerate content distribution through the bottleneck.

2. Coded blocks are not equally important. Each coded block even though is always unique, i.e. rare, in the sense that almost always no two coded blocks are identical, the level of importance of each coded block is different. Coded blocks are created progressively from all the blocks an encoder has downloaded. In the beginning, as there are only a few blocks to encode, the coded blocks created then contain within them only the information from that few blocks. The more blocks an encoder has, the more data are combined to create new coded blocks. Because of that, only at the source or when an encoder has downloaded the full file, are the coded blocks equally important. In other cases, the most recently coded blocks likely contain more information.

To make it clear, we illustrate the problem in two following examples.

[2]In the beginning of the distribution section when peers have no blocks to exchange with others, BitTorrent uses random block selection by which peers choose a random block in the neighborhood to download. Nevertheless, after a peer has acquired some blocks, it switches to rarest block selection.

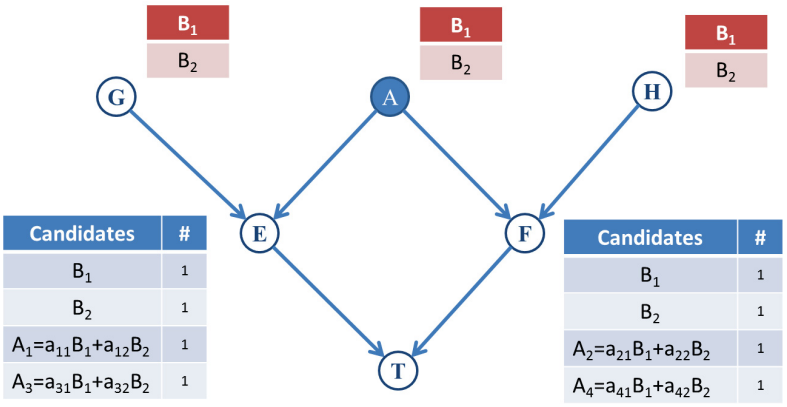

Figure 3.5: Node E and F receive notification from node A, G, and H about the candidate blocks the two nodes can down-load. Of which B_1 and B_2 are non-coded blocks from node G and H; A_1–A_4 are newly encoded blocks from encoder A.

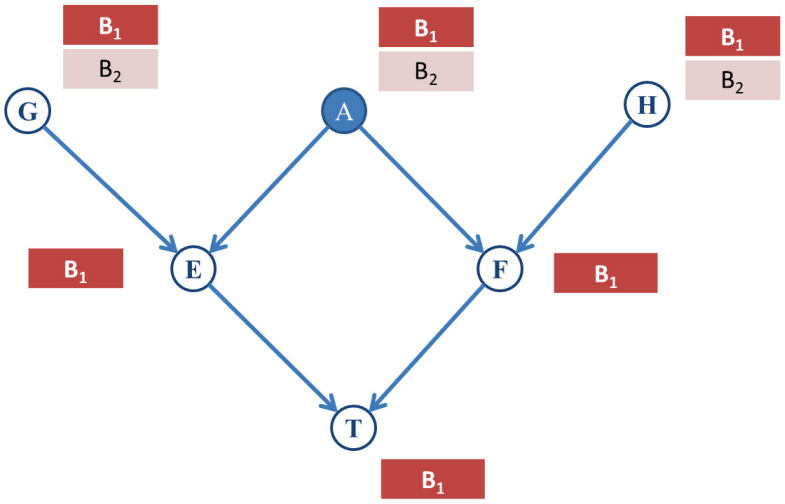

Figure 3.6: With original rarest-first selection, there is a probability 1/8 that node E and F choose the same block B_1 or B_2. The result is that node T can only download one new block while its bandwidth allows two blocks.

Example 1 (Figure 3.5–3.7) illustrates a partial overlay topology with 6 nodes: A, G, H, E, F, T of which A is the only encoder. Nodes A, G, H each has two blocks B_1 and B_2. Encoder A has notified node E, F with 4 blocks A_1–A_4, each node with two newly coded blocks. Nodes G, H have notified E, F with blocks B_1 and B_2. The count of each block in the neighborhood is given in the tables (Figure 3.5). Suppose due to bottlenecks, E and F, each can only download one new block. If E and F select blocks using original rarest-first algorithm, there is 1/8 chance that both will download the same block B_1 or B_2 which results in

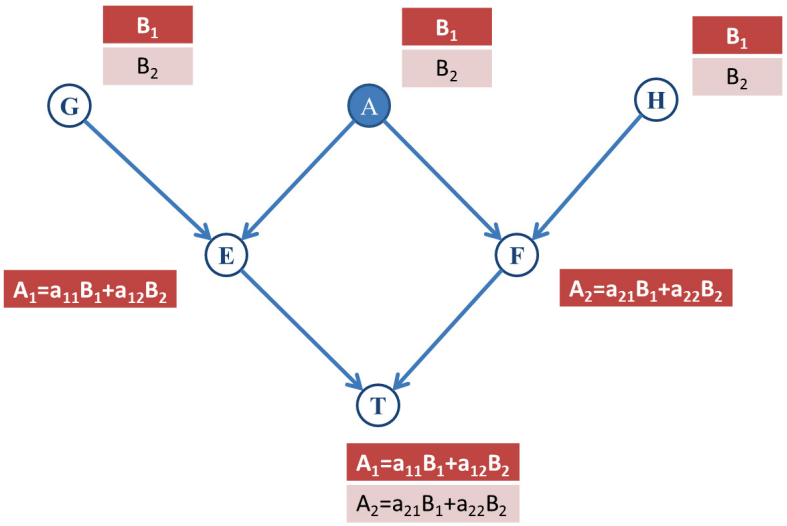

Figure 3.7: If coded blocks from encoder A are preferred, node E and F can always download independent blocks. As a result, node T can utilize all its bandwidth to download 2 new independent blocks.

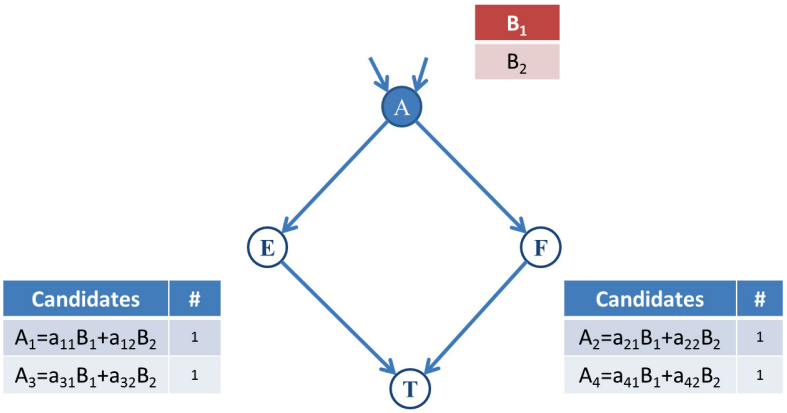

Figure 3.8: Encoder A, having 2 blocks B_1 and B_2, notifies node E and node F with newly encoded blocks A_1–A_4.

node T can only download one new block while its available bandwidth allows two (Figure 3.6). In Figure 3.7, if E and F prefer coded blocks from encoder A over other blocks, T can always download two new blocks.

Example 2 (Figure 3.8–3.11) considers a partial overlay topology in which an encoder A is delivering coded blocks to non-coding nodes E, F, and T. At the beginning, A has two blocks B_1 and B_2, and notifies E and F of 4 newly encoded blocks: A_1–A_4, two blocks for each node (Figure 3.8). Node E and node F then each can download one block, e.g. A_1 and A_2 due to bandwidth limit. In the

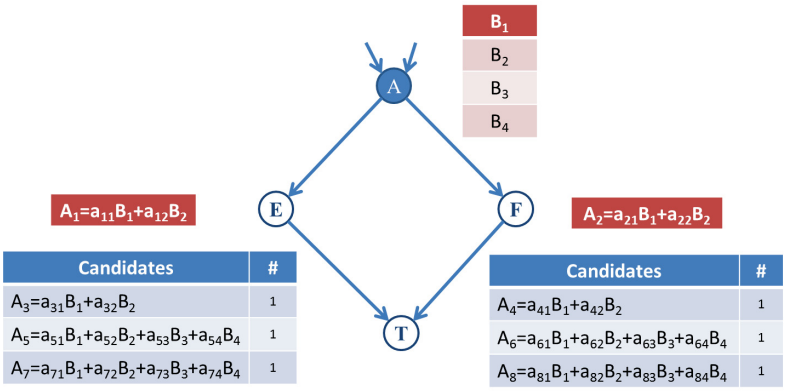

Figure 3.9: Encoder A, after downloading 2 new blocks B_3 and B_4, notifies node E and node F with blocks A_5–A_8 encoded from all 4 blocks B_1–B_4.

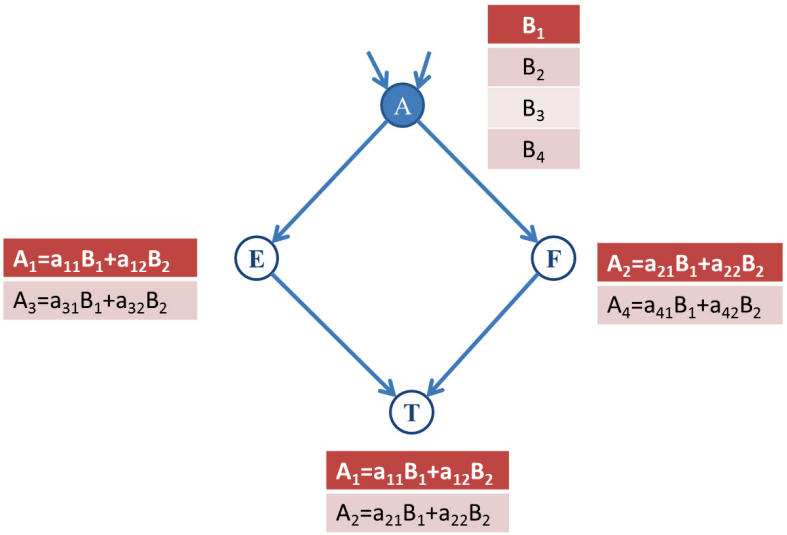

Figure 3.10: With original rarest-first selection, there is a probability 1/9 that node E chooses block A_3 and node F chooses block A_4. The result is node T can only download 2 independent blocks A_1 and A_2 in 2 units of time. Blocks A_3 and A_4 in node E and node F are not useful to node T because they are dependent on A_1 and A_2.

meantime, A downloads two more blocks: B_3 and B_4, and sends new notifications about blocks A_5–A_8 to E and F (Figure 3.9). If E and F select blocks using rarest-first, there is 1/9 chance that E chooses A_3 and F chooses A_4 which results in peer T being only able to obtain 2 independent blocks in 2 units of time (Figure 3.10). In contrast, T can download 4 new blocks if E and F prefer new encoded blocks over old ones (Figure 3.11).

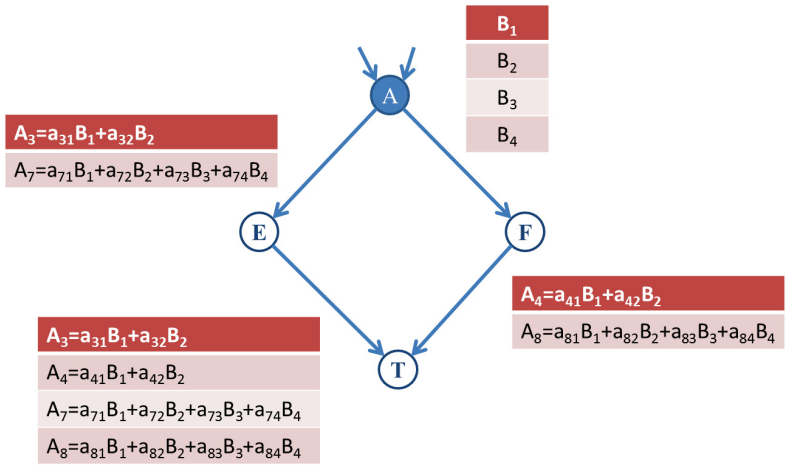

Figure 3.11: If the newest blocks are preferred, the 4 blocks downloaded by node E and node F are independent, which means node T can download in total 4 independent blocks in 2 units of time.

Algorithm 3.1: Proposed Block Selection Algorithm

Input: Set C of candidate blocks and function $count(C_i) \quad \forall C_i \in C$ specifying how many times each block exists in the neighborhood of peer A.
Result: One block for peer A to download

1 Sort blocks in ascending order of their occurrence $count(\cdot)$;
2 Make a list L_1 of all blocks C_i with the lowest $count(C_i)$;
3 Make a list L_2 of all blocks $C_i \in L_1$ which are encoded by a neighbor of peer A in random order: $C_i.encoder\_id \equiv$ encoder B's ID $\forall B, B$ is a neighbor of A;
4 Exchange blocks in L_2 so that blocks from the same encoder are in descending order of their $block\_id$;
5 Make a list L_3 of all blocks $C_i \in L_1 \backslash L_2$ in random order;
6 **if** $L_2 \neq \emptyset$ **then**
7 | **return** the first block in L_2;
8 **end**
9 **else**
10 | **return** the first block in L_3;
11 **end**

The problem therefore is: given a mixture of coded and non-coded blocks in the neighborhood, which blocks should a peer choose to download.

3.3.2 Proposed Block Selection Algorithm

Our proposed algorithm is given in Algorithm 3.1. It works seamlessly in all types of networks: pure non-coding, full-scale network coding, and hybrid network coding. We extend the original rarest-first selection (line 2) to give preference to

coded blocks from immediate neighbors over other ones (line 3). Also, from the same encoding neighbor, newer coded blocks (with larger *block-id*) are preferred over older ones (line 4). In doing so, we allow valuable newly encoded blocks in the neighborhood to be quickly disseminated while preserving the power of rarest-first in distributing new information. Our algorithm improvement is generally significant. Without it, newly coded blocks, virtually with more information, are arbitrarily blocked in the network because neighboring peers may choose not to download them.

3.4 Network Coder Assignment

Given that in the hybrid network coding P2P content distribution, only some peers encode, the questions are which peers will become network coder and who is responsible for assigning them.

In our view, peers at key locations of the network can selectively be assigned as network coders as we have discussed in detail in Chapter 4 and Chapter 5. This approach, however, requires a centralized server to compute and assign coders. Practically, in P2P systems such as BitTorrent [5, 14, 53], we can allow trackers to do that task since the trackers know which peers currently join the torrents.

Network coders can also be assigned in a distributed manner without any centralized server by using, for example, degree information [54]. Given a threshold, peers with degrees higher than the given value will become encoders. Degree-based placement performance, however, is not as good as the proposed placement methods in Chapter 4 and Chapter 5.

In scenarios where computational resources are limited, we can approximately predict the amount of required resources based on which a peer can determine, by itself, to become an encoder if it meets the resource requirements. Such an encoder assignment does not need a centralized server either.

3.5 Performance Evaluation

We implemented a C++ simulator of the hybrid network coding P2P content distribution system. We evaluate the proposed block-selection algorithm in Section 3.3 using either pre-code or post-code protocol in Section 3.2 and compare the performance with a baseline network coding BitTorrent system. The baseline system uses BitTorrent's original rarest first block selection and the pre-code protocol.

A file is distributed from the source to all participating peers, among which a preset number of peers are allowed to encode. The file is divided into smaller fix-sized parts, i.e. blocks. The source and all peers exchange blocks until all peers acquire enough blocks to construct the original file; then the simulation finishes.

The simulations are round-based. Each peer chooses blocks to download according to its available bandwidth, rarest block first selection, and the incentive scheme in the beginning of each round. The chosen blocks are downloaded by the peer at the end of the round and then the system moves to next round. After a peer has collected enough blocks, it stops downloading but keeps staying in the system to serve other peers. A link capacity is measured by block per round, i.e. how many blocks can be transferred through the link in a round. Network coding operations are carried out in finite field $GF(2^8)$. We disregard the negligible overhead of sending encoding coefficients associated with random linear coding in our simulations.

We implemented mutual exchange incentive scheme in the simulations: when there is contention for uploading, a sending peer preferably uploads to the neighbors from whom it is also downloading. After such peers are exhausted, other neighbors are chosen for upload. This kind of incentive schemes has previously been used in [14].

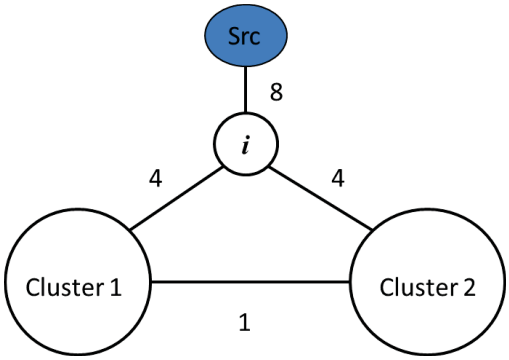

Figure 3.12: A two-cluster topology with a middle node i.

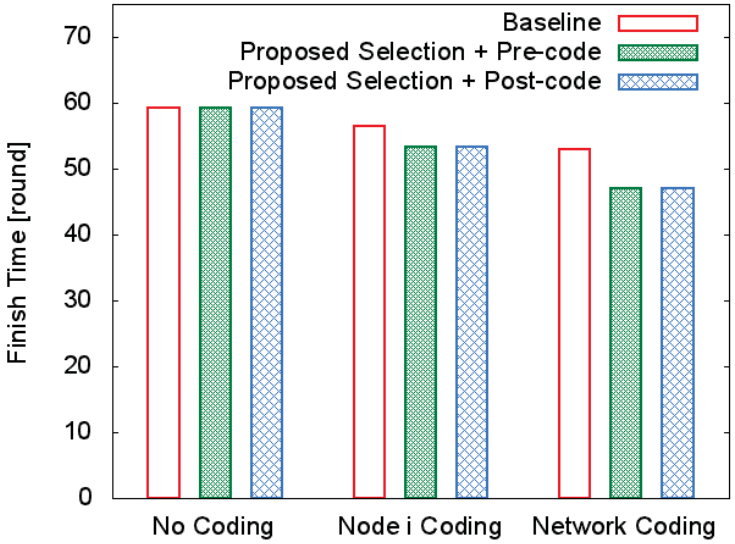

Figure 3.13: Average finish time of the proposed system compared with baseline system in a clustered topology.

3.5.1 Clustered Topologies

We first evaluate performance in a simple topology of two clusters (Figure 3.12). A middle node i intercepts between the source and the clusters to simulate a situation where blocks are coming progressively to node i. Within a cluster, peers are arranged in k-regular random topologies where k is from 3 to 6. Each cluster has 1000 nodes with 1 block per round bandwidth between neighbors within a cluster. Source bandwidth to node i is 8 blocks per round and from node i to each cluster is 4 blocks per round. The two clusters are connected by a link with a capacity of 1 block/round. The source delivers a 200-block file to all peers.

Figure 3.13 compares the finish time of our system using the proposed block selection algorithm (with either pre-code or post-code protocol) and the finish time of the baseline system in three cases: *no coding*, coding at node i, and *network coding*. As we expect, *no coding* finish time is the same for both systems. Finish time improvement of the proposed system becomes evident when node i codes (around 5%) and when all nodes code (around 10%). In this topology, the finish time is the same for both pre-code and post-code protocols.

3.5.2 Small-world Network Topologies

We use Watts and Strogatz *small-world network* model [55] to generate more complex topologies for simulations. The reason is twofold. First, several real-life networks, including P2P overlays, have been reported to exhibit properties of small-world networks [56, 57]. Second, small-world model has a parameter to tune the severity of bottleneck links as explained below.

Nodes in a small-world network are, at the beginning, organized in a ring lattice, each node connects to a predetermined number of nearby nodes, i.e. degree d. The links between nodes are then rewired with some probability p_{rw}, i.e. one endpoint of the (randomly chosen) link is rewired to a new random node, to create shortcuts. With a large rewiring probability, e.g. p_{rw} approaches 1, the network becomes a random one. Small rewiring probabilities result in topologies where peers are highly clustered (due to the original ring lattice) and, despite of that, the average length of shortest paths between peers is low (due to the shortcuts), which means content from one peer can easily reach other peers. When p_{rw}=0 or extremely small, the topology is a regular ring lattice where the average length of paths between peers is relatively long compared with larger rewiring probabilities.

Shortcuts connect different parts of the networks where quite different collections of data blocks exist. Those shortcuts, to a certain extent, equivalent to bottleneck links because the total flow of nearby regular links is bigger than the

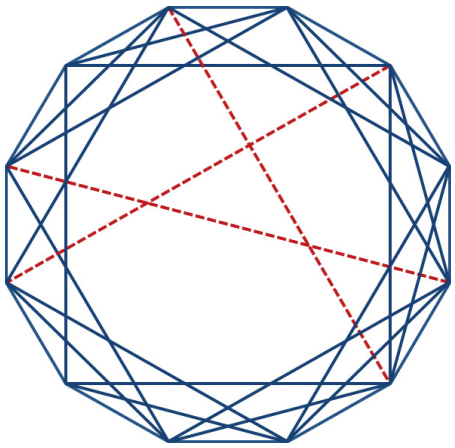

Figure 3.14: A *small-world network* topology with 12 nodes, degree d=6, and rewiring probability p_{rw}=0.05. Random links on the original ring lattice are rewired to new random end-points, which results in the dashed shortcuts.

capacity of the single shortcut link. By adjusting the rewiring probability, we can tune the severity of the bottlenecks. Lower rewiring probabilities mean fewer shortcuts, which in turn mean the bottlenecks are more severe because there are fewer shortcut links for transferring data between different parts of the network.

Figure 3.14 illustrates a small-world network with 12 nodes, degree d=6, and rewiring probability p_{rw}=0.05.

In our simulations, we set overlay link capacity to 1 block per round and change the rewiring probability and degree.[3]

We simulate two real-life scenarios.

1. **Optimization scenario**: encoders are placed at selected peers to minimized distribution time. We use two placement methods:

    - *betweenness centrality* placement: nodes with high betweenness centrality values [48] are chosen as encoders. (We discuss in detail betweenness centrality placement in Section 5.3.)
    - *degree-based* placement: encoders are placed at nodes with high degrees first.

[3]We observe the same results with heterogeneous link capacity.

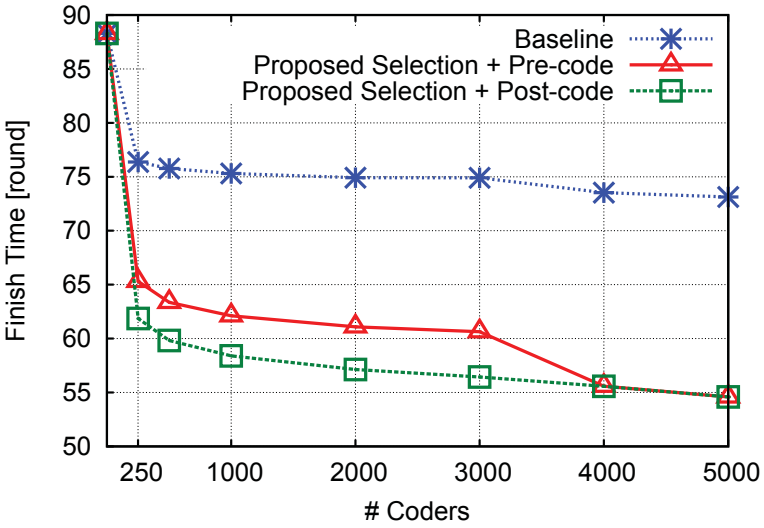

Figure 3.15: Finish time of the proposed system when choosing nodes with highest betweenness centrality as encoders compared with finish time of baseline system.

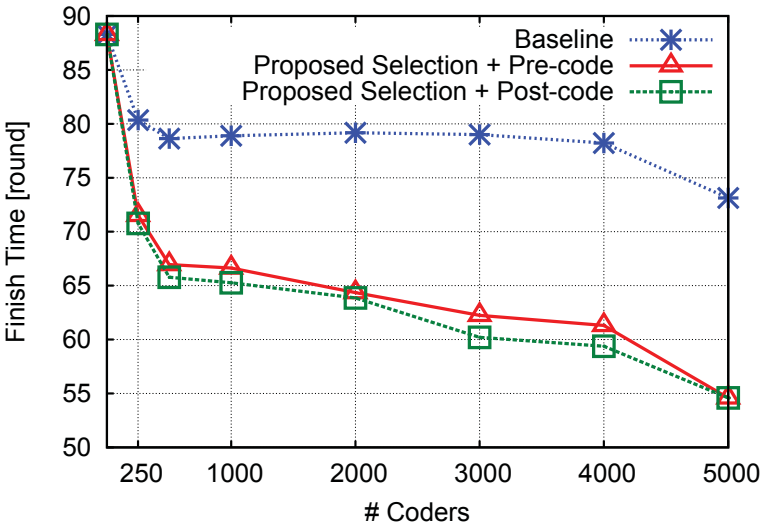

Figure 3.16: Finish time of the proposed system compared with a baseline system in case encoders are placed at high-degree peers.

2. **Resource-constraint scenario**: nodes with higher capacity can encode, nodes having limited resources cannot. Among peers, we set some random ones with rich resources and assign them as encoders.

We increase the number of encoders from 0 (*no coding*) to 5000 (*full network coding*) and compare the performance of the proposed system with the baseline system in two scenarios above. The results are given in Figure 3.15, Figure 3.16, and Figure 3.17 with rewiring probability p_{rw}=0.02.

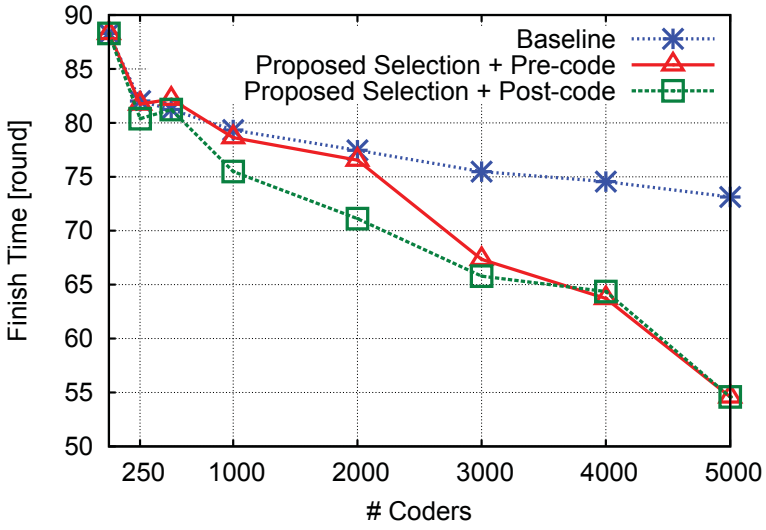

Figure 3.17: The performance of the proposed system compared with baseline system in resource-constraint scenario when only (random) high-capacity peers are allowed to encode.

When betweenness centrality is used to optimize encoder placement, the proposed block selection together with pre-code protocol shortens distribution time by about 15% compared to the baseline system with only 250 encoders (Figure 3.15). With more encoders, the improvement is higher and reaches more than 25% when all nodes encodes. The finish time of no coding, i.e. the number of encoders is zero, is the same regardless of which block selection algorithm and protocol are used.

With 2000 or less encoders chosen at random, i.e. a set of random peers are allowed to encode, there is not much finish time improvement using both rarest-first and the proposed block selection (Figure 3.17). That is because a few encoders at random, without a proper placement, are not effective in improving distribution time. When a large number of encoders, e.g. 3000 or 4000 encoders, are randomly deployed, the proposed block selection with pre-code protocol can improve distribution time by around 15% compared to baseline system.

The finish time using degree-based placement lies between the other two placements. As before, the proposed system achieves noticeable finish time improvement compared to the baseline system using rarest selection (Figure 3.16).

We next change the rewiring probability p_{rw} from 0.02 to 0.5 to evaluate our

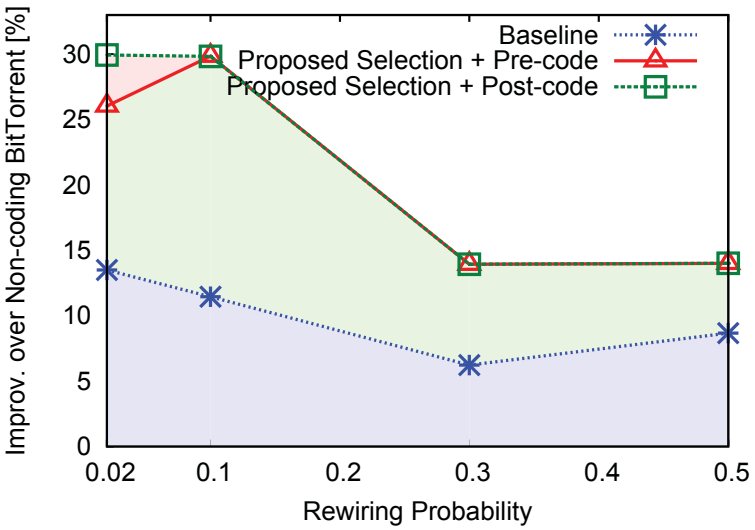

Figure 3.18: Finish time improvement of the proposed system and baseline system using 250 encoder placed at high betweenness centrality peers compared with non-coding BitTorrent.

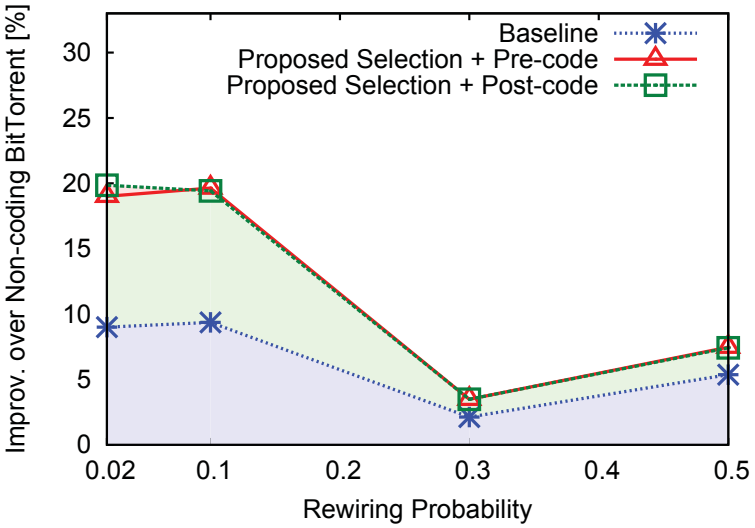

Figure 3.19: Finish time improvement of the proposed system and baseline system using 250 encoder placed at high-degree peers compared with non-coding BitTorrent.

system in a wide range of topologies. Using 250 encoders among the total of 5000 peers, the finish time improvement compared with non-coding Bit-Torrent (with no encoder) is presented in Figure 3.18, Figure 3.19, and Figure 3.20 for betweenness centrality placement, degree-based placement, and random coder placement respectively.

Our proposed system (*proposed selection + pre-code* and *proposed selection +*

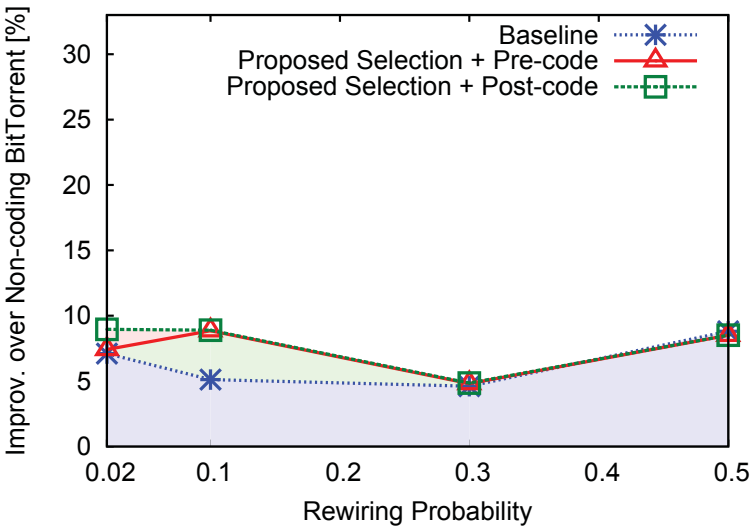

Figure 3.20: Finish time improvement of the proposed system and baseline system using 250 encoder placed at random peers compared with non-coding BitTorrent.

post-code) always achieves improved performance compared with the baseline system. The improvement, however, is more visible in topologies with low rewiring probabilities (p_{rw}=0.02). High rewiring probabilities generate almost random topologies in which the effect of coding, in general, is not so noticeable.

The performance of post-code protocol (*proposed selection + post-code*) is better than pre-code protocol (*proposed selection + pre-code*) in low-rewiring topologies p_{rw}=0.02 (Figure 3.15–3.17). The reason is because with post-code protocol encoders can combine more updated information to send to the receivers as we have discussed before. In topologies with higher rewiring probabilities ($p_{rw} \geq 0.1$) (Figure 3.18, 3.19, and 3.20), since new information can transfer through more rewiring links, post-code protocol has the same performance as pre-code protocol. We note that in the simulations, we have not taken into account the overhead of protocols used to communicate between peers.

3.6 Conclusion

We have proposed information exchange protocols and its associated block-selection algorithm to improve performance of a hybrid network coding P2P

system in which encoding-enabled and non-encoding peers coexist.

Our design is simple, backward compatible to BitTorrent, yet efficient in the way it handles blocks of data: coded and non-coded alike.

We proposed two protocols. The first one, pre-code protocol, is an extension of BitTorrent with the addition of an encoder-id field in the exchanged messages to identify from whom the blocks are generated. The second one, namely post-code, by postponing the encoding process, can combine and deliver more updated information to the receivers and achieve shorter finish time. Post-code protocol is more effective in severely bottlenecked topologies. The trade-off is, however, higher protocol overhead.

Our block-selection algorithm is derived from observation on the benefit of network coding in eliminating data duplication. Most recently encoded blocks are preferred over other blocks since they contain more information, and thus, can accelerate throughput. Using our proposed algorithm, peers can effectively choose blocks to down-load which results in considerable improvement in distribution time.

We believe our proposed solution, which promotes network coding as a method to shorten distribution time even if encoding is not fully enabled at every peer, will be of great use in heterogeneous P2P systems and/or when there is a need to minimize resource consumption.

For future work, we plan to evaluate the proposed design and block selection algorithm in a dynamic setting. We are also interested in implementing the proposal in a real system, especially to evaluate the actual trade-off and effectiveness of the post-code protocol.

We have not addressed incentive issues: how to motivate peers to encode, which is another interesting problem we leave for future work.

Chapter 4

Minimal Delay Coder Placement

Motivated by the question "can we achieve the performance of network coding without requiring all nodes to encode," in this chapter, we first identify the underlying condition for network coding to be effective compared with no coding, and then, given that insight, we propose a novel coder placement algorithm that achieves comparable performance in terms of finish time as network coding while using much less computational resources than network coding does.

We make an elaborate analysis of the network topology to locate nodes in the network where network coding can accelerate content distribution the most. The analysis result is then used to develop a coder placement algorithm which we name *minimal delay* placement or *min-delay* for short. Performance evaluation confirms the effectiveness of min-delay coder placement in shortening distribution time given a constraint on the number of network coding.

Before going into details of our analysis and the proposed network coder placement, we begin with a statement of the network coder placement problem we consider in this chapter.

4.1 Network Coder Placement Problem

Network coding, in achieving optimal distribution time, requires every node to code. Assuming the system uses random linear coding, the encoding complexity of the system is $O(CFK)$ where C is the number of encoders in the system, K is the number of original blocks, and F is the file size. Since F is fixed for a given file, and K cannot be set too low, it is important to minimize the number of coders to reduce the encoding complexity.

Our goal is to devise a coder placement algorithm which substantially reduces the number of coders while, at the same time, effectively shortens distribution time. Our problem can be stated as follows.

Given a P2P content distribution which is defined by

- a network topology $G = \{V, E\}$,

- a source on G with a file of size F to be distributed, and

- a number C $(1 \leq C \leq |V|)$,

where in the network topology can we place C coders in order to shorten distribution time the most?

Since our problem is as hard as the problem of finding a placement with shortest finish time and minimum number of coders which is proved to be NP-hard [39], we aim at heuristic placements which we demonstrate by simulations to achieve comparable finish time as full network coding.

We propose 3 algorithms in total:

1. *minimal delay* placement,

2. *betweenness centrality-based* placement, and

3. *flow centrality-based* placement.[1]

[1] We explain betweenness centrality and flow centrality in Section 5.3 and Section 5.4 respectively.

Table 4.1: Notations

Notation	**Meaning**
$1, 2, ..., i, .., j$	Node IDs. (The source is denoted as node 0.)
t	Time
$R(t)$	The number of redundant blocks transmitted downstream from an intermediate node at time t.
$\alpha_1(t)$	The number of blocks coming to node 2 on $path1$ which node 2 has not downloaded directly from node i by time t in two-receiver case (Figure 4.2).
$\beta_1(t)$	The number of duplicated blocks arriving at node 2 via $path1$ by time t in two-receiver case (Figure 4.2).
$\alpha_2(t)$	The number of blocks coming to node 1 on $path2$ which node 1 has not downloaded directly from node i by time t in two-receiver case (Figure 4.2).
$\beta_2(t)$	The number of duplicated blocks arriving at node 1 via $path2$ by time t in two-receiver case (Figure 4.2).
$f(i, j)$	Actual throughput from node i to node j
$B(i, j)$	Duplication rate from node i to node j
$D(i, j)$	Delay in finish time node i causes to node j due to block duplication.
F	File size in blocks
$maxflow(i, j)$	*Maxflow* from node i to node j.

The first algorithm is proposed in this chapter. The second and third ones are fast placement algorithms taking advantage of *network centrality* [48, 49] to quickly locate key network positions for coder placement which will be presented in the next chapter.

To lay the foundation for our minimal delay network coder placement, we begin with an elaborate analysis of the *duplication* generated in the network topology to determine network nodes at which network coding is needed the most to eliminate that duplication and improve the system performance.

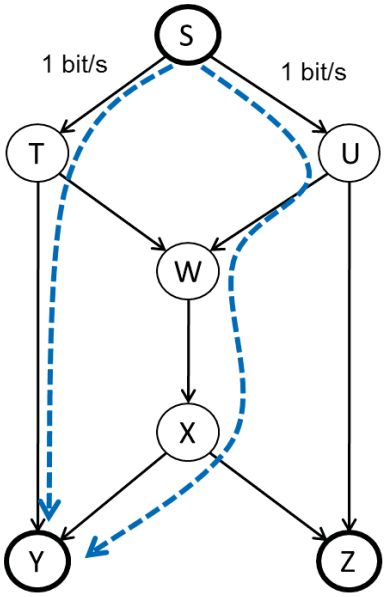

Figure 4.1: Two paths from node S to node Y are marked in dashed lines. All links on the graph have capacity of 1 bit/s. The *maxflow* from node S to node Y is 2 bit/s, which means at most 2 bits can be transferred from node S to node Y in a second.

4.2 Multi-path Delivery Duplication Analysis

In this section, we develop a full analysis of the *block duplication*, i.e. the same, duplicated blocks are delivered towards a given node. Such an analysis is required to find out the nodes from which duplication degrades performance, i.e. finish time, the most. Given that knowledge, we then can place encoders there to eliminate duplication and effectively speed up content distribution.

We begin with terminologies, and then, present our detailed analysis of block duplication. A path, without circles or loops, from node i to node j is a sequence of nodes starting from i and terminating at j in which two adjacent nodes are connected by a link. A flow on a path from node i to node j is a mapping $E \rightarrow \mathbf{R}^+$ which conforms to

- *capacity constraint* of each link: the flow does not exceed the capacity of the link over which it runs through, and
- *flow conservation* at each node on the path: the flow coming to a given node

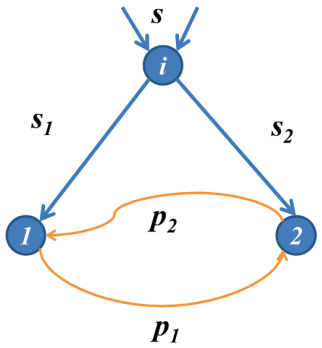

Figure 4.2: A partial graph connecting an intermediate node i and two of its neighbors: node 1 and node 2.

is equal to the flow going out of that node except for the source (node i) and the sink (node j).

A *maxflow* on a path is the maximum flow which can run through the path, and a *maxflow* from node i to node j is the maximum amount of flows passing from node i and node j. Figure 4.1 illustrate two paths from node S to node Y. The maximum flow which can run through each path is 1 bit/s and the *maxflow* from node S to node Y is 2 bit/s.

In P2P content distribution, when there are multiple delivery paths to a particular node, some blocks are transmitted multiple times to those paths, which inefficiently consumes bandwidth and results in insufficiency of new information flow coming to the downstream node. We analyze block duplication on multiple delivery paths in P2P content distribution systems, first in the case the upstream node has two downstream receivers, whose result is then extended to more general cases.

4.2.1 Two Receiver Duplication

Figure 4.2 illustrates data transmission from an upstream node i to two direct neighbors node 1 and node 2. There are two paths, i.e. sequences of nodes, one in each direction, connecting node 1 and node 2: $path1$ from node 1 to node 2,

and $path2$, in the opposite direction, from node 2 to node 1. Denote the incoming throughput to node i as s, the throughput from node i to node 1 over the direct link as s_1, and the throughput from node i to node 2 as s_2. The flow on $path1$ from node 1 to node 2 is p_1, and on $path2$ from node 2 to node 1 is p_2. Since both flows originate from node i, the parameters satisfy following constraints: $p_1 \leq s_1 \leq s$ and $p_2 \leq s_2 \leq s$. In addition, let d_1 is the delay it takes for a block to travel from node 1 to node 2 on $path1$, and d_2 is the delay from node 2 to node 1 on $path2$.

Consider blocks departing from node i. Ideally, to achieve optimal distribution time, a given block should not be transmitted to both node 1 and node 2, except when there are no new blocks available at node i to satisfy requests from node 1 and node 2. Nevertheless, due to delay on connecting paths: $path1$ and $path2$, node 1 and node 2 only know a subset of the blocks which have been downloaded by the other node. As a result, a considerable number of blocks available at node i are requested by and transmitted to both downstream neighbors node 1 and node 2, which we call *redundant blocks*. Those redundant blocks, being transmitted by node 1 and node 2 on the two paths: $path1$ and $path2$, will result in *block duplication* or *duplicated blocks* with the reception at node 2 and node 1 respectively.

Suppose by time t, node 1 has downloaded set S_1 of blocks, and node 2 has downloaded set S_2 of blocks directly from node i. Then the number of redundant blocks received by both node 1 and node 2 by time t is $R(t) = |S_1 \cap S_2|$. At time $t+1$, there are 3 cases in which redundant blocks are transmitted from node i, i.e. the same blocks are downloaded by both node 1 and node 2:

1. blocks in S_1 are downloaded again by node 2,
2. blocks in S_2 are downloaded again by node 1, and
3. the same blocks from node i, which have not been downloaded by either node, are downloaded by both node 1 and node 2.

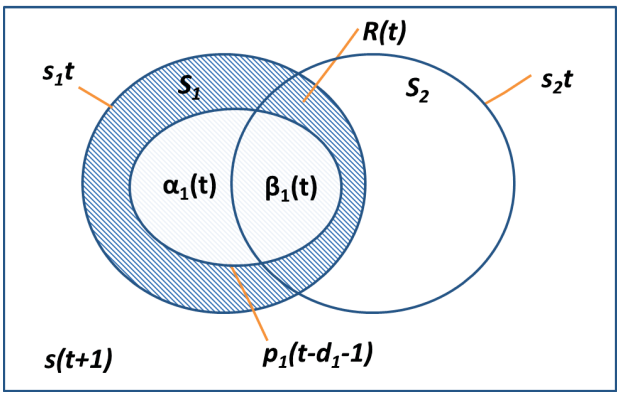

Figure 4.3: Snapshot of data blocks downloaded by node 1 and node 2 at time $t+1$ before the two nodes choose new blocks to download from node i. There are $s(t+1)$ blocks available at node i, of which s_1t and s_2t are the numbers of blocks node 1 and node 2 have downloaded respectively. The inner area $p_1(t-d_1-1)$ represents the number of blocks node 2 received from node 1 over $path1$; a subset of which: $\beta_1(t)$ blocks node 2 has already downloaded from node i, results in *block duplication* at node 2.

Denote $T_1(t)$, $T_2(t)$, and $T_3(t)$ as the number of redundant blocks corresponding to each of these above cases. Let $\beta_1(t)$ be the number of duplicated blocks arriving at node 2 via $path1$ by time t which it has already downloaded from node i. Those duplicated blocks, although not downloaded by node 2 again, will result in under-utilization of $path1$, and consequently node 2's downloading capacity because they consume bandwidth resource along the path from node 1 to node 2.

Denote $\alpha_1(t)$ as the number of blocks coming to node 2 on $path1$ which it has not downloaded directly from node i by time t (refer to Figure 4.3 for an illustration). Since there are a total of $p_1(t-d_1-1)$ blocks[2] coming to node 2 on $path1$ by time t, we have $\alpha_1(t) + \beta_1(t) = p_1(t - d_1 - 1)$.

At time $t+1$, node 2 downloads s_2 new blocks from node i. It can choose s_2 blocks from a set Q_2 of $s(t+1)-s_2t-\alpha_1(t)$ blocks. Within set Q_2, $s_1t-R(t)-\alpha_1(t)$ blocks have been downloaded only by node 1. Therefore, we have the expected

[2]We assume at a given time t a node can only send blocks it has received at time $t-1$ and earlier. For that reason, at time t node 1 sent a total $p_1(t-1)$ blocks through $path1$, of which $p_1(t-d_1-1)$ blocks have arrived at node 2 by time t due to delay d_1 on the path.

number of redundant blocks in case 1:

$$T_1(t+1) = \frac{s_2(s_1 t - R(t) - \alpha_1(t))}{s(t+1) - s_2(t) - \alpha_1(t)}. \tag{4.1}$$

Similarly, the number of redundant blocks in case 2 is:

$$T_2(t+1) = \frac{s_1(s_2 t - R(t) - \alpha_2(t))}{s(t+1) - s_1(t) - \alpha_2(t)} \tag{4.2}$$

where $\alpha_2(t)$ is the number of blocks node 1 received on $path2$ which it has not downloaded directly from node i by time t.

The number of new blocks at node i which have not downloaded by either node 1 or node 2 is $s(t+1) - (s_1 + s_2)t + R(t)$, from which node 1 might download $s_1 - T_2(t+1)$ blocks and node 2 might download $s_2 - T_1(t+1)$ blocks. The expected number of blocks which are downloaded by both node 1 and node 2, i.e. case 3, therefore is

$$\begin{aligned} T_3(t+1) &= \frac{(s_1 - T_2(t+1))\,(s_2 - T_1(t+1))}{s(t+1) - (s_1+s_2)t + R(t)} \\ &= \frac{s_1 s_2 (s(t+1) - (s_1+s_2)t + R(t))}{(s(t+1) - s_1(t) - \alpha_2(t))(s(t+1) - s_2(t) - \alpha_1(t))}. \end{aligned} \tag{4.3}$$

From Eq. (4.1), (4.2), and (4.3), the total redundant blocks at time $t+1$ is

$$\begin{aligned} R(t+1) &= R(t) + T_1(t) + T_2(t) + T_3(t) \\ &= R(t) + \frac{s_2(s_1 t - R(t) - \alpha_1(t))}{s(t+1) - s_2(t) - \alpha_1(t)} + \frac{s_1(s_2 t - R(t) - \alpha_2(t))}{s(t+1) - s_1(t) - \alpha_2(t)} \\ &\quad + \frac{s_1 s_2 (s(t+1) - (s_1+s_2)t + R(t))}{(s(t+1) - s_1(t) - \alpha_2(t))(s(t+1) - s_2(t) - \alpha_1(t))}. \end{aligned} \tag{4.4}$$

We continue to figure the number of duplicated blocks arriving at node 1 and node 2: $\beta_2(t)$ and $\beta_1(t)$ respectively. Those duplicated blocks are the cause of sub-optimal throughput to node 1 and node 2.

At time t, node 2 downloads p_1 new blocks from $path1$. Those blocks are chosen

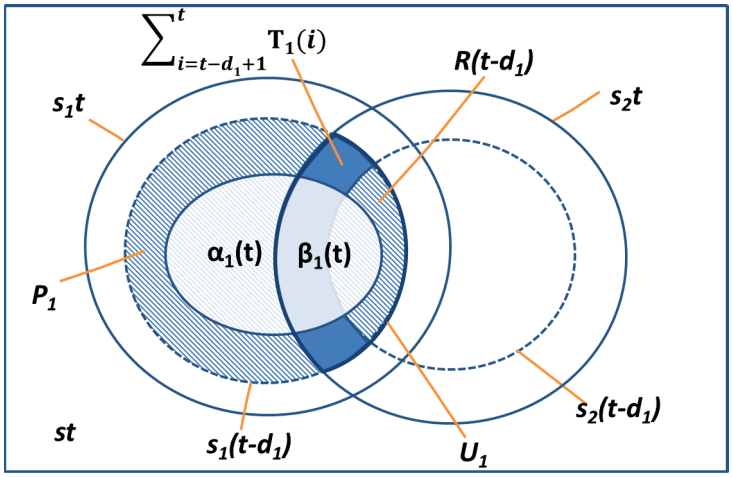

Figure 4.4: The number of duplicated blocks arriving at node 2 via $path1$ by time t: $\beta_1(t)$ is a subset of the union of $R(t-d_1)$ and $\sum_{i=t-d_1+1}^{t} T_1(i)$.

from a set P_1 of $s_1(t-d_1)-p_1(t-d_1-1)$ which have been downloaded by node 1 by time $t-d_1$ but not have yet been sent to node 2 (Figure 4.4).[3]

Notice that the intersection U_1 of $s_1(t-d_1)$ blocks node 1 has downloaded by time $t-d_1$ and s_2t blocks node 2 has downloaded by time t consists of

- $R(t-d_1)$ redundant blocks by time $t-d_1$, and
- $\sum_{i=t-d_1+1}^{t} T_1(i)$ redundant blocks from $t-d_1+1$ to t.

Given that at time t, $\beta_1(t-1)$ blocks in set U_1 have already been sent to node 2, if any blocks in the remaining set $U_1 \setminus \beta_1(t-1)$ were chosen to send to $path1$, those blocks would become duplicated when they arrive at node 2. As a result, the number of duplicated blocks during time t is

$$\Delta\beta_1(t)=\frac{p_1(R(t-d_1)+\sum_{i=t-d_1+1}^{t} T_1(i)-\beta_1(t-1))}{s_1(t-d_1)-p_1(t-d_1-1)}.$$

The accumulative number of duplicated blocks arrived at node 2 on $path1$ by time

[3]We made a simplification to the analysis by ignoring the set of blocks arrived at node 1 from node 2 over $path2$ by time $t-d_1$: $p_2(t-d_1-d_2-1)$ blocks. In reality, those blocks would not be chosen by node 1 to send back to node 2 over $path1$ again.

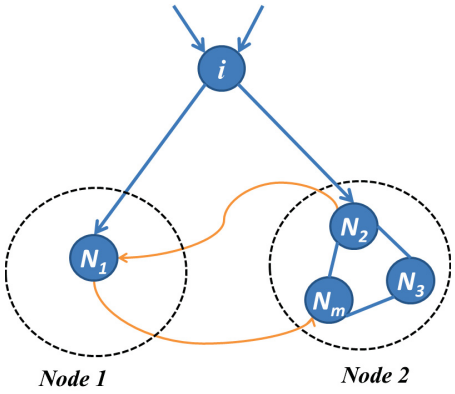

Figure 4.5: A partial graph connecting an intermediate node i and m receiving nodes: node N_1, node N_2 and node N_m. The receiving nodes are represented by two virtual nodes: Node 1 and Node 2.

t, therefore, is

$$\begin{aligned}\beta_1(t) &= \Delta\beta_1(t) + \beta_1(t-1) \\ &= \frac{p_1(R(t-d_1)+\sum_{i=t-d_1+1}^{t} T_1(i)-\beta_1(t-1))}{s_1(t-d_1)-p_1(t-d_1-1)} + \beta_1(t-1),\end{aligned} \tag{4.5}$$

and, the number of non-duplicated blocks is

$$\alpha_1(t) = p_1(t-d_1-1) - \beta_1(t). \tag{4.6}$$

In the opposite direction, using the same reasoning, we have

$$\beta_2(t) = \frac{p_2(R(t-d_2)+\sum_{i=t-d_2+1}^{t} T_2(i)-\beta_2(t-1))}{s_2(t-d_2)-p_2(t-d_2-1)} + \beta_2(t-1), \tag{4.7}$$

$$\alpha_2(t) = p_2(t-d_2-1) - \beta_2(t). \tag{4.8}$$

4.2.2 Multiple Receiver Duplication

We use the two-receiver analysis to approximate block duplication in case there are more than two neighbors who receive blocks from the intermediate node i. Suppose there are m receivers ($m > 2$): node N_1, node N_2, ..., node N_m (Figure 4.5). To analyze duplication to a given receiver, for example node N_1, we can proceed as follows.

- Separate node N_1 and consider it as node 1 in the two-receiver case.
- All the remaining nodes: node N_2, ..., node N_m, are represented by a virtual node 2.
- Figure the topological parameters, i.e. bandwidth and delay, between node i, node 1, and the virtual node 2.
- Apply the two-receiver model to analyze block duplication to node 1.

4.2.3 Delay in Finish Time of Downstream Peers

The above duplication analysis allows us to quantify the actual throughput from node i to its neighbor node 1, as

$$f(i,1) = s_1 + \frac{\alpha_2(t)}{t}. \tag{4.9}$$

Since the maximum throughput from node i to node 1 is $\min(s, s_1 + p_2)$, the delay in finish time of node 1 due to block duplication distributing S blocks from node i is:

$$\begin{aligned} D_1 &= \frac{S}{f(i,1)} - \frac{S}{\min(s, s_1 + p_2)} \\ &= \frac{S}{\min(s, s_1 + p_2) - B(i,1)} - \frac{S}{\min(s, s_1 + p_2)} \end{aligned} \tag{4.10}$$

where $B(i,1) = \min(s, s_1{+}p_2) - f(i,1)$ is the *duplication rate* from node i to node 1.

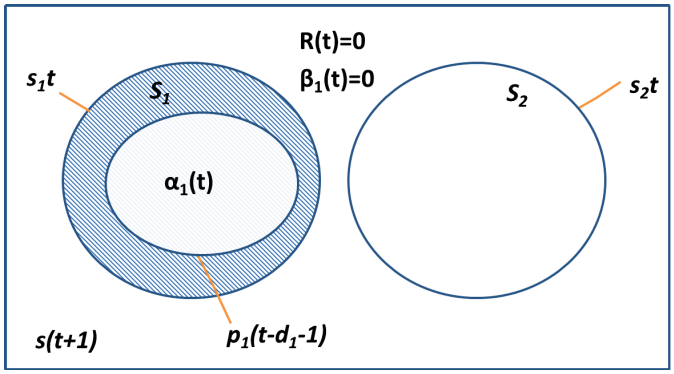

Figure 4.6: When node i encodes, the set S_1 of blocks downloaded by node 1 and the set S_2 of blocks downloaded by node 2 are non-overlapped. The number of redundant blocks $R(t) = |S_1 \cap S_2|$ is, therefore, equal to zero. As a result, number of duplicated blocks coming to node 2: $\beta_1(t) = 0$. (Although not depicted, the same is true for $\beta_2(t)$.)

In general, given the duplication rate from node i to node j: $B(i,j)$, and the $maxflow$ value from the source to node j: $maxflow(0,j)$, the delay node i causes to node j is

$$\begin{aligned} D(i,j) &= \frac{F}{maxflow(0,j) - B(i,j)} - \frac{F}{maxflow(0,j)} \\ &= \frac{B(i,j)}{maxflow(0,j) - B(i,j)} . \frac{F}{maxflow(0,j)} \end{aligned} \tag{4.11}$$

where F is the total number of blocks at the source. The attribute of Eq. 4.11 is that it gives higher values to node j

- with low throughput from the source, i.e. large $\frac{F}{maxflow(0,j)}$, and
- which experiences high duplication rate among nodes with the same $maxflow$ from the source, i.e. $\frac{B(i,j)}{maxflow(0,j)-B(i,j)}$ is high.

4.2.4 The Effect of Network Coding

When a node encodes, each block it sends is uniquely generated: there exist no identical blocks transmitting on different paths originated from it. Suppose in the two-receiver case (Figure 4.2), node i is allowed to encode using random linear

network coding. Because there are virtually no identical encoded blocks, the number of redundant blocks originating from node i is zero[4]. We have $S_1 \cap S_2 = \emptyset$ (Figure 4.6) which means there are no redundant blocks transmitted from node i:

$$\begin{aligned} R(t) &= |S_1 \cap S_2| \\ &= 0. \end{aligned}$$

Since duplicated blocks arriving at node 2 and node 1 are subsets of the set of redundant blocks $S_1 \cap S_2$, we also have

$\beta_1(t) = 0$, and

$\beta_2(t) = 0$ respectively.

Therefore, the delay in finish time of node 1 and node 2 due to duplication from node i:

$D_1 = 0$, and

$D_2 = 0$ respectively.

In the multiple receiver case, when an upstream node i encodes, since there are no duplicated blocks sending from node i, the delay in finish time it causes to a downstream node j due to block duplication is

$$D(i, j) = 0. \tag{4.12}$$

[4]Using random linear network coding, there is a small probability that two encoded blocks are linearly dependent. The probability, however, approaches zero when encoding is done in a large finite field.

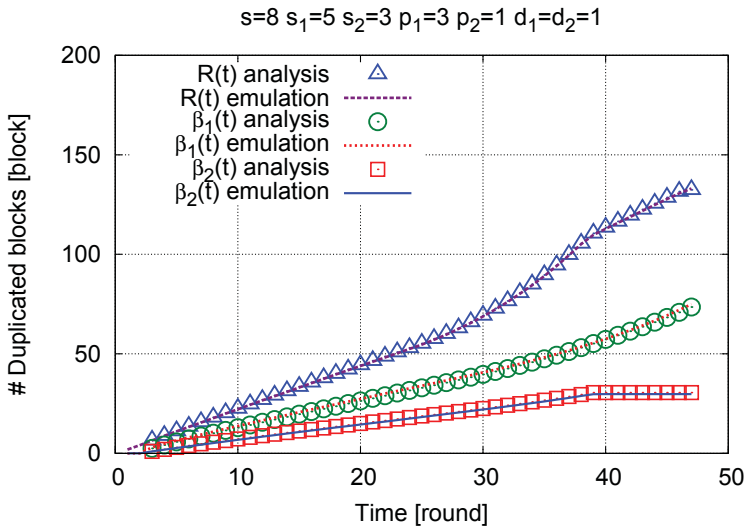

Figure 4.7: Block duplication with different bandwidth settings. The analysis result closely matches emulation result.

4.2.5 Numerical Experiments

Number of redundant blocks $R(t)$, number of duplicated blocks $\beta_1(t)$, $\beta_2(t)$, and delay in finish time D_1, D_2 can be computed from the network topological parameters, i.e. s, s_1, s_2, p_1, p_2, d_1, and d_2 using Eq. ((4.1), (4.2), (4.4)–(4.11). In this subsection, we present some numerical experiments with various topological parameters.

To confirm the correctness of the analysis, we compare the analytical results with a BitTorrent emulation[5] running on the partial topology in Figure 4.2. For each input parameter, we run the emulation 1000 times. The comparison is given in Figure 4.7 where block duplication analysis results (the dots) closely match the emulation results (the lines).

Furthermore, we vary the bottleneck bandwidth p_2 to see how it affects the duplication rate $B(i, 1)$ and delay in finish time D_1 of node 1. With wider bottleneck bandwidth, node 1 experiences higher duplication rate (Figure 4.8) and longer finish time delay (Figure 4.9) because it receives more blocks from the bottleneck

[5]We isolate BitTorrent's block exchange mechanism and emulate it in our emulation.

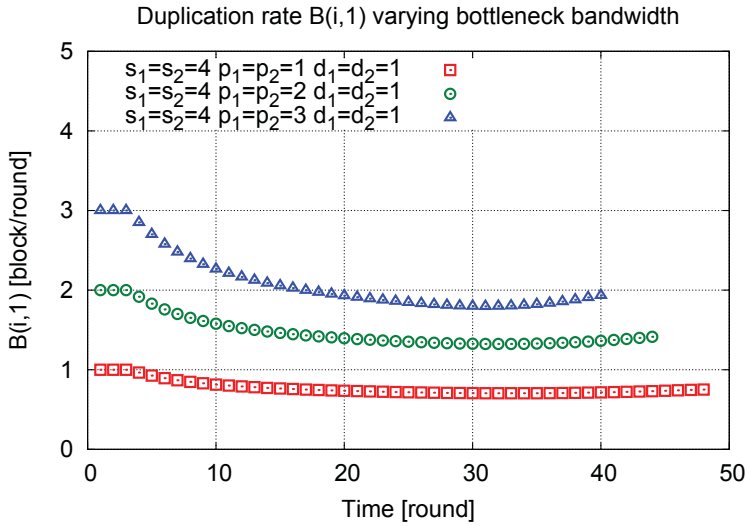

Figure 4.8: Duplication rate with different bandwidth settings. Wider path bandwidth p_1 and p_2 result in higher duplication rate.

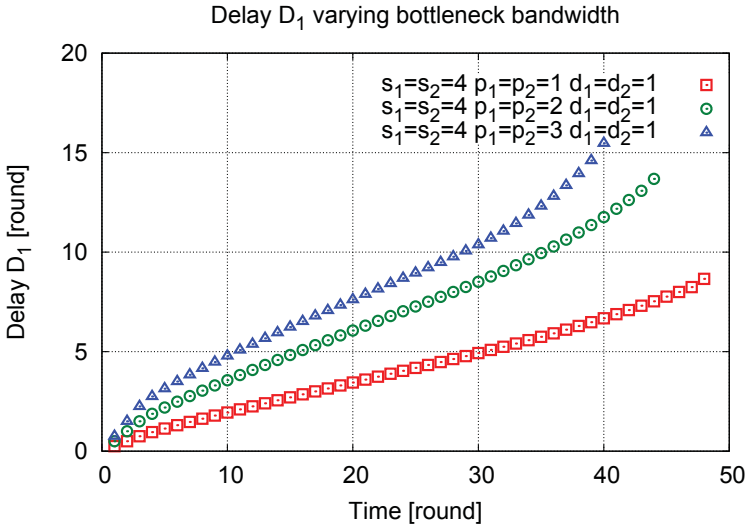

Figure 4.9: Delay in finish time with different bandwidth settings. Wider path bandwidth p_1 and p_2 result in longer delay in finish time of node 1.

path which increases the chance of block duplication.

4.3 Delay Computation and Placement Algorithm

We use block duplication analysis in section 4.2 to figure the number of duplicated blocks originating from each node, and how much delay that particular node

causes to a given *child*, i.e. a neighboring node which receives data from it, and all downstream peers of that child. We then assign the nodes which generate the most delay as encoders. The total delay caused by a node i due to block duplication is:

$$D(i) = \sum_{i \in maxflow(0,j)} D(i,j) \tag{4.13}$$

where $D(i,j)$ is the delay computed using Eq. 4.11.

Our placement algorithm (Algorithm 4.1) starts by computing *maxflow* from the source (node 0) to all peers. During that process, it collects:

- the set $downstream[i \rightarrow j]$ of downstream peers of each link $i \rightarrow j$: peer k is a downstream peer of link $i \rightarrow j$ if link $i \rightarrow j$ belongs to the $maxflow$ from the source to node k,
- the flow passing a link $i \rightarrow j$ to a downstream node k: $flow[i \rightarrow j \rightarrow k]$, and the flow passing a node i to a downstream node k: $flow[i \rightarrow k]$,
- the set $child[i]$ of children of node i: node k is a child of node i if it is a downstream node of node i and the two nodes are neighbors, i.e. directly connected, and
- the flow from a node i to its child node k on link $i \rightarrow k$: $directflow[i \rightarrow k]$.

The main part of the algorithm (lines 23–43) compute the block duplication rate a node i causes to each of its child node j as in section 4.2. The duplication is then propagated to all downstream nodes of link $i \rightarrow j$. Finally, the delay node i causes to each of its downstream nodes is computed using Eq. 4.11 and added to the total delay $D[i]$.

For each link from a node to its child, we have to run $maxflow$ algorithm [58] (lines 31 and 33) which takes $O(VE^2)$ time. Since the main loop is run at most E times, once for each link, the overall complexity therefore is $O(VE^3)$.

Algorithm 4.1: Minimal Delay Placement Algorithm

Input: $G = \{V, E\}$, the source (node 0), file size F, number C
Result: C encoders ($1 \leq C \leq |V|$)

forall the $i, j, k \in V$ **do**
  $flow[i \to j \to k] = 0$;
  $flow[i \to k] = 0$;
  $directflow[i \to k] = 0$;
  $downstream[i \to j] = \emptyset$;
  $child[i] = \emptyset$;
  $D[i] = 0$;
end
foreach $k \in V \backslash \{0\}$ **do**
  compute $maxflow(0 \to k)$ by Edmonds-Karp algorithm [58];
  foreach $i \to j \in maxflow(0 \to k)$ **do**
    $downstream[i \to j] \leftarrow k$;
    $flow[i \to j \to k] = maxflow_{0 \to k}(i \to j)$;
  end
  foreach $i \in maxflow(0 \to k)$ **do**
    $flow[i \to k] = maxflow_{0 \to k}(i)$;
    if $(i \to k) \in E$ **then**
      $child[i] \leftarrow k$;
      $directflow[i \to k] = maxflow_{0 \to k}(i)$;
    end
  end
end
foreach $i \in V$ **do**
  foreach $j \in child[i]$ **do**
    $V_1 \leftarrow j$;
    $V_2 \leftarrow child[i] \backslash \{j\}$;
    $s = flow[0 \to i] - flow[j \to i]$;
    $s_1 = directflow[i \to j]$;
    $s_2 = 0$;
    foreach $k \in child[i] \backslash \{j\}$ **do** $s_2 += directflow[i \to k]$;
    $p_1 = \min(maxflow(V_1 \to V_2), s_1)$;
    $d_1 = \min(pathlen(V_1 \to V_2))$;
    $p_2 = \min(maxflow(V_2 \to V_1), s_2)$;
    $d_2 = \min(pathlen(V_2 \to V_1))$;
    $S = \frac{F.flow[i \to j]}{maxflow(0 \to j)}$;
    compute duplication using Eq. (4.1), (4.2), (4.4)–(4.9);
    $loss\_ratio = \frac{\min(s, s_1 + p_2) - f(i,1)}{\min(s, s_1 + p_2)}$;
    foreach $k \in downstream[i \to j]$ **do**
      $B[i \to k] += loss\_ratio * flow[i \to j \to k]$;
    end
  end
  foreach $k \in V$ **do** $D[i] += \frac{S}{maxflow(0,k) - B[i \to k]} - \frac{S}{maxflow(0,k)}$;
end
Encoders $\leftarrow C$ peers with highest $D[\cdot]$ values;
return Encoders;

4.4 Performance Evaluation

4.4.1 Simulation Settings

We implemented a C++ simulator of the P2P content distribution system using the pre-code protocol in Section 3.2.2 and run simulations over generated topologies distributing a file from the source to all participating peers. The file is divided into smaller fix-sized parts, i.e. blocks. The source and all peers exchange blocks until all peers acquire enough blocks to construct the original file; then the simulation finishes.

The simulations are round-based and peers exchange blocks using the block selection described in Section 3.3 and mutual exchange incentive scheme, the same as in Section 3.5.

We consider three scenarios.

1. *No coding* – coding is not allowed in the system, i.e. all peers send and receive original blocks as in a pure P2P system. A peer finishes when it has collected all the original blocks.

2. *Network coding* – all peers, including the source, are allowed to encode, i.e. combine downloaded blocks to make new encoded blocks and send to other peers. A peer finishes when it has collected enough coded blocks required for decoding.

3. *Selective coding* (proposed) – the same as network coding except that only some peers chosen by the proposed placement algorithm, including the source, are allowed to encode.

For each overlay topology, with the same simulation parameters, we run simulations 100 times[6] distributing a 200-block file from the source and collect the

[6]Although simulation parameters are the same for 100 runs, due to randomness in downloader

Table 4.2: Finish time in a network of 50 nodes placing 4 encoders including the source (values are in *round*). Finish time of network coding (when all 50 encoders are encoders) is included for reference.

	Average Finish Time	Maximum Finish Time
Brute-force Search	64.41	76.50
Min-delay (proposed)	65.32	77.53
Degree-based	68.50	81.46
Network Coding	58.31	75.00

average finish time of all peers (T_{avg}) and maximum finish time among all peers (T_{max}) in each of the 3 scenarios: no coding, network coding, and selective coding when we use the proposed min-delay algorithm to place coders.

We use Watts and Strogatz *small-world network* model [55] to generate P2P network topologies with 5000 peers as described in Section 3.5.2. By varying the small-world network's degree d and rewiring probability p_{rw} we can generate a wide range of network topologies from highly bottlenecked topologies (with low p_{rw}) to random topologies (with high p_{rw}). Capacity of all links is set to 1 block/round.

4.4.2 Performance Compared with Optimal Placement

We first evaluate our algorithm in a small-sized network of 50 peers (degree $d = 4$ and rewiring probability $p_{rw} = 0.05$) with $C = 4$ encoders (including an encoder at the source). Since the size of network is relatively small, we can find an optimal placement by brute-force searching all possible combinations of encoders to find the one which makes shortest finish time.

The performance of the proposed min-delay placement is also compared with that of degree-based placement, i.e. network coders are placed at high-degree nodes first, using the same number of encoders. The result is given in Table 4.2. Both average finish time of all peers and maximum finish times among all peers of min-

selection by the sending peers and block selection by the receiving peers, the result changes with each run.

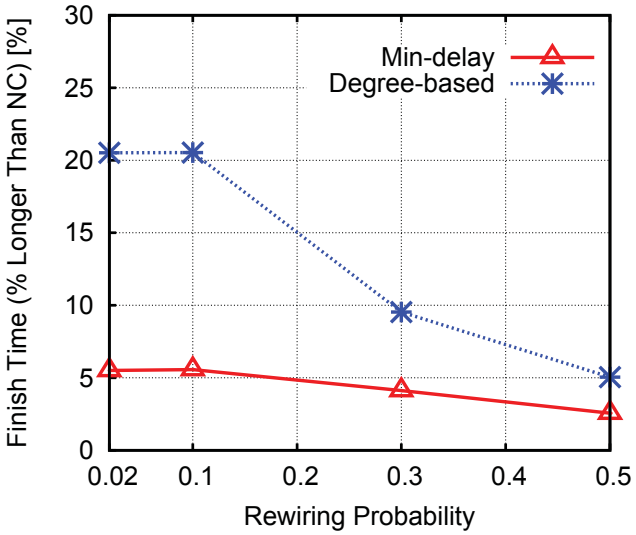

Figure 4.10: Maximum finish time of the proposed min-delay algorithm placing 1000 encoders in 5000-peer topologies with different rewiring probabilities compared with *network coding*. The maximum finish time of degree-based placement is given for reference.

delay placement are close to the optimal maximum finish time found by brute-force search and much shorter than finish time of the degree-based method.

Network coding finish time, included in Table 4.2 for reference purpose, is always shorter than the other given methods because network coding uses all 50 peers as encoders.

4.4.3 Performance in Moderate Bottlenecked Topologies

Topologies with moderate bottleneck are generated using small-world network model with degree $d = 6$ and relatively high rewiring probability $0.02 \leq p_{rw} \leq 0.4$.

Placing 1000 encoders in 5000-peer networks (Figure 4.10 and Figure 4.11), the performance of *min-delay* placement in terms of maximum finish time (Figure 4.10) and average finish time of all peers (Figure 4.11) is as good as network coding's performance. With 20% of the number of encoders, min-delay algorithm can achieve finish time just about 5% longer than finish time of network coding in moderate bottlenecked topologies.

Degree-based placement results in much longer finish time, sometimes as much

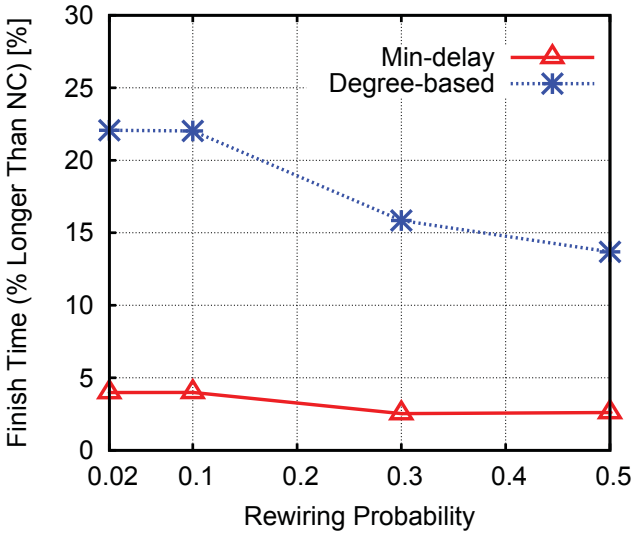

Figure 4.11: Average finish time of the proposed min-delay algorithm placing 1000 encoders in 5000-peer topologies with different rewiring probabilities compared with *network coding*. The average finish time of degree-based placement is given for reference.

as 20% longer than network coding.

4.4.4 Performance in Highly Bottlenecked Topologies

We generate severely bottlenecked topologies by setting the rewiring probability p_{rw} to small values in the range of $[0.002, 0.04]$ and degree $d=8$. The network size is also 5000 peers. The finish time is then compared with that of network coding to evaluate how effectively the proposed algorithm assigns a small number of 250 encoders (excluding the source which always encodes) in such highly bottlenecked networks (Figure 4.12 and Figure 4.13).

Min-delay algorithm achieves maximum finish time within 10% of network coding's finish time using only a small portion of 5% total peers as encoders (Figure 4.12). With such small number of encoders, the average finish time of all peers is about 13% longer than network coding (Figure 4.12). For comparison, in these topologies whose rewiring probabilities are generally low, i.e. network bottlenecks are severe, with a small number of encoders, the finish time of degree-based placement, in some cases, however, is worse than network coding by 30–40%.

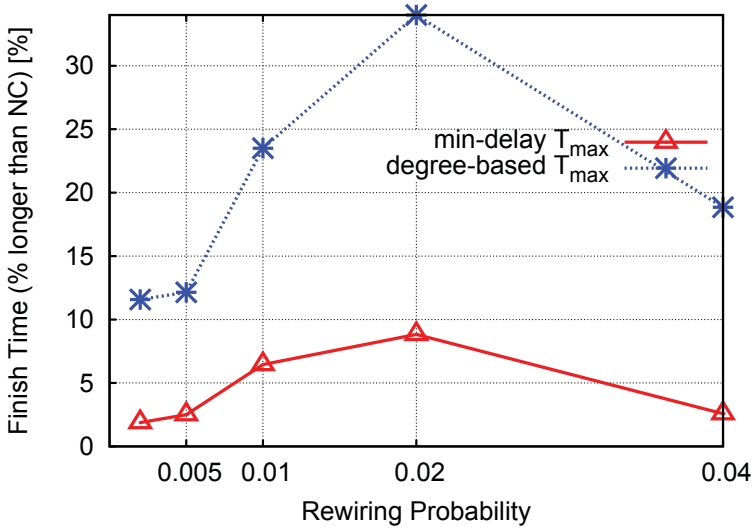

Figure 4.12: Maximum finish time of *min-delay* (newly proposed) and degree-based methods placing 250 encoders compared with *network coding* (NC).

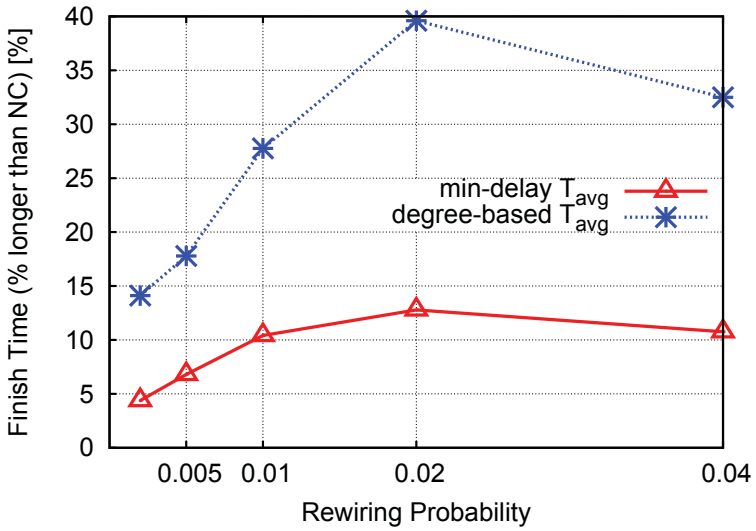

Figure 4.13: Average finish time of *min-delay* (newly proposed) and degree-based methods placing 250 encoders compared with *network coding* (NC).

In regular topologies ($p_{rw} \leq 0.002$) and highly random topologies (large p_{rw}) all algorithms have almost the same finish time as network coding because coding improvement over non-coding is marginal in those topologies.

We vary the number of encoders (excluding the source which always encodes) from 50 to 1000 in a 5000-peer network with $d = 8$ and $p_{rw} = 0.01$. Whereas random and degree-based placements achieve poor performance especially with small numbers of encoders, the proposed min-delay algorithm reaches finish time

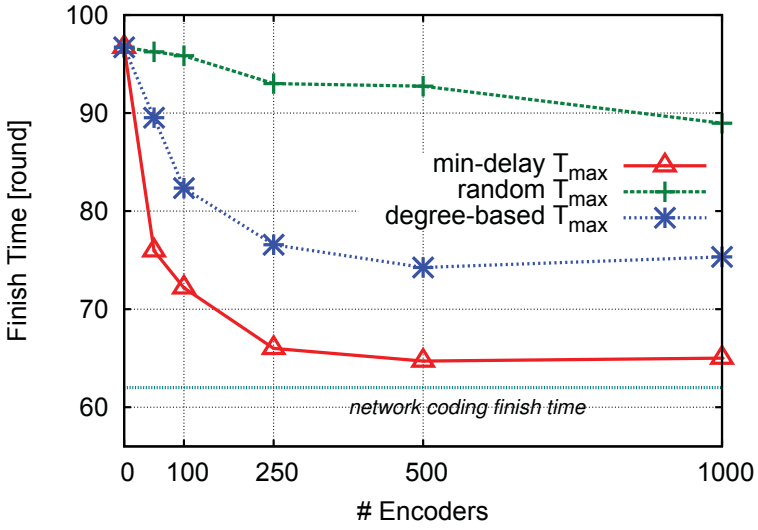

Figure 4.14: Maximum finish time of the newly proposed *min-delay* method compared with random, and degree-based encoder placement ($d = 8, p_{rw} = 0.01$).

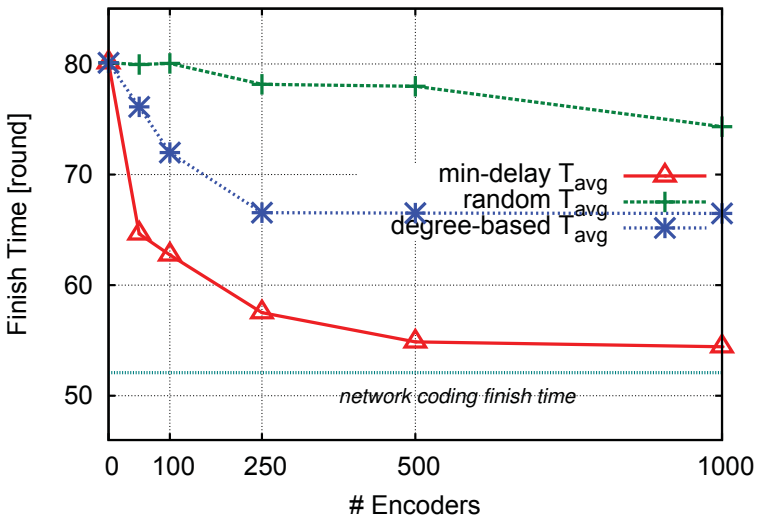

Figure 4.15: Average finish time of the newly proposed *min-delay* method compared with random, and degree-based encoder placement ($d = 8, p_{rw} = 0.01$).

comparable to that of network coding at 500 encoders (Figures 4.14 and 4.15). Deploying 1000 to 5000 encoders using the latter method, there is virtually no more improvement than using 500 encoders.[7]

[7]We only present results deploying 50–1000 encoders in Figures 4.14 and 4.15 to make the figures more focused. Increasing the number of encoders from 1000 to 5000, the finish time of min-delay placement is almost the same.

4.5 Discussion

In this chapter, we have investigated how network coding achieves its robust performance. One reason lies in its ability to eliminate data duplication throughout the network, yet using all peers as encoders. Coding at an upstream peer can eliminate data duplication, and in turn, shorten finish time of downstream peers located behind multiple paths from it. Data redundancy generated by a coder eliminates block duplication, which otherwise unnecessarily consume bandwidth resources and slow down content delivery over bottleneck paths. To reduce the number of encoders, we, therefore, look into the network topology and assign encoders only where large data duplication exists.

We analyze the amount of duplicated data coming to a node and the delay caused by that duplication, given surrounding topological parameters. We make an extensive analysis of how much duplication originated from a given upstream node using a probability model and then translate the duplication to delay in finish time the upstream node causes to its downstream nodes. Based on the analysis, we propose a novel algorithm, namely *min-delay* placement, to place coders within a P2P network to shorten distribution time.

Using the proposed algorithm, the content distribution has comparable performance to network coding, yet with far fewer coders, using much less computational resources compared to network coding which excessively codes everywhere. It can almost reach the performance of network coding in moderate bottlenecked topologies while substantially reducing the number of network coders. *Min-delay* placement demonstrates its full strength in severely bottlenecked topologies where it outperforms other methods we have considered and achieves performance closely matching that of network coding. Noticeably, min-delay placement can approach performance of network coding with just a small portion of encoders.

A limitation of *min-delay* placement is its high complexity $O(VE^3)$ where V is the number of nodes and E is the number of links in the network. When the

network is large or when coder placement is frequently recomputed, algorithms with lower complexity is desirable. We are therefore motivated to devise faster placement algorithms which we present in the next chapter.

One straightforward way to extend our proposed placement to the dynamic case, where peers keep joining and leaving the system, is to redeploy encoders periodically. Developing a distributed algorithm to figure the duplication and delay for coder assignment is also an interesting future work.

Chapter 5

Centrality-based Coder Placement

Minimal delay placement as presented in Chapter 4 can achieve good performance by precisely figuring how much delay an upstream node causes to its downstream nodes, and then, placing encoders at nodes which cause the most delay. Nevertheless, its good performance is accompanied by a high complexity of $O(VE^3)$. In this chapter, in order to reduce the complexity, we aim to find faster heuristic algorithms which can quickly pinpoint important nodes in the network to place network coders.

Our idea is to use *network centrality* [48, 49] as an indicator of where duplication occurs the most and place network coders there to eliminate such duplication.

The new placement algorithms, on the one hand, are derived from our observation that content duplication has close correlations both with the number of paths from an upstream node to a downstream node and with the size of the flows running over those paths. Coding at upstream peers with more and wider paths to other nodes can effectively eliminate content duplication to speed up content delivery. To identify nodes which lie on multiple and wider paths to other nodes to place network coders, our proposed method, on the other hand, exploits *be-*

tweenness centrality [48] and *flow centrality* [49] to quickly locate the desired key locations in the network.

In the following parts, we present the correlation analysis, and after that, our newly proposed centrality-based coder placements based on betweenness centrality and flow centrality.

5.1 Correlation of Duplication with Consisting Flows

BitTorrent P2P content distribution systems [5, 14] are receiver-driven. In such systems, peers choose blocks to download in a distributed manner based on their own perception that those blocks are rare in the neighborhood. Without a global knowledge, when there are multiple downstream paths to a particular node, some blocks are downloaded multiple times by upstream peers on those paths, which results in insufficiency of new information flow coming to the downstream node. Because of duplicated blocks, the downstream node cannot utilize its full downloading capacity. This duplication phenomenon, which we call *block duplication* and analyze in Chapter 4, has been illustrated in [1, 14]. Nevertheless, in this section, we distinctively figure the correlation of block duplication with the number of paths and the size of flows from a upstream peer to a downstream peer, which is the foundation of our newly proposed centrality-based coder placements.

A path, without circles or loops, from node i to node j is a sequence of nodes starting from i and terminating at j in which two adjacent nodes are connected by a link. A flow on a path from node i to node j is a mapping $E \rightarrow \mathbf{R}^+$ which conforms to capacity constraint of each link and flow conservation at each node on the path. A *max-flow* is the flow with maximum value. Figure 5.1 illustrates two paths connecting node i and node j: path 1 and path 2 with two respective flows of p_1 and p_2.

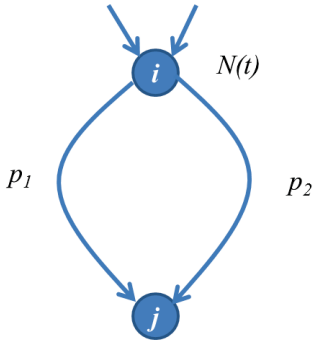

Figure 5.1: A partial graph where two paths connect node i and node j.

Denote $N(t)$ as the total number of blocks available at node i by time t. Since nodes on one path do not know which blocks have been chosen by nodes on the other path, we can assume blocks are picked up at random: $p_1 t$ random blocks are chosen from $N(t)$ to transmit on path 1, and likewise, $p_2 t$ random blocks are transmitted on path 2 by time t. The expected number of duplicated blocks transmitting on the two paths, therefore, is $\frac{p_1 t . p_2 t}{N(t)}$.

The total number of non-duplicated blocks from node i which are delivered to node j by time t is

$$a(t) = p_1 t + p_2 t - \frac{p_1 p_2 t^2}{N(t)}. \tag{5.1}$$

Let s_i be the rate at which blocks coming to node i. We have the number of blocks available at node i by time t: $N(t) = s_i t$. From (5.1), the effective throughput (averaged over time t) from node i to node j is

$$\begin{aligned} p_{eff} &= \frac{a(t)}{t} \\ &= p_1 + p_2 - \frac{p_1 p_2}{s_i}. \end{aligned} \tag{5.2}$$

By the same reasoning, (5.2) can be generalized to get the effective bandwidth

in case there are m paths connecting node i and node j

$$p_{eff} = p_1 + p_2 + .. + p_m - \frac{p_1p_2 + p_1p_3 + .. + p_{m-1}p_m}{s_i} + \frac{p_1p_2p_3 + .. + p_{m-2}p_{m-1}p_m}{s_i^2} - .. - (-1)^m \frac{p_1p_2..p_m}{s_i^{m-1}}. \tag{5.3}$$

Equation (5.3) reveals that due to duplicated blocks on the paths, the effective throughput p_{eff} is smaller than the total flows on all paths from node i to node j:

$$p_{eff} = p_1 + p_2 + .. + p_m - r \tag{5.4}$$

where $r > 0$ is the duplication rate.

From (5.3) and (5.4), we have

$$r = \frac{p_1p_2 + p_1p_3 + .. + p_{m-1}p_m}{s_i} - \frac{p_1p_2p_3 + .. + p_{m-2}p_{m-1}p_m}{s_i^2} + .. + (-1)^m \frac{p_1p_2..p_m}{s_i^{m-1}}. \tag{5.5}$$

There are two observations on the correlations of duplication rate r with consisting flows which contribute to the creation of our coder placement algorithms.

First, duplication rate is higher with larger consisting flows. If we consider a given flow p_i separately and fix all other flows, (5.5) can be converted to

$$r = A_ip_i + B_i \tag{5.6}$$

where A_i and B_i are independent from p_i, and $A_i > 0$, $B_i > 0$. Equation (5.6) shows the correlation of duplication rate and each separate flow from node i to node j: when a given flow p_i increases, duplication rate r also increases.

Second, duplication rate is higher if there are more flows from node i to node j. Let $r(m)$ and $p_{eff}(m)$ respectively be the duplication rate and effective throughput

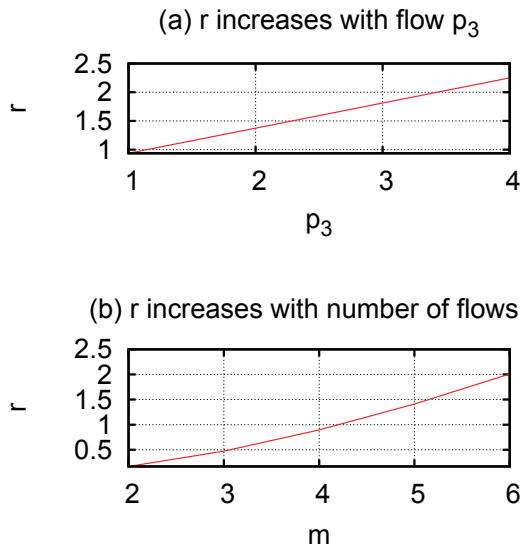

Figure 5.2: Duplication rate increases with flow size and number of flows.

with m flows from node i to node j: $p_1, p_2, .., p_m$ and $r(m+1)$ be the duplication rate when a new flow p_{m+1} is added. It is easy to see that

$$r(m+1) = r(m) + \frac{p_{eff}(m)p_{m+1}}{s_i} \tag{5.7}$$

which means $r(m+1) > r(m)$. Therefore, we have

$$r(l) > r(m) \qquad \forall l > m. \tag{5.8}$$

We illustrate the correlation in Figure 5.2(a) when there are 3 flows p_1, p_2, and p_3 from node i to node j: $s_i = 8$, $p_1 = p_2 = 2$ and p_3 changes from 1 to 4. In Figure 5.2(b), we fix $s_i = 6$, $p_1 = p_2 = 1$ and add more flows with bandwidth equal to 1 to change the number of flows m from 2 to 6.

If a network coder is placed at upstream node i, there are no duplicated blocks transferring on the paths to downstream node j because each coded block is unique. As a result, the duplication rate $r = 0$ when node i encodes.

Algorithm 5.1: Multi-path Coder Placement Algorithm

Input: $G = \{V, E\}$, the source (node 0), number C
Result: C encoders ($1 \leq C \leq |V|$)

```
forall the i ∈ V do
    R[i] = 0;
end
foreach j ∈ V\{0} do
    figure all paths 0 → j;
    foreach i, i on multiple paths 0 → j do
        compute r_i using (5.5);
        R[i] = R[i] + r_i;
    end
end
Encoders ← C peers with highest R[·] values;
return Encoders;
```

5.2 Coding at Network Centrality

Network coding, by generating new coded blocks, can eliminate block duplication, and thus, achieve high throughput and short distribution time. Since block duplication happens on multiple delivery paths, to optimize the whole system, our job is to find a set of C nodes, from which duplicated blocks slow down throughput to other nodes the most, where C is the number of network coders to be deployed. One such algorithm (Algorithm 5.1), which figures all possible path from the source to a downstream node, is given for reference.[1] Algorithm 5.1's running time, however, is prohibitive due to the exponential number of paths.[2]

We propose a heuristic approach instead by quickly looking for nodes which lie on more paths with wider bandwidth. By ensuring that chosen coders lie on paths to many nodes with wider bandwidth we can avoid duplicated blocks

[1]The main difference between Algorithm 5.1 and Algorithm 4.1 in Chapter 4 is that the latter only considers paths with shortest lengths from the source to a given node, which is tractable. Also Algorithm 4.1 concerns with the delay an upstream node causes to its downstream nodes due to duplication while Algorithm 5.1 works with the duplication itself.

[2]The number of paths between two nodes, in general, depends on the topology. In a full mesh topology with V nodes, for example, the number of paths between any two given nodes is $\sum_{k=0}^{V-2} \frac{(V-2)!}{(V-2-k)!}$.

Algorithm 5.2: Centrality-based Coder Placement Algorithm

Input: $G = \{V, E\}$, the source (node 0), number C
Result: C encoders ($1 \leq C \leq |V|$)

foreach $i \in V \backslash \{0\}$ **do**
    compute *centrality* index $C[i]$ using either
        1. Brandes's *betweenness centrality* algorithm [59];
        2. *flow centrality* algorithm (Algorithm 5.3);
end
Encoders $\leftarrow$ C peers with highest $C[\cdot]$ values;
return Encoders;

from transferring to those nodes as shown in Section 5.1. Our idea is to use *network centrality* indices (we will explain shortly) to evaluate how important a node to other nodes if that node becomes a coder. The higher a node's centrality value, the more paths or wider paths between other nodes it stands on, and the more appraisable for it to become a coder. Algorithm 5.2 summarizes our coder placement strategy. Note that the source is always chosen as a coder because it stands on paths to every peer. The algorithm utilizes either *betweenness centrality* [48], or its variant, *flow centrality* [49] to figure the importance of a given network node which we explain in the following section.

Originated in social networks studies, centrality is an essential tool for graph analysis which measures the importance of a node within the graph. Depending on the kind of measures, there are various centrality indices. In this study, however, we are interested in betweenness centrality and flow centrality. Betweenness centrality expresses the degree a node locates on the paths between other nodes, and flow centrality expresses the total bandwidth of flows going through a node. That is, betweenness centrality and flow centrality are indicators of nodes from which duplication occurs. Given that network coding, by generating fresh encoded data to send onto each path, can effectively eliminate such duplication to increase effective bandwidth, placing coders in nodes with high centrality values will speed up content delivery.

5.3 Betweenness Centrality Placement

Betweenness centrality [48] measures the degree that a node stands on the shortest paths between other nodes, which has been applied in different contexts such as routing and cache placement [60, 61] to place a network function in a set of nodes. Since we are interested in distributing data from the source to all peers, all shortest paths under consideration are from the source to other nodes.

Denote σ_k as the number of the shortest paths from the source to node k and $\sigma_k(i)$ as the number of the shortest paths from the source to node k which go through node i. Betweenness centrality of node i is measured by

$$C_B(i) = \sum_{k \neq i \in V} \frac{\sigma_k(i)}{\sigma_k}. \tag{5.9}$$

Nodes with high betweenness centrality locate on more shortest paths to other downstream nodes, and thus, likely generate more duplication as we observe in Section 5.1. If those nodes encode, more duplication can be avoided to speed up content distribution. The limitation of betweenness centrality is that it only considers the number of shortest paths to downstream nodes which does not always reflect correctly the importance of a node in eliminating duplication.

Betweenness centrality of all nodes can be computed with Brandes's $O(VE)$ algorithm [59]. Since we are only interested in shortest paths from the source, the complexity is $O(E)$.

5.4 Flow Centrality Placement

Flow centrality [49], on the other hand, measures the portions of *max-flows* between all pairs of other nodes which go through a given intermediate node. Like betweenness centrality, we are interested in flows from the source to all peers. In our study, we compute flow centrality of node i as the total amount of flows from

the source to all other node k which pass through node i:

$$C_F(i) = \sum_{k \neq i \in V} f(S, i, k). \tag{5.10}$$

where S is the source, and $f(S, i, k)$ is the portion of *max-flow* from the source to node k that passes through node i. As we are interested in both the value of the flow and the shortness of the path, the *max-flow* mentioned above is the one consists of paths with shortest lengths among all paths from the source S to node k.[3]

High flow centrality nodes have larger aggregate flows to downstream nodes. They, therefore, have high probability to stand on more paths to a given downstream node, and in addition, the flows on the paths are likely larger. Since larger duplication results from larger flows as stated in Section 5.1, we expect high flow centrality nodes to generate more duplication to downstream nodes, which justifies the need to place encoders there.

We compute flow centrality using Algorithm 5.3 which is basically Edmonds-Karp's max-flow algorithm [58] with the addition of line 38 where the augmenting flow value (*flow*) is updated to the flow centrality $C_F[i]$ of each node i on that augmenting path.

Flow centrality, however, is more expensive to compute than betweenness centrality. In Algorithm 5.3, the complexity to visit each source-sink pair is $O(VE^2)$. Therefore, to find out flow centrality of all V nodes, the algorithm takes $O(V^2E^2)$ time.

[3]This kind of flow centrality is slightly different from what has been originally proposed in [49] where $f(S, i, k)$ means the portion of *max-flow* from node S to node k that must pass through node i in order that node k achieves its *max-flow*. In other words, if node i is removed, the *max-flow* from node S to node k decreases by $f(S, i, k)$.

Algorithm 5.3: Flow Centrality Computation

Input: $G = \{V, E\}$, the source (node 0)
Result: *flow centrality* $C_F[i] \forall i \in V$

```
1  forall the i, j ∈ V do
2  |  C_F[i] = 0;
3  |  u[i, j]=capacity of link (i, j), (i, j) ∈ E;
4  end
5  s = 0;
6  foreach t ∈ V do
7  |  forall the (i, j) ∈ E do
8  |  |  f[i, j] = 0;
9  |  end
10 |  repeat
11 |  |  Q ← empty queue;
12 |  |  prev[0] = −1;
13 |  |  forall the i ∈ V do
14 |  |  |  color[i]= not visited;
15 |  |  end
16 |  |  enqueue s → Q;
17 |  |  while Q not empty do
18 |  |  |  dequeue i → Q;
19 |  |  |  color[i]= visited;
20 |  |  |  foreach neighbor j of i do
21 |  |  |  |  if color[j]= not visited and u[i, j] − f[i, j] > 0 then
22 |  |  |  |  |  color[j]= queued;
23 |  |  |  |  |  prev[j] = i;
24 |  |  |  |  |  enqueue j → Q;
25 |  |  |  |  end
26 |  |  |  end
27 |  |  end
28 |  |  if color[t]== visited then
29 |  |  |  for j ← t, prev[j] ≥ 0 do
30 |  |  |  |  i = prev[j];
31 |  |  |  |  flow = min(u[i, j] − f[i, j]);
32 |  |  |  |  j ← prev[j];
33 |  |  |  end
34 |  |  |  for j ← t, prev[j] ≥ 0 do
35 |  |  |  |  i = prev[j];
36 |  |  |  |  f[i, j] = f[i, j] + flow;
37 |  |  |  |  f[j, i] = f[j, i] − flow;
38 |  |  |  |  C_F[i] = C_F[i] + flow;
39 |  |  |  |  j ← prev[j];
40 |  |  |  end
41 |  |  end
42 |  until color[t]==visited;
43 end
44 return C_F[·];
```

5.5 Performance Evaluation

We create a round-based simulator in C++ to verify the effectiveness of our proposed algorithm. The simulation settings are the same as in Section 3.5 and Section 4.5. Peers exchange information using pre-code protocol in Section 3.2.2. In the beginning of each round, blocks are assigned to each peer according to available bandwidth, our proposed block selection in Section 3.3 and the *mutual exchange* incentive scheme described in Section 3.5. The assigned blocks are downloaded by the receivers by the end of the round and then the system moves to next round.

A link capacity is measured by block per round, i.e. how many blocks can be transferred through the link in a round. Finite field $GF(2^8)$ is used for encoding and decoding operations. We disregard the negligible overhead of sending encoding coefficients associated with random linear coding in our simulations.

As in Section 3.5.2, we generate 5000-peer overlay topologies using Watts and Strogatz *small-world network* model [55]. By changing the rewiring probability p_{rw} we can generate a wide range of topologies from regular ones (low p_{rw} values) to random ones (high p_{rw} values). When rewiring probability is low, the topology is highly bottlenecked since there are fewer long-distance links connecting parts of the network as we have discussed in Section 3.5.2. All overlay links have capacity of 1 block per round. We run simulations 100 times on each generated topology collecting the average finish time of all peers (T_{avg}) and the maximum finish time among all peers (T_{max}) distributing a 200-block file.

Using the same simulation parameters, the finish time of betweenness centrality and flow centrality coder placements are compared with network coding, i.e. full encoder deployment, and min-delay placement which we previously discussed in Chapter 4. The performance of degree-based coder placement, i.e. coders are placed at high-degree nodes, is also included for comparison.

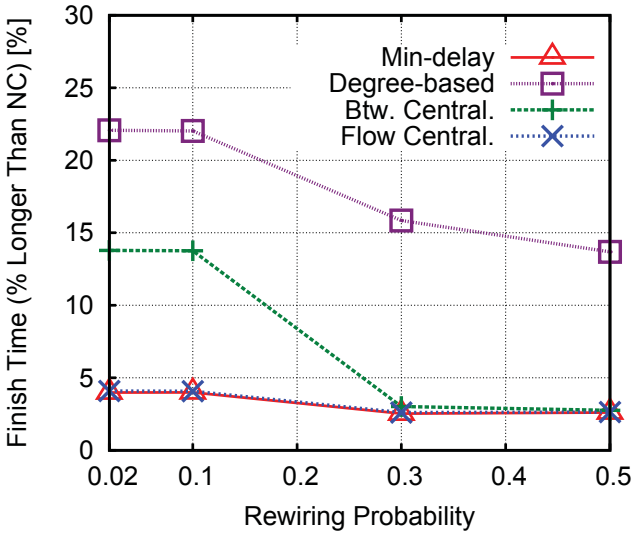

Figure 5.3: Average finish time of *betweenness centrality* and *flow centrality* placements deploying 1000 encoders in 5000-peer topologies compared with *network coding* (NC). The performance of min-delay and degree-based placements are given for comparison.

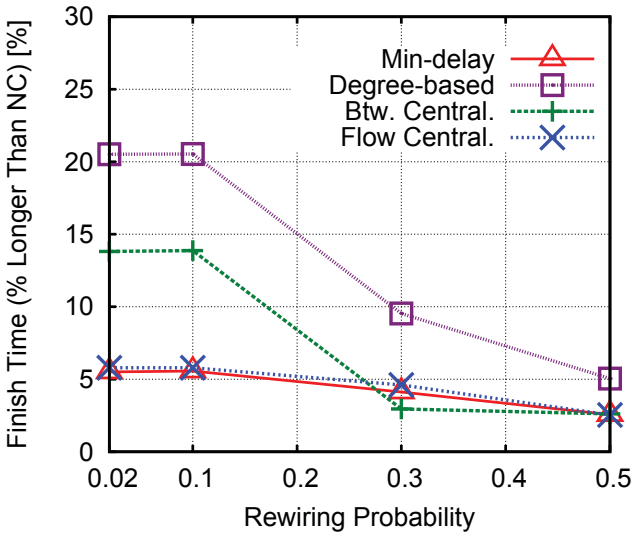

Figure 5.4: Maximum finish time of *betweenness centrality* and *flow centrality* placements deploying 1000 encoders in 5000-peer topologies compared with *network coding* (NC). The performance of min-delay and degree-based placements are given for comparison.

5.5.1 Performance in Moderate Bottlenecked Topologies

We compare the performance of the proposed heuristic placements with full network coding in Figure 5.3 for average finish time of all peers and in Figure 5.4 for maximum finish time among all peers. In moderate bottlenecked topologies

($0.02 \leq p_{rw} \leq 0.5$, $d = 6$), assigning C=1000 peers with highest flow centrality as coders, we can achieve comparable performance to network coding. The average finish time of all peers is approximately under 5% longer than network coding. Using betweenness centrality to appoint the same number of coders with highest betweenness centrality values, the finish time is nearly 15% longer than network coding when the topologies have lower rewiring probability (p_{rw}=0.02 and p_{rw}=0.1) and 5% longer than network coding with higher rewiring probabilities.

The reason for flow centrality's good performance is that, by taking max-flow into account, it reflects more accurately the characteristics of the topology than betweenness centrality. The complexity of flow centrality is, however, higher than betweenness centrality. For comparison, degree-based placement, i.e. encoders are placed at high-degree nodes first, has poor performance. We notice that the performance gain due to coding is negligible (even with full network coding) in regular topologies (p_{rw}=0) and highly random topologies ($p_{rw} > 0.5$) because there are virtually no bottlenecks in such networks.

Flow centrality's performance in moderate bottlenecked topologies is almost the same as min-delay placement (Figures 5.3 and 5.4). Given its lower complexity, flow centrality placement algorithm is certainly preferable in such networks.

We next change parameter C to appoint different numbers of peers with highest centrality values as coders in a topology with 5000 peers and rewiring probability p_{rw}=0.02, degree d=6. Figure 5.5 and Figure 5.6 respectively give the average and maximum finish time when no coders (no coding) to 5000 coders (full network coding)[4] are deployed. We also include the finish time of random placement, which assigns the encoders at random, and degree-based placement for reference. Betweenness centrality and flow centrality placements always achieve improved performance compared with degree-based and random placements. Of the two for-

[4]Using a given placement method, increasing the number of encoders to 5000 means that all peers in the network encode, i.e. network coding. Therefore, in Figure 5.5 and Figure 5.6, finish time is the same for all placement methods when the number of encoders is 5000.

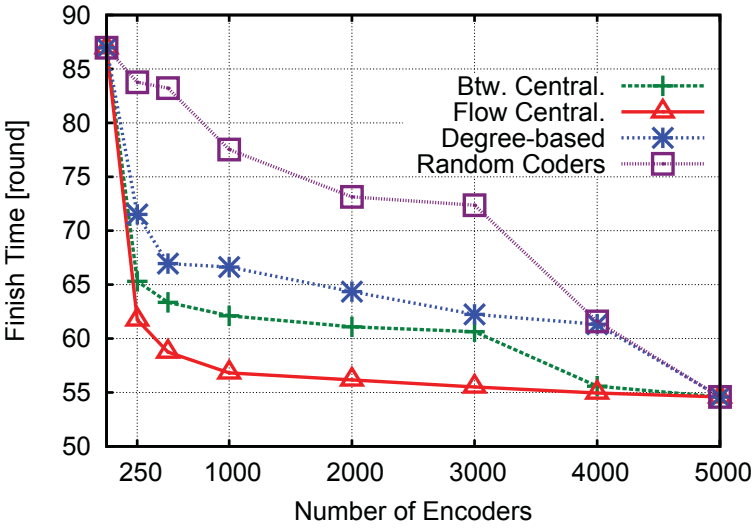

Figure 5.5: Average finish time of *betweenness centrality* and *flow centrality* placements varying the number of encoders compared with degree-based and random placements.

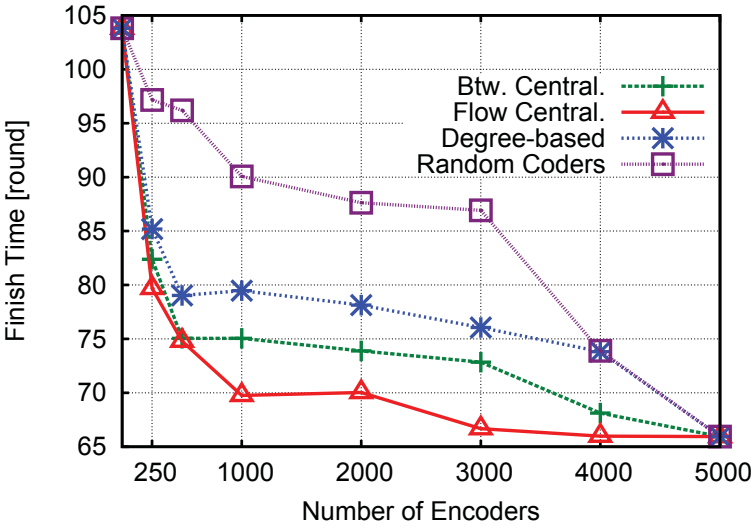

Figure 5.6: Maximum finish time of *betweenness centrality* and *flow centrality* placements varying the number of encoders compared with degree-based and random placements.

mer methods, flow centrality reaches finish time almost equal to network coding with only 1000 encoders and the performance is nearly constant when we increase the number of coders from 1000 to 5000 (Figure 5.5). The result confirms that centrality is good tool to locate a small subset of important nodes for coder placement.

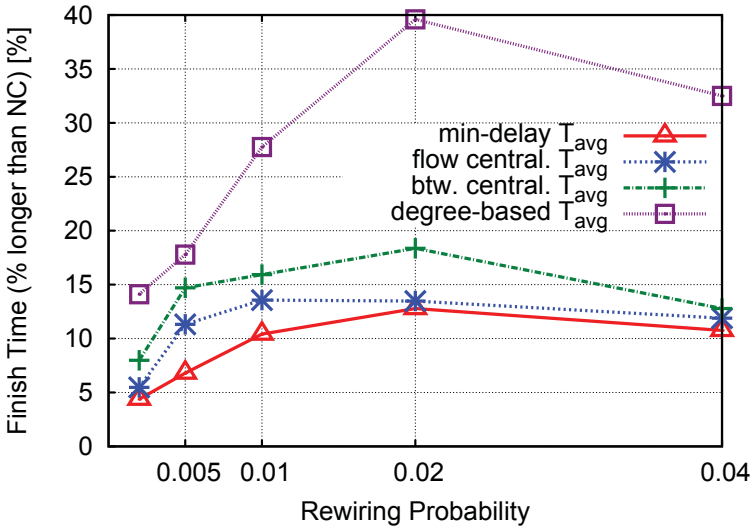

Figure 5.7: Average finish time of *betweenness centrality* and *flow centrality* placements deploying 250 encoders in 5000-peer topologies compared with *network coding* (NC). The performance of min-delay and degree-based placements are given for comparison.

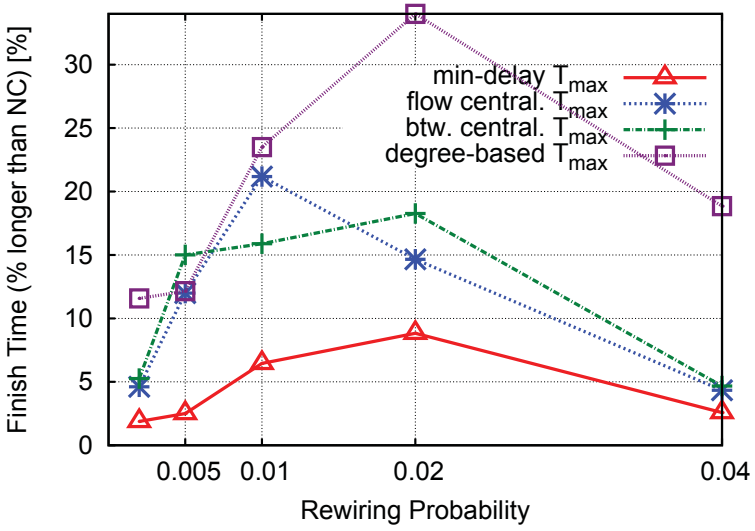

Figure 5.8: Maximum finish time of *betweenness centrality* and *flow centrality* placements deploying 250 encoders in 5000-peer topologies compared with *network coding* (NC). The performance of min-delay and degree-based placements are given for comparison.

5.5.2 Performance in Highly Bottlenecked Topologies

We use low rewiring probability ($0.002 \leq p_{rw} \leq 0.04$) and degree $d = 8$ to generate highly bottlenecked topologies to verify performance of betweenness centrality and flow centrality placements compared with other methods. The average finish time

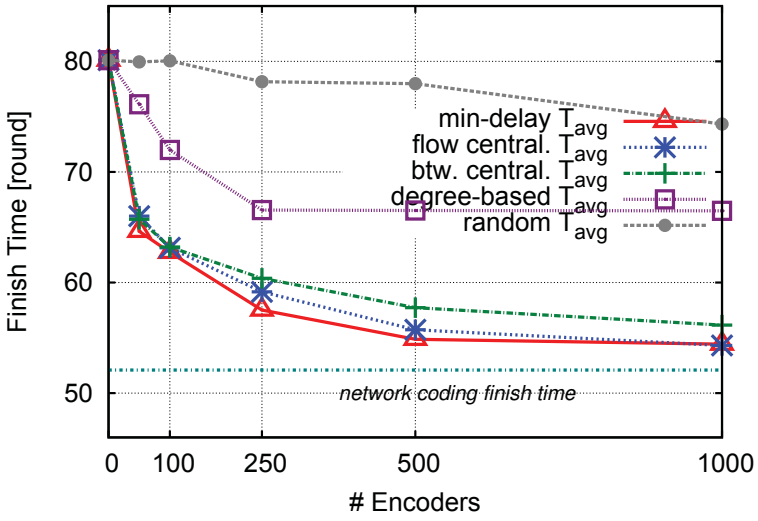

Figure 5.9: Average finish time of *betweenness centrality* and *flow centrality* placements varying the number of encoders in a topology with rewiring probability p_{rw}=0.01 and degree d=8.

and maximum finish time are given in Figure 5.7 and Figure 5.8 respectively using 250 peers as network coders.

Betweenness centrality and flow centrality placements result in average finish time close to min-delay placement (Figure 5.7). In terms of maximum finish time, however, they could not match the performance of min-delay placement (Figure 5.8) which can achieve finish time closer to full network coding. We note that since network coders are placed at a very small portion of 5% of the total peers, the trade-off are longer finish time compared with full network coding which uses all peers as coders.

We increase the number of network coders C from 50 to 1000 coders to see the impact on finish time. With C=500 network coders, flow centrality placement achieves almost the same finish time as min-delay placement which is closely approaches full network coding's performance in terms of both average finish time (Figure 5.9) and maximum finish time (Figure 5.10). Finish time of betweenness centrality placement, however, is almost always longer than flow centrality placement.

Flow centrality and betweenness centrality placements are less robust than min-

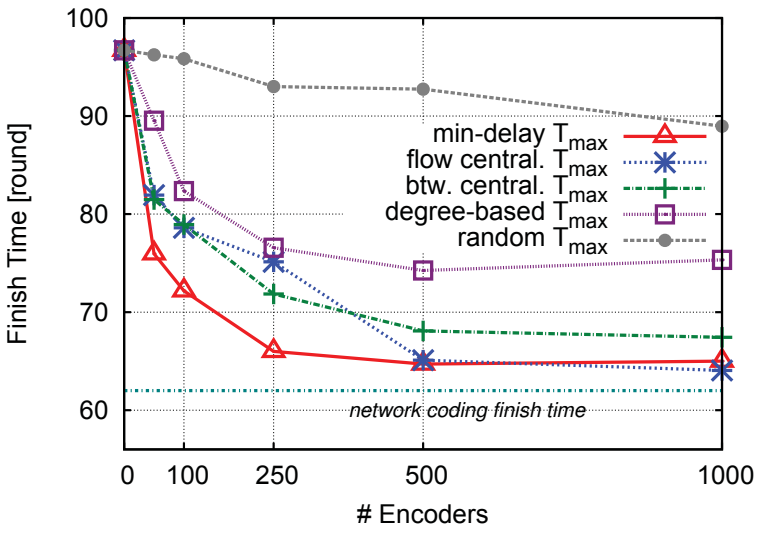

Figure 5.10: Maximum finish time of *betweenness centrality* and *flow centrality* placements varying the number of encoders in a topology with rewiring probability p_{rw}=0.01 and degree d=8.

delay placement in assigning a small number of 50 to 250 encoders with consistent shorter maximum finish time (Figure 5.10). The reason is because min-delay placement accelerates *slow* peers by assigning encoders to shorten their finish time as we have disccused in Chapter 4's Eq. 4.11. The average finish time of the three methods, however, is almost the same (Fig 5.9) since the decrease in finish time of those slow peers is averaged over the total number of peers.

5.5.3 Performance with Different Centrality Thresholds

We furthermore evaluate the placement method in terms of total number of required encoders and average finish time, varying the centrality threshold. With each threshold, all nodes with centrality value higher than or equal to the threshold are chosen as encoders. The results are given in Figure 5.11 and Figure 5.12 in which threshold α=0 means all nodes are chosen as encoders, i.e. full network coding, and threshold infinity means no nodes code, i.e. no coding. In the given topology (d=6, p_{rw}=0.02), choosing all nodes with betweenness centrality higher or equal to α_b=10, which means the total fractions of shortest paths to downstream peers the selected encoders locate on are equal to or greater than 10, gives

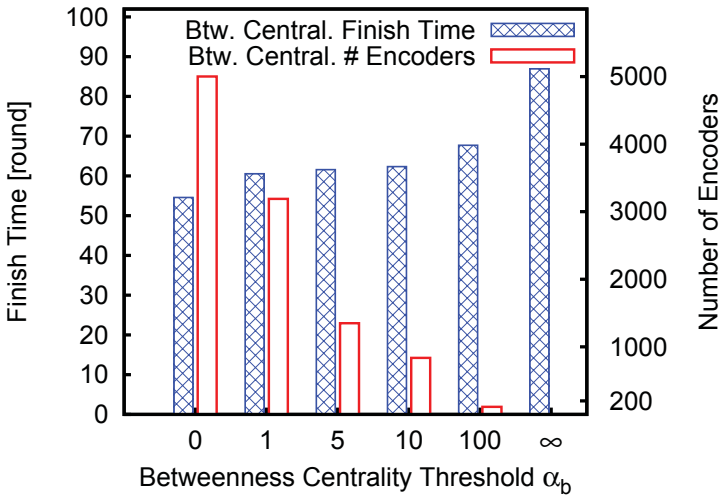

Figure 5.11: Average finish time and number of assigned encoders with different betweenness centrality thresholds.

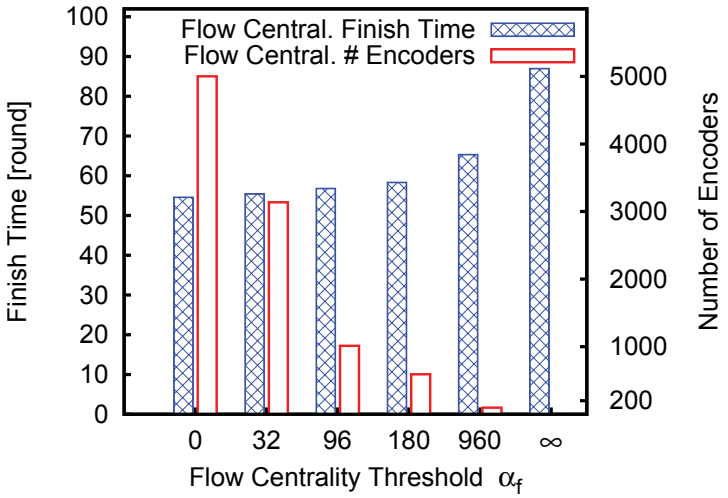

Figure 5.12: Average finish time and number of assigned encoders with different flow centrality thresholds.

an average finish time 14% longer than network coding, however, with a saving of nearly 85% in the number of required encoders compared to network coding (Figure 5.11).

The performance is better using flow centrality. When threshold is set to $\alpha_f = 180$, i.e. each chosen encoder stands on a total flow of 180 to its downstream peers, with nearly 90% saving in encoders, flow centrality placement achieves short finish time, just 7% longer than network coding finish time (Figure 5.12). With

lower thresholds, i.e. more nodes are chosen as encoders, there is not much improvement in finish time, even though the number of encoders is much higher. The result means that while encoders at high centrality nodes can effectively improve performance, those at low centrality nodes are redundant and can be removed to save resources.

5.6 Discussion

In this chapter, we have proposed heuristic algorithms to place coders within a P2P network to shorten distribution time. Unlike previous related work which justifies coding over the whole network topology, our algorithms, in evaluating the centrality value of each node within the topology, look inside the network to find particular places which require network coding. The idea works on the basis that coding at an upstream peer can improve data transmission on multiple paths to downstream peers located behind bottlenecks. Data redundancy generated by a coder eliminates duplicated downloads, which otherwise unnecessarily consume bandwidth resources and slow down content delivery over the paths. By taking advantages of the correlations of duplication with the number and the size of the paths, we can effectively using betweenness centrality and flow centrality to pinpoint the network nodes which generate more duplication for network coder placement.

We have confirmed that betweenness centrality and flow centrality are good indicators to locate important nodes, which lie on multiple paths to other nodes, in a network and deploy them as coders. Flow centrality, by taking flow information into account, achieves a performance closely matched that of full network coding. Betweenness centrality placement, although less effective, has the advantage of much lower complexity which is suitable when encoder placement is frequently computed.

The proposed centrality-based placements have performance comparable to

min-delay placement (which we have presented in Chapter 4) and quite close to network coding performance in moderate bottlenecked topologies. In highly bottlenecked topologies, their performance in terms of average finish time is as good as min-delay placement. However, the maximum finish time is longer than min-delay placement when only a small number of encoders are deployed.

The advantage of centrality-based coder placements is their lower complexity compared with min-delay placement. In practice, in very large network topologies, or in situations where network coder placement are frequently computed, centrality-based algorithms are better choices since they incur less computation time. When performance is preferred, min-delay placement will take the lead.

We can extend our centrality-based placements to the dynamic case, where peers keep joining and leaving the system, is to redeploy encoders periodically. A more elegant extension, however, is to use a distributed approximation algorithm, such as [62, 63], to compute the centrality value at each peer.

Using the method in [62], for example, an alternative centrality value called second order centrality is computed by letting each node keep track of the time elapsed between visits by a random walk. High centrality nodes see more frequent visits compared with other nodes. The approach in [63], on the other hand, figures the centrality level of a node by means of a localized spectral analysis on a small-size neighborhood of the node. In addition, to make the decision whether a node should become a network coder or not fully distributed, the centrality threshold also needs to be determined locally at each node.

Looking at another facet, the performance and the number of network coders depend on the centrality threshold we choose. Lower thresholds result in short finish time with the cost of more network coders. A good centrality threshold is largely controlled by the characteristics of the network topology. Determining such appropriate thresholds is an interesting problem which we leave for our future work.

Chapter 6

Coding Redundancy Ratio

Having located where to place network coders in a given network, in this chapter, we further optimize each network coder itself: we determine the optimal level of redundancy generated by a network encoder to achieve shortest finish time. Each time a network coder generate a new coded data, it put a piece of redundant information into the network which helps accelerate content distribution. However, requiring an encoder to encode all the time is excessive. The level of redundancy generated at each network coder should be carefully considered so that to avoid unnecessary encoding as it consumes the node's resources.

We start with a description of the problem, and then, make an analysis of the optimal redundancy ratio at each network coder. Based on the analysis, we devise an algorithm to compute redundancy ratios of all network coders and evaluate the performance by simulations.

6.1 Redundancy Ratio at a Network Coder

When network coding is enabled at a peer, the peer generates new encoded blocks from what it has received before sending to other peers. Suppose we have a coded

block C_0 with *encoding vector* $(c_{01}, c_{02}, ..., c_{0K})$, and K original blocks, B_1, B_2, ..., B_K. That means

$$C_0 = c_{01}B_1 + c_{02}B_2 + ... + c_{0K}B_K.$$

The coefficients, multiplications, and additions are taken place in a finite field, e.g. $GF(2^8)$.

Now suppose encoder i, having received 2 blocks C_1 and C_2, wants to make a new encoded block to send to a neighboring peer. Using random linear coding [26], encoder i will pick up two random coefficients a_1 and a_2 and generate a new coded block C: $C = a_1C_1 + a_2C_2$, which results in an encoded block with encoding vector $(a_1c_{11} + a_2c_{21}, a_1c_{12} + a_2c_{22}, ..., a_1c_{1K} + a_2c_{2K})$.

The ratio of the number of encoded blocks network coder i generates, namely $enc(i)$, to the number of blocks it received, $recv(i)$, is called the *redundancy ratio* or *expansion factor* at network coder i:

$$e_i = \frac{enc(i)}{recv(i)}. \tag{6.1}$$

After a peer collects K independent blocks (both encoded and original), i.e. the K associated encoding vectors form a full-rank matrix, it can decode to get the K original blocks by solving the set of K linear equations.

6.2 Problem Formulation

Peers in a P2P network form a directed overlay topology, i.e. directed graph $G = \{V, E\}$ where V is the set of peers, or nodes, and E is the set of directed overlay links between peers. Pieces of the file, i.e. *blocks*, are then exchanged between connecting peers, i.e. *neighbors*. A peer finishes when it collects K blocks which form a full-rank matrix. Our problem can be stated as follows.

Table 6.1: Notations

Notation	**Meaning**
$1, 2, ..., n$	Peer IDs. (The source is denoted as S.)
$c(i, j)$	Capacity of overlay link (i, j)
$f(i)$	Max-flow from the source to peer i which is determined by max-flow min-cut theorem [58].
$f(i, j)$	Total throughput from peer i to neighboring peer j over both the direct link (i, j) and the indirect path from i to j. Abbreviated as f_j when i is understood.
$s(i, j)$	Throughput over the direct link (i, j) from peer i to neighboring peer j. Abbreviated as s_j when i is understood.
$b(i, j)$	Bottleneck bandwidth from peer i to peer j, the two peers might not be neighbors. Abbreviated as b_j when i is understood.
K	Number of original blocks
T_i	Shortest finish time of peer i: $T_i = \frac{K}{f(i)}$.
e_i	Redundancy ratio at encoder i

Given a P2P content distribution which is defined by

- a network topology $G = \{V, E\}$,
- a source $S \in V$ with a file of K blocks to be distributed to all peers, and
- a set of encoders C ($C \subseteq V$),

what is the best redundancy ratio at each encoder which shortens distribution time the most?

The notations in Table 6.1 are used in our analysis. Our strategy is to determine the best redundancy ratio at each encoder separately based on the given network topology. We present the detail analysis in the next section.

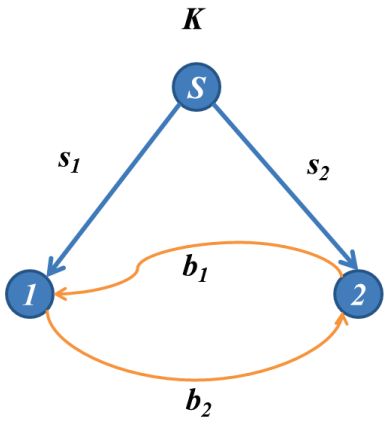

Figure 6.1: The encoder is placed at the source S which is sending data to two neighboring peers: peer 1 and peer 2. Bandwidth utilization over the link from the source to the two peers is s_1 and s_2 respectively. The two peers might not directly connect, i.e. not a neighbor of each other. Bottleneck bandwidth from peer 2 to peer 1 is b_1 and that on the reverse direction is b_2.

6.3 Redundancy Ratio Analysis

6.3.1 Encoder at the Source

The source has two neighboring peers

We first analyze the source's redundancy ratio in a simple topology (Figure 6.1) where the source distributes a file to two receivers: peer 1 and peer 2. An encoder is placed at the source to shorten finish time of the two peers, i.e. the time required for them to download the whole file.

We assume a static network, i.e. there is no change in the physical topology and the overlay topology during a content distribution session as have been stated in Chapter 2. A peer stops downloading when the necessary number of blocks is acquired but keeps staying in the system to serve other peers.

Consider one of the peers, for example peer 2. There are two paths over which content are delivered to it: directly from the source and from the source via peer 1 to peer 2. In ordinary non-coding P2P content distribution, since peers select blocks to download independently based on the local information in the neighborhood, a considerable number of identical blocks are transferred on both paths.

That duplication phenomenon also happens when the source, after sending all K blocks, has to *resend* old blocks into the system. Block duplication results in insufficiency of new information flow coming to peer 2. Consequently, peer 2 cannot utilize its full download capacity.

When the source is allowed to encode, in order for peer 2 to achieve its full download speed, i.e. shortest finish time, the source should generate enough redundancy so that all blocks transferred on one path to peer 2 are different from those on the other path. Denote s_1 and s_2 as the bandwidth utilization over the link from the source to peer 1 and peer 2 respectively. Bottleneck bandwidth from peer 2 to peer 1 is b_1 and that on the reverse direction is b_2. Let T_1 and T_2 be the shortest finish time of peer 1 and peer 2 respectively. That is

$$T_1 = \frac{K}{s_1 + b_1}, \text{and}$$
$$T_2 = \frac{K}{s_2 + b_2}$$

where $s_1 + b_1$ and $s_2 + b_2$ are the throughput from the source to peer 1 and peer 2 respectively and K is the number of original blocks at the source.

Without loss of generality, assume peer 2 is the slower peer, i.e. it finishes after peer 1: $T_2 \geq T_1$. By the time T_2 when peer 2 finishes downloading the whole file, the source has generated and sent

- $s_1 T_1$ encoded blocks to peer 1, and
- $s_2 T_2$ encoded blocks to peer 2.

The total number of encoded blocks the source generated, therefore, is

$$\begin{aligned} enc(S) &= s_1 T_1 + s_2 T_2 \\ &= s_1 . \frac{K}{s_1 + b_1} + s_2 . \frac{K}{s_2 + b_2}. \end{aligned}$$

Since the number of original blocks at the source is K, we have the redundancy ratio at the source:

$$\begin{aligned} e_S &= \frac{enc(S)}{K} \\ &= \frac{s_1}{s_1 + b_1} + \frac{s_2}{s_2 + b_2}. \end{aligned} \tag{6.2}$$

Redundancy ratio smaller than the one given in (6.2) results in some blocks are transfer on both paths, and consequently, peer 2 could under-utilize its download capacity. Larger redundancy ratio, on the other hand, has no effect on the finish time since T_2 is the shortest achievable finish time for peer 2.

The source has m neighboring peers

The result in (6.2) can be extended to the case when the source connects to m neighboring peers: peer 1, peer 2, ..., peer m. We have

$$\begin{aligned} e_S &= \sum_{j=1}^{m} \frac{s_j}{s_j + b_j} \\ &= \sum_{j=1}^{m} \frac{s_j}{f(j)} \end{aligned} \tag{6.3}$$

where $f(j)$ is the total throughput, i.e. *max-flow*, from the source to peer j.

6.3.2 Encoder at an Intermediate Peer

An encoder is placed at an intermediate peer i which delivers blocks to m neighboring peers (Figure 6.2 illustrates the case $m = 2$). The difference in this case is that peer 1, peer 2, ..., and peer m do not need to download all K blocks, which are required to construct the original file, from encoder i. By time t, peer 1 has downloaded $K_1(t) = s_1 t$ blocks from encoder i, peer 2: $K_2(t) = s_2 t$ blocks, ..., and peer m: $K_m(t) = s_m t$ blocks.

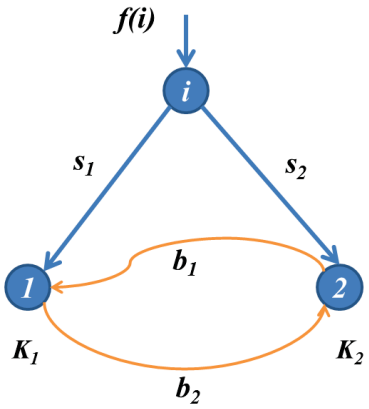

Figure 6.2: The encoder is placed at an intermediate peer i which is sending data to two neighboring peers: peer 1 and peer 2. In this case, intermediate encoder i does not have the whole file but is downloading it at rate $f(i)$; and peer 1 and peer 2 do not need to receive all K blocks, which are required to construct the original file, from encoder i. In total, peer 1 receives K_1 blocks and peer 2 receives K_2 blocks from encoder i.

Assume $T_1 \leq T_2 \leq .. \leq T_m$, the total encoded blocks node i has generated by the time t is

$$enc(i,t) = \begin{cases} s_1 t + s_2 t + .. + s_m t & \text{if } 0 < t < T_1, \\ s_1 T_1 + s_2 t + .. + s_m t & \text{if } T_1 \leq t < T_2, \\ ... & \\ s_1 T_1 + s_2 T_2 + .. + s_m T_m & \text{if } t \geq T_m. \end{cases}$$

Since the number of blocks encoder i received by time t is

$$recv(i,t) = \begin{cases} f(i)t & \text{if } 0 < t < T_i, \\ K & \text{if } t \geq T_i \end{cases}$$

the redundancy ratio at encoder i becomes

$$e_i(t) = \frac{enc(i,t)}{recv(i,t)}$$

$$
\begin{aligned}
&= \begin{cases}
\frac{1}{f(i)t}(s_1t + s_2t + .. + s_mt) & \text{if } 0 < t < T_1, \\
\frac{1}{f(i)t}(s_1T_1 + s_2t + .. + s_mt) & \text{if } T_1 \leq t < T_2, \\
... & \\
\frac{1}{K}(s_1T_1 + s_2T_2 + ..+ & \\
s_{j-1}T_{j-1} + s_jt + .. + s_mt) & \text{if } T_i \leq t < T_j, \\
... & \\
\frac{1}{K}(s_1T_1 + s_2T_2 + .. + s_mT_m) & \text{if } t \geq T_m
\end{cases} \\
&= \begin{cases}
\frac{s_1+s_2+..+s_m}{f(i)} & \text{if } 0 < t < T_1, \\
\frac{s_1T_1}{f(i)t} + \frac{s_2+..+s_m}{f(i)} & \text{if } T_1 \leq t < T_2, \\
... & \\
\frac{s_1T_1+s_2T_2+..+s_{j-1}T_{j-1}}{K} + \frac{(s_j+..+s_m)t}{K} & \text{if } T_i \leq t < T_j, \\
... & \\
\frac{s_1T_1+s_2T_2+..+s_mT_m}{K} & \text{if } t \geq T_m
\end{cases}
\end{aligned} \tag{6.4}
$$

assuming T_j is closest to T_i and $T_j \geq T_i$.

In (6.4), notice that the redundancy ratio $e_i(t)$ is, at first, equal to $\frac{s_1+s_2+..+s_m}{f(i)}$ with $0 < t < T_1$. Then it decreases on $[T_1, T_i)$. After that, the ratio increases on $[T_i, T_m)$ and finally reaches $\frac{s_1T_1+s_2T_2+..+s_mT_m}{K}$ when $t \geq T_m$. Figure 6.3 illustrates the redundancy ratio function $e_i(t)$ in case node i has $m = 3$ neighbors and $T_1 < T_2 < T_i < T_3$.

The maximum redundancy ratio, therefore, is the larger value of the two local maxima $\frac{s_1+s_2+..+s_m}{f(i)}$ in $(0, T_1)$ and $\frac{s_1T_1+s_2T_2+..+s_mT_m}{K}$ in $[T_m, \infty)$:

$$
e_i \;=\; max\left(\frac{s_1 + s_2 + .. + s_m}{f(i)}, \frac{s_1T_1 + s_2T_2 + .. + s_mT_m}{K}\right)
$$

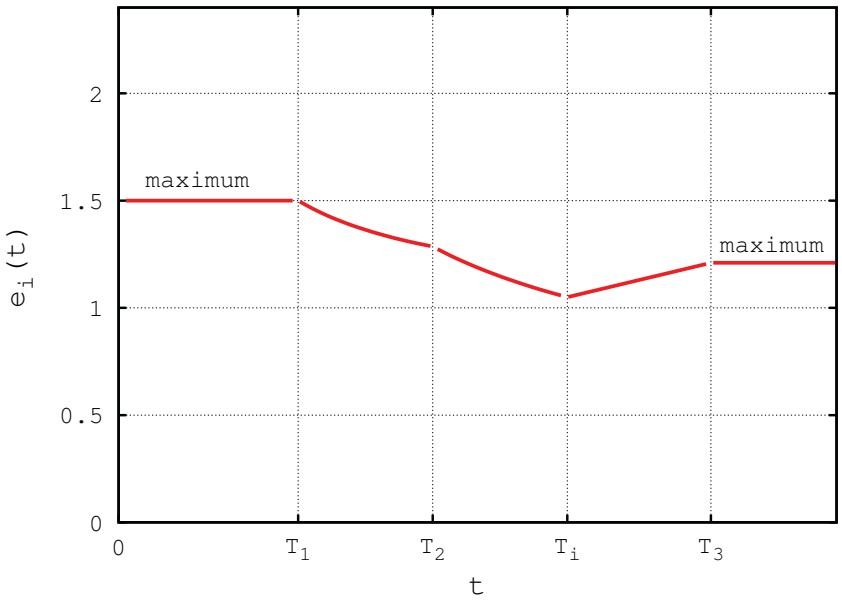

Figure 6.3: Illustration of redundancy ratio at node i with $m = 3$ neighboring peers. The redundancy ratio changes over time and reaches two maxima at $0 < t < T_1$ and $t \geq T_3$.

$$= max\left(\frac{\sum_{j=1}^{m} s_j}{f(i)}, \sum_{j=1}^{m} \frac{s_j}{f(j)}\right). \tag{6.5}$$

6.4 Redundancy Ratio Computation

We implement an algorithm to compute the redundancy ratio of all encoders (Algorithm 6.1) given the overlay topology and capacities of all overlay links.

The algorithm starts by computing *max-flow* from the source to all peers using Edmonds-Karp algorithm [58] (line 7–17). For the sake of presentation, we use notation $maxflow_{S \to j}$ as a set containing all the augmenting paths, i.e. the paths found by repeatedly figuring a path with positive capacity from the source to node j. We note that since Edmonds-Karp [58] uses breadth-first search, the resulting *max-flow* to a given node consists of augmenting paths with shortest lengths from the source to that node.

While computing max-flow to node j, the algorithm collects:

- the set $child[i]$ of children of node i: node j is a child of node i if $i \in$

Algorithm 6.1: Redundancy Ratio Computation

Input: $G = \{V, E\}$, $c(i,j)\ \forall (i,j) \in E$, the source $S \in V$, set C of encoders $(C \subset V)$

Result: *redundancy ratio* $e_i\ \forall i \in C$

forall the $i, j \in V$ **do**
  $f(j) = 0$;
  $f(i,j) = 0$;
  $s(i,j) = 0$;
  $child[i] = \emptyset$;
end
foreach $j \in V \backslash \{S\}$ **do**
  compute *max-flow* from the source to j: $f(j)$ using Edmonds-Karp algorithm [58];
  $maxflow_{S \to j} \leftarrow$ all augmenting paths;
  foreach $i \in maxflow_{S \to j}$ **do**
    if $(i \to j) \in E$ **then**
      $child[i] \leftarrow j$;
      $f(i,j) = maxflow_{S \to j}(i)$;
      $s(i,j) = maxflow_{S \to j}(i \to j)$;
    end
  end
end
foreach $i \in C$ **do**
  if i *is* S **then**
    $e_i = \sum_{j \in child[i]} \frac{s(i,j)}{f(j)}$;
  end
  else
    $e_i = max\left(\frac{\sum_{j \in child[i]} s(i,j)}{f(i)}, \sum_{j \in child[i]} \frac{s(i,j)}{f(j)} \right)$;
  end
end

$maxflow_{S \to j}$ and the two nodes are neighbors, i.e. directly connected,

- the flow passing the corresponding parent node i: $maxflow_{S \to j}(i)$, and
- the flow passing a link from the parent node i to the child node j: $maxflow_{S \to j}(i \to j)$.

Based on the collected information, it then figures the total throughput from node i to node j: $f(i,j)$, and the throughput over the direct link $i \to j$: $s(i,j)$ for

each pair of parent i and child j. Finally, the redundancy ratio at each encoder is computed using either (6.3) at line 20 if the encoder is at the source, or (6.5) at line 23 if the encoder is an intermediate node.

The complexity is dominated by the second loop (line 7–17) computing *max-flow* from the source to all peers which takes $O(V^2E^2)$ time.

6.5 Performance Evaluation

We implemented a C++ simulator of the P2P content distribution system and run simulations over generated topologies distributing a file from the source to all participating peers. The file is divided into smaller fix-sized parts, i.e. blocks. The source and all peers exchange blocks until all peers acquire enough blocks to construct the original file; then the simulation finishes.

The simulations are round-based and peers exchange blocks using our proposed pre-code protocol (Section 3.2.2) and block selection (Section 3.3), and mutual exchange incentive scheme, the same as in Section 3.5.

We generate topologies for simulations using Watts and Strogatz *small-world network* model [55] with degree $d = 8$ and rewiring probability $p_{rw} = 0.01$. All overlay links have capacity of 1 block per round. The network size is 5000 nodes.

We use *min-delay* placement presented in Chapter 4 to determine where to place $|C| = 250$ encoders in the network. Using *min-delay* placement, encoders are placed at nodes from which data duplication causes the most delay in finish time to downstream nodes. We then run Algorithm 6.1 to figure the redundancy ratio of the chosen encoders.

A multiplier λ is used to adjust the actual redundancy ratio at all encoders. At any time, each encoder i is allowed to generate an accumulative number of encoded blocks equal to or less than λe_i times the number of blocks it has received where e_i is node i's maximum redundancy ratio computed by Algorithm 6.1. If it passes the limit, the encoder stops generating new encoded blocks but keeps sending

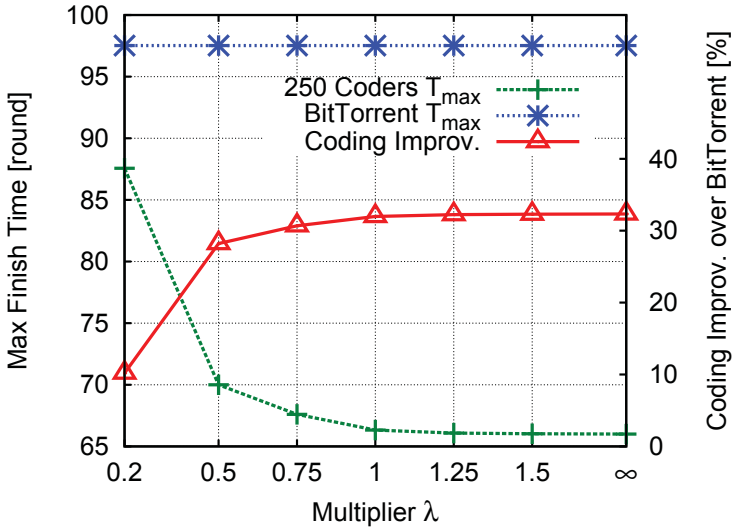

Figure 6.4: Maximum finish time when 250 encoders are deployed using *min-delay* placement compared with maximum finish time of non-coding BitTorrent. Multiplier λ is used to adjust redundancy ratios at all encoders.

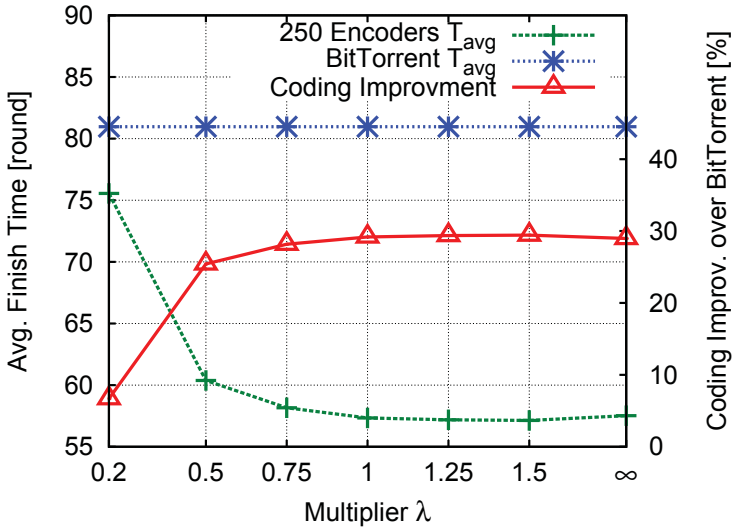

Figure 6.5: Average finish time when 250 encoders are deployed using *min-delay* placement compared with average finish time of non-coding BitTorrent. Multiplier λ is used to adjust redundancy ratios at all encoders.

old encoded blocks for the time being if there are download requests. When the constraint on number of encoded is satisfied, the encoder can start generating new coded blocks again. $\lambda = 1$ means that the encoders generate the same level of redundancy as computed by our analysis.

We run simulations 100 times distributing a 200-block file from the source and collect the maximum finish time T_{max} and average finish time of all peers T_{avg}. We

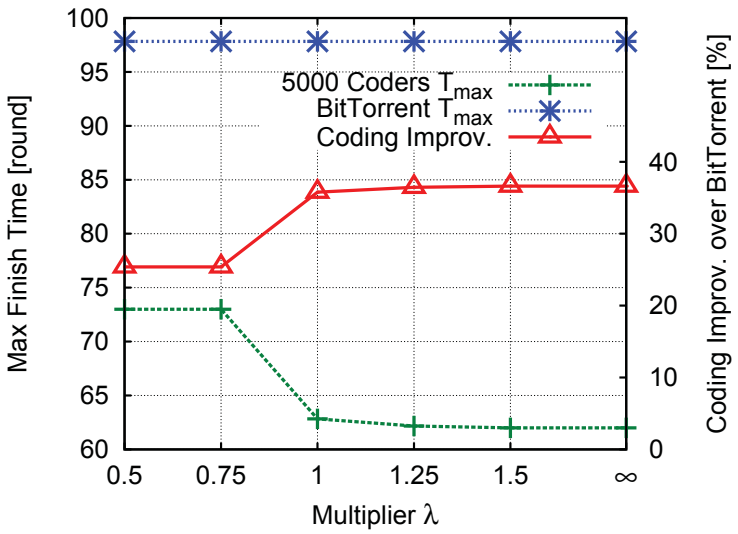

Figure 6.6: Maximum finish time when 5000 encoders (full network coding) are deployed at all peers compared with maximum finish time of non-coding BitTorrent. The finish time improvement increases steeply when multiplier $\lambda = 1$, i.e. encoders use the redundancy ratio computed by our analysis.

compare the finish time of the network coding-enabled P2P system with a non-coding ordinary BitTorrent. To ensure a fair comparison, in both non-coding and coding cases, we enforce at the C chosen nodes *super-seeding* scheme [64]. The nodes try to serve with a block which has never been sent into the network.

The results are given in Figure6.4 for maximum finish time and Figure6.5 for average finish time of all peers. $\lambda = \infty$ means no restriction on the number of encoded blocks an encoder can generate. When $\lambda = 0.2$, i.e. each encoder generates only 20% of the redundancy which has been computed by our proposed algorithm, the finish time is extreme long since there are only a few coded blocks circulating in the system. Increasing λ values results in shorter finish time. There is, however, almost no finish time improvement with $\lambda > 1$ compared with the finish time when $\lambda = 1$. The results confirm that with the analyzed redundancy ratio we can achieve short distribution time.

Notice that a certain level of improvement, though negligible, shows up with redundancy ratios higher than the analyzed one (when $\lambda > 1$). We attribute that improvement to two reasons. First, in actual systems, data might be distributed

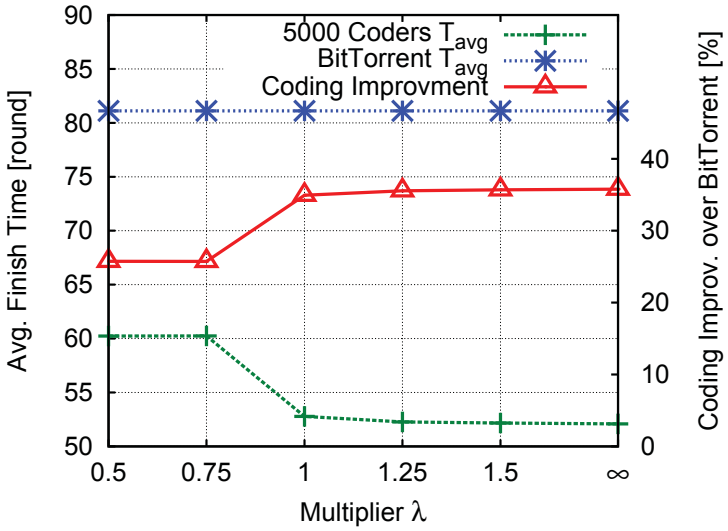

Figure 6.7: Average finish time when 5000 encoders (full network coding) are deployed at all peers compared with average finish time of non-coding BitTorrent. The finish time improvement increases steeply at $\lambda = 1$, i.e. encoders use the redundancy ratio computed by our analysis.

from the source over delivery paths other than the shortest paths which our algorithm considers. When that happens, the actual redundancy ratios of upstream nodes on those paths will be higher. Second, due to delay incurred when blocks are transferred on the paths, a redundancy ratio higher than the analyzed value might be needed to sustain content delivery until downstream peers finish. The difference in performance, however, is insignificant when the network has a small diameter.

We furthermore check the performance when all nodes encode, i.e. all 5000 peers are assigned as encoders (Figure6.6 and Figure6.7).[1] The finish time decreases steeply at $\lambda = 1$, when redundancy ratios are equal to the analyzed values. The reason is because with redundancy ratios lower than the analyzed one, there is not enough block redundancy in the system, which results in high level of duplication and long finish time. Increasing λ higher than 1, and even with unrestricted

[1]We note that in this case, since all peers are controlled by parameter λ, when $\lambda \leq 0.2$, i.e. each peer is allowed to generate at most 20% of the optimal number of blocks, most peers can not finish downloading the file since there are not enough blocks generated and distributed in the system.

encoding ($\lambda = \infty$), there is practically no improvement in finish time.

6.6 Discussion

To further reduce network coding's computational resource consumption, in this chapter, we have proposed a method to compute the redundancy ratio at a given network coder. The proposed redundancy ratio computation is developed based on our analysis of the best level of redundancy each network coder should generate in order to achieve shortest distribution time. Using the analyzed redundancy ratios, the system has short distribution time, virtually the same as in case the encoders are allowed to code unrestrictedly.

The result demonstrates that full network coding, though optimal in achievable throughput, is not actually efficient in terms of resource utilization. By carefully considering the right level of redundancy at each encoder, we can save computational resources while still achieving good performance.

For our future work, we would like to extend the method to a dynamic setting where peers can join and leave the system. To address that, we need a distributed algorithm to assign encoders and figure the redundancy ratio, for example, based on local topology characteristics. We also plan to evaluate performance using traces from live systems.

Another interesting development is to figure when an encoder should encode. Here we let encoders encode as soon as possible which might not be necessarily efficient especially in the beginning of the content distribution session when there are only a few data pieces available in the system.

Chapter 7

Conclusion and Future Work

7.1 Concluding Remarks

We study methods to optimize network coders in P2P content distribution. The main problem we want to solve is to reduce resource consumption of network coding. We address the problem in several aspects.

First of all, we optimize the whole system where network coding-enabled peers and ordinary peers coexist, which we call a hybrid network coding P2P system, by proposing protocols and data selection algorithm for it. Our protocols let peers communicate seamlessly whether they are network coders or ordinary non-coding peers. Our data selection algorithm allows a peer to handle mixtures of coded data and non-coded data gracefully. Together, the proposed protocols and selection algorithm noticeably improve performance of such a hybrid network coding system.

We then, with a target to optimize network coder placement, look into the network topology to determine the locations which most seriously need network coding. We reveal quantitively the condition under which network coding can improve performance. That lies in its ability to eliminate data duplication. When there are multiple delivery paths from an upstream peer to a downstream peer, the same data is transferred multiple times over those paths, which consumes bandwidth and slow down content delivery to the downstream peer. Placing a network

coder at the upstream peer will eliminate data duplication and accelerate transfer throughput. We distinctively figure the amount of duplication originated from each node on a network topology which is the basis for our algorithms to determine where we should place network coders inside a content distribution network.

We proposed algorithms to place network coders at the appropriate nodes based on the insight obtained from our duplication analysis. We place coders where duplication happens and affects performance the most in order to shorten distribution time. The result is that we can eliminate unnecessary network coders to save computational resources. We present three placement algorithms: minimal delay, betweenness centrality, and flow centrality coder placements. The first algorithm, minimal delay placement, elaborately figures the duplication, and delay in finish time, each upstream node causes to its downstream nodes. Based on that, it accurately locates nodes which cause the most delay to downstream nodes due to duplication for network coder placement. The limitation is its higher complexity. The last two methods, on the other hand, exploit centrality concepts to pinpoint the nodes which generate most duplication. Centrality indexes are effective tools for that purpose because using them we can locate nodes which stand on more paths or wider paths to other nodes, which are the places where larger duplication occurs. The advantage of these two methods is their relatively fast speed with good performance in terms of shortening the distribution time.

We furthermore optimize the level of redundancy each network coder should generate, namely *redundancy ratio*. That is we answer the question how much a network coder should encode given the amount of data it has received. By determining the right redundancy ratio at each encoder, we reduce computational resource consumption further, saving the encoders from worthless encoding.

We evaluate the performance of our proposed optimization by simulations.

Our proposed communication protocols and block selection algorithm boost the performance significantly. In simulations, the finish time is 15%–25% shorter when

peers use the proposed protocols to communicate and the proposed block selection algorithm to choose which blocks to download from neighboring peers.

Our placement algorithms achieve comparable performance in terms of finish time as full network coding with only a portion of network coders. As the number of network coders increases the performance is gradually improved, but the finish time almost equal to that of network coding can be achieved with just tenths of the total number of peers assigned as network coders. The result demonstrates that a large number of network coders are redundant and can be removed to save computational resources with virtually no effect on the performance.

With the redundancy ratio from our analysis, network coders can limit their encoding operations while still generating the right redundancy level to shorten finish time. Lower redundancy ratios than the values suggested by the analysis result in longer finish time due to lack of redundancy and higher redundancy ratios practically has no meaningful impact.

Our study offers new insights into what makes network coding's good performance, where in a network can we place network coders, and how much redundancy a network coder needs to generate. In practice, such knowledge equips network designers with an effective tool to deploy network coding in order to improve network performance. We answer the question how to optimally implement network coding from both the network's and each individual node's point of views.

The results of our study can readily be extended to other types of networks where network nodes communicate with one another over multiple paths over which network coding, with the redundancy it generates, can effectively accelerate performance.

7.2 Future Work

Admittedly, we cannot say our work is complete in its current state. There are several improvements we want to carry out in order to strengthen our results.

First of all, we plan to evaluate our proposed optimization in an dynamic environment where peers keep joining and leaving the system. In this work, we focus on understanding the feature of network coding in a static settings. The result should be more convincing if we can verify it in such dynamic networks. We believe our proposals would also work well with node dynamics as network coding, in general, has been shown to increase resilience in such conditions [6, 14].

The algorithms proposed in this dissertation are centralized in the sense that they required a centralized server to do the computation, and then, to assign or communicate the result to participating nodes in the network. We would like to make both of those tasks distributed. As we have discussed in Section 3.4 and Section 5.6, we can decentralize the system, in case of centrality-based placement for example, by, first, devising a distributed algorithm to allow each peer to figure an approximate centrality value by itself which is then used to decide if the peer should encode or not. Second, we would need to determine the centrality threshold and devise a method to inform peers in the system about the threshold. Alternatively, a peer can determine by itself the threshold based on local information or some indication values passing through it from neighboring peers. Peers with centrality values higher than the threshold will become network coders to improve the system performance. Likewise, redundancy ratio can also be computed in a distributed manner by each peer from the information of the traffic flows going through it.

Lastly, we plan to evaluate our proposals using traces from live systems and implement them in real-life networks.

Before closing, we would like to discuss one promising direction to apply the result in this dissertation to the research trend in information-centric networking. Information-centric networking (ICN) [65], currently an active research area, is in a sense a system where content distribution is natively supported. Nevertheless, ICN, especially content-centric networking (CCN) [66], one of the most promising

ICN approaches, is in its early development state and active research is on progress to improve them in several aspects. In terms of a content distribution system, ICN for the time being suffers many drawbacks as pointed out in [20, 67, 68, 69] which result in under-utilization of network resources and sub-optimal delivery throughput. As multiple delivery paths are inherent in ICN, we are, therefore, excited about future research extending the results in this dissertation to see how network coding could help improve ICN efficiency.

Bibliography

[1] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.

[2] S.-Y. Li, R. Yeung, and N. Cai, "Linear network coding," *IEEE Transactions on Information Theory*, vol. 49, no. 2, pp. 371–381, 2003.

[3] A. Legout, G. Urvoy-Keller, and P. Michiardi, "Rarest first and choke algorithms are enough," in *Proceedings of ACM SIGCOMM IMC*, 2006.

[4] D.-M. Chiu and R. Yeung, "Can network coding help in p2p networks," in *Proceedings of NetCod* 2^{nd}, Boston, 2006.

[5] B. Cohen, "Incentives build robustness in bittorrent," in *P2P Economics Workshop*, 2003.

[6] R. Koetter and M. Médard, "An Algebraic Approach to Network Coding," *IEEE/ACM Trans. Netw.*, vol. 11, no. 5, pp. 782–795, Oct. 2003.

[7] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft, "Xors in the air: practical wireless network coding," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 4, pp. 243–254, Aug. 2006. [Online]. Available: http://doi.acm.org/10.1145/1151659.1159942

[8] ——, "Xors in the air: practical wireless network coding," in *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM '06. New York, NY, USA: ACM, 2006, pp. 243–254. [Online]. Available: http://doi.acm.org/10.1145/1159913.1159942

[9] D. Nguyen, T. Tran, T. Nguyen, and B. Bose, "Wireless broadcast using network coding," *Vehicular Technology, IEEE Transactions on*, vol. 58, no. 2, pp. 914–925, 2009.

[10] S. Zhang, S. C. Liew, and P. P. Lam, "Hot topic: physical-layer network coding," in *Proceedings of the 12th annual international conference on Mobile computing and networking*, ser. MobiCom '06. New York, NY, USA: ACM, 2006, pp. 358–365. [Online]. Available: http://doi.acm.org/10.1145/1161089.1161129

[11] J. Sundararajan, D. Shah, M. Medard, M. Mitzenmacher, and J. Barros, "Network coding meets tcp," in *INFOCOM 2009, IEEE*, 2009, pp. 280–288.

[12] A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *Information Theory, IEEE Transactions on*, vol. 56, no. 9, pp. 4539–4551, 2010.

[13] A. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A survey on network codes for distributed storage," *Proceedings of the IEEE*, vol. 99, no. 3, pp. 476–489, 2011.

[14] C. Gkantsidis and P. R. Rodriguez, "Network Coding for Large Scale Content Distribution," in *Proceedings of IEEE INFOCOM*, March 2005.

[15] C. Gkantsidis, J. Miller, and P. Rodriguez, "Anatomy of a P2P Content Distribution System with Network Coding," in *Proceedings of IPTPS'06*, February 2006.

[16] ——, "Comprehensive view of a live network coding P2P system," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, ser. IMC '06. New York, NY, USA: ACM, 2006, pp. 177–188.

[17] 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.

[18] ——, "R2: Random push with random network coding in live peer-to-peer streaming," *Selected Areas in Communications, IEEE Journal on*, vol. 25, no. 9, pp. 1655–1666, 2007.

[19] ——, "Lava: A reality check of network coding in peer-to-peer live streaming," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*. IEEE, 2007, pp. 1082–1090.

[20] M.-J. Montpetit, C. Westphal, and D. Trossen, "Network coding meets information-centric networking: an architectural case for information dispersion through native network coding," in *Proceedings of the 1st ACM workshop on Emerging Name-Oriented Mobile Networking Design - Architecture, Algorithms, and Applications*, ser. NoM '12. New York, NY, USA: ACM, 2012, pp. 31–36. [Online]. Available: http://doi.acm.org/10.1145/2248361.2248370

[21] C. Fragouli, J.-Y. Le Boudec, and J. Widmer, "Network coding: an instant primer," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 63–68, Jan. 2006.

[22] C. Fragouli and E. Soljanin, *Network Coding Fundamentals*, ser. Foundations and Trends in Networking. Now Publishers, 2007.

[23] ——, *Network Coding Applications*, ser. Foundations and Trends in Networking. Now Publishers, 2008.

[24] R. W. Yeung, S.-Y. R. Li, N. Cai, and Z. Zhang, *Network Coding Theory Part I: Single Source*, ser. Foundations and Trends in Communications and Information Theory. Now Publishers, July 2006, vol. 2, no. 4.

[25] T. Ho and D. Lun, *Network Coding: An Introduction*, 1st ed. Cambridge University Press, 2008.

[26] T. Ho, R. Koetter, M. Medard, D. Karger, and M. Effros, "The Benefits of Coding over Routing in a Randomized Setting," in *ISIT 2003*, Yokohama, Japan, 2003.

[27] T. Ho, M. Medard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong, "A Random Linear Network Coding Approach to Multicast," *IEEE Transactions on Information Theory*, vol. 52, no. 10, pp. 4413–4430, 2006.

[28] R. W. Yeung, "Avalanche: A network coding analysis," *Communications in Information & Systems*, vol. 7, no. 4, pp. 353–358, 2007.

[29] T. Locher, S. Schmid, and R. Wattenhofer, "Rescuing Tit-for-Tat with Source Coding," in *Proceedings of IEEE P2P*, September 2007.

[30] D. Nguyen and H. Nakazato, "Peer-to-Peer Content Distribution in Clustered Topologies with Source Coding," in *Proceedings of IEEE GLOBECOM*, Houston, USA, December 2011.

[31] P. Maymounkov and D. Mazires, "Rateless Codes and Big Downloads," in *IPTPS'03*, February 2003.

[32] M. Luby, "LT Codes," in *Proceedings of the* 43^{rd} *Annual IEEE Symposium on Foundations of Computer Science*, 2002.

[33] M. Kim, M. Medard, V. Aggarwal, U.-M. O'Reilly, W. Kim, C. W. Ahn, and M. Effros, "Evolutionary Approaches to Minimizing Network Coding Resources," in *Proceedings of IEEE INFOCOM*, May 2007.

[34] K. Bhattad, N. Ratnakar, R. Koetter, and K. R. Narayanan, "Minimal network coding for multicast," in *Proceedings of IEEE ISIT*, September 2005.

[35] D. Lun, N. Ratnakar, R. Koetter, M. Medard, E. Ahmed, and H. Lee, "Achieving minimum-cost multicast: a decentralized approach based on network coding," in *Proceedings of IEEE INFOCOM*, vol. 3, 2005, pp. 1607–1617.

[36] M. Martal, M. Mohorovicich, G. Ferrari, and C. Fragouli, "Network-coded multihop multicast: Topology and encoding complexity," in *Proceedings of IEEE ICC*, 2012, pp. 2501–2505.

[37] C. Fragouli and E. Soljanin, "Information flow decomposition for network coding," *Information Theory, IEEE Transactions on*, vol. 52, no. 3, pp. 829–848, 2006.

[38] C. Fragouli, E. Soljanin, and A. Shokrollahi, "Network Coding as a Coloring Problem," in *Proceedings of IEEE Annual Conference on Information Sciences and Systems (CISS 2004)*, Princeton, NJ, USA, March 2004.

[39] M. Langberg, A. Sprintson, and J. Bruck, "The encoding complexity of network coding," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2386–2397, 2006.

[40] T. Small and B. Li, "Topology Affects the Efficiency of Network Coding in Peer-to-Peer Networks," in *Proceedings of ICC*, 2008.

[41] D. Niu and B. Li, "Topological Properties Affect the Power of Network Coding in Decentralized Broadcast," in *Proceedings of IEEE INFOCOM*, 2010.

[42] S. Crisostomo, J. Barros, and C. Bettstetter, "Network Coding with Shortcuts," in *Proceedings of IEEE ICCS*, 2008.

[43] S. Maheshwar, Z. Li, and B. Li, "Bounding the coding advantage of combination network coding in undirected networks," *IEEE Transactions on Information Theory*, vol. 58, no. 2, pp. 570–584, 2012.

[44] N. Cleju, N. Thomos, and P. Frossard, "Network Coding Node Placement for Delay Minimization in Streaming Overlays," in *Proceedings of IEEE ICC*, 2010.

[45] P. Maymounkov, N. J. Harvey, and D. S. Lun, "Methods for efficient network coding," in *Proc. 44-th Allerton Conference*, vol. 6, 2006.

[46] M.-L. Champel, K. Huguenin, A.-M. Kermarrec, and N. Le Scouarnec, "LT network codes: low complexity network codes," in *Proceedings of the 5th international student workshop on Emerging networking experiments and technologies*, ser. Co-Next Student Workshop '09. New York, NY, USA: ACM, 2009, pp. 39–40.

[47] D. Silva, W. Zeng, and F. Kschischang, "Sparse network coding with overlapping classes," in *Proceedings of NetCod '09 Workshop*, 2009, pp. 74–79.

[48] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, no. 1, pp. 35–41, 1977.

[49] L. C. Freeman, S. P. Borgatti, and D. R. White, "Centrality in valued graphs: A measure of betweenness based on network flow," *Social Networks*, vol. 13, pp. 141–154, 1991.

[50] D. Nguyen and H. Nakazato, "Centrality-based network coder placement for peer-to-peer content distribution," *International Journal of Computer Networks and Communications*, vol. 5, no. 3, pp. 157–174, May 2013.

[51] ——, "Network coder placement for peer-to-peer content distribution," *IEICE Transactions on Communications*, vol. E96-B, no. 7, July 2013.

[52] P. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proceedings of the Allerton Conference on Communication, Control and Computing*, 2003.

[53] A. Legout, G. Urvoy-Keller, and P. Michiardi, "Understanding BitTorrent: An Experimental Perspective," INRIA-00000156, VERSION 3, Tech. Rep., November 2005.

[54] C. Yin, B. Wang, W. Wang, T. Zhou, and H. Yang, "Efficient routing on scale-free networks based on local information," *Physics Letters A*, vol. 351, pp. 220–224, 2006.

[55] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, pp. 440–442, June 1998.

[56] A. Adamic, "The Small World Web," in *Proceedings of ECDL '99, Springer-Verlag, London, UK*, 1999, pp. 443–452.

[57] N. Leibowitz, M. Ripeanu, and A. Wierzbicki, "Deconstructing the kazaa network," in *Proceedings of WIAPP*, 2003.

[58] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press and McGraw.Hill, 2001, ch. 26.2, pp. 600–663.

[59] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of Mathematical Sociology*, vol. 25, pp. 163–177, 2001.

[60] D. He, W. K. Chai, and G. Pavlou, "Leveraging In-network Caching for Efficient Content Delivery in Content-centric Network," in *Proceedings of the London Communication Symposium*, September 2011.

[61] A. Tizghadam and A. Leon-Garcia, "AORTA: Autonomic Network Control and Management System," in *Proceedings of the 1st IEEE Workshop on Automated Network Management*, April 2008.

[62] A.-M. Kermarrec, E. L. Merrer, B. Sericola, and G. Trdan, "Second order centrality: Distributed assessment of nodes criticity in complex networks," *Computer Communications*, vol. 34, no. 5, pp. 619–628, 2011.

[63] K. Wehmuth and A. Ziviani, "Distributed location of the critical nodes to network robustness based on spectral analysis," in *Network Operations and Management Symposium (LANOMS), 2011 7th Latin American*, 2011, pp. 1–8.

[64] BitTornado. www.bittornado.com.

[65] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A Survey of Information-Centric Networking (Draft)," in *Information-Centric Networking*, ser. Dagstuhl Seminar Proceedings, B. Ahlgren, H. Karl, D. Kutscher, B. Ohlman, S. Oueslati, and I. Solis, Eds., no. 10492. Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2011. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2011/2941

[66] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, ser. CoNEXT '09. New York, NY, USA: ACM, 2009, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/1658939.1658941

[67] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox, "Information-centric networking: seeing the forest for the trees," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, ser. HotNets-X. New York, NY, USA: ACM, 2011, pp. 1:1–1:6. [Online]. Available: http://doi.acm.org/10.1145/2070562.2070563

[68] W. K. Chai, D. He, I. Psaras, and G. Pavlou, "Cache "less for more" in information-centric networks," in *Proceedings of the 11th international IFIP TC 6 conference on Networking - Volume Part I*, ser. IFIP'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 27–40. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-30045-5_3

[69] G. Rossini and D. Rossi, "Evaluating ccn multi-path interest forwarding strategies," *Computer Communications*, vol. 36, no. 7, pp. 771 – 778, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366413000261

List of Academic Achievements

Articles in refereed journals	◯ Dinh Nguyen and Hidenori Nakazato,"Network Coder Placement for Peer-to-Peer Content Distribution," *IEICE Transactions on Communications*, Special Section on Internet Architectures, Protocols, and Management Methods that Enable Sustainable Development, vol. E96-B, no. 07, pp. 1661–1669, July 2013. ◯ Dinh Nguyen and Hidenori Nakazato,"Centrality-based Network Coder Placement for Peer-to-Peer Content Distribution," *International Journal of Computer Networks & Communications*, ISSN 0975-2293, vol. 5, no. 3, pp. 157-174, May 2013. ◯ Dinh Nguyen and Hidenori Nakazato, "Hybrid Network Coding Peer-to-Peer Content Distribution," *Journal of Computing*, ISSN 2151-9617, vol. 5, issue 4, pp. 8-17, April 2013.
Presentations at international conferences	◯ Dinh Nguyen and Hidenori Nakazato, "Rarest-first and Coding are Not Enough," Next Generation Networking and Internet Technical Symposia, IEEE GLOBECOM 2012, Anaheim, California, U.S.A., December 2012. ◯ Dinh Nguyen and Hidenori Nakazato, "Peer-to-Peer Content Distribution in Clustered Topologies with Source Coding," Next Generation Networking and Internet Technical Symposia, IEEE GLOBECOM 2011, Houston, Texas, U.S.A., December 2011.
Presentations at domestic conferences	◯ Dinh Nguyen and Hidenori Nakazato, "Network Coder Placement for Peer-to-Peer Content Distribution," IEICE Tech. Report, vol. 112, no. 309, CS2012-74, pp. 59–64, November 2012. (*IEICE Technical Committee on Communication Systems Award*) Nguyen Quoc Dinh and Hidenori Nakazato, "Peer-to-Peer Content Distribution with Source Coding," IEICE Society Conference, Osaka, Japan, September 2010. Nguyen Quoc Dinh and Hidenori Nakazato, "Transcoder Placement for Peer-to-Peer Streaming," IEICE Tech. Report, vol. 105, no. 627, NS2005-195, pp. 149–152, March 2006.