AN ABSTRACT OF THE DISSERTATION OF

Kien Trung Nguyen for the degree of Doctor of Philosophy in
Electrical and Computer Engineering presented on November 18, 2010.

Title:
Network Coding for Sensor Networks, Distributed Storage and Video Streaming

Abstract approved: _____
Thinh Nguyen Bella Bose

The classical store-and-forward routing has and will continue to be the most important routing architecture in many modern packet-switched communication networks. In a packet-switched network, data is sent in the form of discrete packets that traverse hop-by-hop from a source to a destination. At each intermediate hop, the router stores and examines the packets it receives then forwards them to the next hop until they reach the correct destinations according to some predefined routing algorithms. Importantly, the intermediate routers do not modify but simply store and forward the contents of the packets. In contrast, a new generalized approach to routing called Network Coding (NC) allows the intermediate routers to modify and combine packets from different sources and destinations in such a way that increases the overall throughput. The core idea of NC allowing the intermediate nodes in a network to perform data processing has a wide range of applications well beyond its initial application to routing, impacting different

disciplines from distributed data storage and security to energy efficient sensor networks and Internet media streaming. To that end, this dissertation aims to develop the theories and applications of NC via four main thrusts:

1) Energy efficient NC techniques for sensor networks,

2) Novel NC techniques and protocols for Internet video streaming,

3) Stochastic data replenishment for large scale NC-based distributed storage systems,

4) Real-world implementation of NC-based distributed video streaming system.

In thrust one, we describe a novel cross-sensor coding technique that combines network topology and coding techniques to maximize the life-time of a sensor network, by addressing the uneven energy consumption problem in data gathering sensor networks where the nodes closer to the sink tend to consume more energy than those of the farther nodes. Our approach is based on the following observation from the sensor networks using On-Off Keying and digital transmission: transmitting bit "1" consumes much more energy than bit "0". Our proposed coding technique exploits this difference to reduce the communication energy by limiting the number of bits "1" in the output codeword (*low-weight codeword*) and to use NC-based *cross-sensor* coding technique to equalize the communication energy among the nodes. This cross-sensor coding scheme can significantly extend the network lifetime as compared with traditional (binary) coding by solving the energy-consumption unfairness problem. The theoretical and experimental results confirm that transmission energy can be reduced substantially (e.g., a factor of 15) and the unequal energy consumption among nodes can be practically eliminated.

In thrust two, we describe a rate distortion aware hierarchical NC technique and transport protocol for Internet video streaming. We begin by proposing a NC-based multi-sender streaming framework that reduces the overall storage, eliminates the complexity of sender synchronization, and enables TCP streaming. Furthermore, we propose a Hierarchical Network Coding (HNC) technique that facilitates scalable video streaming to combat bandwidth fluctuation on the Internet. This HNC technique enables receiver to recover the important data gracefully in the presence of limited bandwidth which causes an increase in decoding delay. Simulations demonstrate that under certain scenarios, our proposed NC techniques can result in bandwidth saving up to 60% over the traditional schemes.

In thrust three, we present a theory of NC-based data replenishment to automate the process of data maintenance for large scale distributed storage systems. The data replenishment mechanism is the core of these systems that promises to reduce the coordination complexity and increases performance scalability. The data replenishment automates the process of maintaining a sufficient level of data redundancy to ensure the availability of data in presence of peer departures and failures. The dynamics of peers entering and leaving the network is modeled as a stochastic process. We propose a novel analytical time-backward technique to bound the expected time, the longer the better, for a piece of data to remain in P2P systems. Both theoretical and simulation results are in agreement, indicating that our proposed data replenishment via random linear network coding (RLNC) outperforms other popular strategies that employ repetition and channel coding techniques. Specifically, we show that the expected time for a piece of data to

remain in a P2P system is exponential in the number of peers used to store the data for the RLNC-based strategy, while they are quadratic for other strategies. Furthermore, the time-backward technique can be applied to problems in other disciplines such as gene population modeling in theoretical biology.

Finally in thrust four, we present the architecture, design, and experimental results of an actual NC-based distributed video streaming system. We first implement random linear network coding (RLNC) library and show the feasibility of using RLNC in P2P video streaming applications. Then we design, implement and analyze RESnc - a resilient P2P video storage and streaming over the Internet using network coding. RESnc increases the streaming throughput and data resiliency against peer departures and failures using peer diversity. These improvements are based on three architectural elements:

1) The RLNC scheme that breaks a video stream into multiple smaller pieces, codes, and disperses them throughout peers in the network, in such a way to maximize the probability of recovering the original video under peer departures and failures;

2) The scalable mechanism for automating the data replenishment process using RLNC to maintain a sufficient level of redundancy for video stored in the system;

3) The path-diversity streaming protocol for a client to simultaneously stream a video from multiple peers with minimal coordination.

Experimental results demonstrated that our system adapts well with bandwidth fluctuation, provides significant playback quality improvement and bandwidth saving.

Network Coding for Sensor Networks, Distributed Storage and Video Streaming

by

Kien Trung Nguyen

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented November 18, 2010
Commencement June 2011

Doctor of Philosophy dissertation of Kien Trung Nguyen presented on November 18, 2010.

APPROVED:

Co-Major Professor, representing Electrical and Computer Engineering

---

Co-Major Professor, representing Electrical and Computer Engineering

---

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Kien Trung Nguyen, Author

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor, Prof. Thinh Nguyen, for his incessant guidance, continuous encouragement and financial support. His breadth of knowledge and his enthusiasm for research amazes and inspires me. Working with him is the most enjoyable part of my PhD career. I also would like to thank my co-advisor, Prof. Bella Bose, for his inspiring advice and feedback to my work. I cannot finish this dissertation without their help.

I am grateful to Prof. Mario Magana, Prof. Bechir Hamdaoui and Prof. Mei-Ching Lien for their great effort and significant amount of time to serve on my Ph.D committee. Their useful discussions and feedback help strengthen my work.

I would like to thank all my friends and colleagues at Oregon State University for their friendships and useful discussions. My special thank go to some good friends here who help me get used to the American life.

I am indebted to my parents, Duc Nguyen and Thuan Nguyen, for everything they have given to me. They taught me the value of knowledge, the joy of love and the importance of family. They have stood by me in everything I have done, providing constant support, encouragement, and love. I would like to thank my sisters, Van Nguyen and Linh Nguyen, for their endless love and support, and for taking care of my parents while I am abroad. I also would like to thank my parents-in-law, Thai Tran and Tuyet Nguyen, for their encouragement and unconditional support. They are taking care of our children so that I can concentrate and finish

this work. I am very lucky to have such a wonderful family.

Finally, my greatest thank go to my beloved wife, Thao Tran, for her patience and understanding and for taking care of everything while I was in school. She has been my true companion since the day we met. She and our children, Khoa Nguyen and Anh Nguyen, keep my life filled with love.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF TABLES

DEDICATION

To my father, Duc Nguyen.

To all of my family.

Chapter 1 – Introduction

The classical store-and-forward routing has and will continue to be the most important routing architecture in many modern packet-switched communication networks. In a packet-switched network, data is sent in the form of discrete packets that traverse hop-by-hop from a source to a destination. At each intermediate hop, the router stores and examines the packets it receives then forwards them to the next hop until they reach the correct destinations according to some predefined routing algorithms. Importantly, the intermediate routers do not modify but simply store and forward the contents of the packets. In contrast, a new generalized approach to routing called Network Coding (NC) allows the intermediate routers to modify and combine packets from different sources and destinations in such a way that increases the overall throughput. The core idea of NC allowing the intermediate nodes in a network to perform data processing has a wide range of applications well beyond its initial application to routing, impacting different disciplines from distributed data storage and security to energy efficient sensor networks and Internet media streaming. We begin with the introduction to network coding and its potential benefits in the next section.

1.1 Network coding concept

Communication networks today share the same fundamental principle of operation, where network flow is considered as a commodity flow. That is, independent data streams may share network resources but the information itself is separated. Generally, all the network functions are based on this assumption. In the network,

information is transmitted from the source node to each destination node through a chain of intermediate nodes by a *store and forward* method. With this method, data packets received from an input link of an intermediate node are forwarded to the next node via an output link. Thus, there is no need for data processing at the intermediate node except for data replication.

Network coding is a recent field in information theory that breaks this assumption. It considers network flows as information flows, where information from different data streams can be mixed together at the intermediate nodes. Ahlswede *et. al.* started the field with their pioneering paper [4]. They showed that the network throughput can approach the max-flow min-cut limit of the network graph when the intermediate nodes are allowed to combine their received data instead of simply forwarding them.

The key idea of network coding and its advantage over the traditional routing schemes can be demonstrated by the butterfly network in figure 1.1. In this multicast scenario, two servers S_1 and S_2 want to send packets X_1 and X_2 to both clients C_1 and C_2. Assume all links have capacity of one packet per time slot. Clearly, the optimal links allocation are S_1C_1 and S_1R_1 for transmitting packet X_1; S_2C_2 and S_2R_1 for packet X_2. One has to find the optimal transmission schedule for delivering the packets to both clients.

With traditional routing schemes, the router R_1 only forwards the message it received. Thus, link R_1R_2 will be bottleneck because it can only deliver either X_1 or X_2 but not both at the same time as shown in Figure 1.1a. Therefore one of the clients receives both packets while the other only receives one packet. If the

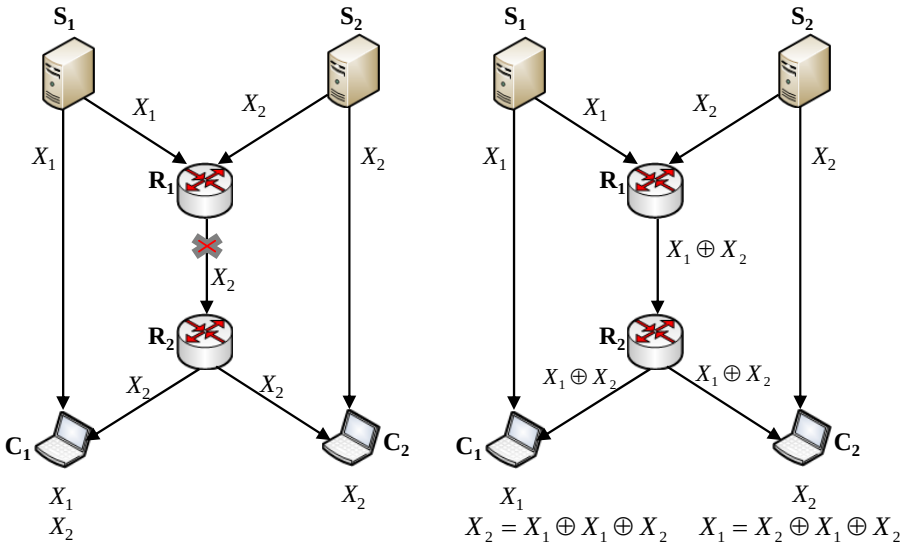


Figure 1.1: Butterfly network: An example of network coding (a) The bottleneck problem exists when using traditional store and forward mechanism, and the maximum rate is 1.5 if both rates are equal (b) Network coding (by XORing the received messages) eliminates bottleneck and the achievable rates are 2 for both sources.

sending rate of both servers are equal, the maximum achievable rate is 1.5.

On the other hand, if the router R_1 is equipped with network coding, it encodes its received packets by any linear combination of X_1 and X_2 and feeds the middle link this encoded message, then there is no more bottleneck. For example, by XORing the two messages, router R_1 sends $X_1 \bigoplus X_2$ to R_2 as shown in Figure 1.1b. Since C_1 already received X_1 and C_2 got X_2 from the side links, both clients C_1 and C_2 are able to obtain both messages X_1 and X_2 by appropriately XORing the network coded packet with the packet they already received. Therefore, the

achievable rates are 2 for both clients, which is the optimal min-cut of this network topology.

Network coding theory has been developed in various directions since first introduced in [4]. Network coding's popularity has increased and many research papers have appeared on the subject [97] [14] [21] [32] [44] [29] [41] [36] [12] [38]. In [49], Li *et. al.* showed that linear codes are sufficient to achieve the maximum flow bounds for the multicast scenarios. Koetter and Medard [46] presented polynomial time algorithms for encoding and decoding, and Ho *et. al.* extended their results to random codes [37].

1.2 Network coding benefits

The elegant network coding paradigm has proved its strength in optimizing the usage of network resources in practical networking systems such as multicast. In multicast networks, all receivers are interested in the same information. As a consequence, only the coding task has to be done inside the network while the decoding process is done at the receivers in order to reconstruct the original information. This is relatively simple to implement compared to the unicast which requires an intelligent encoding/decoding inside the network to be profitable.

Network coding has brought a new paradigm to design architectures and protocols for P2P applications. In contrast to the multicast scenario where the benefits of network coding relate to the specific network topology, the benefits of network coding in P2P systems mostly stem from solving the block scheduling problem at

large scale [54]. More specifically, network coding provides the following benefits. First, it minimizes download times. In such a large scale distributed P2P system, optimal packet scheduling is very complex. Especially, when the participating hosts only have very limited information about the underlying network topology. With network coding, the performance of the system depends much less on the specific overlay topology and schedule. Consequently, very simple mechanisms that construct a random overlay can be used. Second, due to the diversity of the coded blocks, a network coding based solution is much more robust in case the server leaves early (before all peers have finished their downloads) or in the face of high churn rates (where nodes only join for a short period of time or leave immediately after finishing their downloads). Third, in contrast to forwarding based protocols, the network coding protocol suffers only a small performance penalty when incentive cooperate mechanisms are implemented. That said, network coding has been successfully applied to content distribution [28, 29], distributed storage [25] and data dissemination [22]. In which, the most widely known application is Avalanche [54], a Microsoft prototype for large scale content distribution on a peer-to-peer network that uses network coding as the core technology. Microsoft has implemented and showed that Avalanche can improve the expected file download time over BitTorrent by 20 to 30% [1].

Network coding is also highly beneficial to media streaming applications due to its better network resources utilization. However, the application of network coding in multimedia streaming is not a trivial task as streaming applications generally impose strict timing and complexity constraints that limit the coding

opportunities. The application of network coding to media streaming has to properly consider the specificities of multimedia communication applications, such as strict delay constraints, high bandwidth requirements, as well as the unequal importance of the data that further presents some tolerance to packet losses. The design of the system has to take all these parameters into consideration in order to produce efficient solutions with a reasonable complexity. When properly designed, network coding is able to take advantage of the network diversity. Network coding can be used to improve the throughput of a streaming system, to reduce the end-to-end delay, or to increase the robustness. It also provides an efficient solution that reduces the control overhead and avoids the need for reconciliation (reduce coordination between nodes) in distributed systems. Recently, Nguyen *et. al.* proposed and showed that network coding also helps multimedia storage and streaming applications over peer-to-peer networks to improve the quality of streaming, reduce storage redundancy and increase network robustness again peer dynamics [59] [58] [60] [61].

1.3 Summary of contributions

Network coding is a novel mechanism that promises optimal utilization of the resources of a network topology. The core idea of NC allowing the intermediate nodes in a network to perform data processing has a wide range of applications well beyond its initial application to routing, impacting different disciplines from distributed data storage and security to energy efficient sensor networks and Inter-

net media streaming. To that end, this dissertation aims to develop the theories and applications of NC via four main thrusts:

1) Energy efficient NC techniques for sensor networks,

2) Novel NC techniques and protocols for Internet video streaming,

3) Stochastic data replenishment for large scale NC-based distributed storage systems,

4) Real-world implementation of NC-based distributed video streaming system.

In thrust one, we describe a novel cross-sensor coding technique that combines network topology and coding techniques to maximize the life-time of a sensor network, by addressing the uneven energy consumption problem in data gathering sensor networks where the nodes closer to the sink tend to consume more energy than those of the farther nodes. Our approach is based on the following observation from the sensor networks using On-Off Keying and digital transmission: transmitting bit "1" consumes much more energy than bit "0". Our proposed coding technique exploits this difference to reduce the communication energy by limiting the number of bits "1" in the output codeword (*low-weight codeword*) and to use NC-based *cross-sensor* coding technique to equalize the communication energy among the nodes. This cross-sensor coding scheme can significantly extend the network lifetime as compared with traditional (binary) coding by solving the energy-consumption unfairness problem. The theoretical and experimental results confirm that transmission energy can be reduced substantially (e.g., a factor of 15) and the unequal energy consumption among nodes can be practically eliminated.

In thrust two, we describe a rate distortion aware hierarchical NC technique

and transport protocol for Internet video streaming. We begin by proposing a NC-based multi-sender streaming framework that reduces the overall storage, eliminates the complexity of sender synchronization, and enables TCP streaming. Furthermore, we propose a Hierarchical Network Coding (HNC) technique that facilitates scalable video streaming to combat bandwidth fluctuation on the Internet. This HNC technique enables receiver to recover the important data gracefully in the presence of limited bandwidth which causes an increase in decoding delay. Simulations demonstrate that under certain scenarios, our proposed NC techniques can result in bandwidth saving up to 60% over the traditional schemes.

In thrust three, we present a theory of NC-based data replenishment to automate the process of data maintenance for large scale distributed storage systems. The data replenishment mechanism is the core of these systems that promises to reduce the coordination complexity and increases performance scalability. The data replenishment automates the process of maintaining a sufficient level of data redundancy to ensure the availability of data in presence of peer departures and failures. The dynamics of peers entering and leaving the network is modeled as a stochastic process. We propose a novel analytical time-backward technique to bound the expected time, the longer the better, for a piece of data to remain in P2P systems. Both theoretical and simulation results are in agreement, indicating that our proposed data replenishment via random linear network coding (RLNC) outperforms other popular strategies that employ repetition and channel coding techniques. Specifically, we show that the expected time for a piece of data to remain in a P2P system is exponential in the number of peers used to store the

data for the RLNC-based strategy, while they are quadratic for other strategies. Furthermore, the time-backward technique can be applied to problems in other disciplines such as gene population modeling in theoretical biology.

Finally in thrust four, we present the architecture, design, and experimental results of an actual NC-based distributed video streaming system. We first implement random linear network coding (RLNC) library and show the feasibility of using RLNC in P2P video streaming applications. Then we design, implement and analyze RESnc - a resilient P2P video storage and streaming over the Internet using network coding. RESnc increases the streaming throughput and data resiliency against peer departures and failures using peer diversity. These improvements are based on three architectural elements:

1) The RLNC scheme that breaks a video stream into multiple smaller pieces, codes, and disperses them throughout peers in the network, in such a way to maximize the probability of recovering the original video under peer departures and failures;

2) The scalable mechanism for automating the data replenishment process using RLNC to maintain a sufficient level of redundancy for video stored in the system;

3) The path-diversity streaming protocol for a client to simultaneously stream a video from multiple peers with minimal coordination.

Experimental results demonstrated that our system adapts well with bandwidth fluctuation, provides significant playback quality improvement and bandwidth saving.

1.4 Organization of the Dissertation

The dissertation contains seven chapters. Each chapter presents a completed work with sufficient background, detail proofs and main results. In Chapter 2, we provide the background and preliminaries on network coding, sensor networks, P2P networks and media streaming. Chapter 3 presents the energy efficient NC techniques for sensor networks. Next, we propose and investigate the novel NC techniques and protocols for Internet video streaming in Chapter 4. In Chapter 5, we describe a stochastic data replenishment for large scale NC-based distributed storage systems. Then we present a real-world implementation of NC-based distributed video streaming system in Chapter 6. Finally, we conclude the dissertation with some remarks and future work in Chapter 7.

Chapter 2 – Background and Preliminaries

2.1 Network Coding

The network coding field started with the pioneering paper by Ahlswede *et. al.*, who showed that maximum capacity in a network can be achieved by appropriately mixing data at the intermediate nodes [4]. The most elegant result of network coding is that the maximum network capacity is achievable using some *random network coding* techniques, while this is not usually possible with the traditional store and forward routing. Li *et. al.* [49] showed that linear coding with finite symbol size is sufficient for multicast. We will discuss their work shortly.

2.1.1 Linear Network Coding

Consider a system that acts as information relay, such as a router, a node in an ad-hoc network, or a node in a peer. Traditionally, when forwarding an information packet destined to some other nodes, it simply repeats it. With network coding, the relay node combines a number of packets it has received or created into the outgoing packets. Li *et. al.* proposed linear network coding [49], where outgoing packets are linear combinations of the original packets and performed over the field F_q. Linear combination is not concatenated, i.e. if we linearly combine packets of length L, the resulting encoded packet also has length L.

The encoding process can be described as follows. Assume that there are m original packets M_1, ...,M_m generated by one or several sources in the system. Using linear network coding, each relay node in the network generates the new

forwarded packet X_i by combining m original packets as:

$$X_i = \sum_{j=1}^{m} g_{ij} M_j \tag{2.1}$$

where the coefficients g_{i1}, ..., g_{im} are chosen from a finite field F_q. That node then includes the information about the coefficients $\{g_{ij}\}$ in the header of the new packets and sends the new packet to its neighbors. Encoding can be performed recursively, namely, to already encoded packets. This operation may be repeated at several nodes in the network and doesn't affect the decoding process.

Assume that the receiver node has received n encoded packets. It decodes to the original packets by solving the set of n following equations

$$\{X_i = \sum_{j=1}^{m} g_{ij} M_j, i = 1, ...n\} \tag{2.2}$$

for the unknowns $\{M_j\}$. This is a linear system with n equations and m unknown variables. In order to recover all original data packets, the receiver has to collect at least m linearly independent encoded packets, which means that the number of received packets needs to be at least as large as the number of original packets ($n \geq m$).

The question of network code design is how to select linear combinations at each node of the network. A simple algorithm is to let each node in the network select uniformly at random the coefficients over the field F_q [35], in a completely independent and decentralized manner as shown in Figure 2.1. If the field size q is

large enough, the probability of selecting linearly dependent combinations will be very small and can be negligible. Thus, the receiver will get enough independent encoded packets with high probability and small redundancy. In other words, it can retrieve the original packets with high probability.

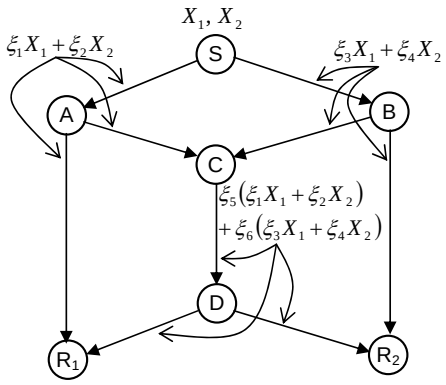


Figure 2.1: An example of random linear network coding in multicast scenario. X_1 and X_2 are the original packets that source S want to multicast to the receivers R_1 and R_2. The coefficients ξ_i are randomly chose elements of a finite field F_q. The label on each link represents the encoded packet being transmitted on the link.

2.1.2 Algebraic Network Coding

Another network code construction method was proposed by Koetter and Medard in [46]. They introduced an algebraic framework for network coding that extended previous results from Li *et. al.* [49] to arbitrary networks and robust networking. They proved the achievability with time-invariant solutions of the min-cut max-flow bound for networks with delay and cycles. They also gave an algebraic characterization of the feasibility of a multicast problem and the validity of a net-

work coding solution in terms of transfer matrices.

In the scalar algebraic coding framework of [46], the source originated packets X_j, the encoded packets transmitted on each link Y_j and the receiver output packets Z_j are treated as elements of a finite field F_q. The encoded packets Y_j transmitted on a link j is constructed by a linear combination in F_q of link j's input packets. For the delay-free case, this is represented by the equation:

$$Y_j = \sum_{source(i)=input(j)} \alpha_{ij} X_i + \sum_{link(l)=input(j)} \beta_{lj} Y_l \tag{2.3}$$

The i^{th} output packet at receiver node, Z_i, is a linear combination of the encoded packets on its terminal links, represented as:

$$Z_i = \sum_{link(l)=input(k)} \gamma_{li} Y_l \tag{2.4}$$

where all the coefficients $\{\alpha_{ij}\}$, $\{\beta_{lj}\}$ and $\{\gamma_{li}\}$ are elements of the finite field F_q. This code construction mechanism can be demonstrated by the following example. Consider the delay free network which has only one source S and one sink R as shown in Figure 2.2. The information transferring process from the source to the

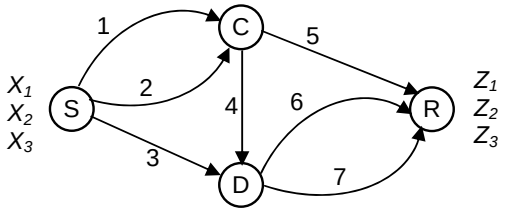


Figure 2.2: A simple network with one source and one sink, and their input and output packets.

sink can be represented by the following algebraic equations.

$$\begin{aligned} Y_1 &= \alpha_{11}X_1 + \alpha_{21}X_2 + \alpha_{31}X_3 \\ Y_2 &= \alpha_{12}X_1 + \alpha_{22}X_2 + \alpha_{32}X_3 \\ Y_3 &= \alpha_{13}X_1 + \alpha_{23}X_2 + \alpha_{33}X_3 \\ Y_4 &= \beta_{14}Y_1 + \beta_{24}Y_2 \\ Y_5 &= \beta_{15}Y_1 + \beta_{25}Y_2 \\ Y_6 &= \beta_{36}Y_3 + \beta_{46}Y_4 \\ Y_7 &= \beta_{37}Y_3 + \beta_{47}Y_4 \end{aligned} \tag{2.5}$$

$$\begin{aligned} Z_1 &= \gamma_{51}Y_5 + \gamma_{61}Y_6 + \gamma_{71}Y_7 \\ Z_2 &= \gamma_{52}Y_5 + \gamma_{62}Y_6 + \gamma_{72}Y_7 \\ Z_3 &= \gamma_{53}Y_5 + \gamma_{63}Y_6 + \gamma_{73}Y_7 \end{aligned} \tag{2.6}$$

Let matrices $\mathbf{A}$ and $\mathbf{B}$ be defined as

$$\mathbf{A} = \begin{bmatrix} \alpha_{11} & \alpha_{21} & \alpha_{31} \\ \alpha_{12} & \alpha_{22} & \alpha_{32} \\ \alpha_{13} & \alpha_{23} & \alpha_{33} \end{bmatrix} \tag{2.7}$$

$$\mathbf{B} = \begin{bmatrix} \gamma_{51} & \gamma_{61} & \gamma_{71} \\ \gamma_{52} & \gamma_{62} & \gamma_{72} \\ \gamma_{53} & \gamma_{63} & \gamma_{73} \end{bmatrix} \tag{2.8}$$

then the transfer matrix describing the relationship between the output and input processes $\underline{z} = \mathbf{M}\underline{x}$ can be calculated as

$$\mathbf{M} = \mathbf{B} \begin{bmatrix} \beta_{15} & \beta_{25} & 0 \\ \beta_{14}\beta_{46} & \beta_{24}\beta_{46} & \beta_{36} \\ \beta_{14}\beta_{47} & \beta_{24}\beta_{47} & \beta_{37} \end{bmatrix} \mathbf{A} = \mathbf{BGA} \tag{2.9}$$

For a network code to exist, the above equation must be solvable, i.e. the transfer matrix $\mathbf{M}$ must be invertible. Therefore, we can choose the coefficients $\{\alpha_{ij}\}$, $\{\beta_{lj}\}$ and $\{\gamma_{li}\}$ in the finite field F_q so that the determinant of $\mathbf{M}$ is nonzero over F_q. One such solution is choosing $\mathbf{M}$ to be an identity matrix. There exists an infinite number of solutions to this problem, namely all assignments of parameters which render a nonzero determinant of the transfer matrix $\mathbf{M}$.

2.1.3 Applications of Network Coding

Network coding is a novel mechanism that promises optimal utilization of the resources of a network topology. With network coding, intermediate nodes in the network mix information from different flows rather than just forward them independently. This elegant transmission paradigm has proved its strength in optimizing the usage of network resources and might have interesting applications in practical networking systems such as multicast. In multicast networks, all receivers are interested in the same information. As a consequence, only the coding task has to be done inside the network while the decoding process is done at the receivers in order to reconstruct the original information. This is relatively simple to implement compared to the unicast case which requires an intelligent encoding/decoding inside the network to be profitable.

The other scenario that profited from the power of network coding is file sharing applications running on top of peer-to-peer networks. With network coding, every transmitted packet is a linear combination of all or a subset of the packets available at the sender. And the encoded packets can be further recombined to generate new linear combinations, which enables nodes to generate encoded packets without having the full file. The original information can be reconstructed after receiving enough linearly independent packets.

Network coding is of great use in large-scale distributed systems [29]. The most widely known application is Avalanche [54], a Microsoft prototype for large scale content distribution on a peer-to-peer network that uses network coding as the core

technology. Microsoft has implemented and shown that Avalanche can improve the expected file download time over BitTorrent [1] by 20% to 30%. Recently, Nguyen *et. al.* proposed and showed that network coding also helps multimedia storage and streaming applications over peer-to-peer networks to improve the quality of streaming, reduce storage redundancy and increase network robustness again peer dynamics [59] [58] [60].

2.2 Sensor network

A sensor network is typically composed of a large number of small-size, low-cost, low power sensor nodes which are randomly deployed and communicated with one another to send data to the sink. Because of the limited capability of sensor nodes, data can only be transmitted to the neighboring nodes, which then further relay the data to its neighbor in the direction to the sink. Traversing hop by hop, the report finally reaches the sink to be analyzed and taken further action as required by the application (as shown in Figure 2.3).

Sensor networks have a variety of applications such as environmental monitoring, habitat monitoring, geophysical monitoring, military surveillance etc. A typical use of sensor networks is to have a kind of streaming data, in which little amount of data (typically just a few bytes) are transmitted periodically as in temperature measurement system. The large number of nodes allows taking advantage of short-range, multi-hop communication to conserve energy, especially when data aggregation is applied.

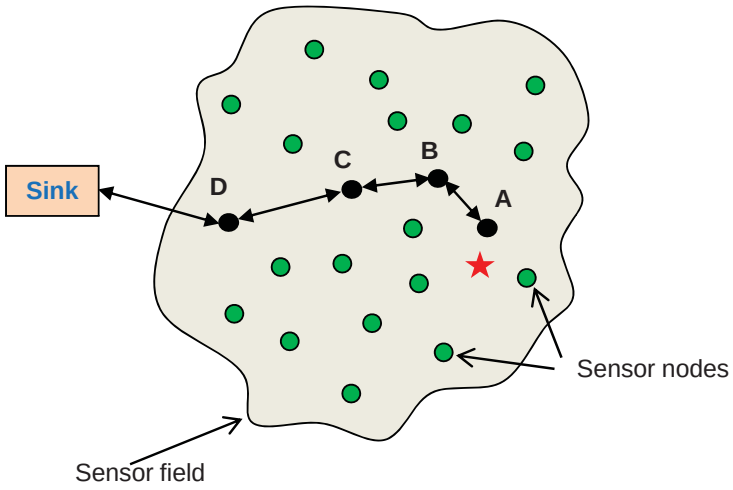


Figure 2.3: Sensor network and multi-hop transmission.

Sensors networks are most valued for their small size and low processing capabilities. A small sensor can only be equipped with limited power source. Also, in most cases, the environment to be monitored does not have an existing infrastructure for either energy or communication. It becomes imperative for sensor nodes to survive on limited energy and communicate through a wireless communication channel. Energy efficiency is a major concern in wireless sensor networks.

Typically, a wireless sensor node includes four basic components [72]: (i) a sensing subsystem for data acquisition from the physical surrounding environment; (ii) a processing subsystem for local data processing and storage; (iii) a wireless communication subsystem for data transmission, and (iv) a power supply subsystem. In which, communication subsystem is the main source of energy consumption. The energy cost of transmitting a single bit of information is approximately the

same as that needed for processing a thousand operations in a typical sensor node [69]. The energy consumption of the sensing subsystem depends on the specific sensor type. In many cases it is negligible with respect to the energy consumed by the processing and, above all, the communication subsystems. In other cases, the energy expenditure for data sensing may be comparable to the energy needed for data transmission. In general, energy-saving techniques focus on two subsystems: the communication subsystem (i.e., energy management is taken into account in the operations of each node, as well as in the design of networking protocols), and the sensing subsystem (i.e., techniques are used to reduce the amount or frequency of energy-expensive samples).

Energy optimization in wireless sensor networks becomes more complicated because it involved not only the reduction of energy consumption at individual node but also prolonging the network lifetime. Thus, energy awareness must be incorporated into the entire network, not only at the individual node. These necessitate energy-awareness at all layers of networking protocol stack. The issues related to physical and link layers are generally common for all kind of sensor applications. Many solutions have been proposed and focused on system-level power awareness such as dynamic voltage scaling, radio communication hardware, low duty cycle, system partitioning, energy aware MAC protocols [34] [55] [93] [94] [83]. At the network layer, the main aim is to find ways for energy efficient route setup and reliable relaying of data from the sensor nodes to the sink so that the lifetime of the network is maximized [89] [40] [20]. In our work, we tackle the problem at the upper layer. Specifically, we exploit the delay-energy trade-

off and the known network topology in order to reduce the transmission energy consumption of the network. Using idea of network coding, we are able to design an efficient cross-sensor coding technique to prolong the network lifetime of the low-energy sensor networks.

2.3 Internet media streaming

Recent years have witnessed an explosive growth of the Internet with greater network bandwidth and increasing access to networks. The advances in computer networking together with powerful personal devices make multimedia streaming practical and affordable for ordinary consumers. The demand of multimedia services over the Internet is expanding from PC into new user platforms such as hand-held wireless devices, interactive television set-top boxes, games consoles, 3G mobile phones, etc. It has received tremendous attention from academia and industry to accommodate this growing demand.

Multimedia streaming over the Internet is challenging due to bandwidth fluctuation, delay and packet loss. The current best-effort Internet does not offer any quality of service (QoS) guarantees to multimedia streaming. It does not provide any guarantees that media packet will reach its destination, and the packets which do arrive may not arrive in the same order to which they were sent. Furthermore, it is difficult to achieve both efficiency and flexibility in streaming. To address these challenges, extensive research has been conducted and many solutions have been proposed, ranging from source and channel coding to network protocols and archi-

tecture. For example, to combat the fluctuating and limited bandwidth, a scalable video bit stream is used to allow a sender to dynamically adapt its video bit rate to the available bandwidth at any point in time [86]. To reduce packet loss and the associated delay due to the retransmissions of the lost packets, Forward Error Correction (FEC) techniques have been proposed to increase reliability at the expense of bandwidth expansion [53]. However, all the existing approaches have their own drawbacks that can not fully utilize the benefit of distributed computing on the Internet. There is a need to develop a new framework for multimedia storage and streaming over the Internet. One promising approach is the recent development of network coding (NC) paradigm which has been shown to improve the performance of the networks, such as bandwidth efficiency and robustness to network dynamics. Before discussing the benefits of network coding in multimedia streaming, we first begin with a review of the existing multimedia streaming architectures.

2.3.1 Streaming architecture

The traditional architecture for multimedia streaming over the Internet is client-server model. It allocates servers and network resources for each client request. One variation of client-server model is Content Delivery Network (CDN). CDN aims to improve the throughput by pushing content to the servers strategically placed at the edge of the network. This allows a client to choose the server that results in shortest round-trip time and/or least amount of congestion (which is called edge streaming). The major challenge of server based multimedia streaming

is its scalability. The server load and network bandwidth at server side must grow proportionally with the client population. This makes the client-server approach very expensive and not scalable.

For example, Youtube, the most popular video storage and streaming system on the Internet where its users can upload, view and share their video clips with others, employs CDN to stream video to end users. In January 2008 alone, nearly 79 million users had viewed over 3 billion videos. Although the numbers are impressive, majority of YouTube videos are rather short and of low-quality, resulting in manageable amounts of storage and streaming bandwidth. This allows YouTube to centrally control and coordinate their servers. However, its bandwidth cost is estimated at one million dollars a day. This does not scale since the server computational capability and the network bandwidth at the server side ultimately limit the number of clients that can be served at any given time.

Recently, the multi-sender streaming paradigm has been proposed as an alternative to edge streaming to provide smooth video delivery [74, 63, 8]. The main idea is to have each server storing an identical copy of the video. The video is partitioned into multiple disjoint parts, each part is then streamed from separated server to a single receiver simultaneously. Having multiple senders is in essence a diversification scheme in that it combats unpredictability of congestion in the Internet. However, the above schemes are all rely on the dedicated server and not storage efficient. Specifically, smooth video delivery can be realized if we assume independent routes from various senders to the receiver, and argue that the chances of all routes experiencing congestion at the same time is quite small. If the route

between a particular sender and the receiver experiences congestion during streaming, the receiver can re-distribute rates among the existing senders, or recruit new senders to provide the required throughput. This requires a careful synchronization among the senders to ensure that distinct partitions of a video are sent by different senders in order to increase the effective throughput. In other words, one must design an optimal partition algorithm to dynamically assign chunks of different lengths to different senders based on their available bandwidths. This dynamic partition algorithm, however is often suboptimal due to the lack of accurate available bandwidth estimation.

The recent development of Peer-to-Peer (P2P) networks opens a new possibility of building completely distributed systems that can eliminate the computational and bandwidth bottlenecks existed in the client-server architecture. The basic design philosophy of P2P is to encourage users to act as both clients and servers (peers), where a peer not only downloads data from the network, but also uploads its data to other peers in the network. The uploading bandwidth of peers is efficiently utilized to reduce the bandwidth burdens placed on the servers.

The advantage of P2P systems resides in their capability for self organization, scalability, and path diversity, which are all very attractive features for effective delivery of media streams over the networks. However, providing streaming service over P2P networks is still a challenging task because of their inherent instability and unreliability. To that end, a fair amount of distributed system research has recently been focused on using P2P platforms to build reliable, large scale distributed systems for media storage and delivery over the Internet [76, 85, 73]. In

these systems, data is stored distributedly at different peers. In some P2P multimedia media streaming systems, original data such as a video file does not have to be stored entirely at a peer. Rather, a video is broken into multiple pieces, coded properly, then dispersed to a number of peers in the network. Given a policy on data redundancy, each peer may contain only a fraction of the video, but collectively, the entire network has the entire video plus some redundancy to safeguard data against peer failures or departures. Furthermore, the dispersion of data over the network provides robustness against the single point of failure. In this setting, to watch a video, a client requests simultaneous transmissions from multiple peers that collectively have the complete requested video. This approach creates multiple Internet routes for transmitting a video to a client, resulting in larger throughput.

However, the specificity of media applications in terms of bandwidth, delay, and reliability are not completely addressed by the characteristics of unstructured P2P systems. The lack of coordination of such systems, the limited peer capabilities, and the low system stability over time represent a great challenge for the deployment of high quality P2P streaming applications. The replacement or extension of conventional media delivery infrastructures with P2P systems clearly requires the adaptation of existing coding, routing, and scheduling algorithms to unreliable network environments. Network coding is a promising approach which has recently emerged as an alternative to traditional routing algorithms.

2.3.2 Peer-2-Peer streaming

Peer-to-peer (P2P) systems are becoming increasingly popular due to their ability to deliver large amounts of data at a reduced deployment cost. In which, BitTorrent system [1] is the most popular application. A BitTorrent file is partitioned into multiple distinct pieces, and these pieces are then exchanged among the peers to increase the receiving throughput as compared to using a single server to send the pieces to multiple receivers. Thus, at any point in time, a peer can receive multiple pieces from different peers. BitTorrent can be viewed as multi-sender system.

P2P systems also represent a scalable and cost-effective alternative to traditional media streaming services that enables extended network coverage in the absence of IP multicast or expensive content distribution networks (CDN). As a P2P system does not provide any guaranteed support to streaming services, these must rely on self-organized and adaptive network architectures to meet their stringent quality requirements. The delivery architectures for streaming applications can be broadly classified into two categories: (1) tree-based overlay for streaming sessions from media sources to a pool of client peers, and (2) mesh overlay for massive parallel content distribution among peers.

2.3.2.1 Tree-based topology

Tree-based overlay organizes the peers as a single or multiple trees overlay rooted at the media source as shown in figure 2.4. Clients are leaf nodes in the distribution tree, while intermediate peers push the content from the source. A peer can

simultaneously be a leaf in some distribution trees and an intermediate node in others.

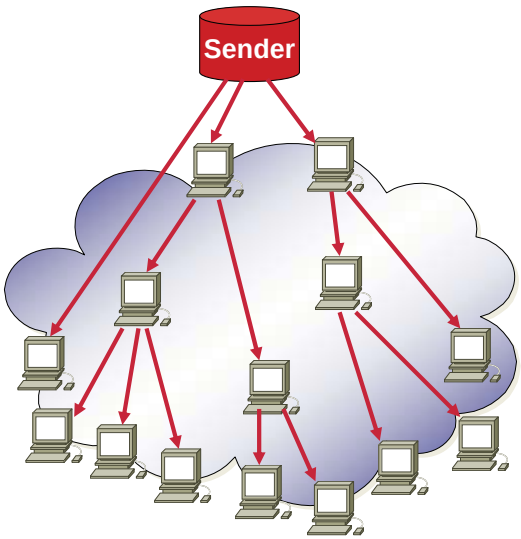


Figure 2.4: Tree-based topology.

The first tree-based approaches to p2p streaming derived from early research on application-level overlays [9] [87]. The main motivation was to provide the benefits of an easily deployable multicast infrastructure. In which, no native support for IP multicast was available, and an unique overlay tree was used to convey data. End System Multicast (ESM) [15] is the first large-scale video distribution system based on a single-tree multicast infrastructure had been deployed. NICE [9] organizes nodes in a hierarchical, cluster-based single tree overlay, trying to optimize the average delay through appropriate node management policies and cluster-head selection criteria. ZIGZAG [87] improves on this design by adding redundancy to

the cluster management mechanisms to better cope with node churn. ZIGZAG organizes receivers into a hierarchy of bounded-size clusters and builds the multicast tree based on that.

The fundamental shortcoming of all tree-based systems is due to the limitations imposed by the tree structure. Single tree architectures are easy to implement and maintain. However, they are highly vulnerable to peers joining/leaving the system (peer churn). A departure peer will temporarily disrupt video delivery to all peers in the subtree rooted at the departed peer. The maintenance, optimization, and recovery of failed overlay tree links can become a daunting task under heavy churn. In addition, the delivery rate is limited by the minimum upload bandwidth of the intermediate peers in the branch, as each client is connected to the source through a single tree branch. Each internal node is a possible bandwidth bottleneck for the subtree it serves, and packet losses do accumulate while descending the tree.

Multiple-tree based overlays have been proposed to address the above problems by providing redundancy in network paths. They encode the stream as multiple independent MDC (multiple description coding) stripes [12] and stream them over several trees. For example, Splitstream [13] creates multiple interior-node-disjoint trees. Every peer is an interior node in at most one tree, thus mitigating the bottleneck data loss problem (a single failure results in the interruption of at most one stripe), and the capacity limiting problem (each node is likely to be an interior node in at least one tree). In Coopnet [67], Padmanabhan *et al.* used multiple overlay multicast trees to stream multiple descriptions of the video to the clients. Each multicast tree transmits a description of the video. When a large number

of descriptions are received, higher video quality can be achieved. Recently, Li *et al.* proposed MutualCast [48] which focuses on throughput improvement for data dissemination in P2P network. MutualCast employed partitioning techniques and a fully connected topology to ensure that the upload bandwidth of all the nodes is fully utilized.

However, designing and maintaining such multiple-tree based systems becomes less trivial. These also can lead to solving contradictory issues such as minimizing tree depth, while simultaneously provisioning network path diversity. Most importantly, the underlying physical topology must be carefully considered to achieve efficient content dissemination [66].

2.3.2.2 Mesh topology

Mesh overlay architecture is based on self organization of nodes in a directed mesh to deliver media to clients as shown in Figure 2.5. The original media content from a source is distributed among different peers. A peer is connected to the mesh through one or more parent peers, where it retrieves media information and a set of child peers, to which it serves media packets.

Mesh-based designs aim to further reduce the structural constraints in media streaming systems to improve their resilience against churn and node transience. The stream is divided into a number of chunks, which are distributed by the source to few nodes in the system. Nodes must exchange chunks to retrieve a sufficiently complete stream before the play-out deadline. Bullet [47] is an early approach

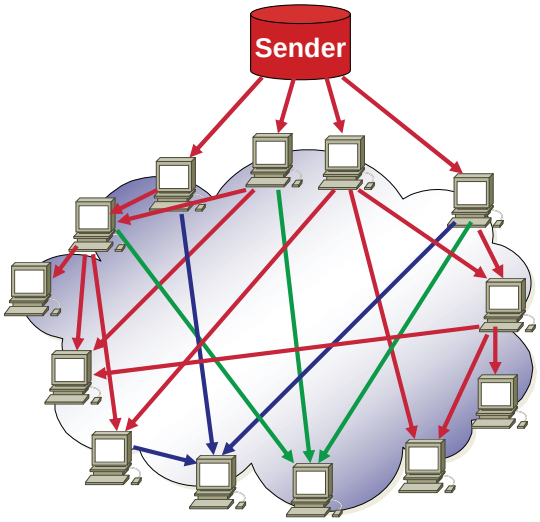


Figure 2.5: Mesh topology.

that combines a single-tree and a mesh: the tree is used to convey both data chunks and control information, while the mesh is created independently by the nodes based on the control information. An advantage of this scheme is that the control protocol running on the tree can have a very low overhead. On the other hand, no mechanism to encourage bandwidth contribution has been implemented in the system. Coolstreaming/DONet [95] introduced a better chunk scheduling algorithm that takes into account the individual chunk deadlines for play-out. While each node performs a periodical long-term optimization of the mesh by replacing the least contributing neighbors, the system is not meant to address the combined effects of upload bandwidth heterogeneity under global shortage and high churn rates. User experiences with the Coolstreaming application seem to indicate

that it suffers from a high play-out latency, probably due to conservative data buffering. PULSE [68] constantly optimizes its mesh of data connections using a feedback-driven peer selection strategy that is based on pairwise incentives. Its dynamically optimized mesh supports high bandwidth heterogeneity and churn, while its buffering requirements are typically low even under a moderate shortage of global serving capacity.

However, these systems do not use any sophisticated coding. Thus the performance can be shown *theoretically* lower than those of P2P systems that employ coding to cope with the variation in unstable P2P networks. For example, video can be compressed and packaged in a form that facilitates adaptation to variable network bandwidth, packet loss, and delay; error control mechanism in the form of packet retransmission and forward error correction can help increase robustness of streaming applications.

Systems that employ coding techniques include the work of Byers *et al.* [11] on informed overlay network. In this work, Byers *et al.* proposed to partition data and make use of peers to increase the throughput of the system. In this approach, each node randomly sends different coded partitions on different links. Data reconciliation techniques are then used to reduce data redundancy sent between nodes.

Recently, network coding helps increase network capacity (by reducing the amount of replication) and its resilience to packet loss. Random network coding technique has been previously used in Avalanche system for file distribution in P2P network [29], in distributed storage system [3], and in live peer-to-peer streaming [91]. R^2 [91] incorporates random network coding along with a random pushing

algorithm to smooth the latency problems, reduce buffer level and increase system scalability.

The potentials of network coding in peer-to-peer video streaming have yet to be explored. There are additional issues arising from the unique features of streaming applications. First, unlike file, video is loss-tolerant. With network coding, however, available data blocks might not be decodable if one or more blocks are missing before playback deadline. Second, given the unbounded session time of live streaming, the buffer at each node has to be updated over time to remove obsolete data. This is different from bulk file download where the buffer is just allocated for the file with minimizing the filling up time being the key objective.

Chapter 3 – Energy Efficient Coding Techniques for Sensor Networks

1. Kien Nguyen and Thinh Nguyen, "*Cross-sensor Coding Techniques for Low Energy Sensor Networks*", The 6^{th} International Wireless Communications and Mobile Computing Conference (IWCMC10), June 2010, Caen, France.

2. Kien Nguyen, Thinh Nguyen, and Senching Cheung, "*On Reducing Communication Energy Using Cross-Sensor Coding Technique*", submitted to International Journal of Distributed Sensor Networks, 2010.

3.1 Introduction

Recent years have witnessed an explosive growth of sensor networks designed for environmental data gathering and monitoring [5] [42] [71]. A typical sensor network for data gathering consists of a large number of battery operated sensor nodes, each is capable of sensing and forwarding its data wirelessly to the processing node through its neighboring nodes. The sensor networks are often designed to operate at low power in order to prolong their life times. However, building a truly energy efficient sensor network is still a challenging problem. There has been a large body of work on reducing energy consumption of sensor networks using different approaches, from low-voltage hardware designs [71] and transmission schemes [33] to in-network processing and routing algorithms [93].

Recently, several energy efficient algorithms for sensor networks based on the correlation characteristic of data have been studied. From signal processing perspective, if the measured data among the nodes are spatially correlated, a node can jointly compress its data and its neighbor's data to increase the capacity of the network [18] [19] [17] or to minimize the energy to transmit the data [51] [82]. The larger correlation results in larger energy saving [84]. The argument for energy reduction using data compression is based on the entropy concept from information theory [16]. The entropy represents the amount of information in terms of the number of bits. Therefore, by compressing the data to a smaller number of bits, less energy is required to send these bits. However, in many situations with practical sensor networks, the assumption of data correlation is not applicable.

An important problem in sensor network is the uneven energy consumption of the nodes, specifically the energy spent on communication. The nature of data forwarding in a sensor network implies that sensor nodes closer to the processing node (the sink) tend to spend more energy than those of the farther nodes in order to relay the accumulated data. Figure 3.1 shows an example in which data are collected and relayed through the internal nodes 2 and 3, and eventually to the processing node 4. Thus, nodes 2 and 3 consume two and three times more energy than node 1, respectively because they have to relay the data of other nodes. This energy unfairness problem can significantly shorten the life time of a sensor network.

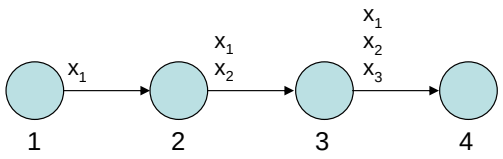


Figure 3.1: A simple sensor network. All measured data from nodes 1, 2, and 3 are relayed to node 4

To see the seriousness of this problem, we assume that all the nodes have identical batteries designed to continuously measure and transmit *one sample of data* to the next hop for 3 months. Then, node 2 will run out of battery in 1.5 months, and node 3 will run out of battery in 1 month. Thus, while nodes 1 and 2 still function after one month, the sensor network is unusable due to node 3's failure. In other words, the life time of this sensor network depends on the life time of node 3. Therefore, it is extremely important to incorporate the knowledge

of the network topology into the design of a long-lived sensor network. We will show that *by allowing the internal nodes to re-encode the data they receive with appropriate codes based on its position in the network, the energy consumption can be significantly reduced.*

This work proposes a low-weight *cross-sensor* coding technique to reduce the communication energy and alleviate the problem of unequal energy consumption among nodes in a sensor network, thus improving its lifetime. Our approach is based on the following observation. Assume that digital transmission is used, and transmitting bit "1" consumes much more energy than bit "0". This is typically true in optical communication where transmitting a bit "1" requires much energy to generate a laser pulse, while being silent represents a bit-"0" transmission. Our proposed coding technique exploits this difference to reduce the communication energy by limiting the number of bits "1" in the output codeword (*low-weight codeword*) and to use a *cross-sensor* coding technique to equalize the communication energy among the nodes. The proposed technique is designed to reduce the communication energy for fiber-optic sensor networks similar to the one used by Selker *et al.* as shown in Fig. 3.2 [81]. This sensor network consists of temperature sensors connected to each other through a fiber-optic cable. Each sensor can measure temperature at temporal resolution of fractions of a minute.

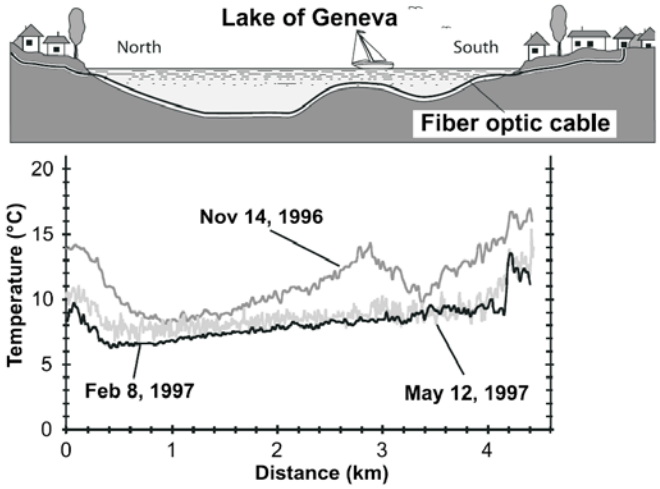


Figure 3.2: An fiber-optic sensor network deployed in lake Geneva.

3.2 Related Work

There has been a large amount of research on designing energy efficient sensor networks, from hardware designs and modulation schemes to MAC/routing protocols and in-network processing techniques. In this section, we list a few works in this area.

Many data aggregation algorithms for sensor networks have been recently proposed for energy optimization [19, 17, 18, 51]. In [19], Cristescu *et al.* apply information network theory to sensor network by using a simple model for data compression and devise approximate algorithms for constructing a shortest path tree to aggregate data from multiple sensors to a sink in order to minimize the total energy. While this approach can greatly reduce the energy consumption, it relies on large correlation of data at different sensors to achieve low energy consumption,

which may not be true in many real-world scenarios.

Similar work on aggregation of correlated data in sensor network are studied by Enachescu and Sharaf in [27] and [82]. In [27], the authors propose a simple randomized algorithm for routing data on a grid of sensors in a way that promotes data aggregation. They show that their randomized algorithm is a constant factor approximation to the optimal aggregation tree. On other hand, our work does not rely on data correlation.

Several researchers work on constructing the optimal prefix codes with unequal letters costs [43, 10, 30]. They solved this problem by transformation to integer linear programming to find the bound and necessary coefficients. In [43], Karp also proved and proposed a method to construct the optimal codes from those coefficients.

Our approach is similar to the works of Liu *et al.* [52], Kim *et al.* [45], and Prakash *et al.* [70], in which the authors optimize the energy consumption by exploiting different transmission energies for bits "1" and "0". Our work is also similar to the silence based communication paradigm proposed in [98]. In [23], Dhulipala *et al.* also characterized the complexity of silence based communication. On the other hand, unlike previous approaches, our approach introduces a novel concept of coding data across different sensors which can alleviate the problem of uneven energy consumption at different sensors, thus prolonging the lifetime of a sensor network.

3.3 Sensor Network Model

This work does not address the overall energy consumption problem for the general class of sensor networks. Instead, it is useful for reducing communication energy for a number of sensor networks in which the following assumptions hold:

1. Data is transmitted digitally using On-Off Keying (OOK) technique with bit "1" using more energy than that of bit "0".
2. Largest source of energy consumption is transmission.
3. All the sensor nodes are equipped with a loosely synchronized clock.
4. The sensor network is designed to collect data at low frequency, i.e. low bandwidth requirement.
5. Sensors are aligned and relay data in a linear fashion.

Assumption 1 is critical to our coding technique since we utilize the characteristic of OOK that more energy is required to transmit bit "1" than bit "0". Assumption 2 typically holds in many sensor networks, and thus, minimizing the transmission energy significantly improves the network lifetime. Therefore, in our energy performance evaluation, we only consider the energy required to transmit, ignore the energy consumed by the nodes to listen and to perform any necessary computations. We can easily add these other sources of energy consumption in future work.

For this scheme to work, both receiver and sender must have a synchronized clock, thus assumption 3 follows. As for assumption 4, our coding technique is

more effective in term of energy reduction when there is no or little restriction on the bandwidth requirement. This typical assumption often holds since sensor networks collect data at the large intervals, e.g., on the order of minutes, resulting in a very low bandwidth. Therefore, the clocks at different nodes are only needed to be synchronized to a certain resolution.

Assumption 5 enables a node to code the data effectively. This assumption arises with the fundamental characteristic of sensor networks where data is only sent in one direction i.e., from sensor nodes to the sink as most of the sensor networks are used for data aggregation. Furthermore, our work focuses on the environmental sensor applications in which a typical sensor network includes only hundreds of sensors.

Finally, the discussion on transmitted bit errors is outside the scope of this research.

3.4 Coding Schemes

3.4.1 1-bit Coding

Typical entropy coding techniques such as Huffman coding, aim to minimize the number of bits per symbol, given a probability distribution of the source data. The symbols with frequent occurrences are coded using fewer number of bits than those of the rare symbols. As a result, the average number of bits per symbol is reduced. However, Huffman coding only minimizes the number of bits, regardless

whether the bit is "0" or "1". This may result in higher transmission energy. To reduce transmission energy, one can use the 1-bit coding scheme [70] in which, every codeword consists of at most 1 high bit as below:

$$\begin{array}{lcl} 0 & \rightarrow & 00000 \\ 1 & \rightarrow & 00001 \\ 2 & \rightarrow & 00010 \\ 3 & \rightarrow & 00100 \\ 4 & \rightarrow & 01000 \\ 5 & \rightarrow & 10000 \end{array}$$

Assume that it takes no energy to transmit bit "0", then the transmission energy per symbol is much smaller for the 1-bit coding than the traditional coding schemes such as direct binary coding or Huffman coding. The 1-bit coding scheme employs fixed-length codes for differentiation of symbol boundary. Furthermore, the receiver and sender's clocks must be synchronized in order for the receiver to detect the *silent* interval of bit "0". As a result, the 1-bit coding scheme results in longer average time to transmit a symbol.

The 1-bit coding technique can be generalized to k-bit coding which uses slightly more energy per symbol in order to reduce the delay. A k-bit coding symbol consists of at most k bits "1" per codeword. Therefore, the number of possible k-bit codewords of length n is:

$$M(n,k) = \sum_{i=0}^{k} \left(\begin{array}{c} n \\ \\ i \end{array} \right) \tag{3.1}$$

The individual k-bit codewords of length n are then simply the enumeration of the

combinations $\binom{n}{i}$ for $i = 0...k$. Clearly, larger k results in larger energy consumption. We note that if the number of possible symbols exceeds 2^n where n is the length of a k-bit symbol, then n will be increased, but k remains the same in order to keep the transmission energy constant.

3.4.2 Cross-Sensor Coding Technique

The previous coding scheme reduces the energy consumption at each sensor, however, it cannot solve the problem of energy unfairness as explained in Section 3.1. One may think that the relay architecture used in the sensor network fundamentally produces this energy unfairness problem since some nodes, (e.g., the internal nodes) will have to transmit more data, thus resulting in higher energy consumption. However, this is not necessarily the case. The cross-sensor coding technique described below aims to solve this problem.

We first begin with the definition of energy unfairness which ultimately affects the life time of a sensor network. The energy unfairness is defined as follows:

Definition 3.1

$$F_E = \frac{1}{n} \sum_{i=1}^{n} \left(X_i - \frac{\sum_{j=1}^{n} X_j}{n} \right)^2 \tag{3.2}$$

where n denotes the number of sensors, X_i denotes the average transmission energy at node i which is proportional to the number of 1's in a codeword.

Intuitively, the energy unfairness represents the variance of the average energy consumption by the nodes. The lower value of F_E indicates higher fairness and

thus is desirable in a long-lived sensor network. We will show that our cross-sensor coding technique can reduce F_E significantly.

To illustrate our coding techniques and their advantages, we first consider the traditional data relay in a simple sensor network consisting of four sensors and a processing node as shown in Figure 3.3.

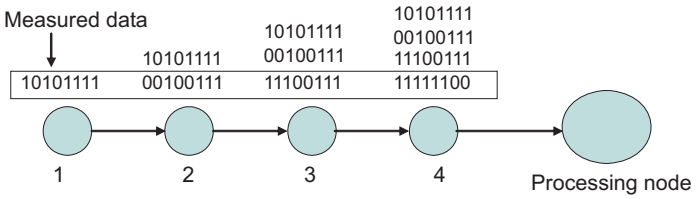


Figure 3.3: All measured data from nodes 1, 2, 3, 4 are relayed to the processing node.

3.4.2.1 Traditional Data Relay

Suppose the sensors are designed to measure the data with 8-bit resolution. Assume that at time t, the measured data for nodes 1, 2, 3, and 4 are 10101111, 00100111, 11100111, and 11111100, respectively. Assuming that propagation delay is negligible, then to transfer these data to the processing node (from left to right in Figure 3.3), the traditional relay method requires: node 1 transmits 6 bit 1's, node 2 transmits 10 bit 1's, node 3 transmits 16 bit 1's, and node 4 transmits 22 bit 1's. Therefore, the total number of transmitted bits equals to $6 + 10 + 16 + 22 = 54$ bits. Assume that the energy for transmitting a bit "1" is 1 joule and for transmitting bit "0" is 0 joule, then the total energy consumption per one round

of data gathering in this sensor network is 54 joules. Furthermore, the energy consumption of node 4 is almost 4 times the larger than that of node 1. In general, it is easy to show that the average number of transmitted bits per sample, hence the energy per sample, is $O(mn^2)$ where m is the number of bits per sample (data resolution) and n is the number of sensor nodes.

3.4.2.2 Direct 1-bit Coding

Now suppose we apply 1-bit coding to the 8-bit data directly, and each node relays the data in the same way as the traditional method. Since each sample data is coded with only one bit "1", the total number of transmitted bits for this scheme scheme equals $1 + 2 + 3 + 4 = 10$ bits, resulting a factor of 5 in energy reduction compared to that of the traditional coding technique. The direct 1-bit coding data scheme results in much higher delay. Assume that a node relays the data in the next time slot after it receives a data sample completely, then there is an additional 255 bit delay per hop. This relay technique (also called *store and forward* scheme) is employed in packet-switched networks. The *store and forward* scheme waits until all bits in a packet are received before transmitting the first bit of the packet onto the next link. In our case, a packet is a data sample (255 time slots). Consequently, the time it takes for the first bit of node 1's data to arrive at the processing node equals to $255 \times 3 = 765$ time slots as compared to only $8 \times 3 = 24$ time slots for traditional binary coding. In general, we prove the following theorem regarding the direct 1-bit coding scheme for sensor networks.

Proposition 3.1 *Given the following*

1. *A sensor network consisting of n sensor nodes and one processing node, all are aligned and forward data in a linear fashion.*

2. *Data gathering is based on store and forward scheme with a packet being a data sample of m-bit resolution.*

3. *Measured data are randomly and uniformly distributed.*

Then, using direct 1-bit coding scheme with time slot of T second per bit, results in:

1. *The average number of transmitted bits TB per sample data and node is*

$$TB = \left(1 - \frac{1}{2^m}\right) \frac{n+1}{2} \tag{3.3}$$

2. *The achievable bandwidth B is*

$$B = \frac{mn}{T(2n-1)(2^m-1)} \qquad [bps] \tag{3.4}$$

3. *The energy fairness is*

$$F_E = \left(1 - \frac{1}{2^m}\right)^2 \left(\frac{n(n+1)(n-1)}{12}\right) \tag{3.5}$$

Proof 3.1 *To prove Equation (3.3), one notes that the first node transmits its sample data to node 2 using at most 1 bit "1". There is a $1/2^m$ chance that node*

1 does not transmit any bit "1". This happens when its measured value maps to a 1-bit codeword that contains all zeros. Thus, the average number of bits transmitted by node 1 is $(1-1/2^m)$. *Since node 1's bit must be relayed through* $n-1$ *nodes before arriving at the processing nodes, the total number of transmissions for this bit is therefore* $(1-1/2^m)n$. *Similarly, the bit from a node 2, will have to be transmitted through* $n-2$ *nodes, resulting in* $(1-1/2^m)(n-1)$ *transmissions. Thus the average number of transmissions for all the sample data is the total number of transmitted bits per* n *sample data:*

$$\left(1-\frac{1}{2^m}\right)n+\left(1-\frac{1}{2^m}\right)(n-1)+...\left(1-\frac{1}{2^m}\right)=\left(1-\frac{1}{2^m}\right)\frac{n(n+1)}{2}$$

Divided the result by n, *we obtain the average number of bits per sample data and node*

$$TB=\left(1-\frac{1}{2^m}\right)\frac{n+1}{2}$$

To prove Equation (3.4), one notes that 2^m-1 *is the number of time slots required to encode* m*-bit data using 1-bit coding. Since there are* n *nodes, the time for the first data bit of the first node to arrive at the last node is* $(n-1)(2^m-1)T$. *All the subsequent bits will arrive at the processing node at the rate of* $1/T$ *bps. There are a total of* n *data sample transmitted from the last node to the processing node, the time it takes for all* n *data samples to arrive at the processing node is* $n(2^m-1)T$. *Adding the delay of the first bit and the transmission time of* n *samples together, we obtain the total time to receive all* n *samples of* m *bits each. Thus the*

bandwidth is

$$\frac{mn}{T(2n-1)(2^m-1)}$$

The second sample data of the first node can be sent after all the processing node has received completely the first data samples from all the nodes.

To prove Equation (3.5), one can explicitly compute the formula in Equation (3.2). To compute X_i*, one notes that* $X_{i+1} = X_i + (1 - 1/2^m)$*. We know* $X_1 = 1 - 1/2^m$*, thus* X_i *can be shown to be* $i(1 - 1/2^m)$*.*

We note that using direct 1-bit coding results in lower energy consumption, but the energy unfairness is still on the order of $O(n^2)$. In other words, as the number of relay nodes increases, the life time of the sensor network decreases significantly. Thus, we propose the following cross-sensor coding technique that saves communication energy as well as resolve the energy unfairness issue.

3.4.2.3 Cross-Sensor 1-bit Coding

Unlike the previous approaches in which each sensor encodes its data separately, cross-sensor coding technique allows each sensor to encode its data and its neighbor's data jointly using variable-length codewords where codeword's length is based on the location of a sensor.

Figure 3.4 illustrates the cross-sensor coding technique for 8-bit sample data. For easy visualization, the 8-bit resolution data at each sensor nodes are now arranged vertically with the most significant bits (MSB) at the top, and the least

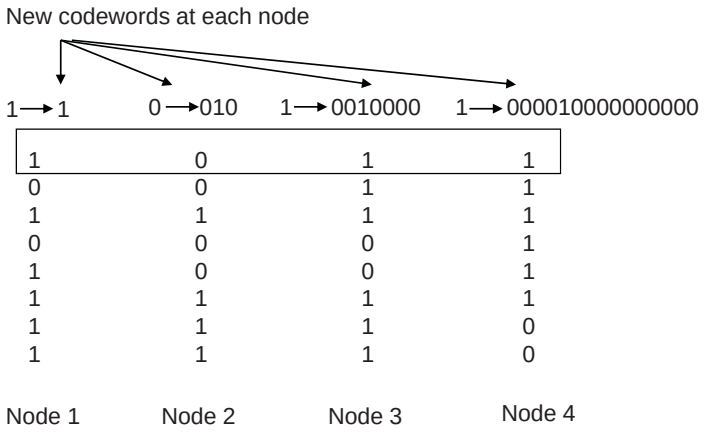


Figure 3.4: Individual bits are transmitted through each node. Each sensor combines the cumulative data and produces a new codeword with larger length.

significant bits at the bottom. First, we consider sending the MSBs of the data measured at different sensors to the processing node. The MSB of the measured data at the node 1 is "1", so node 1 will send the codeword "1" to the node 2. The MSB of the measured data at node 2 is 0, so node 2 will send the codeword "010", representing both MSBs at nodes 1 and 2 ("10"), to node 3. The MSB of the measured data at node 3 is 1, so node 3 will send the codeword "0010000" to node 4, representing MSBs at nodes 1, 2, and 3 ("101"). Finally, the MSB of the measured data at node 4 is "1", so node 4 will send the codeword "000010000000000" to the processing node. Basically, the code sent by node n is the 1-bit codeword, representing all the possible bit patterns at the nodes 1 to n.

We note that the length of the codeword increases roughly by a factor 2 after each hop. This is because there are 4 maximum possible data patterns that node 2 can send to node 3, i.e., 0 or 1 from node 1, and 0 or 1 from node 2. Similarly,

there are 8 possible data patterns that node 3 can send to node 4. In general, there are 2^n possible data patterns that represent different MSB patterns for nodes 1 to n. Thus, 1-bit codeword sent out by node n will have a length of $2^n - 1$ bit-times.

Using the above scheme, when the processing node receives a codeword "000010 000000000", it will be able to infer that the MSBs at nodes 1, 2, 3, and 4 are 1, 0, 1, and 1, respectively, by simply converting "000010000000000" (11 in decimal) to a binary coding, i.e., 1011. After the processing node receives the MSBs of all the nodes, the first node can start to transmit the second bit, and the process repeats until all the 8-bit data are transmitted. Clearly, using cross-sensor 1-bit coding, each sensor only needs to send at most 1 bit "1" per hop, thus eliminates the problem of uneven transmission energy. However, the length of an 1-bit codeword increases roughly by a factor 2 after each hop, i.e., 1-bit codeword sent out by node n will have a length of $2^n - 1$ bit-times, resulting in longer delay, or equivalently bandwidth.

The cross-sensor coding technique achieves energy fairness and efficiency by exploiting the individual node's knowledge of the topology. In particular, each sensor needs to know its position in the topology, and the data from other sensors in order to encode the new codeword with appropriate length. Furthermore, to be able to receive the correct codeword, each sensor must have a synchronized clock to enable correct timing of each bit. Cross-sensor coding also introduces additional delay per hop due to the store and forward scheme. Figure 3.5 shows that the delay increases exponentially as data traverse to the processing nodes. Note that for two consecutive nodes, the transmission time slots of one node are

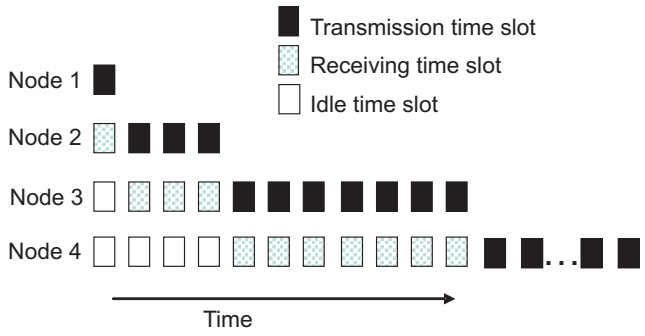


Figure 3.5: Delay in cross-sensor coding. Staggering delay is introduced at each node as shown by the squares representing time slots. The closer the nodes to the processing node, the higher delay. Blank squares represent the idle time slots, the wavy squares represent the transmission time slots, and the brick squares represent the time slots during which the node is receiving information.

the receiving time slots of another node. We emphasize that a transmission time slot of a node does not mean that node has to transmit bit "1" during this time slot. Rather, a transmission time slot means that this time slot is used to indicate the transmission of the data bits either "0" or "1". There will not be an actual transmission if the data bit is "0". Due to 1-bit coding, there are also idle time slots which are necessary for the node to differentiate the codewords. The following theorem characterizes the bandwidth, the transmission energy, and the energy fairness associated with 1-bit cross-sensor coding.

Proposition 3.2 *Given the same conditions as those of Proposition 3.1, using 1-bit cross-sensor coding scheme with time slot of T second per bit results in:*

1. *The average number of transmitted bits (TB) per sample data and node is*

$$TB = m\left(1 - \frac{1}{n} + \frac{1}{n2^n}\right) \tag{3.6}$$

2. *The achievable bandwidth B is*

$$B = \frac{mn}{T(2^{n+1} - n - 2)} \qquad [bps] \tag{3.7}$$

3. *The energy fairness is*

$$F_E = \frac{1}{3n}\left(1 - \frac{1}{4^n}\right) - \frac{1}{n^2}\left(1 - \frac{1}{2^n}\right)^2 \tag{3.8}$$

Proof 3.2 *To prove Equation (3.6), one notes that with probability 1/2, the first node will not transmit anything, i.e., the first bit is bit "0". With probability 1/4 node 2 will not transmit anything to node 3 (pattern "00"). With probability 1/8, node 3 will not transmit anything to node 4 (pattern "000"). In general, with probability $1/2^n$, node n will not transmit anything. Thus, the expected transmission energy equals to*

$$\sum_{i=1}^{n}\left(1 - \frac{1}{2^i}\right) = n - 1 + \frac{1}{2^n}$$

Hence, the average number of transmissions per node is

$$1 - \frac{1}{n} + \frac{1}{n2^n}$$

If the data is m-bit resolution, we just need to transmit a total of

$$m\left(1-\frac{1}{n}+\frac{1}{n2^n}\right)$$

bits per data sample.

As for Equation (3.7), it is clear that each node i introduces additional 2^i-1 *bit times as shown in Figure 3.5. Hence, the total number of bit times after n is*

$$\sum_{i=1}^{n}\left(2^i-1\right)=2^{n+1}-n-2$$

Thus, the bandwidth is

$$B=\frac{mn}{T(2^{n+1}-n-2)}$$

Finally, Equation (3.8) can be obtained by computing the formula in Equation (3.2) with $X_i=1-1/2^i$.

Note that energy unfairness from using cross-sensor coding scheme now asymptotically approaches 0 as the number of sensors approaches infinity. We can also extend cross-sensor 1-bit coding to cross-sensor k-bit coding.

3.4.2.4 Cross-Sensor k-bit Coding

To reduce the delay or increase the bandwidth, one can use k-bit coding technique where the maximum number of high bits in a codeword is k. The code length n is

a variable such that:

$$2^m \leq \binom{n}{0} + \binom{n}{1} + ... + \binom{n}{k}, \tag{3.9}$$

where, m is the data resolution.

3.5 Optimal Low-weight, Cross-sensor Coding

Until now, the transmission energy of bit "0" is assumed to be 0 which may not be true. We now show how to find an appropriate low-weight, variable-length, prefix code for each sensor based on (a) its position, (b) the transmission energy of bits "0" and "1", and (c) a certain delay-energy consumption trade-off. The term *low-weight* means a small number of bits "1", leading to small transmission energy. We note that the delay-energy consumption trade-off is equivalent to the bandwidth-energy consumption trade-off when using the cross-sensor coding technique.

Intuitively, if bits "0" and "1" require the same transmission energies, then the delay is proportional to the length of the code which is also proportional to the energy consumption. Therefore, Huffman coding can minimize the energy consumption and delay simultaneously. However, since sending bits "0" and "1" require different amounts of energy, it makes sense to design a different code. One can use an approach similar to that proposed by Karp [43]. Here, transmission energy can be thought as the cost of a letter (alphabet) in a code. Thus, the problem can be formulated as finding a prefix codebook whose letters (alphabets) have unequal costs which results in minimum cost. On the other hand, by doing so, the average code length, i.e., the delay may exceed a desirable threshold. Therefore,

we will consider the delay-energy consumption trade-off in our formulations. Our approach is based on Karp's work [43] which can be formulated as an integer programming problem [1]. Karp's approach assumes that the cost of each letter is discrete, and finds the necessary and sufficient conditions for the prefix code in the form of a set of linear inequalities involving integer solutions. The key to Karp's formulation is to design a tree as shown in Figure 3.6, where a leaf node represents a codeword with its letters corresponding to labels of a branch leading to it. The length of a branch is proportional to a letter cost. Once the integer solutions are found, they can be mapped directly to a valid prefix code. We refer the interested readers to [43] for more details.

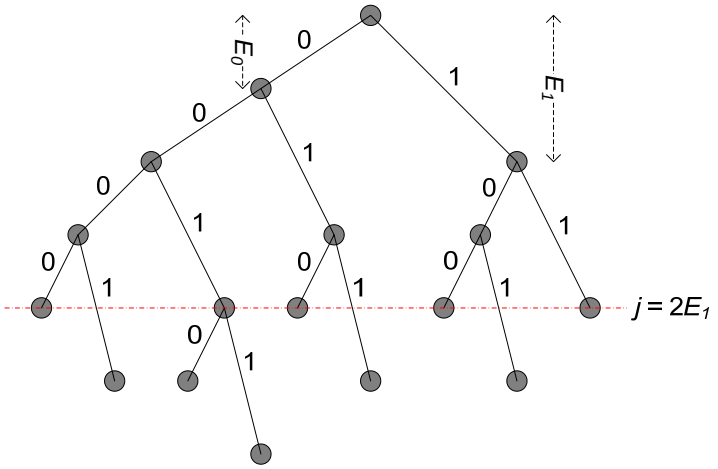


Figure 3.6: Prefix code as a tree with the cost of a symbol represented by the length of its branch.

Our formulation is a variant of Karp's in which, we introduce new optimization

[1] To the best of our knowledge, it is currently unknown whether the problem of optimal unequal cost letters is NP hard or polynomial time solvable.

variables to handle the delay and the multi-sensor aspects of our problem. Define the following notations:

- N: The number of sensors in the network.
- n_k: The number of possible symbols observed at sensor k. For a straight-line topology, $n_i = 2^i - 1 > n_j$ if sensor i is closer to the sink than sensor j.
- X_i^k: The input symbol i at node k, $i = 1...n_k$.
- C_i^k: The codeword for symbol X_i^k.
- N_{ki}^0, N_{ki}^1: The number of bits "0" and bit "1" in a codeword C_i^k, respectively.
- p_i^k: The probability of the symbol X_i^k observed at node k and $i = 1...n_k$.
- E_0 and E_1: Transmission energies of "0" and "1", respectively. We assume that E_0 and E_1 are positive integers. Note that one can approximate *real* cost with *integer* cost by scaling the costs appropriately.
- m_k: The maximum cost of a codeword at node k. m_k can be set to a large value to guarantee a solution.

To study the delay-energy consumption trade-off of a code, we first find the expression for transmission energy in terms of the optimization variables. We observe that the cost (transmission energy) d_i^k of transmitting a symbol X_i^k by node k is

$$d_i^k = E_0 N_{ki}^0 + E_1 N_{ki}^1, \tag{3.10}$$

and the average transmission energy per symbol at node k is given by

$$E^k = \sum_{i=1}^{n_k} p_i^k d_i^k \tag{3.11}$$

Karp has shown that the condition of existence of a uniquely decipherable and prefix code (Kraft's inequality for binary symbols)

$$\sum_{i=1}^{n_k} 2^{-l_i^k} \leq 1 \tag{3.12}$$

with l_i^k denoting the length of C_i^k at node k, is equivalent to: $a_j^k + b_j^k \leq b_{j-E_0}^k + b_{j-E_1}^k$ where a_j^k denotes the number of codeword (leaf nodes) of cost j and b_j^k denotes the number of non-leaf nodes at cost j as shown in Figure 3.6. The proof of the equivalence between Kraft's inequality and these linear constraints can be found in [43].

Now, let us define

$$y_{ij}^k = \begin{cases} 1 & \text{if } d_i^k = j, \\ 0, & \text{otherwise.} \end{cases}$$

then

$$d_i^k = \sum_{j=1}^{m_k} j y_{ij}^k, \quad a_i^k = \sum_{i=1}^{n_k} y_{ij}^k$$

Thus, the transmission energy E^k at a node k can be expressed as:

$$E^k = \sum_{i=1}^{n_k} \sum_{j=1}^{m_k} p_i^k j y_{ij}^k \tag{3.13}$$

Our goal is to find a formulation for minimizing the average transmission energy for the entire sensor network subject to the delay constraint (codelength) or vice versa. To do so, we first find a mathematical expression for the length of a prefix code. Denote l_i^k as the length of a prefix code C_i^k at node k, then

$$l_i^k = N_{ki}^0 + N_{ki}^1$$

Therefore, the average cost (transmission energy) to transmit the code C_i^k is:

$$d_i^k = E_0(N_{ki}^0 + N_{ki}^1) + (E_1 - E_0)N_{ki}^1 \tag{3.14}$$

Define a new variable x_{ij}^k such that

$$x_{ij}^k = \begin{cases} 1 & \text{for } j \leq N_{ki}^1, \\ 0 & \text{for } j > N_{ki}^1. \end{cases}$$

then

$$N_{ki}^1 = \sum_{j=1}^{m_k} x_{ij}^k$$

and

$$d_i^k = E_0 l_i^k + (E_1 - E_0)N_{ki}^1 = E_0 l_i^k + (E_1 - E_0)\sum_{j=1}^{m_k} x_{ij}^k$$

$$\Rightarrow \sum_{j=1}^{m_k} j y_{ij}^k = E_0 l_i^k + (E_1 - E_0)\sum_{j=1}^{m_k} x_{ij}^k$$

Thus we can represent the output codeword length of node k as a linear combination of the two integer variables x_{ij}^k and y_{ij}^k as:

$$l_i^k = \frac{1}{E_0} \sum_{j=1}^{m_k} j y_{ij}^k - \frac{E_1 - E_0}{E_0} \sum_{j=1}^{m_k} x_{ij}^k \tag{3.15}$$

Given the expressions for code length (delay) and transmission energy, we can formulate a variety of optimization problems regarding the energy consumption and delay (bandwidth). As an example, for a straight-line network consisting of N nodes, we can formulate the problem of finding a prefix code that minimizes the average transmission energy of the network subject to (a) the transmission energy of any node may not exceed a certain value W and (b) the average delay has to be smaller than a value D as follows:

Minimize

$$\sum_{k=1}^{N} \sum_{i=1}^{n_k} \sum_{j=1}^{m_k} p_i^k j y_{ij}^k$$

Subject to,

$$b_0^k = 1; \quad b_j^k = 0 \quad for \quad j < 0$$

For $1 \leq k \leq N$ and $1 \leq j \leq m_k$,

$$\sum_{i=1}^{n_k} y_{ij}^k + b_j^k \leq b_{j-E_1}^k + b_{j-E_0}^k$$

For $1 \leq k \leq N$ and $1 \leq i \leq n_k$,

$$\sum_{j=1}^{m_k} y_{ij}^k = 1$$

And $\sum_{i=1}^{n_k} \sum_{j=1}^{m_k} p_i^k j y_{ij}^k \leq W$

$$\sum_{k=1}^{N} \left(\frac{1}{E_0} \sum_{i=1}^{n_k} \sum_{j=1}^{m_k} p_i^k j y_{ij}^k - \frac{E_1 - E_0}{E_0} \sum_{i=1}^{n_k} \sum_{j=1}^{m_k} p_i^k x_{ij}^k\right) \leq D,$$

Where b_{ij}^k, x_{ij}^k and y_{ij}^k are the integer and two binary optimization variables, respectively. Once the variables b_j and y_{ij}^k are found, the optimal prefix code can be obtained via a polynomial time mapping [43].

We can also find the optimal prefix code that minimizes the average delay (bandwidth) such that the average transmission energy of each sensor does not exceed W as follow:

Minimize

$$\sum_{k=1}^{N} \left(\frac{1}{E_0} \sum_{i=1}^{n_k} \sum_{j=1}^{m_k} p_i^k j y_{ij}^k - \frac{E_1 - E_0}{E_0} \sum_{i=1}^{n_k} \sum_{j=1}^{m_k} p_i^k x_{ij}^k\right)$$

Subject to,

$$b_0^k = 1; \quad b_j^k = 0 \quad for \quad j < 0$$

For $1 \leq k \leq N$ and $1 \leq j \leq m_k$,

$$\sum_{i=1}^{n_k} y_{ij}^k + b_j^k \leq b_{j-E_1}^k + b_{j-E_0}^k$$

For $1 \leq k \leq N$ and $1 \leq i \leq n_k$,

$$\sum_{j=1}^{m_k} y_{ij}^k = 1$$

For $1 \leq k \leq N$, and for $1 \leq i \leq n_k$

$$\sum_{i=1}^{n_k} \sum_{j=1}^{m_k} p_j^k j y_{ij}^k \leq W \tag{3.16}$$

Where b_{ij}^k, x_{ij}^k, and y_{ij}^k are the optimization variables with b_{ij}^k taking on integer values, while x_{ij}^k and y_{ij}^k are binary variables.

Similarly, one can also replace the constraint in (3.16) by:

$$\sum_{j=1}^{m_k} x_{ij}^k \leq K \tag{3.17}$$

to guarantee that the number of bits "1" in any codeword cannot exceed K. This is useful when $E_0 << E_1$ and one wants to keep the transmission energy low even when the symbol distribution changes.

3.6 Performance Evaluation

We now present a few results on the trade-off between the delay and transmission energy for a straight-line topology. These results are based on the different problem formulations which are made possible by having the expressions of delay and transmission energy (Equations (3.13) and (3.15)) in Section 3.5. We use CPLEX

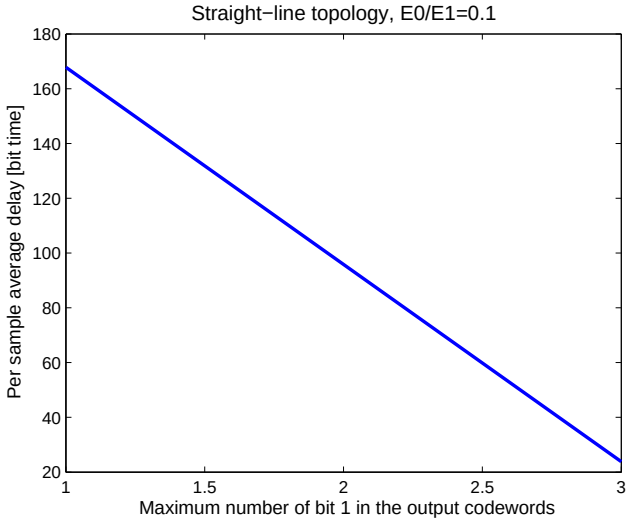
- an optimization package to obtain our solutions. The running times to obtain these solutions are negligible in all our experiments.

First, we assume that the data sample is of 10-bit resolution. Each bit has one-half probability of being 1. Figure 3.7(a) shows the average transmission delay versus K, the maximum number of bit "1" allowed in any codeword. As seen, the delay decreases with the increase of K, i.e. more transmission energy is allowed. Similarly, Figure 3.7(b) shows the average transmission energy as a function of delay when the transmission energy is minimized subject to the delay constraint.

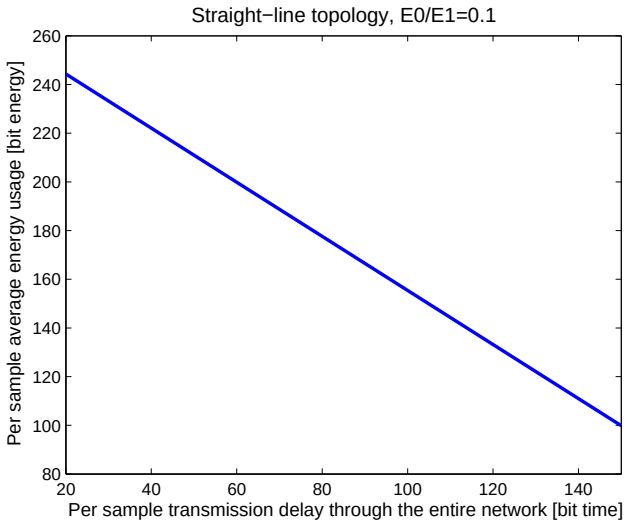
Figure 3.8 shows the average delay or code length for different values of K (the constraint on the number of bits 1 in a codeword) as a function of the number of sensors. As expected, higher K results in lower delay at the expense of energy increase.

We now present the results of applying cross-sensor coding on the actual air temperature being collected by an array of sensors developed at Oregon State University. The aim of this project is to monitor the temperature and humidity levels of the forest at different elevations. These sensors are placed along a tower to continuously log the current temperatures at every 15 minute interval. The sensor network is designed to collect data in real time, and the data is sent from the higher sensors to the lower sensors, and finally to the processing node on the ground. The measured temperature data has 10-bit resolution.

Figure 3.9 shows the transmission energy at each node (with node 1 being the node at the top) for different relative transmission energy levels of bits "0" and "1". If the transmission energy is a significant part of the overall energy, then



(a)



(b)

Figure 3.7: (a) Per sample delay as a function of K (delay optimization); (b) Average energy as a function of per sample delay (energy optimization)

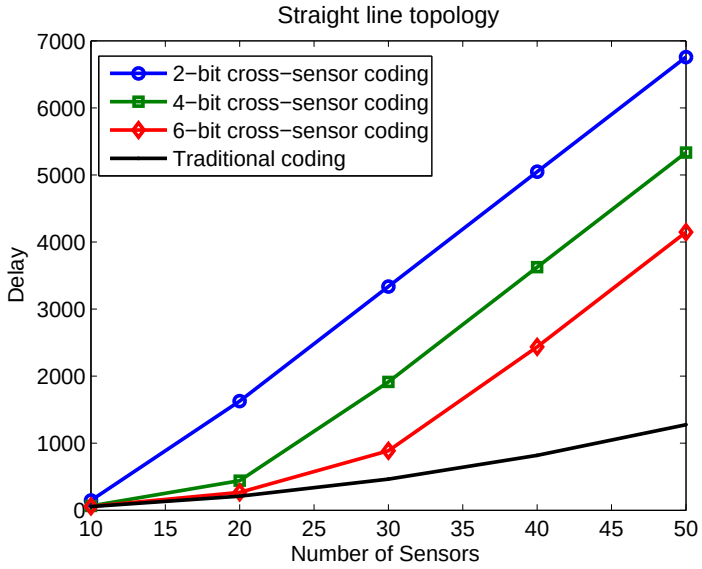
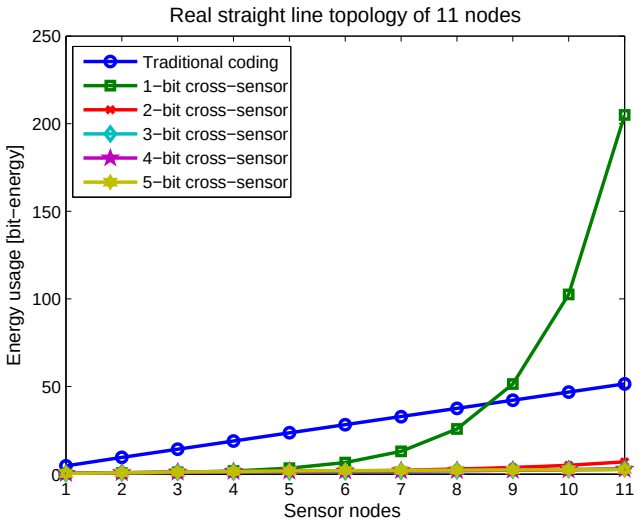
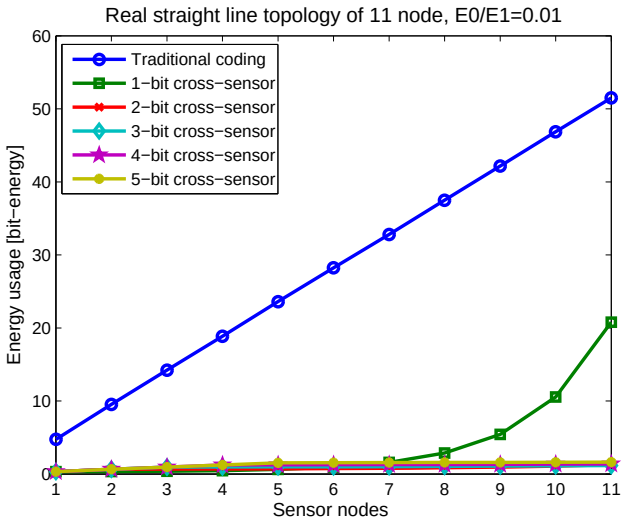


Figure 3.8: Delay (number of time slots) versus number of nodes for straight line topology.

the network lifetime depends on the transmission energy of node 11. As shown in Figure 3.9(a), node 11 uses more energy with the traditional (binary) coding than with the cross-sensor coding for $K > 1$. This is intuitively plausible as with $K = 1$, the length of a codeword needs to be a lot longer, i.e. more zeros which results in more energy usage, since transmission energy of bit "0" is not negligible, i.e., 10 times less than that of bit "1". When $K > 1$, a typical codeword resulted from cross-sensor coding is shorter, leading to lower overall transmission energy. For certain applications, one can find a suitable K-bit cross-sensor code that achieves the best performance both in terms of the delay and network lifetime.



(a)



(b)

Figure 3.9: Per node energy usage of cross-sensor and traditional coding schemes as a function of the node number for (a) $E_0/E_1 = 0.1$ and (b) $E_0/E_1 = 0.01$

3.7 Conclusions

We have designed and analyzed a cross-sensor energy-efficient coding technique for the ultra low energy sensor networks using On-Off Keying. Cross-sensor coding can significantly extend the network lifetime as compared with traditional (binary) coding by solving the energy-consumption unfairness problem. We have presented the theoretical and experimental results to show that transmission energy can be reduced substantially (e.g., a factor of 15) and the unequal energy consumption among nodes can be practically eliminated.

Chapter 4 – Video Streaming with Network Coding

1. Kien Nguyen, Thinh Nguyen, and Senching Cheung, "*Peer-to-Peer Streaming with Hierarchical Network Coding*", The IEEE International Conference on Multimedia & Expo (ICME'07), July 2007, Bejing, China.

2. Kien Nguyen, Thinh Nguyen, and Senching Cheung, "*Video Streaming with Network Coding*", The Journal of Signal Processing Systems, Vol. 57, Issue 3, Pages 319-333, Jun. 2010.

4.1 Introduction

Multimedia streaming over the Internet is challenging due to packet loss, delay, and bandwidth fluctuation. Thus, many solutions have been proposed, ranging from source and channel coding to network protocols and architecture. For example, to combat the fluctuating and limited bandwidth, a scalable video bit stream is used to allow a sender to dynamically adapt its video bit rate to the available bandwidth at any point in time [86]. To reduce packet loss and the associated delay due to the retransmissions of the lost packets, Forward Error Correction (FEC) techniques have been proposed to increase reliability at the expense of bandwidth expansion [53]. Content Delivery Network (CDN) companies such as Akamai attempt to improve the throughput by pushing content to the servers strategically placed at the edge of the Internet. This allows a client to choose the server that results in shortest round-trip time and/or least amount of congestion.

Recently, the multi-sender streaming paradigm has been proposed as an alternative to edge streaming to provide smooth video delivery [74][63][8]. The main idea is to have each server storing an identical copy of the video. The video is partitioned into multiple disjoint parts, each part is then streamed from separate servers to a single receiver simultaneously. Having multiple senders is in essence a diversification scheme in that it combats unpredictability of congestion in the Internet. Specifically, smooth video delivery can be realized if we assume independent routes from various senders to the receiver, and argue that the chances of all routes experiencing congestion at the same time is quite small. If the route between

a particular sender and the receiver experiences congestion during streaming, the receiver can re-distribute rates among the existing senders, or recruit new senders so as to provide the required throughput.

This multi-sender streaming framework is particularly well suited for CDN and P2P networks since multiple copies of a video are often present at these servers/peers either through a coordinated distribution of the video from an original CDN server, or through an uncoordinated propagation of contents in a P2P network such as KaZaa [2]. However, there are a number of drawbacks with the current multi-sender framework. First, many of the current multi-sender streaming schemes assume that identical copies of a video must be present at different servers/peers. This implies an increase in the overall storage. Second, a careful synchronization among the senders is needed to ensure that distinct partitions of a video are sent by different servers/peers in order to increase the effective throughput. In other words, an optimal partition algorithm must be able to dynamically assign chunks of different lengths to different servers based on their available bandwidths. This dynamic partition algorithm, however is often suboptimal due to the lack of accurate available bandwidth estimation. Third, for ease of controlling the sending rates as well as data partition, many multi-sender schemes assume a UDP-like transport protocol, which often cannot be used for computers behind a firewall in many networks. That said, we propose a multi-sender streaming framework using network coding technique that reduces the overall storage, the complexity of sender synchronization, and enables TCP streaming. Furthermore, we propose a Hierarchical Network Coding (HNC) technique that facilitates scalable video

streaming.

4.2 Preliminaries

In this section, we discuss some background and motivation for video streaming via path diversity framework. Based on these discussions, we will highlight several important research issues associated with path diversity framework. The goal of these discussions is to bring about an abstract model for multi-sender streaming which is general enough, yet sufficient to characterize the performance in various settings.

4.2.1 CDN and P2P Networks

A typical Internet application sends packets that follow one and only route at any instance. An application has no control over which route its packets traverse, rather, the route is determined by the underlying Internet routing protocols. In recent years, overlay networks have been proposed as an alternative to enable an application to control its route to some extent [7]. The idea is, instead of sending the packets to the destination, an application sends its packets to an intermediate host belonged to an overlay network. This intermediate host then forwards the packets to the intended destination on the behalf of the sender. As a result, the packets will take a different route than the one determined by the underlying routing protocols. Path diversity framework takes one further step by allowing an

application to send packets on multiple routes simultaneously. When packets are partitioned and/or coded properly, this path diversity framework has been shown to improve the visual quality of video streaming applications [65][8].

That said, P2P networks are overlay networks where two peers are connected together via a TCP connection. To send data from one peer to another, the data may go through a number of intermediate peers to get to the intended peer. This provides a natural framework for path diversity streaming via forcing the packets through intermediate peers. In addition, if a peer wants to view a video stream, and presumably a number of its neighbors (direct connected peers) have either the complete or partial video, it can simultaneously request different parts of the video from different neighbors. Effectively, the video packets will traverse on different routes to the peer, thus congestion on one route will not have much effect on a peer's viewing experience when the remaining routes together, provide sufficient throughput.

Content Delivery Networks (CDNs) is also a natural framework for path diversity streaming. CDN aims to improve the application's performance by placing the servers near the customers in order to increase throughput and reduce latency. In a CDN, contents are distributed to a number of servers which are strategically placed around the edge of the Internet. When a customer requests a content, the nearest server with the desired content is chosen to serve that customer. This framework can be easily enhanced to allow multiple servers to deliver the content simultaneously to a customer, and thus obtaining the benefits of path diversity streaming, or more precisely, multi-sender streaming.

On the other hand, the advantages of multi-sender streaming framework come with many research issues to be resolved. In what follows, we will discuss network protocols to accommodate multi-sender streaming framework.

4.2.2 Network Protocols

TCP vs. UDP. Many interactive and live video streaming systems use UDP whenever possible as the basic building block for sending packets over the Internet. This is because UDP allows the sender to precisely control the sending rate, and if the network is not too much congested, a receiver would receive the data at approximately the same rate. This property is also desirable for live video streaming applications where minimal throughput often must be maintained for high quality viewing experience.

On the other hand, UDP is not a congestion aware protocol, in the sense that it does not reduce its sending rate in presence of heavy traffic load. As a result, when a large amount of UDP traffic is injected into a network, it can cause a global congestion collapse where majority of packets are dropped at the routers. For this reason, non-real time applications often use TCP that can adapt the sending rate to the network conditions automatically. This rate adaptivity prevents congestion collapse, and results in a fair and efficient throughput allocation for each application even when the network is congested. Furthermore, TCP-based applications are preferable since many networks actively filter out UDP packets which are often thought as a sign of possible flooding attacks from malicious automated software.

Based on these, it makes sense to use TCP when TCP's delay and throughput fluctuation can be minimized. As will be discussed shortly, our proposed network coding technique is designed for efficient TCP based transmission.

Push vs. Pull. In multi-sender streaming framework, the data come from multiple senders which leads to the question of how to coordinate the multiple transmissions to prevent or mitigate data duplication. One possible approach is for the receiver to request disjoint data partitions from multiple senders. The protocols based on this approach are called pull-based protocol, and they work well in many scenarios. In other scenarios, it may be better to use push-based approach where the senders simply send packets to a receiver without its request.

Majority of P2P systems use pull-based protocols [75][95][88] because of their robustness against peer joining and leaving the network. Pull-based protocols also use bandwidth efficiently, in the sense that a receiver does not receive any duplicate data from the senders. However, they have many drawbacks that might be unsuitable for some video streaming scenarios.

First, using pull-based protocols may result in lower throughput for a receiving peer due to lack of disjoint data from its neighboring peers. To illustrate this, consider a streaming a video from the source 0 to two receivers 1 and 2. Suppose these peers are connected to each other. Using the pull-based protocol, receiver 1 would request data from 0 and 2 while receiver 2 would request data from 0 and 1. Since these receivers are acting independently, both may request the same packets from the source 0. If they do, most of the time, the two receivers would have the same data, thus they cannot exchange new data with each other, resulting

in lower throughput. Now, consider a simple push-based protocol in which the source simply pushes the odd packets to receiver 1 and even packets to receiver 2. Each receiver then pushes the data it receives from one node to the other node. Effectively, the receiver 1 pushes the odd packets to receiver 2, and receiver 2 pushes even packets to receiver 1. Clearly, using this protocol, the throughput at each receiver is larger than that of using the pull-based protocol. Typically, when network topology is well-defined and relatively unchanged over time, the push-based protocols result in higher throughput than their pull-based counterparts. Also, the pull-based protocols often introduce high latency due to the requests, this may not be appropriate for media streaming applications.

Second, a careful coordination on which to be sent by which sender (pull-based protocol) is required to achieve optimal performance from the perspective of a particular receiver. As an example, assuming that two senders are used for streaming, then sender 1 can stream the odd packets while the other streams the even packets, starting from the beginning of the file. As described, this approach is roughly optimal when the throughputs from the two senders are somewhat identical. On the other hand, when the throughput of server 1 is twice as large as that of server 2, then the pattern of packets received at the receiver will look like (0, 2, 1, 4, 6, 3, 8, 10, 5). Clearly, the gap between even and odd packets will grow with time. This is problematic for streaming applications where packets are played back in order, and the playback rate is larger than the throughput of the slow link. For example, if the playback rate is 2 packets/s, then even with the pre-buffering technique, the video player eventually has to stop to wait for odd packets since the arrival rate of

odd packets is only 1 packet/s. We note that the total receiving rate at a receiver is 3 packets/s which, in principle, should be sufficient for a 2 packets/s stream. However, the suboptimal packet partition creates the scenario where the receiver receives many packets to be playbacked in the far future, but not enough packets in the near future for playback in time. A solution to this problem is to let the receiver dynamically requests the packets it needs. When there are many servers with different available bandwidths and are varied with time, complex dynamic coordination between the client and the servers is needed to achieve the optimal throughput.

Third, even when complex coordination is possible, this only works well if all the senders have the complete file or data segments of interest, so that a receiver can choose which packets from which senders based on the sender's available bandwidths which presumably can be observed by the receiver. In a P2P network, it is not always the case that the sending peers would have the complete file. In fact, previously discussed example showed that using the pull-based approach may result in lower throughput due to the duplication of data among the peers. In a CDN, it is possible to store duplicated versions of a video stream at different servers before a streaming session. However, this technique results in larger overall servers' storage.

As such, we believe that for certain scenarios, push-based protocols are better-suited for multimedia streaming since they are simple, and can provide high throughput and low delay. Although to be bandwidth efficient, one must ensure that the data duplication at the receiver is minimal. As will be discussed shortly,

our approach is to employ network coding to achieve this property.

4.2.3 Source Coding

Often, one must employ source coding techniques, in particular, scalable video coding, to mitigate the effect of Internet throughput variations on the viewing experience. Scalable video coding enables a sender to adapt its sending rate to the current network conditions while allowing a graceful degradation of the video quality. A typical scalable video bit stream consists of frames. Each frame consists of bits with different importance levels in terms of the visual quality. These bits are categorized into a hierarchy of layers with different importance levels. Thus, when the available bandwidth is small, sending bits from the most important layers and ignoring others would result in a smooth video playback, albeit slightly lower video quality. That said, we will discuss the hierarchical network coding technique designed for efficient multi-sender transmission of scalable video bit streams.

4.3 Streaming Model

To motivate the proposed abstract model for multi-sender streaming framework, let us first consider the distribution of a live or non-live video to the clients in a CDN. For a non-live setting, the origin server can distribute a video to a number of assisted servers prior to the start of a video streaming session. A client then randomly connects to a small number of these assisted servers in parallel to view

the video. If each of the assisted server has the entire video, using a pull-based protocol, a client can request different parts of the video from different servers as discussed previously. However, requiring the video to be on every server implies much redundancy. Thus, an interesting question is how to distribute the video to the assisted servers such that, even when each server does not have the complete video, there is still high probability that a client can get the complete video from all the servers that it connects to. Intuitively, the key to a good distribution scheme is to ensure that the assisted servers share as little information as possible while allowing a client to obtain the complete video.

Another interesting question is how to best distribute a scalable video to the assisting servers. Intuitively, for a given redundancy, a good distribution scheme should provide a high chance for a client to obtain the base layer bits, perhaps at the expense of lower probability of its getting the enhancement layer bits.

That said, a simple distribution scheme could be as follows. At each time step, the origin server would pick a packet in playback order and randomly chooses a server to send the packet to. This process repeats until a specified number of packets (redundancy level) has been sent. This scheme, however, tends to produce many duplicated video parts among the servers chosen by a client for streaming, and thus reducing the chance of a client to obtain high throughput from multiple servers. On the other hand, from a client's viewpoint, when scalable video is used, having multiple servers storing duplicated base layer bits is beneficial. This is because a client now has a higher chance of obtaining the base layer bits from a random number of servers that it connects to. We note that because of the

randomness in the way servers were chosen, the client may or may not have the packets it wants. *Thus the goal of the origin server is to code and distribute packets in such a way to result in high availability of packets needed by a client for smooth video playback.* Furthermore, when scalable video is used, the origin server may decide that it would distribute the important layer packets with high probability, i.e., more important layer bits end up at the assisted servers, thus increasing the probability of a client obtaining these bits.

In addition to CDN setting, let us consider a video broadcast session from a single source to multiple receivers (peers) in a P2P network. We assume a push-based protocol, in which the source pushes the packets to its neighboring peers who in turn push the data to other peers. Packets are pushed out by the source in some order. To reduce the playback delay, the source may want to send packets with earlier playback deadlines first. A peer then pushes the packets out to its peers in the order that these packets were received.

Since streaming is of concern, it is important to consider the set of packets available for a receiver at any point in time. To achieve smooth video playback, this set of packets must contain the packets that are used to playback the current video frame. From a receiver's viewpoint, this implies that its neighbors must have the packets it wants in a timely manner. Unfortunately, due to many factors, e.g., variations in round trip time (due to topology), peer joins and leaves, bandwidth heterogeneity of peers, these packets arrive at the neighbors of a receiver in different order than the one they were sent by the source. Thus, within a small window of time, from a receiver's viewpoint, we assume these packets arrive at its neighbors in

a somewhat random manner. The neighbors then randomly push the packets to the receiver. Clearly, the distribution of packets at these neighbors can be controlled to some extent by the source. For example, a source may push duplicated packets containing base layer bits to ensure their availability for the receiver. This scheme, however might take away the throughput used for transmissions of enhancement layer bits otherwise.

Based on these discussions, we are interested in the following abstract model. A source has a file. It allows to code and distribute the file in whatever way to a number of intermediate nodes (servers in CDNs and peers in P2P networks). A receiver then randomly connects to some of these intermediate nodes to obtain the file as shown in Figure 4.1. Thus, we model the process into two stages: the initial distribution of the packets to the intermediate nodes and the transmissions of packets from the intermediate nodes to a receiver. The arrival patterns of packets at the intermediate nodes are assumed to be somewhat random, and can be controlled to some extent by the source. In a CDN, these packet patterns are direct result of how an origin server send packets to these assisting servers. On the other hand, in a P2P network, how the source send packets has an indirect effect on the distribution of packets at the intermediate nodes, i.e., neighboring peers of receiver. For scalability, we also assume that the intermediate nodes do not communicate with each other. Instead, these nodes simply push packets to a receiver in some random manner. Thus, one major concern is how to mitigate the data duplication when using push-based protocols. We note again that the push-based protocols can eliminate the packet partition problem that can reduce throughput while min-

imizing the coordination overhead as argued in Section 4.2.2. That said, in this work, we describe network coding approaches for the distribution of packets from a source to the intermediate nodes in order to minimize the storage redundancy (in CDNs) and bandwidth usage (in P2P networks). Furthermore, we describe a TCP-based streaming protocol that employs network coding technique to allow a receiver to achieve high throughput while minimizing the coordination overhead. We now introduce the necessary background on network coding techniques.

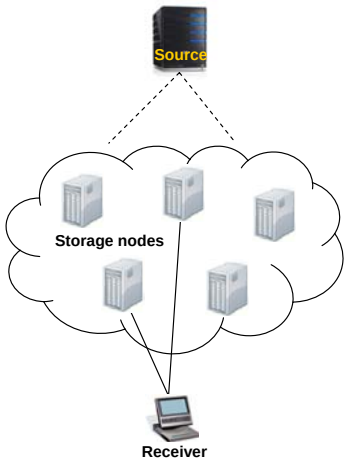


Figure 4.1: An abstract model for path diversity streaming.

4.4 Network Coding

4.4.1 Random Network Coding

In their seminal network coding paper, Ahlswede *et al.*, showed that maximum capacity in a network can be achieved by appropriate mixing of data at the intermediate nodes [4][46]. The most elegant result of network coding is that the maximum network capacity is achievable using some *random network coding* [35][**?**] techniques, while this is not usually possible with the traditional store and forward routing.

Using random network coding (RNC), a peer encodes a new packet p_i by linearly combining n original packets as: $p_i = \sum_{j=1}^{n} f_{ij} c_j$ where f_{ij} are the random elements belonging to a finite field F_q having q elements. A node then includes the information about the f_{ij} in the header of the new packets and sends these new packets to its neighbors. If a receiver receives n encoded packets p_i's that form a set of n linearly independent equations, it will be able to recover n original packets. The advantage of using this random network coding in CDN or P2P networks can be seen in the following simple CDN scenario.

Assuming that an origin server distributes a file to a number of assisting servers in a CDN. To increase the throughput, the origin server can first divide a file into n different chunks and randomly distributes these chunks to the assisting servers. A client then connects to these servers to get the file. Since each server randomly pushes pieces of the file simultaneously to a client, the time for a client to recover all n chunks is potentially much shorter than having the only origin server pushing

the file. Note that this design scales well since no coordination among the servers is required. However, it is not optimal. Because of the random packet pushing, some of the packets received at a client may be duplicated, resulting in wasteful bandwidth. For example, an origin server may divide a file into 4 chunks c_1, c_2, c_3, and c_4, and randomly distributes to a number of assisting servers. As a result, assume that the chunks at server A are c_1, c_2, c_3, and at server B are c_2, c_3, and c_4. Now suppose that a receiver R connects to both servers A and B to stream the file from. Suppose further that A pushes out packets c_1, c_2, c_3 and B pushes out packets c_2, c_3, and c_4 in that order. After the first time slot, R obtains both c_1 and c_2. In the second time slot, R obtains c_2 and c_3, but since it already obtained c_2 from the previous time slot, it discards c_2. In the third time slot, it obtains c_3 and c_4, and discards c_3. As seen, R needs to download six chunks in three time slots to be able to receive the complete file.

Now let us consider the case where the origin server is allowed to use network coding. In particular, the origin server produces coded packets as a linear combination of the original packets, and distributes them to the servers A and B randomly. Formally, the coded packets are encoded as follows:

$$a_i = \sum_{j=1}^{3} f_{ij}^{a} c_j \ , \qquad b_i = \sum_{j=2}^{4} f_{ij}^{b} c_j,$$

where f_{ij}^{a} and f_{ij}^{b} are random elements belonging to a finite field F_q. Because of the randomness, each server is likely to have different packets, and thus R is also likely to receive different packets. For example, during the first two time slots, it is likely that R would receive different packets, two from each server. Suppose that R receives $a_1 = f_{11}^{a} c_1 + f_{12}^{a} c_2 + f_{13}^{a} c_3$ and $a_2 = f_{21}^{a} c_1 + f_{22}^{a} c_2 + f_{23}^{a} c_3$ from A, and

$b_1 = f^b_{12}c_2 + f^b_{13}c_3 + f^b_{14}c_4$ and $b_2 = f^b_{22}c_2 + f^b_{23}c_3 + f^b_{24}c_4$ from B, then clearly, it will be able to recover c_1, c_2, c_3, c_4 if these four equations are linearly independent and f^a_{ij} and f^b_{ij} are known. It can be shown that if the field size is large enough, the probability of obtaining these independent equations is close to 1. For this scheme to work, the information about f^a_{ij} and f^b_{ij} must be included in the data packets. The number of bits required to specify f^a_{ij} and f^b_{ij} are $n\log(q)$ where n is the number of original packets while q is the size of the finite field. If the chunk size $m >> n$ then these bits are negligible. Therefore, for most practical purposes, this network coding scheme can speed up the download time (4 packets as compared to 6 packets) without the overhead of coordination.

One important observation is that network coding incurs an additional delay before any of the original data can be recovered. Without network coding, R will be able to recover c_1 and c_2 during the first time slot. On the other hand, using network coding, c_1 and c_2 cannot be recovered until the second time slot, although after the second time slot, all c_1 through c_4 can be recovered simultaneously. In general, if a network coded packet is a combination of n packets, then a receiver will have to receive at least n coded packets in order for it to recover any one of the original packets. This potentially introduces unnecessary delay for video streaming applications. Therefore, we propose a network code structure that enables a receiver to recover the important data gracefully in the presence of limited bandwidth which causes an increase in decoding delay.

4.4.2 Hierarchical Network Coding

To increase the probability that the most important data (base layer bits) are available at the servers, and therefore can be pushed down to a receiver, a straightforward scheme is for a source to send more duplicates of the important data. For given bandwidth and storage requirements, this implies taking away some of the bandwidth and storage that might be used for the enhancement layer bits otherwise. For example, let us consider a two layer video bit stream, instead of sending every packet with equal chance (0.5), the source may want to first group the base layer bits and enhancement layer bits into to different types of packets: the base layer packets and enhancement layer packets. Next, it can push the base layer packets to the assisting servers with higher probability, e.g. 0.7 than those of an enhancement layer packets. For a limited redundancy, a receiver will likely to recover the base layer information. Also even when every server has the complete file (high redundancy), the receiver will be able to recover the base layer information faster since the assisted server pushes the packets randomly to a receiver. This method seems promising, however, as will be shown later, it is still far from optimal.

We now describe a hierarchical network coding scheme to overcome the decoding delay of the RNC and duplication of Uncoded schemes, while increasing the chance for a receiver to decode the important bits of the video in time [59]. Let us consider a r layers scalable video stream. We first divide the stream into a number of relatively large consecutive chunks. Each chunk consists of the bits from all the

layers. Now, within each chunk, we group all the bits from the same layer i into a number of packets m_i. Denote these packets as b_1^i, b_2^i, ... $b_{m_i}^i$. Next, we code the packets within a chunk using one of the following r structures:

$$p_i = \sum_{j=1}^{m_1} f_j^1 b_j^1 + \sum_{j=1}^{m_2} f_j^2 b_j^2 + .. + \sum_{j=1}^{m_i} f_j^i b_j^i \tag{4.1}$$

where f_j^i are the non-zero random elements of a finite field F_q and b_j^i are the original packets of layer l_i. Assuming that l_1 and l_r are the most and least important layers, then a coded packet p_i would always contain the information from the base layer. In essence, the coded packets belongs to one of the r classes. Let us denote these classes as N_1 to N_r. The packets belonging to the most important class N_1 contain only information about the base layer. The packets belonging to second most important class contain the information about the base layer and the first enhancement layer. In general, the packets belonging to a k class contain information about layer 1 to k.

Using this encoding structure, given a random number of coded packets, the probability of recovering original packets from a base layer is always larger than those of other layers. In fact, the probability of recovering a packet from an important layer is always larger that of a less important layer.

To fine tune the probability of receiving a certain type of packets, one can also control the number of packets belonging to a certain types. For example, one can increase the probability of receiving base layer packets by generating more packets of N_1 type.

Table 4.1: Compare coding schemes with 2 layers data

Uncoded	WLNC	Hierarchical NC	RNC
a_1	a_1	a_1	a_1
a_2	a_2	a_2	a_2
	$a_1 + a_2$	$a_1 + a_2$	$a_1 + a_2$
b_1	b_1	$a_1 + b_1$	$a_1 + b_1$
b_2	b_2	$a_1 + b_2$	$a_1 + b_2$
	$b_1 + b_2$	$a_1 + b_1 + b_2$	$a_1 + b_1 + b_2$
		$a_2 + b_1$	$a_2 + b_1$
		$a_2 + b_2$	$a_2 + b_2$
		$a_2 + b_1 + b_2$	$a_2 + b_1 + b_2$
		$a_1 + a_2 + b_1$	$a_1 + a_2 + b_1$
		$a_1 + a_2 + b_2$	$a_1 + a_2 + b_2$
		$a_1 + a_2 + b_1 + b_2$	$a_1 + a_2 + b_1 + b_2$
			b_1
			b_2
			$b_1 + b_2$

To illustrate our approach, let us consider a simple example involving only 4 packets belonging to one base and one enhancement layer. Let us denote the four packets as a_1, a_2, b_1, and b_2 with a_i's and b_i's belonging to the base and enhancement layers, respectively. Further suppose that the coefficients have binary values only. Table I shows possible encoded packets for four coding schemes: Uncoded, Within Layer NC (WLNC), HNC, and RNC. The WLNC scheme produces coded packets which are linear combinations of the original packets belonging to the same layer.

For each scheme, assuming that the origin server randomly encodes a total of M packets which can be any of these packets. With the exception of RNC, before coding any packet, the origin server decides whether a packet should be coded as

the base layer packet with probability P, or as the enhancement layer packet with probability with $1-P$. After the packet class has been chosen, a packet is randomly and uniformly generated. Equivalently, the packet is chosen uniformly from all the possible packets within a class. By choosing appropriate value of P, one can tune the probability of getting packets from certain classes. For the RNC, there is no class, thus a packet is randomly generated as a random linear combination of the original packets from the entire chunk.

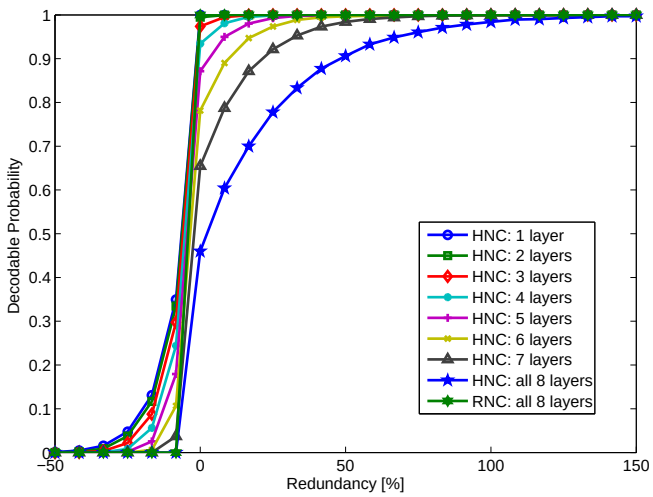
Suppose that three encoded packets (M) are to be randomly generated by each scheme. It is clear to note that when using the non-network coding, exactly two of these packets have to be a_1 and a_2 in order to recover all the base layer packets (a_1 and a_2). For the WLNC scheme, to recover the base layer packets, two distinct packets have to come from the N_1 class. For the HNC scheme, the probability for recovering both a_1 and a_2 is larger than that of WLNC. This is because in addition to being able to recover the N_1 packets from two distinct packets from the N_1 class, this scheme is also able to recover the base layer packets with an appropriate combination of 1 N_1 packet and 2 N_2 packets, e.g., (a_1,a_1+b_1, a_2+b_1). Finally, for the RNC scheme , the probability of recovering base layer packets is approximately equal to that of HNC for this particular example. In more general scenario, RNC scheme would have lower probability of obtaining important layers when small number of packets are chosen.

We note that if the origin server only generates packets from N_1 class, the receiver has the largest probability of recovering the base layer packets. However by doing so, the receiver will never be able to recover packets from the enhancement

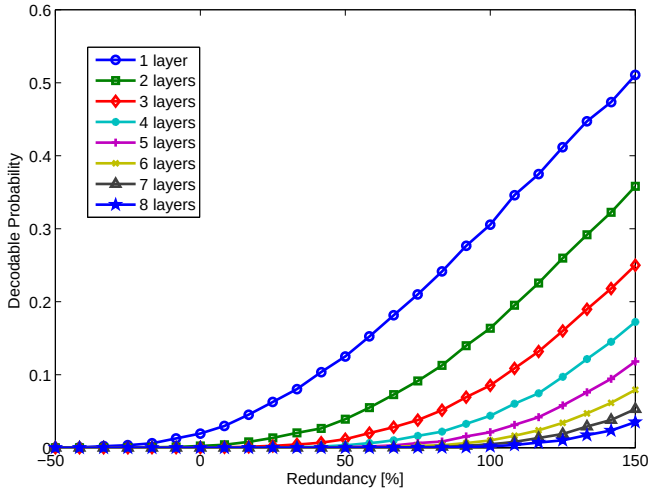
layer. HNC enables receiver to recover both base and enhancement layer packets with different probabilities. For RNC in a general setting, as argued in Section 4.4.1, it may take longer time (more packets) to be able to recover any of the original packets. But when it does so, it can recover all the packets simultaneously.

As a simple example to show the benefits of HNC, we use a scalable stream with 8 layers. The base layer contains 8 packets while other 7 enhancement layers contain 4 packets each. P is set to 1/8, i.e., the probability of generating a packet from any layer is the same. We compare the layer recoverability of non-network coding and HNC schemes as a function of the total number of random packets generated. The more packets are generated, the higher recoverability at the expense of larger redundancy. Redundancy is the number of additional packets received for decoding all the packets in a layer. Figure 4.2(a) shows this decodable probability for every layer as a function of redundancy when using HNC and RNC. Similarly, Figure 4.2(b) shows the decodable probability when no coding is used.

As seen, when the redundancy is less than 0, using HNC, the receiver is able to decode the most important layer with higher probability than that of using RNC. Redundancy less than zero represents the case where a receiver does not receive all $1 \times 8 + 7 \times 4$ packets. This could be due to the servers do not have enough packets or simply the throughput from all the servers to a receiver is not sufficient for the receiver to receive all the packets in time. In this case, a receiver is still able to recover packets from the important layers with some high probability. However, HNC incurs additional bandwidth when the redundancy ranges from 0 to 150%. After 150%, both HNC and RNC results in approximate performance.



(a)



(b)

Figure 4.2: Layer decodable probabilities as a function of redundancy using (a) HNC and (b) non-network coding.

On the other hand, when Uncoded scheme is used, the decodable probability is substantially smaller for a same specified redundancy level. Even with the redundancy of 150%, a receiver still fails to decode the layers with high probability.

In section 4.6, we will show more detail performances of network coding schemes for realistic scenarios and compare their performances with traditional Reed-Solomon codes.

4.5 Joint Network Protocols and Coding Schemes

We now propose network coding schemes for multi-sender streaming framework that reduce the coordination among servers in CDN or peers in P2P networks. We first discuss the RNC-based protocol.

In this scheme, a video stream F is randomly network coded and dispersed to a number of servers/peers in the network. As described above, a file is partitioned into N chunks $c_1, c_2, ..., c_N$. Each chunk c_i is further divided into n small packets $p_{i1}, p_{i2}, ..., p_{in}$. Now, for each chunk c_i, the origin sender will randomly network code the packets within it, to produce a number of coded packets. These packets will be randomly sent to the assisting servers in a CDN or peers in a P2P network. Note that each server/peer does not need to keep all n coded packets. They may keep only a fraction of the coded packets, but each server/peer will have some coded packets from every chunk c_i. Therefore, the total amount of storage of this scheme is small than that of the traditional CDN.

Using this approach, the receiver first requests all the senders to send their

packets p_{1i}'s from the first chunk c_1. The sender then pushes the packets within a chunk to a receiver in a random manner. After the receiver receives roughly n coded packets, it will be able to recover n original packets. It then immediately sends a request to all the senders to start streaming the packets from the second chunk c_2. In the meanwhile, the receiver can start playback the video. The process continues until the end of the stream is reached. Clearly, there is a delay at the beginning due to the time for the receiver to receive n independent packets. The attractive feature of this scheme is that no dynamic packet partition is required. All senders are sending at their available time-varying bandwidth until the receiver sends an *end of chunk* request to move to the next chunk. Therefore, TCP can be employed for streaming. The effective throughput at the receiver is roughly equal to the total throughputs from all the senders. At any point in time, one sender may have a slow connection, but as long as the total throughput is larger than the playback rate, the receiver will be able to playback the video smoothly.

We emphasize that this scheme achieves maximum throughput without the complex coordination for allocating the packets. However, it may not work well when the aggregate throughput of all the senders is smaller than the video bit rate. Thus, one cannot playback the video smoothly.

We now describe a HNC-based protocol that employs scalable video bit stream to solve this problem. Similar to the RNC-based scheme, the video file is partitioned into chunks. However, instead of using random network coding, we employ HNC technique. As discussed previously, HNC packets are coded based on the importance levels of the bits in a scalable bit stream.

The advantage of HNC is that, given a smaller number of received coded packets due to smaller throughput during some period of time, the probability of decoding the base layer using HNC is higher than that of RNC scheme. Thus, when a receiver determines that it may not have enough time to wait for all the coded packets within a chunk, it can readily decode the packets from the important layers, and ignore other undecodable packets. It then signals the senders to send packets from the next chunk. This technique allows a receiver to playback a low quality but smooth video.

4.6 Simulation Results

In this section, we investigate the performances of the proposed schemes. To be concrete, our simulations assume a CDN scenario in which, there is an origin server with the original video stream. This server distributes either uncoded or coded packets to a number of assisting servers which are then responsible for streaming the video to a client as shown in Figure 4.3.

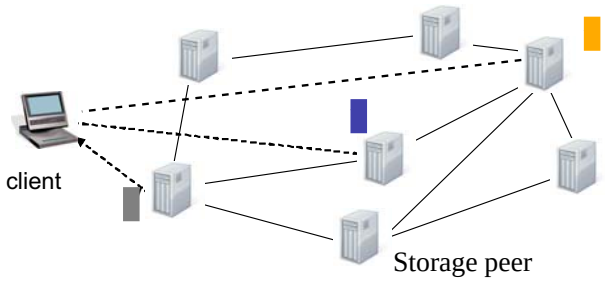


Figure 4.3: Simulation setup.

In this simulation, the origin server has a 3 layers scalable video bit stream with a rate of 432kbps. The base layer rate is 192kbps, while the rates for the two enhancement layers are 120kbps each. The original stream is divided into a number of chunks c_i of length 1 second. Thus each chunk consists of 36 packets of size 1500 bytes. As a result, the base layer and each of two enhancement layers has 16 and 10 packets, respectively. The origin server distributes the packets to 3 servers using some schemes. Next, TCP is employed to transmit data from these three servers to a single receiver simultaneously. We use the finite field size of 2^8 for all the network coding operation.

We consider the following push-based transmission protocol with different coding schemes. In particular, the schemes of interest are: Uncoded , Reed Solomon coding, RNC, WLNC, and HNC. Except for the non-network coding techniques, the protocol used in these schemes are identical to the one described in Section 4.5.

Uncoded. Packets are not coded, however, they are divided into three classes corresponding to the number of video layers. The origin server randomly pushes packets to the assisting servers with different probabilities P_1, P_2, and P_3, based on the classes that the packets belong to.

Reed Solomon (RS). Using this scheme, m original packets within a chunk are coded into $m(1 + b)$ coded packets and distributed randomly to the servers. In this case, $m = 16, 10, 10$ for the base layer, first enhancement layer, and second enhancement layer, respectively. Similar to the Uncoded scheme, the packets of three different classes are generated and pushed to the servers with different

probabilities.

RNC. The origin server randomly generates a number of packets as linear combinations of 36 packets for each chunk, and distributes these packets to the assisting servers. As a result, each assisted server keeps a fraction of coded packets which are pushed to the receiver randomly. Note that packets are not grouped into class, thus all coded packets are randomly generated with equal probability.

WLNC. The origin server applies network coding to the packets belonging to the same layer. The coded packets are then generated and pushed to the assisting servers with different probabilities according to their classes.

HNC. The origin server employs HNC which results in three classes of packets as described in Section 4.4.2. These coded packets are generated and pushed to the assisting servers with different probabilities.

First, we characterize the probability of a receiver being able to decode certain layer as a function of storage redundancy for different coding schemes. A layer is decodable if all of its packets are recoverable. This implies that there are enough distinct coded packets for this layer on the servers. With the exception of RNC, one important parameter for these schemes are the probabilities for which the packets from certain classes are generated and sent to a receiver. Intuitively, higher probability of sending packets from a class results in higher probability for a receiver being able to decode all the packets from that class. That said, we present the simulation results on the decodability for different transmission (equivalently, generation) probabilities for different classes.

Figure 4.4 shows the decodable probabilities for different layers when using

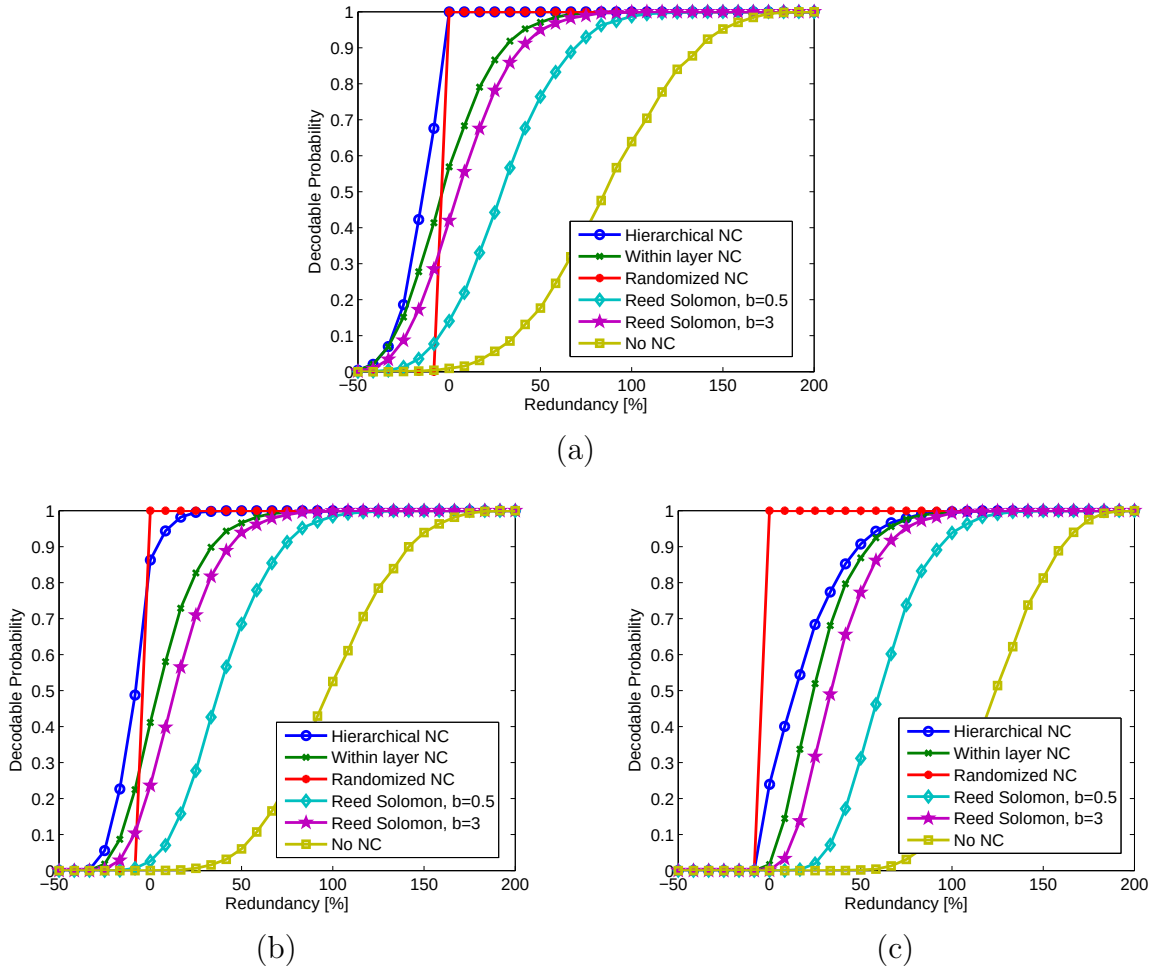


Figure 4.4: The probability of a receiver being able to decode (a) first layer; (b) first and second layers; (c) All 3 layers; $P_1 = P_2 = P_3 = 1/3$.

different schemes. The transmission probabilities for each layer are set to equal to each other, i.e., $P_1 = P_2 = P_3$. As seen, when the redundancy level is less than zero using HNC has the largest probabilities of recovering layers 1 and 2, followed by WLNC, RS, RNC, and Uncoded schemes. On the other hand, when

the redundancy level is greater than zero, RNC scheme has the largest probability of recovering layers 1 and 2, followed by HNC, WLNC, RS, and Uncoded schemes. When the redundancy level is 150%, HNC, WLNC, RS, and RNC schemes can recover all three layers with a probability close to 1, but the Uncoded scheme requires redundancy of almost 200% to accomplish the same goal.

Similarly, Figures 4.5 and 4.6 show the decodable probabilities of different schemes when the transmission probability for layer 1 increases. In particular, the transmission probabilities for the layers are now set as $(P_1, P_2, P_3) = (0.4, 0.3, 0.3)$ and $(P_1, P_2, P_3) = (0.5, 0.25, 0.25)$, respectively. As seen, the decodable probability for layer 1 increases for all the schemes. This is intuitively plausible as more packets of layer 1 are likely to be sent to the assisting servers, and thus a receiver is likely to be able to decode all the packets from this layer. On the other hand, this comes at the expense of not getting packets from other layers as shown in Figure 4.5 and 4.6.

From the simulation results, it is best to employ RNC and HNC when the redundancy is greater or smaller than zero, respectively.

We now consider a scenario where each server has much redundancy, thus a receiver will be able to recover the packets if there is sufficient throughput between itself and the servers. One problem arises, however, when the total throughput at a receiver is less than the video playback rate due to network congestion. In that case, using HNC may allow a receiver to receive and decode the most important layers early and in time for playback. A receiver then can request the next chunk in a video stream from the servers when it determines that there is not enough

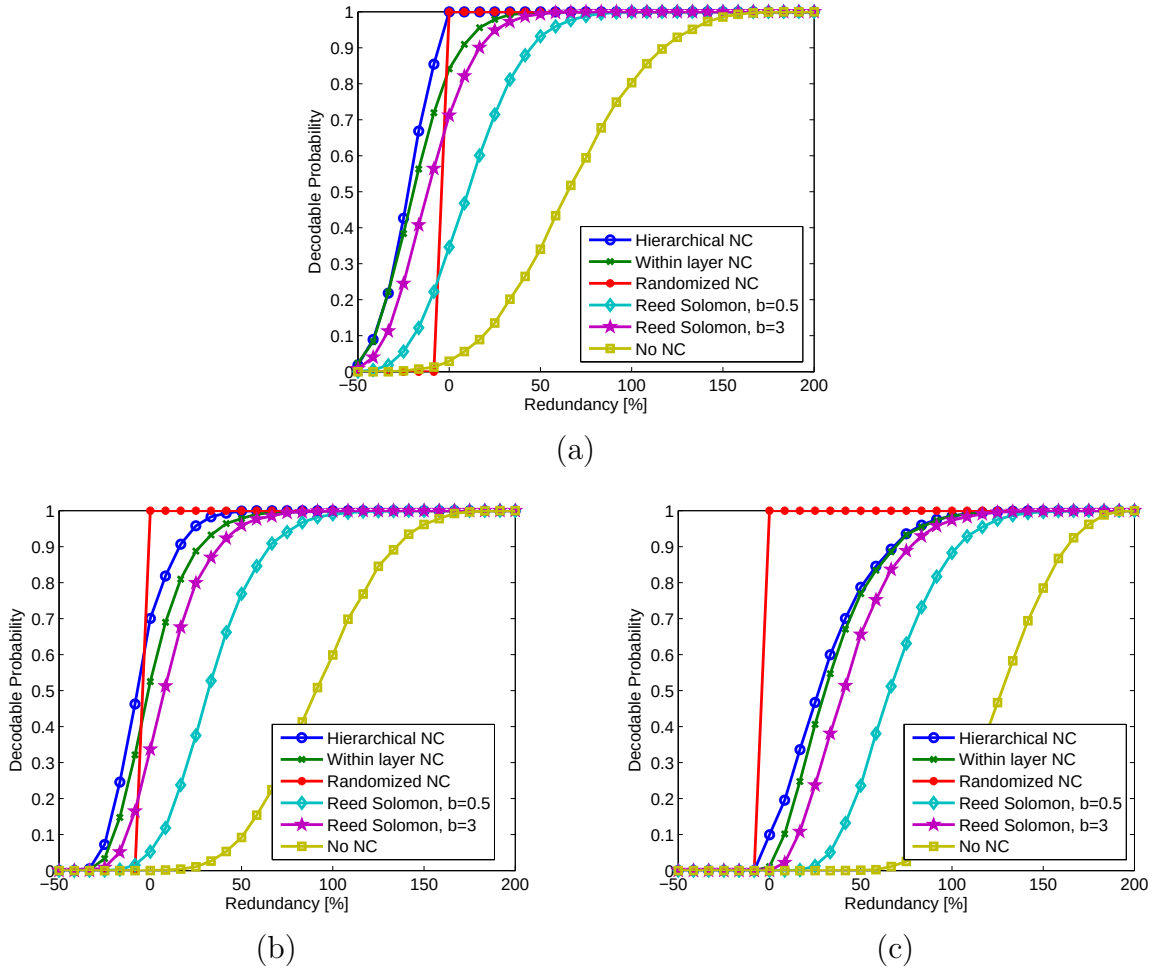


Figure 4.5: The probability of a receiver to be able to decode (a) first layer; (b) first and second layers; (c) All 3 layers; $P_1 = 0.4$, $P_2 = 0.3$, $P_3 = 0.3$.

throughput to receive the enhancement packets.

To simulate this scenario, we use network simulator NS [39]. We use a number of servers to transmit data to a receiver as shown in Figure 4.3. Heavy traffics between the servers and the receiver are generated using on-off exponential distribution

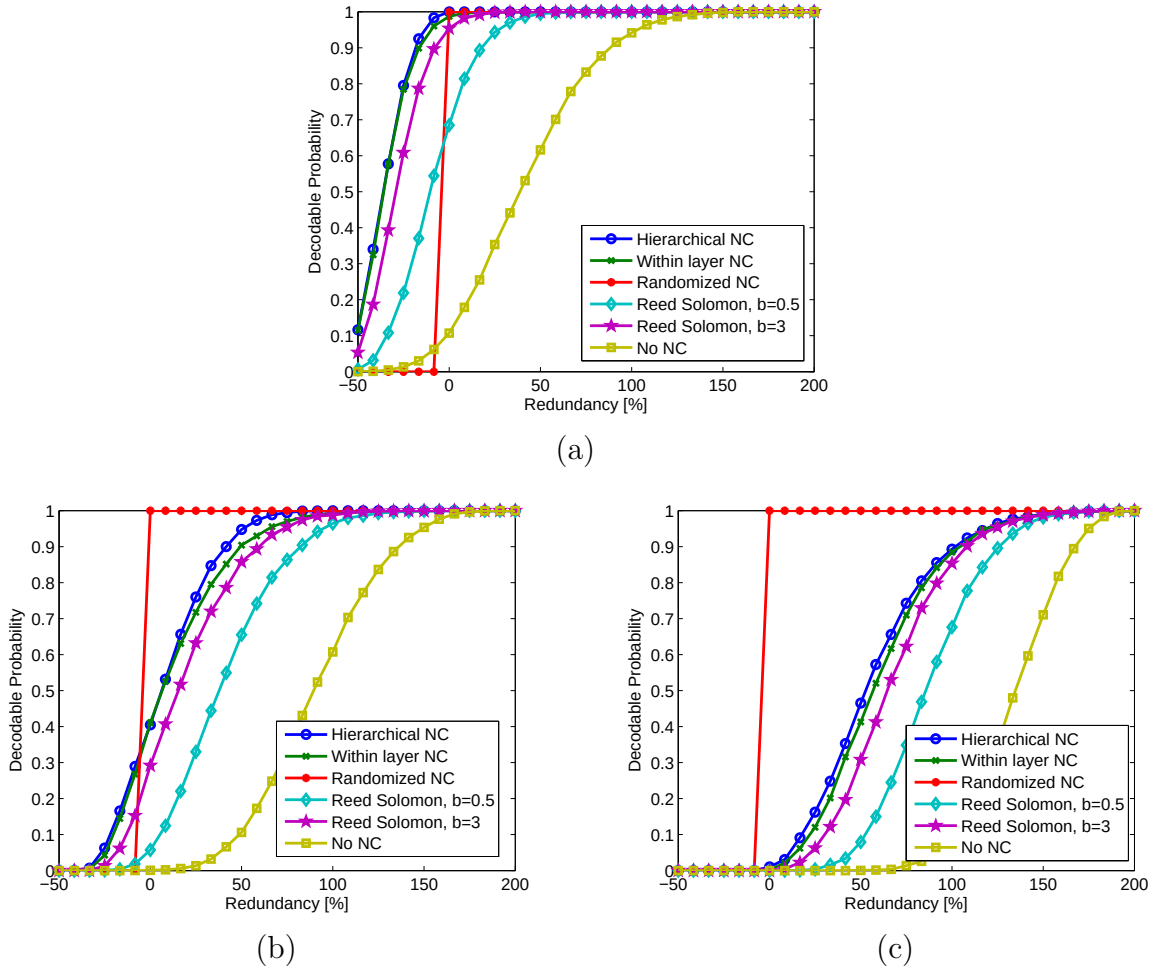


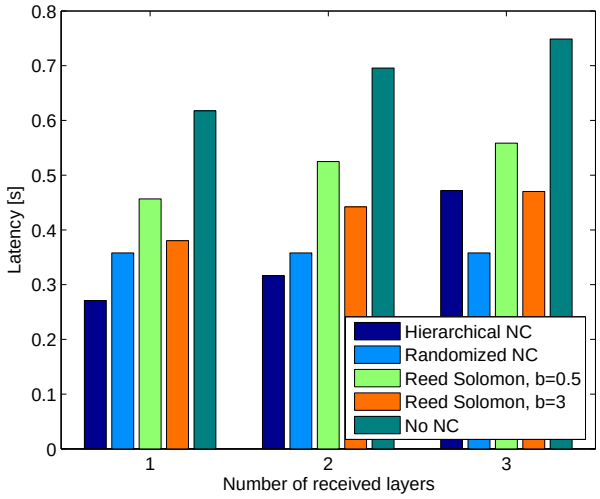
Figure 4.6: The probability of a receiver to be able to decode (a) first layer; (b) first and second layers; (c) All 3 layers; $P_1 = 0.5$, $P_2 = 0.25$, $P_3 = 0.25$.

with mean of 300kbps. The on and off periods are set to 50 ms each. The physical bandwidth for the links between the servers and the receiver are set to 500 kbps. Since TCP is used for transmission the data, the available throughputs of different connections vary with time, resulting in different number of packets received per

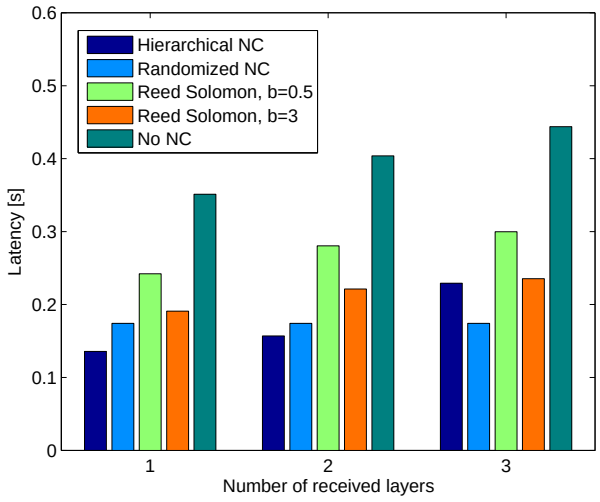
unit time. Figure 4.7 (a) and Figure 4.7(b) show the average time before a client can decode different layers within a chunk for different schemes when using 3 and 6 servers, respectively. As seen, for the Uncoded scheme, the time to decode any layer is the largest due to the high probability of getting duplicated packets. The performances of RS schemes are better than Uncoded, but worse than those of RNC and HNC schemes. For the RNC scheme, the time to decode all 3 layers is the smallest. However, the time to decode 1 and 2 layers are longer than those of the HNC. This is due to the fact that RNC scheme mixes all the packets together; thus it requires a larger number coded packets to decode any packets. However, when enough number of packets are received, it can decode all the packets simultaneously. On the other hand, HNC allows a receiver to recover the important packets early, but pays extra overhead to recover all the packets. This is suitable for scalable video streaming since if there is not enough bandwidth as automatically dictated by TCP congestion control, the receiver can instruct the servers to start sending packets from the next chunk. In the meanwhile, the receiver can playback the important layers that it has received.

Similar results are obtained when the physical bandwidth of the links are reduced to 70 kbps and 60 kbps as shown in Figure 4.8, respectively.

We now compare HNC technique with the coordinated technique in which the receiver instructs the servers 1, 2, 3 to send its packets with equal rates. Packets from more important layers are sent first, followed by the less important ones. It is easy to see that if the available bandwidth of all the servers are equal to each other, this coordinated technique is close to optimal. However, when the available

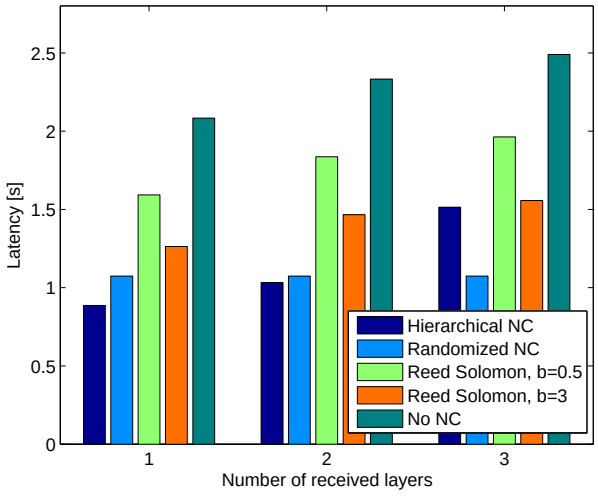


(a)

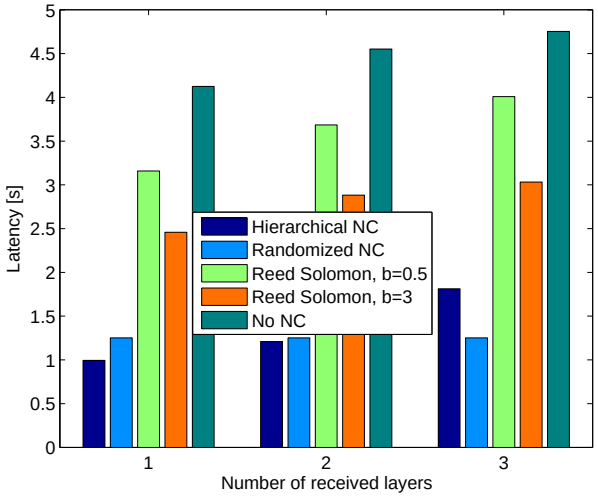


(b)

Figure 4.7: Latencies to receive one chunk using non-network coding, RS, RNC, and HNC for (a) 3 servers; (b) 6 servers; Link bandwidth = 500 kbps.



(a)



(b)

Figure 4.8: Latencies to receive one chunk using non-network coding, RS, RNC, and HNC using six servers (a) physical bandwidth per link = 70 kbps; (b) physical bandwidth per link = 60 kbps.

throughputs of these servers are not equal and varied with time, the HNC technique can outperform this coordinated technique significantly. In particular, we assume that relatively large disjoint partitions of data among the senders are used in this coordinated technique. Furthermore, a receiver can dynamically request a new data allocation from the senders due to the change in the estimated throughput. However, the new allocation request is only sent after the receiver has received all the data from the current allocation. Therefore, if the partition size is large, when the available throughputs change, a receiver may have to wait some time before requesting a new data allocation from the servers. This may result in suboptimal throughput.

We simulate the unequal throughputs by injecting on-off exponential traffic with the mean of 450 kbps for one link, and 250 kbps for each of the other two links. The physical bandwidth for each link is set to 500 kbps. As seen in Figure 4.9, both schemes are able to obtain the base layer packets in reasonable short time. However, the coordinated scheme takes along time to receive the enhancement packets. This is due to the congestion at one link. During this congestion period, for the coordinated scheme, two other servers are idle since they already sent all of their packets. However, the partition sent by the server with the congested link takes a long time to arrive at the receiver. This happens because the receiver cannot dynamically repartition the packets fast enough to accommodate the change in the available throughput. As a result, the coordinated approach takes up to 60% more time to obtain a complete chunk.

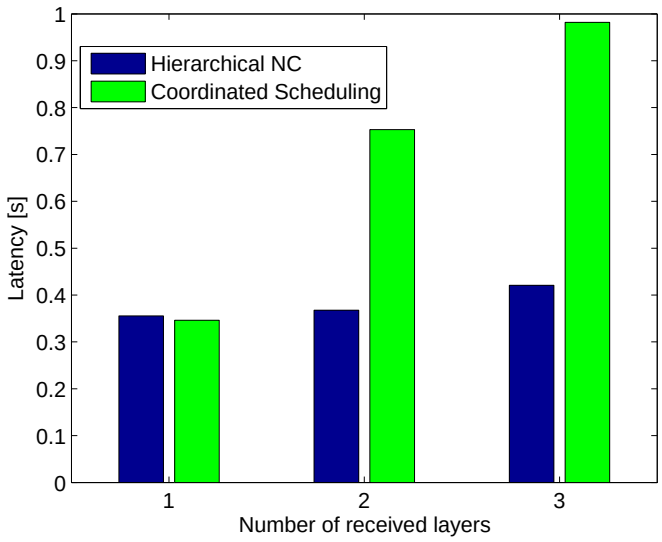


Figure 4.9: Latencies for coordinated transmission and non-coordinated HNC based transmission.

4.7 Conclusions

We have proposed a *network coding* framework for efficient media streaming in CDNs or P2P networks in which, multiple servers/peers are employed to simultaneously stream a video to a single receiver. Our framework reduces the redundancy storage and simplifies the tight synchronization between the senders and receivers. Furthermore, we proposed an HNC technique to be used with scalable video bit stream to enable a receiver to adapt to the available bandwidth. Simulation results demonstrate that under certain scenarios, our proposed schemes can result in bandwidth saving up to 60% over the traditional schemes.

Chapter 5 – Distributed Data Replenishment

1. Kien Nguyen, Thinh Nguyen, Yevgeniy Kovchegov, Viet Le, "*Distributed Data Replenishment*", submitted to IEEE Transactions on Parallel and Distributed Systems, 2010.

5.1 Introduction

Recent development of Peer-to-Peer (P2P) networks opens a new possibility for building large-scale distributed systems over the Internet. Typically in such systems, data are replicated across multiple nodes (peers) at different network locations such that network failures in some parts of the Internet will not prevent a user from accessing the data stored in other parts of the Internet. To that end, many recent research efforts have been focused on using P2P platforms to build reliable, large scale distributed systems for Internet services [73, 85, 96, 6]. That said, one critical drawback of distributed systems, specifically P2P systems, is the overhead of managing data stored at multiple peers who at times, can be highly unreliable. In this paper, we will investigate the theoretical underpinnings and examine the simulated performance for a class of large-scale distributed systems based on a randomized P2P approach via coding techniques. Before describing such systems in detail, we first highlight the fundamental differences between centralized and distributed systems. We also point out the salient features for the proposed class of distributed system based on randomized P2P approach.

Centralized vs. Distributed. Roughly speaking, a centralized system has the ability to access all the information and to perform all the calculations *locally* using a single computational unit. The ability to access all information locally is a marked attribute of a centralized system that enables it to carry out efficient computations. On the other hand, given the current storage and CPU technology trends, the centralized system architecture often does not scale well with storage

and computational capabilities. Importantly, because a centralized system often has all of its resources at one location and its components are tightly coupled, it is more susceptible to bottleneck failures. These drawbacks motivate the distributed system architecture that alleviates both the scalability and bottleneck problems. In a nutshell, a distributed system over the Internet is an overlay network of storage and computing nodes, linked together in such a way to allow computational, storage, and bandwidth resources to be shared. Popular P2P networks such as BitTorrent [1] and KaZaA [2] for example, are distributed systems that enable their users to share data and bandwidth. Because these overlay nodes are located geographically apart, i.e., each node has different network access, and data are replicated across multiple nodes, the bottleneck failures are of less concerned. However, if not properly designed, a typical Internet-wide distributed system will incur substantial communication/coordination overhead among nodes in order to guarantee operational correctness. Thus, the main challenge in realizing an efficient distributed system is to design efficient coordination/communication protocols to minimize such overheads while guaranteeing correctness.

Principle of Minimum Communication and Coordination. In a distributed system, nodes often make decision with limited information. Specifically, each node does not know about the state information of all other nodes. Instead, it only keeps track of the information about its immediate neighbors. This reduces the communication overhead as well as the complexity of maintaining accurate state information at every node in the network. Domain Name Server (DNS) systems is an example of such a system whose each local DNS server only stores

only a limited number of IP addresses. We will discuss shortly how one can employ the minimum communication and coordination principle for an Internet-wide distributed storage system via randomness and coding techniques.

Data replenishment. The fundamental tasks of a distributed storage system is to index the data, maintain the data, and retrieve the data. In this paper, we will not discuss various aspects of indexing and retrieving data in a distributed system. These topics have been well investigated in [76, 77, 85, 73, 31, 79, 80, 78]. Rather, we will focus on scalable methods for maintaining data in a highly volatile environment such as P2P networks. Suppose the data is stored on a peer's hard drive. Then, when a peer departs the network, so does the data it carries. Therefore, it is preferable to employ some form of data replenishment mechanism which ensures that at any time, the requested data is available at one or multiple peers collectively. Furthermore, the data replenishment mechanism should be simple for it to be effective in highly dynamic and distributed environments. Data replenishment mechanism is the focus of this paper.

Approach Overview. Traditionally in a distributed storage system, a file is replicated in its entirety at one or multiple locations. However, for the same overall storage redundancy, a more robust approach is to break up a single file into many pieces, code these pieces properly, then disperse them to multiple nodes in a network [58, 60, 3]. A user recovers the file by downloading its many pieces simultaneously from different locations. In this paper, we consider the following variant of the setup described in [3, 26]. In this setup, a file to be stored, is first broken up into multiple pieces or packets, coded using either Reed-Solomon,

repetition, or random linear network codes (RLNC), then dispersed to a number of peers in the network. We will discuss these techniques shortly. Now, any peer can depart the network along with its data. If a new peer joins, it can be recruited to help replenish the missing data. There are many ways to replenish the missing data. One way is better than the others. We will show that the proposed data replenishment via RLNC is much more efficient than the strategies using repetition and traditional channel code such as Reed-Solomon code. We note that the concept of data replenishment is very much similar to data repair as termed in [26, 25].

The outline of this chapter is as follows. In Section 5.2, we elaborate on different replenishment strategies for a synchronous network model. In Section 5.3, we model the evolution of a piece of data through time for different replenishment schemes as discrete stochastic processes. These processes however, have an exponential large number of states. Because of that, it is difficult to analytically examine their behaviors of functions of time. We then proposed a *time-backward* model based on which, we are able determine an approximate closed form expression for the elapsed time that a piece of data is expected to remain in the system. For the rest of this chapter, we call this expected time as the mean absorption time. Our analytical results show that using the proposed data replenishment via RLNC, the absorption time is exponential in the number of peers used to store the data. This is much more robust than other data protection strategies based on repetition or channel coding techniques whose absorption times are quadratic in the number of peers. As an extension, in Section 5.7, we present the analysis for an asynchronous model which describes peer arrivals and departures as Pois-

son processes. Essentially, our theoretical results show that the performance of the asynchronous model depends critically on the peer arrival and departure rates. Finally, we conclude with some remarks on applying the proposed time-backward model to other problems including gene and population evolutions.

5.2 Synchronous Network Model and Data Replenishment Strategies

As briefly discussed in the Introduction, our distributed storage systems of interest are the types whose files are not stored in their entirety at a specific location. Rather, they are broken up into many pieces, coded for redundancy, and dispersed to multiple locations (peers) in a P2P network. When a particular file is requested, its pieces are downloaded simultaneously from multiple locations. It can be shown that this method increases data availability and reduces congestion bottlenecks [58, 60].

That said, when a peer leaves the network, so does its data. This effectively reduces the robustness of the system temporarily or permanently if the peer never rejoins or rejoins without its data. To avoid this, a peer can transfer its data to some other peers before its departure. However, if the data is large, a peer is less willing to wait until the transfer completes. Thus, without any proactive data replenishment, the redundancy level of a piece of data in the network is continuously reduced. After some period of time, the data of interest is not likely to be recoverable.

Theoretically, if one is to replace the exact missing data in the network, the redundancy level would remain the same. However, this requires global knowledge. Specifically the system needs to know which peer leaves the network and which pieces of data that it has. Then, a precise coordination and communication mechanism is needed to reproduce the equivalent state of the network, more precisely the redundancy amount for a piece of data prior to a peer's departure. This could potentially create significant communication and coordination overheads. Therefore, we take a more scalable, randomized approach that aims to approximately reproduce the state of the network prior to a peer's departures, i.e. *data replenishment.* In this chapter, we explore novel techniques for this approach to maintain the data in the network for as long as possible while minimizing the coordination and communication overheads.

To capture how data redundancy in the network evolves over time, it is important to model the peer arrival and departure processes. For the majority of this chapter, we will study a synchronous model for peer arrival and departure. In this synchronous model, for every peer that leaves the network, the system can find another peer to take over the responsibility of the departed peer. Under certain setting, this model approximates the dynamics of a network with constant number of peers since the departures and arrivals are synchronized. This is the most interesting model as the advantage of Random Linear Network Coding can be clearly demonstrated over other popular channel and repetition coding techniques. That said, an analysis of a more general, but less interesting model with Poisson arrival and departure of peers will be presented in Section 5.7.

We will describe three replenishment strategies in this chapter. Each strategy has to follow the basic rules which model the limited communication and storage capacities of the peers. We abstract the replenishment process as the following game:

The game involves N peers. The objective of the game is for the N peers to collectively maintain some given data, e.g., a file of C bits for as long as possible, subject to the following rules:

1. Each peer is allowed to carry a maximum of T bits.
2. At every time step, a peer is selected uniformly at random to leave the game. Thus the T bits that it carries will also be deleted.
3. A new peer is recruited to replace the departed peer. It is allowed to communicate with a maximum of M peers in an attempt to replenish the data.
4. Peers can modify the data in any way, as long as they do not exceed their storage capacity of T bits.

Given these rules, what is the optimal strategy for the system to maintain a piece of data for as long as possible? Note that, if $M = N - 1$, i.e., the new peer is able to communicate with every other peers, thus it will know exactly what the missing data is, and will be able to restore the missing data quite easily. When $M < N - 1$, it is impossible to know with certainty what data is missing. However, some form of data replenishment might be sufficient to maintain certain level of redundancy in the system. We now describe three replenishment strategies, starting first with the repetition coding technique.

Repetition Code Based Strategy: To be specific, suppose a file to be stored is C bits long, and there are N peers, each can store up to $C/2$ bits. The repetition strategy divides the peers into two groups. Peers in one group are assigned to store the first half of the file, while peers in the other group store the remaining half. Note that the redundancy ratio is the total storage divided by the file size. In this particular case, the redundancy is $\frac{NC/2}{C} = N/2$. Whenever a peer departs, a new peer joins, and communicates with $M = 2$ other peers selected uniformly at random. Since the new peer's capacity is only $C/2$ bits, even it contacts two peers, it will only copy the data from one of these peers, or effectively, $M = 1$. The game is played repeatedly until all the peers have the same piece of data which is either the first half or the second half of the file. It is easy to see that this will happen with probability 1. When this happens, the file is no longer recoverable even with the help of all the peers. But before this happens, all the peers collectively will be able to provide the file.

Reed-Solomon Code Based Strategy: Intuitively, a better strategy is to employ the standard channel coding techniques such as the Reed-Solomon code. Using this strategy, a file of C bits is first divided into three equal parts, which are then channel coded to produce N codewords of length $C/3$ bits. Each peer then keeps a codeword. The redundancy in this case is $N/3$. The property of $RS(N, 3)$ code ensures that a file can be recovered using any of three distinct codewords [50, 92]. Now, the game is played in exactly the same way as before. When a peer departs, the new peer joins, and is allowed to communicate with $M = 2$ peers in an attempt to replenish the missing data. With $M = 2$, the new peer would not

be able reconstruct the entire file. Therefore, its best strategy is to choose one of the codewords from the two contacted peers at random, and copies this codeword to itself to increase data redundancy in the system.

Random Linear Network Code Based Strategy: A yet intuitively better strategy is to employ Random Linear Network Coding technique [35][4]. Using this strategy, a file of C bits is first divided into three equal parts. N codewords are produced, each is a random linear combination of the three original parts of the file. Each peer then keeps a codeword. The redundancy is $N/3$, identical to that of RS code strategy. Mathematically, an $n - bit$ pattern can be viewed as an element from a finite field. Thus, a codeword can be viewed as a vector of elements from a finite field. A codeword A is a random linear combination of codewords B and C, then

$$A = c_1 B + c_2 C, \tag{5.1}$$

where c_i's are elements drawn uniformly at random from a finite field. Assuming that coefficients c_i's are known, it is clear that if all peers have at least three independent codewords (which are formed by three linear independent equations), then the file can be recovered. Note that the number of bits that represents the coefficients is included in the codewords, and is negligible for sufficiently long codewords.

Now, the game for RLNC is played a bit different from the previous two. When a peer departs, the new peer randomly chooses $M = 2$ peers, then copies their data. However, since the new peer's storage capacity is only $C/3$ bits, it generates and

stores only one new codeword as a random linear combination of the two codewords it just copied.

In all of these strategies, the game ends at the moment when all the peers together cannot recover the original file. We will show theoretically that the RLNC based strategy is much better than the others, i.e., it will take longer to play the game. In practice, one can view the distributed storage system based on the RLNC strategy as letting individual peers to perform a simple task in a random manner. However after some time, the data might no longer be recoverable. Thus, a practical system will allow the simple replenishments to continue until the level of redundancy is deemed to fall below a certain threshold, then a more expensive, full-blown replenishment is performed to restore the full level of redundancy. This kind of systems is more scalable than ones that perform expensive replenishments at every single peer departure.

Mathematically, the RLNC game is equivalent to the following procedure. Given an $N \times N$ matrix with rank 3, at each time, a row is replaced by a new row produced by a random linear combination of two other rows. The question is how long does it take for the rank of the matrix to reduce to 2.

5.3 Discrete Stochastic Model for Random Linear Network Coding Based Replenishment Strategy

In this section, we describe a discrete stochastic model for the RLNC based replenishment strategy. This model is analytically intractable due to a large number

of states. However, it serves as a motivation for the proposed time-backward technique that approximates the expected time until the file is no longer recoverable.

As described in Section 5.2, each replenished codewords can be viewed as a row in a matrix, and is linearly dependent on the other rows that were used to generate it. Therefore, over time one would expect the number of linearly independent rows decreases. Eventually, the rank of the matrix will reduce below the number of original data packets. At this point, even when all rows are used, the data cannot be recovered. Our objective is to determine the average time until this happens.

To help with the modeling process, we start with the following claim:

Given N codewords, each is a vector of L elements in a finite field $\mathbb{F}$. A new codeword of the same length is generated with the elements drawn uniformly at random from the same field. The probability that this new codeword is linearly independent from any combination M codewords from the given N codewords is almost unity if L and $|\mathbb{F}|$ are sufficiently large. [1]

Another way to view this is that if the elements are drawn from $\mathbb{R}$[1], then a randomly drawn row will definitely be independent from any other M equations because $|\mathbb{R}|$ is infinite. We will make this approximation to model the replenishment process as follows.

For simplicity, we will focus on the following simple scenario. A file is broken up into $K = 3$ parts, then N codewords are generated by linearly combining these three parts (codewords) at random. Now, at every time step, a new peer is chosen uniformly at random to depart. A new peer joins. Two distinct remaining peers

[1]It is straightforward to show that the lower bound for this probability is $1 - \frac{\binom{N}{M}}{|\mathbb{F}|^{L-1}}$.

($M = 2$) are then uniformly chosen at random to have their codewords copied to the new peer. The new peer then generates its new codeword by linearly combining these two codewords with random coefficients. In general, if $M \geq K$, it will almost always be possible to recover the file, independent of the number of replenishments. This is because we can almost always get K linearly independent codewords, unless with a small probability, the generated codeword happens to be linearly dependent on some $M' < K$ codewords.

We use the diagram in Figure 5.1 to visually depict how the dependencies among the codewords progress in discrete time steps. The meanings of the solid and non-solid circles will become clear shortly when we discuss the time-backward process. For now, at time step $n = 0$, there are 7 codewords. Any of these codewords can be represented as a linearly combination of any three other codewords due to the initial mixing. At time step $n = 1$, the codeword 6 is replaced by a random linear combination of codewords 5 and 7. At this stage, the file can be recovered using any triplet of codewords except (5,6,7) since these three codewords are not linearly independent. At $n = 2$, codeword 2 is replaced by a random linear combination of codewords 1 and 3. As such, one cannot use triplets (5,6,7) or (1,2,3) to recover the file at this time. The process repeats, and eventually, all codewords will be some linear combinations of some two codewords, and the file will no longer be recoverable. A discrete time Markov chain representation, specifically a transition probability matrix can be used describe this replenishment process. However, a direct application of this method requires an exponentially large number of states (N) where a state denotes a configuration in the diagram. For example, at any

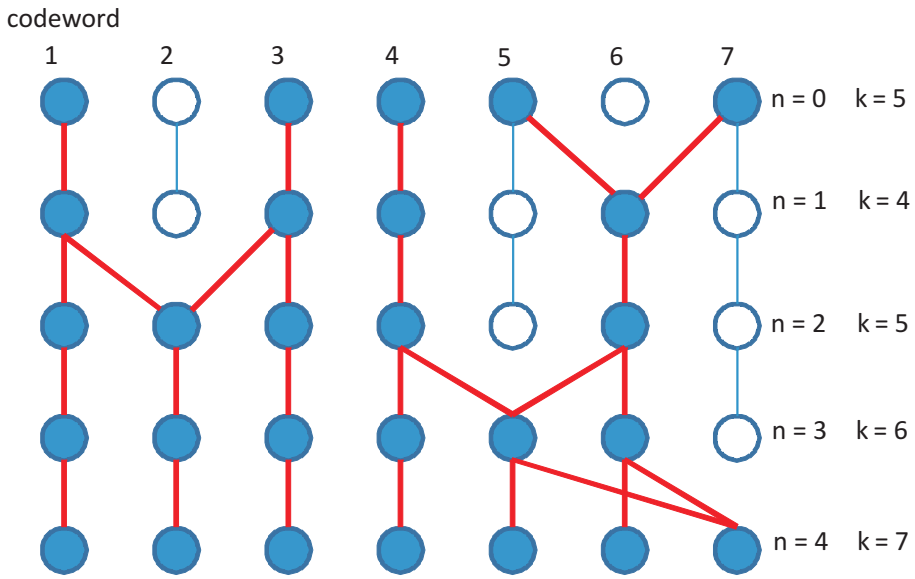


Figure 5.1: Progression of codewords for seven peers, $N = 7$, $M = 2$ in discrete time steps n. Codewords are represented by the circles. A circle in the current time step that is connected to two circles in the previous time step, represents a codeword that is a linear combination of two codewords. Solid circles are parent codewords that are part of the linear combinations in the current codewords, while non-solid circles are not part of the linear combinations of the current codewords.

time step, there are approximately $N \times \binom{N}{2}$ states that the chain can transition to, making this approach analytically intractable.

Our contributions is a modeling technique that produces an approximate but closed form solution for the expected number of time steps to get from any state to any other, including the state in which the file is no longer recoverable. Furthermore, we can bound the error on this approximate time by a factor of 2. The key to this modeling technique is to consider a more tractable time-backward model in which the replenishments are performed backward in time. We now describe the

time-backward model.

5.4 Time-Backward Model

We consider the time-backward process where we begin with $n = 4$ and end with $n = 0$. Fig. 5.1 shows two types of circles. The solid circles denotes the parent nodes which are the codewords involving in the linear combination at different time steps. The non-solid circle represent codewords that are not parents nodes. Let redefine n as the number of time steps going backward from some initial state. Now, let X_n denote the number of parent nodes at time n where n denotes the number of time steps from the initial state with the number of parent nodes $X_0 = N$. For example, in Figure 5.1, $X_0 = 7$ and $X_4 = 5$. Clearly, all the codewords at time $n = 0$ are linearly dependent on the codewords 1,2,3,4,5,6 at time $n = 1$. All the codewords at time $n = 1$, are linearly dependent on the codewords 1, 2, 3, 4, 6 at time $n = 2$, and so on. With this setup, one can view the time-backward process as a one dimensional random walk X_n on $2, 3, \ldots, N$. For the case where $M = 2$, one can write down the following transition probabilities:

$$\begin{aligned}
P(X_{n+1} = k-1 | X_n = k) &= \frac{k}{N}\left(\frac{k-1}{N-1}\right)\left(\frac{k-2}{N-2}\right) \\
P(X_{n+1} = k+1 | X_n = k) &= \frac{k}{N}\left(\frac{N-k}{N-1}\right)\left(\frac{N-1-k}{N-2}\right) \\
P(X_{n+1} = k | X_n = k) &= 1 - \frac{k}{N}\left(\frac{k-1}{N-1}\right)\left(\frac{k-2}{N-2}\right) \\
&\quad - \frac{k}{N}\left(\frac{N-k}{N-1}\right)\left(\frac{N-1-k}{N-2}\right)
\end{aligned}$$

Furthermore, when $X_n = M$, we can artificially stop the process, i.e., setting the transition probabilities from this state to all other states to 0. At this stage, the file is no longer recoverable.

That said, X_n can only take on values between M and N, so the size of the transition matrix $\mathbf{P}$ is $(N-M+1) \times (N-M+1)$, thus is much more manageable as compared to modeling the time-forward process.

For example, with $M = 2$, $N = 7$, the corresponding transition probability matrix is:

$$\mathbf{P} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1/35 & 4/5 & 6/35 & 0 & 0 & 0 \\ 0 & 4/35 & 27/35 & 4/25 & 0 & 0 \\ 0 & 0 & 2/7 & 2/3 & 1/21 & 0 \\ 0 & 0 & 0 & 4/7 & 3/7 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Note that this type of matrices has one recurrent state $X_n = 2$ (the first row in the matrix), the rest of the states are transient.

5.4.1 Mean Absorption Time

Based on $\mathbf{P}$, one can immediately compute the mean absorption time, i.e., the expected number of time steps, starting from an initial transient state to a recurrent state, using a standard technique. Specifically, let $\mathbf{Q}$ be the submatrix of $\mathbf{P}$ that includes only the rows and columns corresponding to the transient states, then by rearranging the order of the states, one can write the transition probability matrix $\mathbf{P}$ as:

$$\mathbf{P} = \begin{pmatrix} \tilde{\mathbf{P}} & \mathbf{0} \\ \mathbf{S} & \mathbf{Q} \end{pmatrix}.$$

Let

$$\mathbf{M} = (\mathbf{I} - \mathbf{Q})^{-1}, \tag{5.2}$$

Let B_i be the random variable denoting the absorption time starting in a transient state i, the the mean absorption time is:

$$\mathbb{E}B_i = \sum_j \mathbf{M}_{ij} \tag{5.3}$$

5.4.2 Variance of Absorption Time

Given $\mathbf{P}$, we can also explicitly compute the variance of absorption time, starting in any transient state. To the best of our knowledge, we have not found any result on computing the variance of random walk in such a closed form as the one presented

below. We have the following proposition:

Proposition 5.1

$$Var(B_i) = (\mathbf{I} - \mathbf{P})^{-1}(\mathbf{I} + 2\mathbf{PM1}) - (\sum_j \mathbf{M}_{ij})^2 \tag{5.4}$$

Proof 5.1 *Using the well-known recursion, we have:*

$$B_i^2 = \sum_j p_{ij}(1 + B_j)^2. \tag{5.5}$$

Thus,

$$\begin{aligned} \mathbb{E}B_i^2 &= \mathbb{E}[\sum_j p_{ij}(1 + B_j)^2] \\ &= 1 + \sum_j p_{ij}\mathbb{E}B_j^2 + 2\sum_j p_{ij}\mathbb{E}B_j. \end{aligned}$$

Rearranging the terms, we have

$$\mathbb{E}B_i^2 - \sum_j p_{ij}\mathbb{E}B_j^2 = 1 + 2\sum_j p_{ij}\mathbb{E}B_j \tag{5.6}$$

Let $\mathbf{B^{(1)}} = (\mathbb{E}B_1, \mathbb{E}B_2, \ldots, \mathbb{E}B_N)^T$ *and* $\mathbf{B^{(2)}} = (\mathbb{E}B_1^2, \mathbb{E}B_2^2, \ldots, \mathbb{E}B_N^2)^T$*, then one can write Equation (5.6) in matrix form as:*

$$\begin{aligned} (\mathbf{I} - \mathbf{P})\mathbf{B^{(2)}} &= \mathbf{1} + 2\mathbf{PB^{(1)}} \\ &= \mathbf{1} + 2\mathbf{P}(\mathbf{I} - \mathbf{Q})^{-1}\mathbf{1}. \end{aligned} \tag{5.7}$$

Thus,

$$\begin{aligned}\mathbf{B^{(2)}} &= (\mathbf{I}-\mathbf{P})^{-1}(\mathbf{I}+2\mathbf{P}(\mathbf{I}-\mathbf{Q})^{-1})\mathbf{1} \\ &= (\mathbf{I}-\mathbf{P})^{-1}(\mathbf{I}+2\mathbf{PM})\mathbf{1} \\ &= (\mathbf{I}-\mathbf{P})^{-1}(\mathbf{1}+2\mathbf{PM1}) \qquad (5.8)\end{aligned}$$

Finally, we finish the proof by noting that $Var(B_i) = \mathbb{E}B_i^2 - (\mathbb{E}B_i)^2$ *and* $(\mathbb{E}B_i)^2 = (\sum_j \mathbf{M}_{ij})^2$.

Note that this method can be easily extended to compute a recursive expression for higher moments.

5.4.3 Bounding Mean Absorption Time of Time-Forward Process with Time-Backward Walk

We have shown that, in contrast to the time-forward process, modeling the corresponding time-backward walk is quite tractable. We now show that the expected absorption time of the time-forward process can be approximated well by that of the corresponding time-backward walk. Specifically, we have the following Proposition:

Proposition 5.2 *Let* F_i *and be* B_i *be the random variables denoting the absorption times of the time-forward process and time-backward walk, starting in state* i*, respectively, then*

$$\mathbb{E}B_i \leq \mathbb{E}F_i < 2\mathbb{E}B_i \qquad (5.9)$$

Proof 5.2 *We first prove the lower bound. As shown in Fig. 5.2, a sequence of forward walk that results in all the codewords being the children of only two codewords must contain a sequence of backward walk that reaches these two codewords. Thus,* $\mathbb{E}B_i \leq \mathbb{E}F_i$.

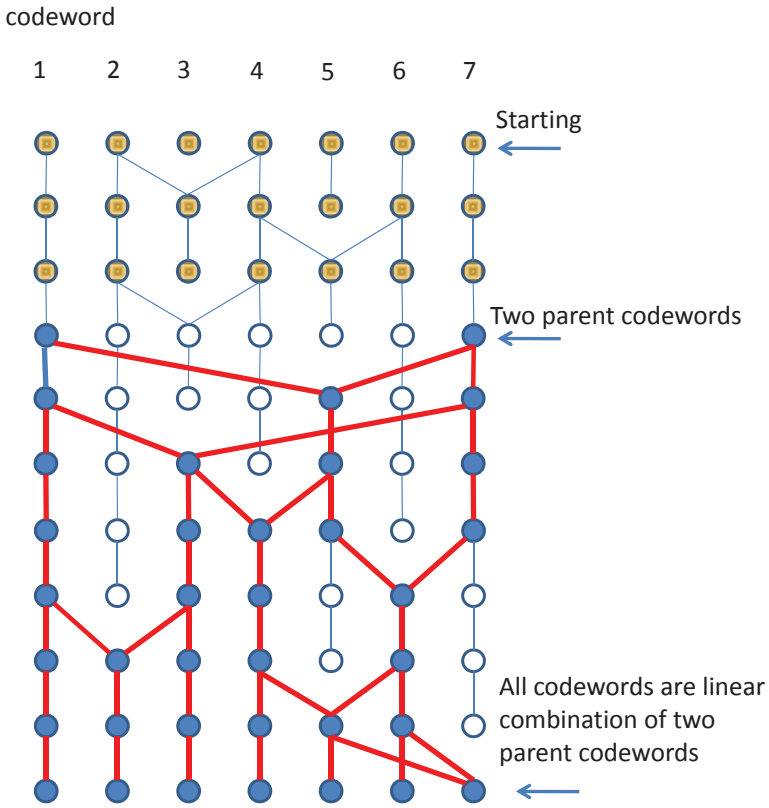


Figure 5.2: Illustrating diagrams for proof of lower bound.

We now prove the upper bound. Suppose we walk backward from the state $X_0 = k = N$ *until there are only two parents nodes* ($k = 2$). *We then reset* $k = N$, *then walk backward one more time as shown in Figure 5.3. The total expected*

number of time steps to reach the second merge, counting from the beginning, is $2\mathbb{E}B_i$. *Now if the forward walk starts earlier than* $2\mathbb{E}B_i$ *time steps, then it must have encountered at least two instances where every node is a descendant of only two parents node. But, by definition,* $\mathbb{E}F_i$ *is the number of time steps for the chain to reach* $k = 2$, *and stays at* $k = 2$ *permanently. Thus,* $\mathbb{E}F_i < 2\mathbb{E}B_i$.

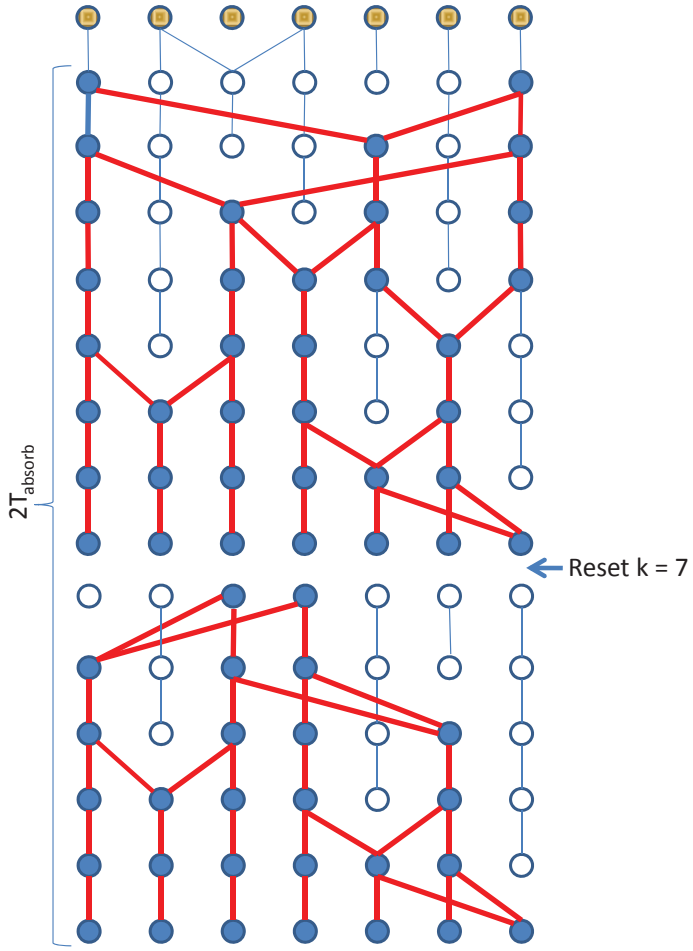


Figure 5.3: Illustrating diagrams for proof of upper bound.

Although the theoretical upper bound on $\mathbb{E}F_i$ is a bit loose, simulation results in the Section 5.5.2 reveal that $\mathbb{E}F_i$ is quite close to $\mathbb{E}B_i$.

5.5 Exponential Rate for Data Replenishment via Random Linear Network Coding

5.5.1 Analysis of Exponential Mean Absorption Time

While the time-backward model allows one to compute a closed-form solution for the expected absorption time starting from any state, it relies on the computation of $(\mathbf{I} - \mathbf{Q})^{-1}$ which often cannot be used to examine the asymptotic behavior of the absorption time as a function of N. We now show how to compute a closed form expression for the expected absorption time in terms of N which shows that the mean absorption time is exponential in N. We have the following Proposition:

Proposition 5.3 *Given N node and M, the mean absorption time for the time backward walk is at least exponential in N, and thus the mean absorption time for the time-forward walk is also at least exponential in N.*

Proof 5.3 *We present a proof based on the classical method for computing hitting time of a discrete Markov chain. In the Appendix, we provide an alternative proof that further confirms the exponential mean absorption time. To be concrete, we consider the case of $M = 2$, i.e., make a new codeword as a random linear combination of two codewords. It is straightforward to show that the mean absorption time for the case $M > 2$ must be greater than that of $M = 2$.*

Denote h_k as the mean absorption time starting in the state $X_0 = k$, for $k = 3, 4, \ldots, N-2$, and ending in state $X_n = 2$ for some n. Then, we can write down the following recursion:

$$\begin{aligned} h_k = 1 \quad &+ \quad h_{k-1}\frac{k}{N}\left(\frac{k-1}{N-1}\right)\left(\frac{k-2}{N-2}\right) + h_{k+1}\frac{k}{N}\left(\frac{N-k}{N-1}\right)\left(\frac{N-1-k}{N-2}\right) \\ &+ \quad h_k\left[1 - \frac{k}{N}\left(\frac{k-1}{N-1}\right)\left(\frac{k-2}{N-2}\right) - \frac{k}{N}\left(\frac{N-k}{N-1}\right)\left(\frac{N-1-k}{N-2}\right)\right] \end{aligned}$$

Letting $y_k = h_{k+1} - h_k$, and after some term re-arrangements, we have

$$y_{k-1} = y_k\frac{(N-k)(N-1-k)}{(k-1)(k-2)} + \frac{N(N-1)(N-2)}{k(k-1)(k-2)}$$

Now, this is a difference equation with the following initial conditions:

$$y_{N-1} = h_N - h_{N-1} = 1,$$

and

$$y_{N-2} = h_{N-1} - h_{N-2} = \frac{N}{N-3}.$$

The first initial condition is true because the first step always reduces the number of parents nodes by 1. The second condition is true because in the special state $X_n = N-1$, the chain can either go to $(N-2)$ or stay at $(N-1)$. It will never go back to N as seen in the Fig. 5.1. The probability that the chain will go to $N-2$ given that it is currently in $N-1$ is the probability that all three selected nodes are

the parent nodes in the current time step. This probability is

$$\frac{(N-1)(N-2)(N-3)}{N(N-1)(N-2)} = \frac{N-3}{N}.$$

Thus the expected number of trials before moving to $N-2$ *is* $\frac{N}{N-3}$.

The difference equation is of the form:

$$y_{k-1} = a_k y_k + b_k, \tag{5.10}$$

where

$$a_k = \frac{(N-k)(N-1-k)}{(k-1)(k-2)} \tag{5.11}$$

and,

$$b_k = \frac{N(N-1)(N-2)}{k(k-1)(k-2)}, \tag{5.12}$$

for $k = 3, 4, \ldots, N-2$.

By performing a few recursions, starting at any N_0*, we have*

$$\begin{aligned} y_{N_0-k} &= a_{N_0} a_{N_0-1} \ldots a_{N_0-k+1} y_{N_0} \\ &+ a_{N_0-1} a_{N_0-2} \ldots a_{N_0-k+1} b_{N_0} + \cdots + a_{N_0-k+1} b_{N_0-k+2} + b_{N_0-k+1}, \end{aligned} \tag{5.13}$$

which can be written as:

$$y_{N_0-k} = \Big(\prod_{i=N_0-k+1}^{N_0} a_i\Big) y_{N_0} + \sum_{j=N_0}^{N_0-k+2} b_j \prod_{i=j-1}^{N_0-k+1} a_i + b_{N_0-k+1}. \tag{5.14}$$

Let $N_0 = N-2$, *we have* $y_{N_0} = y_{N-2} = \frac{N}{N-3}$, $k = 0, 1, \ldots, N-4$, *and*

$$\begin{aligned} y_{N-2-k} &= \frac{N}{N-3} \prod_{i=N-k-1}^{N-2} \frac{(N-i)(N-1-i)}{(i-1)(i-2)} \\ &+ \sum_{j=N-2}^{N-k} \frac{N(N-1)(N-2)}{j(j-1)(j-2)} \prod_{i=j-1}^{N-k-1} \frac{(N-i)(N-1-i)}{(i-1)(i-2)} \\ &+ \frac{N(N-1)(N-2)}{(N-k-1)(N-k-2)(N-k-3)}. \end{aligned} \tag{5.15}$$

Next, to find mean absorption time h_k, *i.e. the expected number of time steps for the chain to reach* $X_n = 2$ *from* $X_0 = k$, *for* $k = 3, 4, \ldots, N-2$, *we have*

$$\sum_{j=2}^{k-1} y_j = h_k - h_2 = h_k, \tag{5.16}$$

since $h_2 = 0$. *The expected number of time steps before the file is no longer recoverable is:*

$$h_N = h_{N-2} + 1 + \frac{N}{N-3} = \sum_{j=2}^{N-3} y_j + 1 + \frac{N}{N-3} \tag{5.17}$$

We now show that h_N *is at least exponential in* N.

We consider

$$\begin{aligned} z_{N_0-k} &= a_{N_0} a_{N_0-1} \ldots a_{N_0-k+1} \\ &\quad + a_{N_0-1} a_{N_0-2} \ldots a_{N_0-k+1} + \cdots + a_{N_0-k+1} \end{aligned}$$

for $k = 2, 3, 4, \ldots, N_0$, *with* $N_0 \leq N-2$.

Note that z_k is a modified version of y_k as defined in Equation (5.15). It is easy to see that

$$y_k > z_k, \tag{5.18}$$

for $k = 2, 3, \ldots, N_0$, since $y_{N_0} > 1$ and $b_k > 1$.

Therefore it is sufficient to show that if z_2 is exponential in N_0 then y_2 is at least exponential in N_0, and thus h_{N_0} as defined in Equation (5.17) must be at least exponential in N_0. We proceed as follows.

First, we note that

$$\begin{aligned} a_k &= \frac{(N-k)(N-1-k)}{(k-1)(k-2)} \\ &> \frac{(N-k)(N-1-k)}{k(k-1)} \\ &\approx \left(\frac{N-k}{k}\right)^2 \end{aligned} \tag{5.19}$$

Now let $a'_k = \left(\frac{N-k}{k}\right)^2$*, we have*

$$\begin{aligned}
z_2 &= a_3 + a_3a_4 + a_3a_4a_5 + \cdots + a_3a_4 \ldots a_{N_0} \\
&> a'_3 + a'_3a'_4 + a'_3a'_4a'_5 + \cdots + a'_3a'_4 \ldots a'_{N_0} \\
&= \frac{a'_1 + a'_1a'_2 + a'_1a'_2a'_3 + \cdots + a'_1a'_2 \ldots a'_{N_0}}{a'_1a'_2} - \frac{1 + a'_2}{a'_2} \\
&= \frac{\sum_{i=1}^{N_0} \prod_{k=1}^{i} a'_k}{a'_1a'_2} - \frac{1 + a'_2}{a'_2} \\
&= \frac{4\sum_{i=1}^{N_0} \prod_{k=1}^{i} \left(\frac{N-k}{k}\right)^2}{(N-1)^2(N-2)^2} - \frac{4}{(N-2)^2} - 1 \\
&= \frac{4\sum_{i=1}^{N_0} \binom{N-1}{i}^2}{(N-1)^2(N-2)^2} - \frac{4}{(N-2)^2} - 1 \\
&= \frac{4\sum_{i=1}^{N_0} \binom{N-1}{i}\binom{N-1}{N-i-1}}{(N-1)^2(N-2)^2} - \frac{4}{(N-2)^2} - 1 \\
&> \frac{4\sum_{i=1}^{N_0} \binom{N_0}{i}\binom{N_0}{N_0-i}}{(N-1)^2(N-2)^2} - \frac{4}{(N-2)^2} - 1 \\
&= \frac{4[\binom{2N_0}{N_0} - 1]}{(N-1)^2(N-2)^2} - \frac{4}{(N-2)^2} - 1 \\
&\approx \frac{2^{2(N_0+1)}}{(N-1)^2(N-2)^2} - \frac{4}{(N-2)^2} - 1 \qquad (5.20)
\end{aligned}$$

where the Stirling's approximation is used in Equation (5.20).

By Proposition 5.2, the mean absorption time for the time-forward model must be exponential in N.

5.5.2 Simulation results for RLNC based data replenishment

In this section, we show the simulation results that demonstrate the theoretical exponential mean absorption time using the RLNC-based data replenishment. Furthermore, our simulation results show that the mean absorption time of the time-forward model is very close to that of time-backward model, i.e. much smaller closer to the lower bound than the upper bound given in Proposition 5.2. Therefore in many settings, replacing the time-forward model with the time-forward model would not sacrifice much accuracy on how long a piece of data is to remain in the system.

First, we want to show that the mean absorption time is indeed exponential. Figure 5.4 shows the log of mean absorption time as a function of N, the number of nodes for both the time-forward and time-backward models. Recall that N is the number of nodes used to store the data. In this simulation, a newly arrival node always connects to $M = 2$ existing nodes to download and mix their data. Originally, there is a total of three pieces of independent information, i.e., the redundancy ratio is $N/3$. The mean absorption time is the expected time (the number of pairs of departures and arrivals) until all the nodes collectively contain exactly two pieces of independent information. From the previous stochastic model, this happens when the number of parents nodes equal to two. The graphs in Figure 5.4 shows two relatively straight line segments. Since the y-axis is in log scale, this indicates that both the mean absorption times of the time-forward and time-backward models are exponential in the number of nodes used to store the data.

Furthermore, they are almost identical in the log scale. This closeness also exhibits in the linear scale graphs (not shown). It is noted that the mean absorption time for the time-forward model is always larger than that of the time-backward model, following the lower bound in Proposition 5.2.

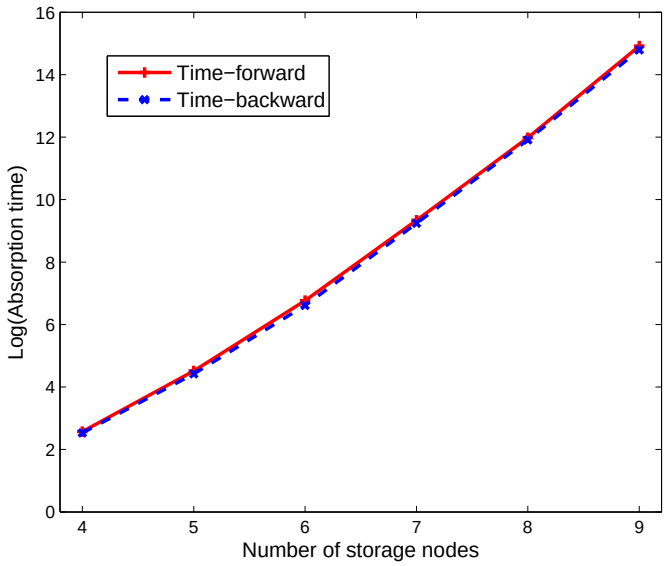


Figure 5.4: Log of mean absorption time vs. the number of nodes for $N = 4, .., 9; M = 2$ for the RLNC strategy.

Next, we investigate the mean absorption time as a function of redundancy given a constant number of nodes $N = 9$. Specifically, if at the start, there are N pieces of independent information, then the mean absorption time is the expected time that the number of parents nodes reduces to k. Every newly arrival peer still connects to $M = 2$ peers for data replenishment. Figure 5.5 shows log of mean absorption time versus k, the number of parents nodes. Again, the mean absorption

time for the time-forward model is higher than that of the time-backward model as predicted, but they are close to each other.

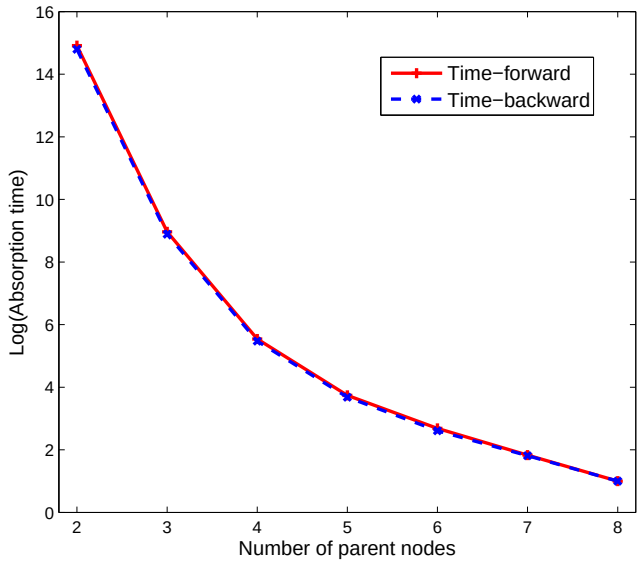


Figure 5.5: Log of mean absorption time vs. the number of parent nodes for $N = 9; M = 2$ for the RLNC strategy.

Figure 5.6 show the absorption times (minimum, maximum, median, 25 and 75 percentile) for the time-forward and time-backward models when the number of nodes N varies from 6 to 9 and $M = 2$ connections for data replenishment. Again, they are very much in agreement with each other.

Finally, we also investigate the performance of the RLNC-based data replenishment scheme when $M \geq 2$ connections are used. Specifically, a newly arrival peer chooses two, three, or four peers uniformly at random to download the data and perform replenishment. Figure 5.7 shows the log of the mean absorption time for

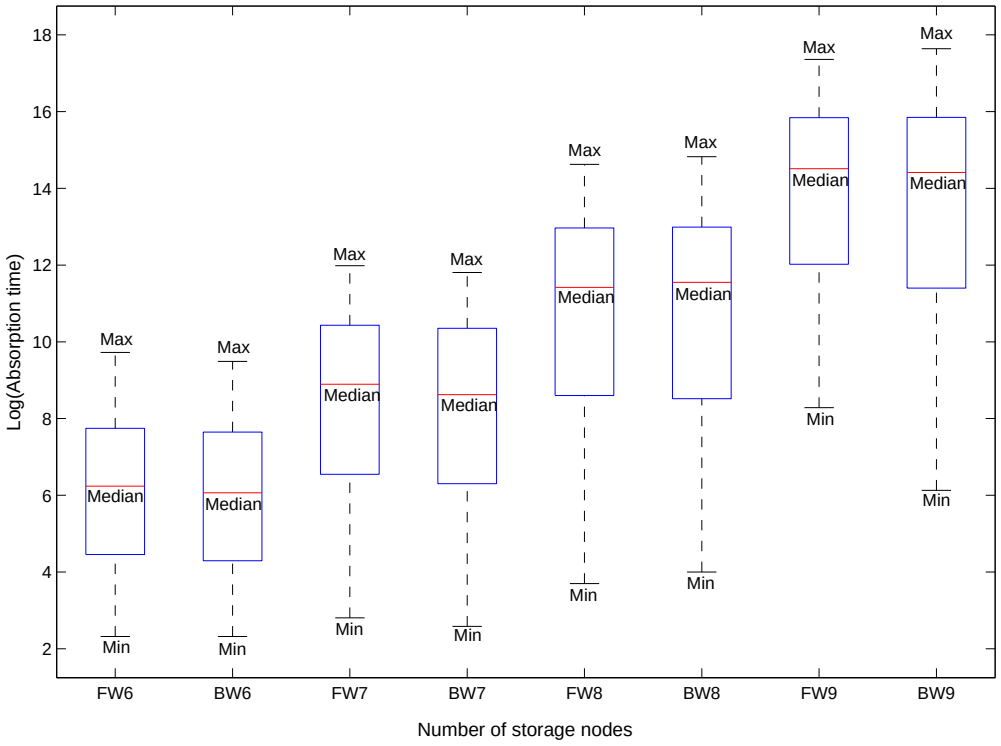


Figure 5.6: Log of absorption time vs number of original nodes for $N = 6, .., 9; M = 2$ for the RLNC strategy.

the time-backward and time-forward models vs. k, the number of parents nodes, for the case of $N = 8$ nodes.

The simulation results show that for larger values of M, one can expect a longer mean absorption time. This is intuitively plausible as a larger M would reduce the chance of creating dependency at each replenishment steps.

5.6 Quadratic Rate for Data Replenishment via RS and Repetition Codes

In this section, we show that using replenishment based on the RS and repetition codes, the number of time steps before a file is no longer recoverable is of $O(N^2)$, and thus is less effective than that of the RLNC based strategy. We begin with the RS based strategy.

5.6.1 Mean Absorption Time for RS-based Strategy

A file is divided into three parts, coded using $RS(N, 3)$. Each peer keeps a codeword, resulting in a redundancy level of $N/3$. A new peer is allowed to contact with $M = 2$ peers. Since with $M = 2$, the new peer cannot recover the file, thus it will randomly copy the codeword from one of the two peers. In this special case, it is just as good as communicating with only $M = 1$ peer. Note that it is also straightforward to analyze the general case where $RS(N, K)$ with $(2 < M < K)$ is used. In this general case, the new peer still cannot reconstruct the file, however it is potentially better to copy the data that is least duplicated among the M peers, so as to increase the diversity. For simplicity, let us consider the case where $M = 2$, or effectively $M = 1$.

Using the time-backward model, it is straightforward to model the RS-based strategy as shown in Fig. 5.8.

Similar to the RLNC strategy, let k denote the number of parents nodes. A file

is then irrecoverable when $k = 2$. We can write the mean absorption time using the recursion as

$$\begin{aligned} h_k &= \frac{k}{N}(\frac{k-1}{N-1})h_{k-1} + (1 - \frac{k}{N}\frac{k-1}{N-1})h_k + 1 \\ h_k &= h_{k-1} + \frac{N(N-1)}{k(k-1)} \end{aligned} \tag{5.21}$$

for $k = 3, 4, \ldots, N$.

Now $h_2 = 0$ hence,

$$\begin{aligned} h_N &= \sum_{k=2}^{N-1} \frac{N(N-1)}{k(k+1)} \\ &= N(N-1) \sum_{k=2}^{N-1} \left(\frac{1}{k} - \frac{1}{k+1} \right) \\ &= \frac{(N-1)(N-2)}{2} \end{aligned} \tag{5.22}$$

By Proposition 5.2, the mean absorption time of the time-forward process cannot be less than h_N.

Note that if $RS(N, 2)$ is used, i.e., redundancy is increased to $N/2$. Then, using the same method, we have $h_N = (N-1)^2$. Thus, the mean absorption time for the time-forward walk cannot be more than $2h_N$. For large N, this is a small improvement over the repetition code strategy for the same redundancy as shown below.

5.6.2 Mean Absorption Time for Repetition Code Strategy

Suppose a file is split into two parts and there are N peers, each containing either parts of the file. For this strategy, whenever a peer leaves and a new peer enters, a peer is picked uniformly at random out of $N-1$ existing peers, and its data is copied to the new peer. The process is of birth-and-death type on $\{0, 1, \ldots, N\}$ with two absorbing states, 0 and N. We would like to estimate the mean absorption time.

In the above birth-and-death process the forward probabilities are given by

$$p_k = \frac{k(N-k)}{N(N-1)} \quad \text{for } k = 1, 2, \ldots N, \quad \text{and } p_0 = 0$$

and the backward probabilities are

$$q_k = \frac{k(N-k)}{N(N-1)} \quad \text{for } k = 0, 1, \ldots N-1, \quad \text{and } q_N = 0.$$

The expected absorption time $h_k = E_k[T_0 \wedge T_n]$ solves the following recurrence equation: For $k = 1, 2, \ldots, N-1$,

$$\begin{aligned} h_k &= 1 + \frac{k(N-k)}{N(N-1)} h_{k-1} + \frac{k(N-k)}{N(N-1)} h_{k+1} + \left(1 - \frac{2k(N-k)}{N(N-1)}\right) h_k \\ h_0 &= h_N = 0 \end{aligned} \tag{5.23}$$

The above equation can be rewritten as follows

$$y_k = -(N-1)\left(\frac{1}{k} + \frac{1}{N-k}\right) + y_{k-1},$$

where $y_k = h_{k+1} - h_k$.

Thus

$$\begin{aligned} y_k &= -(N-1)\left(1 + \frac{1}{2} + \cdots + \frac{1}{k}\right) \\ &\quad -(N-1)\left(\frac{1}{N-k} + \cdots + \frac{1}{N-1}\right) + y_0. \end{aligned} \tag{5.24}$$

Now, by symmetry,

$$-y_0 = y_{N-1} = -2(N-1)\left(1 + \frac{1}{2} + \cdots + \frac{1}{N-1}\right) + y_0$$

and therefore

$$y_0 = (N-1)\left(1 + \frac{1}{2} + \cdots + \frac{1}{N-1}\right). \tag{5.25}$$

Plugging y_0 into (6.6), and after some algebraic manipulations, we obtain:

$$\begin{aligned} h_{k+1} &= (N-1)k\left(1 + \frac{1}{2} + \cdots + \frac{1}{N-1}\right) \\ &\quad -(N-1)(k+1)\left(1 + \frac{1}{2} + \cdots + \frac{1}{k}\right) \\ &\quad +(N-1)(N-k-1)\left(\frac{1}{N-k} + \cdots + \frac{1}{N-1}\right). \end{aligned} \tag{5.26}$$

Taking $k+1 = \frac{N}{2}$, we obtain

$$h\left(\frac{N}{2}\right) \approx \ln 2 \cdot N^2 \tag{5.27}$$

5.7 Extension: Asynchronous Network Model

In Section 5.2, we consider the network model in which, peer departures and arrivals are synchronized. In this section, this assumption is relaxed. Instead, we model the peer arrivals and departures as Poisson processes with rate $N\lambda$ and $N\mu$ [2] where N denotes the current number of peers in the network. It is easy to show that if $\lambda \geq \mu$, a piece of data will be almost certain to remain in the network even when using the repetition coding technique, a weaker form of data replenishment. Intuitively, this is because the amount of storage increases with time outweighs the data dependency due to replenishment. When $\lambda < \mu$, the number of peers will reduce to zero quickly (linear with N) so that any form of replenishment will be ineffective in this scenario. Therefore, in this section, we will mainly address this model under the repetition coding technique for the case when $\lambda \geq \mu$, and show that there is a chance that the data will be disappeared , but this probability is exponentially small with N, the number of peers.

In particular, suppose the data is divided into r distinct equal-sized parts to be replicated at multiple nodes. We consider a system of $m \geq r$ peers, each peer

[2] One can also make the peer arrival rate as λ instead of $N\lambda$ to reflect that the peer arrival rate is independent of the number of existing peers. However, this implies that no peer will survive after a short period of time (order of $\log N$), and thus this scenario is not interesting.

stores exactly one part of the data. Redundancy is created by having multiple peers store the same part of the data. Peer arrivals and departures follows Poisson processes. Whenever a new peer arrives, it picks uniformly at random a peer from the existing peers and copies its data. Clearly, the data cannot be recovered if at least one part of the data is missing in system. *We are interested in computing this irrecoverable probability given that a distributed storage system begins with N peers.*

One important observation is that this problem can be viewed as a classical parallel queuing system consisting of r statistically independent queues with different arrival and service rates. The number of packets in each distinct queue represent the number of peers storing the same parts of the data. Specifically, let n_1, n_2, ..., n_r be the queue sizes, i.e., the number of peers carrying the data parts 1 to r at any point in time. Then, it is sufficient to analyze the occupancy of any one queue. The departure rate of peers storing parts i is $k_i\mu/N$. The arrival rate of peer of type i is then $k_i\lambda/N$. We now analyze only one queue of interest, say queue i. To simplify the notation, let us denote $n = n_i$ as the number of packets in the queue i. Then, the number of packets in the queue (number of peers carrying the same parts of the data) evolves according to a birth-and-death process on $\{0, 1, \dots\}$ with the recurrence state 0. It has the transition probabilities:

$$p(n, n-1) = \frac{n\mu}{n\lambda + n\mu} = \frac{\mu}{\lambda + \mu} \tag{5.28}$$

and

$$p(n, n+1) = \frac{n\lambda}{n\lambda + n\mu} = \frac{\lambda}{\lambda + \mu} \tag{5.29}$$

Let $a(n)$ be the probability that the system starting at state n ever reach state 0. Note that $a(0) = 1$ and the values of $a(n)$ is the same whether one considers the continuous-time or discrete-time system. It is satisfied the following recursive equation:

$$\begin{aligned} a(n) &= a(n-1)p(n, n-1) + a(n+1)p(n, n+1) \\ &= a(n-1)\frac{\mu}{\lambda + \mu} + a(n+1)\frac{\lambda}{\lambda + \mu}, \ n \geq 1 \end{aligned} \tag{5.30}$$

Equation 5.30 can be rewritten as

$$a(n) - a(n+1) = \frac{\mu}{\lambda}[a(n-1) - a(n)], \quad n \geq 1 \tag{5.31}$$

We obtain

$$a(n) - a(n+1) = \frac{\mu^n}{\lambda^n}[a(0) - a(1)] \tag{5.32}$$

Hence,

$$\begin{aligned} a(n+1) &= a(n+1) - a(0) + a(0) \\ &= \sum_{j=1}^{n} [a(j+1) - a(j)] + a(0) \\ &= [a(1) - 1] \sum_{j=0}^{n} \left(\frac{\mu}{\lambda}\right)^j + 1 \end{aligned} \tag{5.33}$$

where the $j = 0$ term of the sum equals 1 by convention. We can find a non-trivial solution if the sum converges, i.e. the system is transient if and only if

$$\sum_{j=0}^{\infty} \left(\frac{\mu}{\lambda}\right)^j < \infty \tag{5.34}$$

The sum converges if and only if $\mu < \lambda$. For the case of $\mu \geq \lambda$, the system is recurrent with recurrent state 0, and the data will disappear quickly (linear with n). Thus we only consider the case $\lambda > \mu$.

We note that in this case the mean absorption time is infinite since the Markov chain is transient. So a better metric is the probability that the number of packets in the queue will reach zero, given that there are n packets initially.

Since $a(n)$ is probability, we look for a solution of the form $a(n) = \theta^n$, where $0 \leq \theta \leq 1$. Substitute this into Eq.5.30 we have:

$$\theta^n = q\theta^{n-1} + p\theta^{n+1} \tag{5.35}$$

where

$$q = \frac{\mu}{\lambda + \mu}$$

and

$$p = \frac{\lambda}{\lambda + \mu}$$

Cancel out the power θ^{n-1}, the Eq.5.35 now becomes

$$\theta = q + p\theta^2$$

which has 2 roots

$$\theta_1 = 1$$

and

$$\theta_2 = \frac{q}{p} = \frac{\mu}{\lambda} < 1$$

Thus

$$a(n) = c_1\theta_1^n + c_2\theta_2^n = c_1 + c_2\theta_2^n = c_1 + c_2\left(\frac{\mu}{\lambda}\right)^n \tag{5.36}$$

We can find the value of c_1 and c_2 based on two initial conditions:

$$1 = a(0) = c_1 + c_2$$

and

$$0 = a(N) = c_1 + c_2\left(\frac{\mu}{\lambda}\right)^N$$

where N is very large. Then,

$$c_1 = -\frac{(\frac{\mu}{\lambda})^N}{1-(\frac{\mu}{\lambda})^N} \approx 0$$

$$c_2 = \frac{1}{1-(\frac{\mu}{\lambda})^N} \approx 1$$

Therefore,

$$a(n) = \left(\frac{\mu}{\lambda}\right)^n \tag{5.37}$$

Note that this probability is exponentially small with n, the initial number of peers.

Now, consider a system of r queues, each queue contains n_k pieces of data type k, where $N = \sum_{k=1}^{r} n_k$ is the total number of data pieces stored in the system. Since all the queues are independent, the probability that a piece of data is not recoverable is:

$$P = \sum_{i=1}^{r} \binom{r}{i} \prod_{k=1}^{i} a(n_k) = \sum_{i=1}^{r} \binom{r}{i} \prod_{k=1}^{i} \left(\frac{\mu_k}{\lambda_k}\right)^{n_k} \tag{5.38}$$

5.8 Concluding Remarks

In conclusion, we suggest that, to maintain data for as long as possible in a distributed setting with limited peer communication and storage, it is better to mix the data as proposed in the RLNC strategy. We show that the average number of replenishments before a file is no longer recoverable is exponential in the number of peers used store the data distributedly for RLNC-based strategy and quadratic for other traditional strategies. We also propose a novel time-backward technique to approximate the mean absorption time. We believe this technique is general as such, it can be applied to other problems such as gene population in which, it is intractable to directly model an exponentially large number of states using Markov chain representations. For example, one simple direct application of our time-backward technique is to model the diversity of a gene pool after several generations of inbreeding and without mutation. Taking the view of this chapter, each gene can be considered as a codeword. One would expect that after many generations of inbreeding, the gene population would then become homogeneous.

Equivalently viewed, many codewords are linear combinations of a few codewords. One can also extend the model to include other factors, e.g., making the population a random variable and using other distributions other than the uniform distribution for selecting the departed nodes.

5.9 Appendix: Alternative Proof of Exponential Mean Absorption Time for RLNC-based Replenishment

Denote h_k as the mean absorption time starting in the state $X_0 = k$, for $k = 3, 4, \ldots, N-2$, and ending in state $X_n = 2$ for some n. Then, we can write down the following recursion:

$$\begin{aligned} h_k = 1 \quad &+ \quad h_{k-1}\frac{k}{N}\left(\frac{k-1}{N-1}\right)\left(\frac{k-2}{N-2}\right) + h_{k+1}\frac{k}{N}\left(\frac{N-k}{N-1}\right)\left(\frac{N-1-k}{N-2}\right) \\ &+ \quad h_k\left[1 - \frac{k}{N}\left(\frac{k-1}{N-1}\right)\left(\frac{k-2}{N-2}\right) - \frac{k}{N}\left(\frac{N-k}{N-1}\right)\left(\frac{N-1-k}{N-2}\right)\right] \end{aligned}$$

Using continuous approximation of discrete random variable k, we have

$$\begin{aligned} h(x) = 1 \quad &+ \quad h(x-1)p(x) + h(x+1)q(x) \\ &+ \quad h(x)\left(1 - p(x) - q(x)\right) \end{aligned} \tag{5.39}$$

for $3 \leq x \leq N-2$ and

$$\begin{aligned} p(x) &= \frac{x}{N}\left(\frac{x-1}{N-1}\right)\left(\frac{x-2}{N-2}\right) \\ q(x) &= \frac{x}{N}\left(\frac{x-N}{N-1}\right)\left(\frac{x-N+1}{N-2}\right) \end{aligned}$$

Then,

$$h'(x) \approx \Delta h(x) = h(x) - h(x-1) \tag{5.40}$$

and

$$\begin{aligned} h''(x) &\approx \Delta^2 h(x) = \Delta h(x+1) - \Delta h(x) && (5.41) \\ &= h(x+1) - h(x) - (h(x) - h(x-1)) \\ &= h(x+1) - h(x) - h'(x) && (5.42) \end{aligned}$$

Thus, Eq.(5.39) can be rewritten as

$$\begin{aligned} 0 &= 1 - p(x)\left[h(x) - h(x-1)\right] + q(x)\left[h(x+1) - h(x)\right] \\ &\approx 1 - p(x)h'(x) + q(x)\left[h''(x) + h'(x)\right] \\ &\Rightarrow \; q(x)h''(x) + \left[q(x) - p(x)\right]h'(x) = -1. \end{aligned} \tag{5.43}$$

This is a second order non-homogeneous differential equation:

$$h''(x) + f(x)h'(x) = g(x) \tag{5.44}$$

where

$$\begin{aligned} f(x) &= \frac{q(x)-p(x)}{q(x)} = \frac{-(N-2)(2x-N-1)}{(x-N)(x-N+1)} \\ &= \frac{(N-2)(N-3)}{x-N+1} - \frac{(N-1)(N-2)}{x-N} \end{aligned}$$

and

$$\begin{aligned} g(x) &= -\frac{1}{q(x)} = -\frac{N(N-1)(N-2)}{x(x-N)(x-N+1)} \\ &= \frac{N(N-2)}{x-N+1} - \frac{(N-1)(N-2)}{x-N} - \frac{N-2}{x} \end{aligned}$$

Let $y(x) = h'(x)$, Eq. (5.44) becomes the first order non-homogeneous differential equation:

$$y'(x) + f(x)y(x) = g(x) \tag{5.45}$$

With following solution:

$$y(x) = \frac{1}{\mu(x)} \left[\int^x \mu(t)g(t)dt + c \right] \tag{5.46}$$

where

$$\mu(x) = e^{\int^x f(t)dt}$$

Since

$$\begin{aligned}\int^x f(t)dt &= \int^x \left(\frac{(N-2)(N-3)}{t-N+1} - \frac{(N-1)(N-2)}{t-N}\right) dt \\ &= (N-2)(N-3)\log(x-N+1) - (N-1)(N-2)\log(x-N) \\ &= \log \frac{(x-N+1)^{(N-2)(N-3)}}{(x-N)^{(N-1)(N-2)}}\end{aligned}$$

Then

$$\begin{aligned}\mu(x) &= \frac{(x-N+1)^{(N-2)(N-3)}}{(x-N)^{(N-1)(N-2)}} \\ &\approx (x-N)^{-2(N-2)} \end{aligned} \tag{5.47}$$

Substitute into Eq. (5.46), we have

$$\begin{aligned}y(x) &= (x-N)^{2(N-2)} \times \left(\int^x -(t-N)^{-2(N-2)} \frac{N(N-1)(N-2)}{t(t-N)(t-N+1)} dt + c\right) \\ &\approx c(x-N)^{2(N-2)} - (x-N)^{2(N-2)} N(N-1)(N-2)\left(\int^x (t-N)^{-2N+1} dt\right) \\ &\approx c(x-N)^{2(N-2)} + (x-N)^{-2}\frac{N(N-2)}{2} \\ &\approx (x-N)^{2(N-2)} + \frac{N(N-2)}{2(x-N)^2}\end{aligned} \tag{5.48}$$

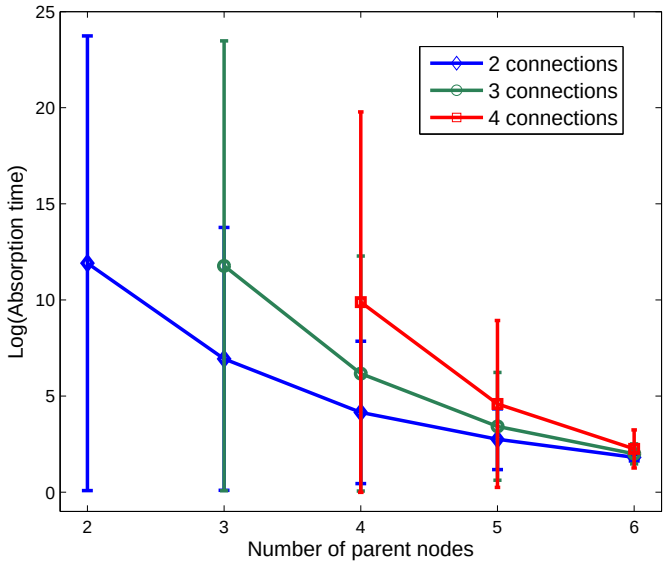
Finally, the explicit solution of Eq. (5.39) is

$$\begin{aligned} h(x) &= \int^x y(t)dt + c \\ &\approx \int^x \left((t-N)^{2(N-2)} + \frac{N(N-2)}{2(t-N)^2} \right) dt + c \\ &\approx (x-N)^{2N-2} - \frac{N(N-2)}{2(x-N)} \end{aligned} \tag{5.49}$$

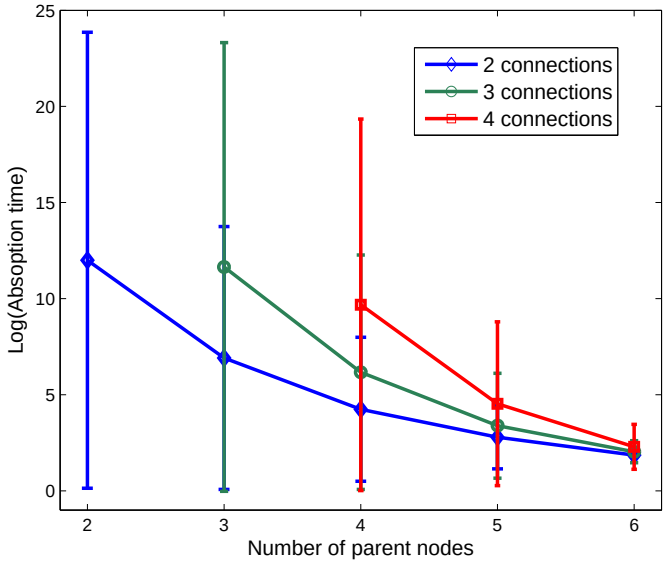
Thus, the mean absorbed time if one starts from state $x = N-2$ is

$$h(N-2) \approx 2^{2N-2} + \frac{N(N-2)}{4} \tag{5.50}$$

which is exponential in N.



(a)



(b)

Figure 5.7: Log of mean absorption time and its standard deviation vs number of parent nodes for $N = 8; M = 2, 3, 4$ for the RLNC strategy with (a) Time-forward model and (b) Time-backward model.

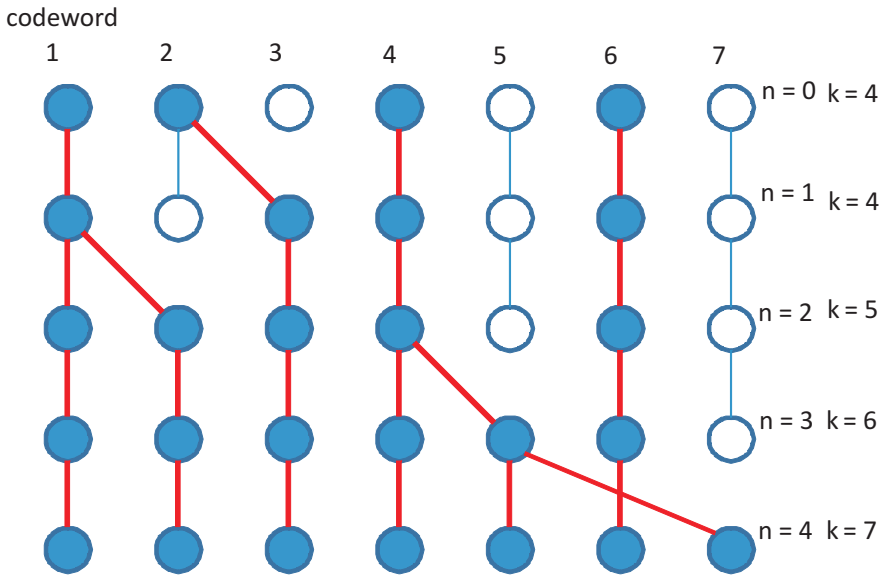


Figure 5.8: Progression of codewords for seven peers, $N = 7$, $M = 2$ in discrete time step n. At each time step, a codeword is replaced by another.

Chapter 6 – RESnc: A Resilient P2P Streaming with Network Coding

1. Kien Nguyen, Thinh Nguyen, and Yevgeniy Kovchegov, "*A P2P Video Delivery Network (P2P-VDN)*", The IEEE Workshop on Grid and P2P Systems and Applications (GridPeer'09), Aug. 2009, San Francisco, CA, USA.

2. Kien Nguyen, Thinh Nguyen, Yevgeniy Kovchegov, and Ian Milligan, "*Resilient P2P Video Storage and Streaming with Network Coding*", submitted to the Journal of Parallel and Distributed Computing, 2010.

6.1 Introduction

Multimedia streaming over the Internet is challenging due to packet loss, delay, and bandwidth fluctuation. Thus, many solutions have been proposed, ranging from source and channel coding to network protocols and architecture. In which, multi-sender streaming paradigm has been proposed to provide smooth video delivery [63, 8]. The main idea is to have each server storing an identical copy of the video. The video is partitioned into multiple disjoint parts, each part is then streamed from separated server to a single receiver simultaneously. Having multiple senders is in essence a diversification scheme in that it combats unpredictability of congestion in the Internet. However, these schemes all rely upon a dedicated server and not storage efficient.

The recent development of Peer-to-Peer (P2P) networks opens a new possibility of building completely distributed systems that have the potential to eliminate the computational and bandwidth bottlenecks existing in the traditional client-server architecture. To that end, a fair amount of distributed system research has recently been focused on using P2P platforms to build reliable, large scale distributed systems for Internet services [73] [85]. In these systems, data is stored distributedly at different peers, which provides robustness against the single point of failure. To watch a video, a client requests simultaneous transmissions from multiple peers that collectively have the complete requested video. This approach creates multiple Internet routes for transmitting a video to a client, resulting in larger throughput.

To build such a system, one must address many important issues associated with a decentralized overlay network such as communication, indexing, and searching. Notably, in Chord, Stoica *et al.* employ a Distributed Hash Table (DHT) structure in order to provide large scale Internet services, e.g., DNS in a scalable manner. In this chapter, we describe a peer-to-peer video-on-demand system (RESnc) for providing video streaming services over the Internet using Chord as its indexing architecture. The proposed RESnc system has the potential to achieve both high scalability and performance based on three key designs.

First, in our proposed RESnc system, a video is broken into multiple pieces, coded properly, then dispersed to a number of peers in the network. This is in contrast with the current video storage and streaming systems such as YouTube or Akamai whose individual videos are stored in their entireties at a video server, or at multiple servers if video replication is employed. As will be described subsequently, using a proper indexing architecture such as Chord, a video publisher will be able to publish his video, i.e. to disperse his coded video pieces to the RESnc network. This is done in such a way to allow a client to efficiently locate the pieces of a previously published video, and use them to recover the original video. From a user perspective, this is much like uploading one's video to one of the YouTube server. However, unknown to the uploader, his video will be broken up into parts, coded properly, and dispersed to multiple peers in the RESnc. And unknown to a client, his requested video is being retrieved not from a single server, but from multiple peers.

Second, in the RESnc system, videos are stored at peers who, unlike servers,

are unreliable due to their frequent departures and failures. As a result, video availability is a major concern. Therefore, the RESnc employs a data replenishment mechanism to either proactively or passively fill in the missing data due to peer departures or failures. It does so by recruiting the incoming peers to take part in sharing the burden of storage and streaming resources. Specifically, data replenishment mechanism is designed to automatically maintain a constant level of redundancy for videos in the network over time, by allowing incoming peers to store appropriate *generated* data. This compensates for the data loss due to a departing peer. Its aim is to increase the probability that, at any moment, peers in the network, collectively store a sufficient number of coded pieces of a particular video that allows a perfect reconstruction of that video. Furthermore, the data replenishment is designed to be distributed, scalable, and does not require the presence of the original videos. Peers take part in the data replenishment process in a random and independent manner, but yet collectively, the video in the network is robust against peer dynamics. As will be discussed subsequently, the key to such a design is to employ Random Network Coding (RNC).

Third, to watch a video, a client requests simultaneous transmissions from multiple peers in the RESnc network that collectively have all the pieces to reconstruct the requested video. This approach creates multiple Internet routes for transmitting a video to a client, resulting in larger throughput and higher video quality. To accomplish this, we describe a path-diversity streaming protocol using network coding technique that reduces the complexity of sender synchronization while enabling TCP streaming.

Those three key designs are three main operational phases in our proposed system: data dispersion, data replenishment and path-diversity streaming protocol, respectively. In this chapter, we present analysis and implementation of RNC library and show the feasibility of using RNC in P2P video streaming applications. We also create a application framework to test those three designs in the real world scenario. We focus on throughput, initial buffering delay, playback quality and downloading bandwidth.

6.2 Background and Related Work

Since the emergence of network coding in information theory [4], it has attracted a substantial amount of attention in networking research. In [35], Ho *et. al.* shown that the maximum min-cut bound of information flow rates in a multicast session can be achieved with network coding. It has also been shown that network coding helps achieve provably good overall performance in P2P networks. In [29] Gkantsidis *et. al.* have shown how to use network coding to improve download time by a 30% in peer-to-peer file content distribution. In this context, network coding helps to eliminate the need for finding rare data blocks. In [90], Wang *et. al.* have shown that network coding is beneficial in P2P video streaming, especially when server bandwidth supplies barely meet user demand.

The main advantage of the current P2P video streaming systems is the bandwidth saving for a live video broadcaster. In a broadcast session, there are multiple peers (participants) who desire to watch the same video. Therefore, the broad-

caster may need only to send a single video stream to one or a few peers who then relay the same video stream to other peers. The process repeats itself until every peer receives the video stream. Effectively, the P2P streaming technique above only requires one or few video streams from the broadcaster, rather than N streams for N users as required in the client-server model. Thus, P2P streaming eliminates bandwidth and computational bottlenecks.

On the other hand, for video-on-demand applications, it is likely that at any moment, there are only one or a few users who want to watch the same video. In this case, the bandwidth advantage of P2P systems seems to be significantly diminished. For this reason, the video-on-demand market has been dominated by Content Delivery Networks (CDN). Because of the on-demand characteristic, many more videos must be stored on CDN servers, in order to satisfy a potentially large number of different client's requests at any moment. Therefore, from both network bandwidth and storage perspectives, using a single video streaming server is not possible. As such, many CDNs such as Akamai use multiple servers to optimize the streaming performance for a large number of users at different locations in the Internet. Specifically, Akamai servers are strategically placed at the edge of the Internet such that the nearest server to a client is chosen for streaming, thus improving the client's viewing experience.

That said, an Akamai-like approach might not be an optimal approach since the overall storage amount, bandwidth capacity, and computational capability still scale linearly with the number of clients. We advocate a new P2P approach to designing video-on-demand systems in which participated peers contribute not only

bandwidth but also storage resources. However, by nature, peers join and leave the network frequently. As a result, videos might be lost temporarily or permanently. Thus, there is a need to develop techniques and policies for providing sufficient redundancy to ensure that, when a video is requested, it is available in the network.

A simple strategy is to replicate a video at some number of peers. This approach, however, is storage inefficient. A more effective approach is to use FEC. A FEC code with rate n/k ensures that if k or more distinct packets are received out of a FEC block of n packets, then the original k data packets can be recovered [92]. Thus, for every k packets belonging to a video, one can generate additional $n - k$ parity packets, and distribute all n packets to a number of peers in the network. For example, using Reed Solomon code $RS(8, 4)$, one may distribute 1 packet to each of the 8 peers. As long as 4 of these peers are in operation, a client will be able to recover the entire data block.

However, this approach is suboptimal since it only works for a given value of n and k. In a very unstable P2P network, it is hard to design an optimal FEC scheme. Consider the same example above for a very unstable network. As such, we want to distribute those packets to 16 peers rather than just 8 peers. The problem is, for each FEC block, we have only 8 data packets, thus it is necessary that some peers must share identical packets. Because of this, if at some given time, only four peers are in the network, they might collectively have fewer than 4 *distinct* packets. As a result, a client cannot recover the original block from these peers.

The work by Acendanski *et al.* [3] and Nguyen *et al.* [59] [58] use the Random

Network Coding (RNC) technique to overcome this problem. Using RNC, a coded packet is a random linear combination N original packets. Each peer then can keep a fraction of the total number of coded packets. If at any given time, the peers in the network collectively have a set of coded packets that form a set of N linearly independent equations, then a client connecting to these peers will be able to recover N original packets. Using the random coefficients from a large finite field, one can generate "infinite" number of coded packets in which the probability of getting linearly independent packets is close to 1. This effectively eliminates the duplicate packets problem posed by using RS code.

In a real-world scenario, without any active injection of redundancy into the network content will eventually disappear since peers will depart or the file will be deleted with some non-zero probability. Dimakis *et al.* [26] [24] proposed a way of restoring the missing redundancy from the remaining peers. Specifically, their work show that, using RNC for storage, when a new peer joins the network, it can download minimal amount of data from other peers to restore the missing redundancy. We will describe a similar data replenishment process in our proposed system.

6.3 Design Objectives

The recent development of P2P networks and network coding theory [4] provide us the powerful tools to design a distributed system for media storage and streaming over the Internet, that has the potential to eliminate the computational and

bandwidth bottlenecks. Network coding is a novel mechanism that promises optimal utilization of the resources of a network topology. The most elegant result of network coding is that the maximum multicast capacity is achievable using some *random network coding* [35] techniques, while this is not usually possible with the traditional store and forward routing. Recently, we proposed and showed that network coding also helps to improve the quality of media streaming in P2P network [59].

In the process of designing the RESnc system, we seek to achieve the following design objectives.

6.3.1 Server bandwidth

Dedicated server bandwidth is finite and expensive resource, even with the explosive growth of Internet infrastructure. For example, the bandwidth costs of YouTube is estimated at 1 million dollar a day. There is a need to find a method of scaling limited server bandwidth in order to serve as many users as possible. Using a peer-assisted streaming can substantially reduce server bandwidth use. We design the *data dispersion* mechanism to code and distribute data to the intermediate peers with minimum storage redundancy and bandwidth usage. It helps conserve server bandwidth by maximizing peer bandwidth contributions to serve other peers. Together with the redundancy introduced by *data dispersion*, the *data replenishment* process is designed to counter the data deletion due to peer dynamics. The server/publisher only needs to inject data sporadically to conserve its

bandwidth. We will discuss more about the *data replenishment* process in Section 6.4.

6.3.2 Robustness

To provide a on-demand streaming service, one has to ensure that when content is requested it is available on the network. In RESnc, media data is stored distributedly on peers which join and leave the network frequently. When a peer leaves the network, so does its data. This effectively reduces the robustness of the system temporarily or permanently if the peer never rejoins or rejoins without its data. RESnc employs a *data replenishment* process to increase data resiliency again peer departures and failures with minimal access to the media server.

6.3.3 Initial buffering delay

In on-demand video streaming systems, one of the most important performance metrics is the initial buffering delay, which is the time a user would have to wait before being able to playback the requested video. By fine tuning the network coding schemes and redundancy level, we are able to find the minimum point of the average initial buffering delays for the given network.

6.3.4 Playback quality

In streaming systems, one needs to maintain a consistently satisfactory level of user playback quality by making sure that the average download rate for a video stream is higher than its playback rate. This requires a complicated scheduling mechanism in multi-sender streaming. With help of network coding, we design a *path diversity streaming* protocol which requires minimal or no coordination among senders. The receiver can connect to arbitrary number of sender peers to get a video stream. This protocol utilizes all available bandwidth of all senders. With enough redundancy, it provides consistent playback quality if the total available bandwidth of all sender-to-receiver connections is greater than the playback rate. It is oblivious to senders' qualities. Multiple senders serving one receiving peer can collaborate with ease, and there is no need to design an elaborate protocol to select some high quality senders. As long as a sender has the requested chunk in its cache, its upload capacity can be utilized to serve that chunk.

To get a chunk, the receiving peer simply signals the senders and waits until it has enough data to decode that chunk. Using Gauss-Jordan elimination, the receiving peer can even progressively recover original blocks so that the entire chunk is immediately playable after the last block is received from any of the senders. This is due to the fact that randomly generated coefficient vectors from different senders are linearly independent with high probability. One can generate "infinite" number of coded blocks then disperse to a number of senders. As long as n linearly independent coded blocks are received, they are sufficient to recover

the original chunk.

We will describe in detail on our design to achieve all the above objectives in the next section.

6.4 The RESnc Architecture

In this section, we describe the overall architecture of the proposed RESnc system to be used as a platform for video-on-demand applications. As mentioned in the Introduction, the RESnc system is built on three key designs: data dispersion, data replenishment, and path-diversity streaming protocol. These three designs correspond to the three main operational phases in the RESnc system. The data dispersion is used when a user publishes his video to the RESnc network, similar to the video uploading done by a YouTube user. The difference is that the video is broken into multiple pieces, coded and dispersed to many peers, rather than being stored in its entirety at a single YouTube server. The data replenishment is a continual data maintenance process that ensures high probability of a video being recoverable by a client when it is requested. This data replenishment is needed for RESnc since videos are stored at unreliable peers which enter and depart the network frequently. On the other hand, no such mechanism is necessary for YouTube since by assumption, its servers are reliable. The path diversity streaming protocol is used when a client requests a video from the RESnc. The client might or might not belong to the RESnc network. In this case, multiple peers in the RESnc network will cooperate among each other via the path diversity streaming

protocol, to simultaneously stream the requested video to the client. For YouTube, no special protocol is needed. Rather, TCP is sufficient to stream a video from a single YouTube server to the client. We now describe and motivate each of the three key designs.

6.4.1 Data Dispersion

In addition to employing participating peers rather than dedicated servers, a key difference between RESnc and existing CDNs is found in the way in which videos are stored. Akamai, for example, replicates videos in their entireties at multiple edge servers. This allows for proximity optimization, i.e., a nearest server to the client with the requested video in term of round trip time, will be chosen to stream that video to the client. Thus, it is necessary that the videos are replicated sufficiently at different servers. However, as discussed in Section 6.2, replicating videos is not a storage efficient method. Therefore, in RESnc, a video is broken up into multiple pieces. These pieces are coded using the RNC technique. The coded pieces are then dispersed randomly to multiple peers. We emphasize that data dispersion is good not only because it avoids the central point of failure, but also results in better load balancing.

Using the RNC technique, a video publisher first breaks a video stream into a number of chunks. Each chunk is further broken up into a number of packets. Each packet $\mathbf{p_j}$ can be viewed as a vector of elements belonging to a finite field. For example, if $GF(2^{16})$ is used, then each element of the finite field can be represented

by an 16-bit pattern. Therefore, a packet of length $16L$ bits can be viewed as a vector of L elements from $GF(2^{16})$. A coded packet or equivalently, a vector of finite elements, $\mathbf{c_i}$ is produced by linearly combining k original packets as:

$$\mathbf{c_i} = \sum_{j=1}^{k} f_j \mathbf{p_j} \tag{6.1}$$

where f_j are the random elements belonging to a finite field F_q having q elements.

For every k original packets, the publisher generates $n > k$ coded packets. The value of n depends on the desired redundancy, the larger n, the more redundancy. The publisher also includes the information of $\{f_j\}$ in the header of each of coded packets. Therefore, one should note that if a client has access to any of the k encoded packets $\mathbf{c_i}$'s that form a set of k linearly independent equations, then it will be able to recover the k original packets by solving a set of linearly independent equations. We can show that if the finite field is sufficiently large, the probability of having linearly independent equations is essentially 1. Furthermore, if the packet size is large, the overhead in storing the extra bits in the packet header for of $\{f_j\}$ is negligible.

Now, after generating these coded packets, the publisher randomly sends these packets to a number of peers. It is important to note that each peer does not need to store all k coded packets. Rather, to save space a peer may store only a fraction of a coded video packets.

Next, in order for a client to be able to recover these pieces, we use a modified version of Chord for indexing these pieces. In particular, we use a mapping that

hashes a searchable video title into a value, e.g. 64-bit value. Using Chord, the peer whose hash of its IP address is closest to this value, is used to store the set of IP addresses of the peers that were randomly picked for storing the coded video packets. Effectively, the client would be able to determine which peers store the desired video by searching the DHT in $\log(N)$ where N is the number of participating peers. Since peers join and leave frequently, the DHT structure is updated accordingly as described in [85].

We note that for some period of time there may not be a sufficient number of peers with the requested video available in the network to enable the client to recover the video. Thus an automatic data replenishment technique for countering the effect of peer departures is described below.

6.4.2 Data Replenishment

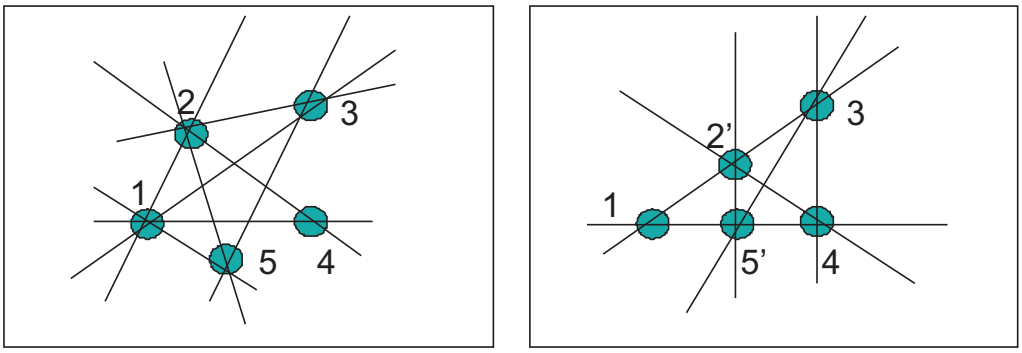
When a peer leaves the network, so does its data. This effectively reduces the robustness of the system temporarily or permanently if the peer never rejoins or rejoins without its data. To avoid this, a peer can transfer its data to some other peers before its departure. However, media data such as a video tends to be large, making a peer less willing to wait until the transfer completes. This process implies that, without any proactive data replenishment, the redundancy level of a video in the network is continuously reduced. At some point, the video is not likely to be recoverable.

Theoretically, if one can replace the exact missing data in the network, the

redundancy level would remain the same. However, a typical peer may not have the complete video that allows it to reproduce an arbitrary missing portion, nor does it know what the leaving peer has. Instead, we propose the following data replenishment scheme. Assume that the network knows the peer departure or failure rate. This can be estimated empirically, e.g., using some kind of periodic sampling. Thus, we can determine an appropriate replenishment rate to counter the information deletion rate. In this chapter, we do not focus on the replenishment rate. Rather, we describe how replenishment is performed.

Assume that when a peer leaves the network there is on average at least one peer who joins the network or there is at least one existing peer with spare storage capacity. This allows for another peer to take over the storage responsibility of the departed peer, to maintain the same redundancy of a piece of data. The new peer will randomly connect to a number of peers and download some fraction of their data. It then generates its new packets as some random linear combinations of the packets it obtains from other peers, in an attempt to maintain the same level of redundancy for the video. This method is attractive for its distributed nature and scalability. Also, the original data is not needed to be present in the network. Given such data replenishment scheme, we have to analyze the data robustness in a network as its evolves. In other words, given piece of data, what is the probability of being able to recover this piece of data as a function of the number of replenishments?

Clearly, the replenished data is linearly dependent on the data that are used to generate it. So if the number of packets used to generate a new packet is fewer



(a) (b)

Figure 6.1: a) Original robust state of the network; (b) Less robust state after peer departures and data replenishments.

than the number of packets in the original piece of the data and replenishments are repeated over time, then there may be a non-negligible chance that all peers contain relatively few independent data, making the network less robust.

To illustrate this, consider using RNC on a video and dispersing the coded packets to five peers, each peer stores one coded packet. Assuming that a client is able to recover the video if it obtains three or more independent packets. Furthermore, assuming that every time a peer leaves, two randomly chosen peers will send their packets to the replacement peer. The replacement peer then randomly combines the two packets to generate its own packet. Fig. 6.1(a) shows the geometric representation of the original robust state of the network where the circles represent packets. If there are three or more points lay on one line, they are co-linear. In other work, there are only two independent points on any line. In this state, if a client connects to any three peers, it will be able to recover the video, or geometrically, recover the entire plane. It is possible because none of the 3

points are co-linear. Now, suppose peers 2 and 5 depart the network. By chance, peers 1 and 3 are chosen to replenish data for peer 2, while 1 and 4 are chosen to replenish data for peer 5. As a result, the generated data are now 2' and 5'. 2' are linearly dependent on 1 and 3 while 5' are linearly dependent on 1 and 4 as shown in Fig.6.1(b). Now, if a client connects to peers 1, 2', and 3, it will not be able to recover the video since these points are co-linear. A similar situation arises when it connects to 1, 5', and 4. Effectively, the robustness on data recoverability in the network has been reduced. Eventually, this piece of data become unavailable when all the points lay on the same line.

This robustness, i.e., the recoverable probability of a video depends on how many peers are chosen to replenish the data and the amount of data each peer has. Figure 6.2 shows the simulation results on the recoverable probability as the network evolves under repeated replenishments. This simulation uses two peers for replenishment and the given parameters (a, b, c) is (total number of network coded packets, number of independent packets needed to recover a video, number of packets stored at each peers). As shown, the higher number of peers, the longer it takes before a file cannot be recovered. It is interesting to note that the robustness decreases quickly if the initial data redundancy is not sufficient.

One can use a discrete time Markov chain representation, specifically a transition probability matrix can be used to describe this replenishment process [61]. We will show the detail derivation in the next section.

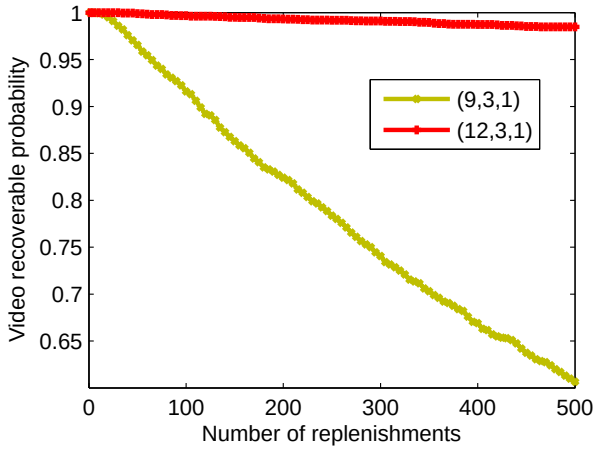


Figure 6.2: Data recoverable probability as a function of replenishments.

6.4.2.1 A Markov Model for Data Replenishment

Using RNC, each packet is represented by a vector $\mathbf{p} = (p_1, p_2, \ldots, p_j)$ where $p_i \in GF(2^K)$. A piece of data is split into l packets $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_l$. Using RNC, a network coded packet $\mathbf{c}_i$ is generated by randomly and linearly combining $\mathbf{p}_i$'s, i.e., $\mathbf{c}_i = \sum_{j=1}^{l} a_j \mathbf{p}_j$ where a_j is randomly chosen from $GF(2^K)$. n network coded packets $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_n$ are generated, and each network coded packet is dispersed to exactly one peer. Using a large finite field $GF(2^K)$, it can be shown that the probability that l randomly chosen network coded packets (vectors) being mutually independent, is approximately 1. Thus, if $n > l$, one can recover all the original packets $\mathbf{p}_i$'s using any l network coded packets from any l peers.

Recall that every time a peer leaves, a new peer will connect to r randomly chosen remaining peers, download their packets, and generate its packet using

RNC. Clearly if $r = l$, then the probability that the newly generated network coded packet is mutually independent with any of the l linear combinations from the rest of the packets in the network is approximately 1. And thus, the network will likely contain sufficient linearly independent packets that allows a client connecting to the network, to recover all the original packets. On the other hand if $r < l$, there is a non-negligible chance that after some number of replenishments, any packet (vector) in the network can be spanned by only r or fewer vectors. Thus, a client connecting to the network will not be able to recover the original data. We are interested in the data replenishment using small values of r, specifically $r < l$, since we want to reduce the amount of data sent to the new peer during the replenishment. We also assume that whenever a peer leaves the network, the probability of any peer being the departure peer is $p = 1/n$. That said, at any point in time, the data robustness of a network depends on the interdependency level of the packets in the network. So we model the network state using a Markov chain as follows.

Initially, there are n network coded packets: $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_n$. At any time later, any network coded packet resulted from replenishments must be a linear combination of r or more of these original network coded packets. Since there are n possible original network coded packets, a network coded packet must be in one of the $\binom{n}{r}$ combinations. Furthermore, we make an assumption that a network coded packet can belong to exactly one of these combinations. This can be justified when using a large finite field and long packets. In general, for any r, a network coded packet must be in one of the $\sum_{i=1}^{n} \binom{n}{i} = 2^n - 1$ combinations. Thus a network

state can be represented by the following $(2^n - 1)$-tuple.

$$S = \{|A_1|, |A_2|, \ldots, |A_{2^n-1}|\}$$

where A_i represents the set containing all the network coded packets that are from a particular linear combination. Because there are n packets in the network at any time, $\sum_{i=1}^{2^n-1} |A_i| = n$. One convenient way to map A_i to a particular linear combination is using the binary expansion of $i = (b_1, b_2, \ldots, b_n)$ where $b_i = 1$ implies that $\mathbf{c}_i$ is present in the linear combination, $b_i = 0$ otherwise.

Having defined the state, we now show how to compute the transition probabilities. At time t, the network state

$$S_t = \{|A_1|, |A_2|, ..., |A_{2^n-1}|\}$$

Suppose a packet $\mathbf{d}_j \in A_j$ is deleted due to a peer departure. Then, the temporary state is

$$S_{temp} = \{|A_1|, |A_2|, ..., |A_j| - 1, ..., |A_{2^n-1}|\}$$

Now suppose r packets $[\mathbf{d}_{s_1}, ..., \mathbf{d}_{s_r}]$ are randomly chosen to generate the replacement for lost packet $\mathbf{d}_j$. Assuming k_i of r chosen packets are in A_{s_i}. Then there are $\binom{|A_{s_i}|}{k_i}$ possible ways of choosing k_i packets from A_{s_i}. Therefore, the probability of choosing this r packets from different A_{s_i} is

$$\prod_{i=1}^{r} \frac{\binom{|A_{s_i}|}{k_i}}{n-i}$$

For a given set of r packets $[\mathbf{d}_{s_1}, ..., \mathbf{d}_{s_r}]$, there are $r!$ ways of choosing r packets by different orders. Thus, the total probability of choosing this r packets is

$$r! \prod_{i=1}^{r} \frac{\binom{|A_{s_i}|}{k_i}}{n-i}$$

The new packet will be in the set A_m where $m = \biguplus_{i=1}^{r} s_i$ and $\biguplus$ is a bitwise OR operator. That is, we expand s_i in binary form then perform bitwise OR. With the above chosen set of r packets $[\mathbf{d}_{s_1}, ..., \mathbf{d}_{s_r}]$, the network state will move to state

$$S_{t+1} = \{|A_1|, |A_2|, ..., |A_j| - 1, ..., |A_m| + 1, ..., |A_{2^n-1}|\}$$

with probability

$$p(j,m) = \frac{|A_j|}{n} r! \prod_{i=1}^{r} \frac{\binom{|A_{s_i}|}{k_i}}{n-i} = \frac{|A_j|}{n} \frac{\prod_{i=1}^{r} \binom{|A_{s_i}|}{k_i}}{\binom{n-1}{r}} \tag{6.2}$$

However, there are many ways to choose the set of r packets $[\mathbf{d}_{s_1}, ..., \mathbf{d}_{s_r}]$ that all lead to the state S_{t+1}, i.e.

$$\biguplus_{i=1}^{r} s_i = m \tag{6.3}$$

We define a set T_m that contains all the possible selections of r packets that satisfy equation 6.3. Then, the transition probability from state S_t to state S_{t+1} is:

$$P(j,m) = \sum_{s=1}^{|T_m|} p(j,m) \tag{6.4}$$

Theoretically, with this model, one can estimate the robustness of the network over time by examining the hitting times of a particular state using standard recurrence equations. However, this model is analytically intractable due to a large number of states. Interestingly, the time-backward model proposed in [61] can help to produce an approximate closed form solution for the expected number of time steps to get from any state to any other, including the state in which the file is no longer recoverable. In the next section, we show a simpler analysis on the performance of a simple RNC-based replenishment versus repetition code strategy.

6.4.2.2 Repetition code

Suppose a video is split into two input vectors (packets) $\mathbf{p}_1$ and $\mathbf{p}_2$, and there are n peers, each containing either $\mathbf{p}_1$ or $\mathbf{p}_2$. Whenever a peer leaves, the new peer randomly picks one out of $n-1$ existing peers, and the input vector (either $\mathbf{p}_1$ or $\mathbf{p}_2$) is copied to the new peer. The process is of birth-and-death type on $\{0, 1, \ldots, n\}$ with two absorbing states, 0 and n. We would like to estimate the mean absorption time.

In the above birth-and-death process the forward probabilities are given by

$$p_k = \frac{k(n-k)}{n(n-1)} \quad \text{for } k = 1, 2, \ldots n, \quad \text{and } p_0 = 0$$

and the backward probabilities are

$$q_k = \frac{k(n-k)}{n(n-1)} \quad \text{for } k = 0, 1, \ldots n-1, \quad \text{and } q_n = 0 \ .$$

The expected hitting time $h(k) = E_k[T_0 \wedge T_n]$ can be found by solving the following recurrence equation:

For $k = 1, 2, \ldots, n-1$,

$$\begin{aligned} h(k) &= 1 + \frac{k(n-k)}{n(n-1)} h(k-1) + \frac{k(n-k)}{n(n-1)} h(k+1) + \left(1 - \frac{2k(n-k)}{n(n-1)}\right) h(k) \\ h(0) &= h(n) = 0 \end{aligned} \tag{6.5}$$

The above equation can be rewritten as follows

$$y_k = -(n-1)\left(\frac{1}{k} + \frac{1}{n-k}\right) + y_{k-1},$$

where $y_k = h(k+1) - h(k)$.

Thus

$$\begin{aligned} y_k &= -(n-1)\left(1 + \frac{1}{2} + \cdots + \frac{1}{k}\right) \\ &- (n-1)\left(\frac{1}{n-k} + \cdots + \frac{1}{n-1}\right) + y_0. \end{aligned} \tag{6.6}$$

Now, by symmetry,

$$-y_0 = y_{n-1} = -2(n-1)\left(1 + \frac{1}{2} + \cdots + \frac{1}{n-1}\right) + y_0$$

and therefore

$$y_0 = (n-1)\left(1 + \frac{1}{2} + \cdots + \frac{1}{n-1}\right). \tag{6.7}$$

Plugging y_0 into (6.6), and after some algebraic manipulations, we obtain:

$$\begin{aligned} h(k+1) &= (n-1)k\left(1 + \frac{1}{2} + \cdots + \frac{1}{n-1}\right) \\ &- (n-1)(k+1)\left(1 + \frac{1}{2} + \cdots + \frac{1}{k}\right) \\ &+ (n-1)(n-k-1)\left(\frac{1}{n-k} + \cdots + \frac{1}{n-1}\right). \end{aligned} \tag{6.8}$$

Taking $k+1 = \frac{n}{2}$, we obtain

$$h\left(\frac{n}{2}\right) \approx \ln 2 \cdot n^2 \tag{6.9}$$

6.4.2.3 Random linear network coding

A video is split into three input vectors $\mathbf{p}_1$, $\mathbf{p}_2$ and $\mathbf{p}_3$. Each peer contains a different linear combination of the three input vectors. Each time a peer is replaced, it picks two random peers and takes a random linear combination of the two vectors. The question is, how long will it take before the rank of $n \times 3$ matrix of coefficients falls below three?

We model the process by splitting the peers into cliques, each having rank two. There are originally $\begin{pmatrix} n \\ 2 \end{pmatrix}$ cliques. Observe that a clique can be absorbed by another clique only when it has two peers in it. It might reappear later. The

state space is therefore $\{absorbed,\ 2, 3, \ldots, n\}$. In fact a clique is a birth and death process on the above state space with the following forward and backward probabilities, given there are k peers in a clique:

$$p_k = \frac{n-k}{n} \cdot \frac{\begin{pmatrix} k \\ 2 \end{pmatrix}}{\begin{pmatrix} n-1 \\ 2 \end{pmatrix}} = \frac{k(n-k)(k-1)}{n(n-1)(n-2)} \tag{6.10}$$

and

$$q_k = \frac{k}{n} \cdot \left[1 - \frac{\begin{pmatrix} k-1 \\ 2 \end{pmatrix}}{\begin{pmatrix} n-1 \\ 2 \end{pmatrix}} \right] = \frac{k(n-k)(n+k-3)}{n(n-1)(n-2)} \tag{6.11}$$

Let $\phi(k)$ be the associated probability harmonic function:

$$\phi(k) = 1 + \sum_{m=2}^{k-1} \frac{q_2 \cdots q_m}{p_2 \cdots p_m} \quad k = 2, 3, \ldots, n \tag{6.12}$$

Plugging in, we obtain

$$\phi(k) = 1 + \sum_{m=2}^{k-1} \begin{pmatrix} n+m-3 \\ n-2 \end{pmatrix} \tag{6.13}$$

$$= \sum_{j=0}^{k-2} \begin{pmatrix} n-2+j \\ n-2 \end{pmatrix} \tag{6.14}$$

$$= \begin{pmatrix} n+k-3 \\ n-1 \end{pmatrix}$$

via basic combinatorics.

If we denote k_t is the size of a given clique at time t, then $\phi(k_t)$ is a martingale, and by the Stopping Theorem, the probability that the clique, starting with three peers expands to all n peers while never shrinking to two is

$$\frac{\phi(3) - \phi(2)}{\phi(n) - \phi(2)} = \frac{n-1}{\begin{pmatrix} 2n-3 \\ n-1 \end{pmatrix} - 1}$$

Thus, it will take an average of

$$\frac{\begin{pmatrix} 2n-3 \\ n-1 \end{pmatrix} - 1}{n-1}$$

trials for a clique to start growing, and reach the size of n before shrinking to 2.

Here, by Stirling's formula,

$$\frac{\begin{pmatrix} 2n-3 \\ n-1 \end{pmatrix} - 1}{n-1} \sim \frac{1}{\sqrt{\pi} n^{3/2}} 2^{2n}$$

Since there are no more than $\begin{pmatrix} n \\ 2 \end{pmatrix}$ cliques at one time, it will take an average of at least

$$\begin{pmatrix} n \\ 2 \end{pmatrix}^{-1} \cdot \frac{\begin{pmatrix} 2n-3 \\ n-1 \end{pmatrix} - 1}{n-1} \sim \frac{1}{\sqrt{\pi} n^{7/2}} 2^{2n-2} \tag{6.15}$$

replenishments before the video is no longer recoverable.

Thus, the proposed replenishment mechanism is theoretically much better than the naive random duplication technique in terms of maintaining data redundancy for recovery.

6.4.3 Path-diversity Streaming Protocol

In this section, we discuss how distributed storage using RNC can help to facilitate path diversity streaming. In a traditional video streaming application, a video is streamed from a server to a client. However, if the path between the server and the client experiences heavy congestion, the quality of the video can degrade significantly. To overcome congestion, many researchers have proposed the path-diversity

streaming technique in which the different parts of the video are simultaneously streamed from multiple servers to the client on multiple distinct routes [64]. With appropriate channel and source coding techniques and rate allocation among the servers, the video quality at the receiver can be improved significantly. In addition, video streaming using multiple servers also allows fine-grained load balancing to improve network performance. Unfortunately, current approaches to multi-sender video streaming requires a careful coordination between a client and server to achieve optimal performance. In particular, assuming that two servers are used for streaming, then server 1 can stream the odd packets while the other streams the even packets, starting from the beginning of the file. This approach works when there are only two servers, and that their average bandwidth are equal and constant throughout the streaming session. When there are many servers with different available bandwidth, and these bandwidths are varied with time (e.g. TCP is used for streaming), then obtaining the optimal packet partitions for each server requires a complex dynamic coordination between the client and the servers. Even when complex coordination is possible, the inaccurate estimation of available bandwidth is often not possible which results in sub-optimality.

We now describe the network coding scheme for multi-sender streaming framework that reduces the coordination among servers. In this scheme, a video stream F is randomly network coded and dispersed to a number of peers in the network. In this model, a stream is partitioned into N chunks. Each chunk is further divided into k small packets $\mathbf{p_1}, \mathbf{p_2}, ..., \mathbf{p_k}$. Now, for each chunk, the video publisher will randomly network code the packets within it, to produce a number of coded pack-

ets. These packets will be distributed to a number of peers at random as described in Section 6.4.1. Note that each peer does not need to keep all k coded packets. They may keep only a fraction of the coded packets, but each peer will have some coded packets from every chunk. Therefore, the total amount of storage of this scheme is smaller than other approaches.

To stream a video, the client connects to a number of peers with the fraction of the desired video. It first requests these peers to send their packets p_i's from the first chunk. After the client receives roughly k coded packets, it will be able to recover k original packets. It immediately sends a request to the senders to signal them to start streaming the packets from the second chunk. In the meanwhile, the client can start video playback. The process continues until the end of the stream is reached. Clearly, there is a delay at the beginning due to the time for the client to receive k independent packets. The attractive feature of this scheme is that no dynamic packet partition is required. All sending peers send packets at their available time-varying bandwidth until the client sends an *end of chunk* request to move to the next chunk. Therefore, TCP can be employed for streaming. The effective throughput at the receiver is roughly equal to the total throughputs from all the senders. At any point in time, one sender may have a slow connection, but as long as the total throughput is larger than the playback rate, the receiver will be able to playback the video smoothly. We emphasize that this scheme achieves maximum throughput without the complex coordination [58].

6.5 Implementation

We implement our proposed RESnc system in C++/Linux and test it in the real-world environment. In our model, the RESnc streaming system contains of a number of storage peers and receiving peers. When a source peer has a video file to share, it is allowed to code and distribute that file to a number of storage peers in such a way that each peer has some blocks from all the chunks. A receiver then randomly connects to some of these storage peers to obtain the file as shown in Figure 6.3. Thus, we model the process into three phases: the initial distribution of the packets to the intermediate nodes (data dispersion), the re-generation of lost packets due to peer failures and departures (data replenishment), and the transmissions of packets from the storage nodes to a receiver (path diversity streaming). These operations are on top of the existing P2P infrastructure, which performs indexing, searching and communicating among peers.

The RESnc streaming system consists of a set of peers interconnected through a Chord network [85] (Figure 6.3). The Chord network maintains connectivity among peers, manages peer membership, and performs object lookup. We note that the original Chord returns only one peer for an object lookup request, if the object exists in the system. In our prototype, we modify Chord to return multiple peers for each lookup request. Unlike the original Chord, our modified Chord stores a list of nodes that contains some parts of the video. When a user requests a video, it first sends a look up request to Chord to get back the list of nodes storing that video. Then it randomly chooses some of those storage nodes to download the

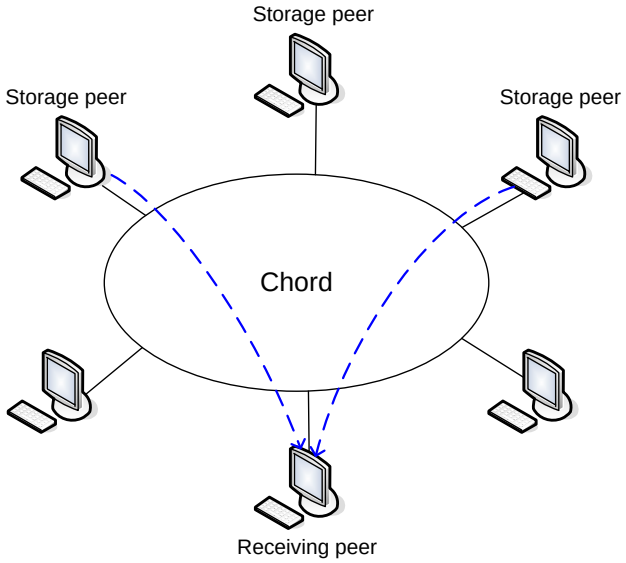


Figure 6.3: RESnc model: Peers are interconnected through a P2P substrate (CHORD) and multiple peers are serving one receiving peer.

video.

We use a tracking server (Chord server) to maintain information about participating peers in the Chord network, including their IP addresses and listening ports. When a peer wants to watch a video in the RESnc network, the tracking server will give a Chord node for it to send look up request. In RESnc, a peer is required to contribute not only its uploading bandwidth but also its storage capacity.

We also need to handle the following challenges when implementing RESnc:

6.5.1 Block size

A coded block is the transmission unit of a chunk. Small block size can accommodate slower seeds so that their slow link can transmit at least one block. However, small block size makes the network coding coefficients overhead more significant. For example, with random linear coding on $GF(2^{16})$, each coefficient occupies two bytes, and such overhead depends on the network coding size. Furthermore, to utilize the TCP link, a peer should fill up at least one TCP frame. Thus, investigation of the optimal block size is needed.

6.5.2 Chunk size

Large chunk size brings long delay and coding computational cost. Small chunk size leads to a different type of overhead since a receiving peer has to send start and stop request for each chunk. Due to the latency for these "stop" messages, the receiver might receive additional blocks after the chunk is complete. The overhead of such redundant blocks is more significant with smaller chunk size. Thus, study on optimal chunk size is also needed.

6.5.3 Redundancy level

When dispersing video, we introduce some redundancy to increase the streaming speed and data robustness. We need to find a good trade off between storage redundancy and robustness.

In the next section, we present the experimental results for a wide range for these parameters, to understand how the system performance varies as a function of these parameters.

6.6 Experimental Results

In this section, we investigate the practicality of the proposed system. First, we will be concerned with the computational overhead of performing network coding, and whether NC can be realized in practical system.

6.6.1 Network Coding rate

In this section, we want to measure the actual speeds of network encoding and decoding rates. In this experiment, we set the packet size of 1024 bytes and vary the network coding size (the number of packets involved in each operation). Figure 6.4 shows the computation rates of encoding and decoding operation versus the network coding size, and figure 6.5 shows the network coding delay versus chunk size in $GF(2^{16})$. We see that the encoding time is less than the decoding time but they are not much different. The reason is we implemented NC library using the table-lookup method, which makes multiplication and division quite similar in term of complexity. Importantly, when the number of packets per encoding increases, the encoding rate and the decoding rate decrease. It is necessary to design an appropriate coding structure so that the data transmission speed matches with the

encoding/decoding speeds, especially for P2P video streaming applications. For example, with network coding size of 20 packets, the encoding rate and decoding rate reach 28 Mb/s and 23 Mb/s respectively. Which is more than enough for any P2P streaming applications.

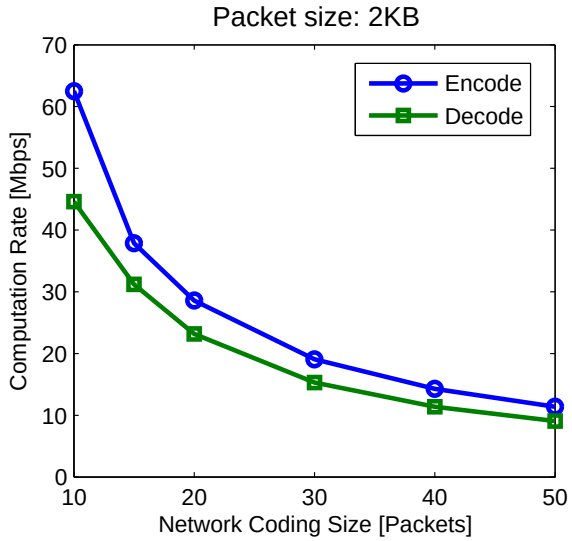


Figure 6.4: Computation rate of Network coding.

6.6.2 Linear dependence rate

Though it has been theoretically proved that coded packets generated by random network coding are linearly independent with high probability, in a realistic system, more linear dependence may appear due to a number of practical implementation issues, e.g., a sequence of coding coefficients generated from a random seed may not be as random as those uniformly and randomly selected from Galois Field.

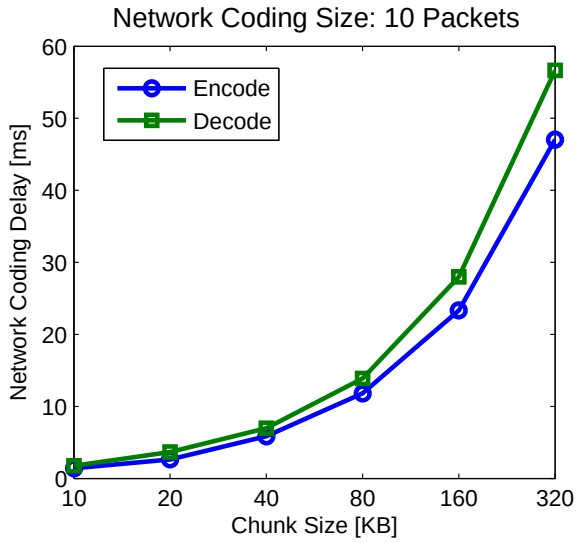


Figure 6.5: Network coding delay.

Therefore we need to evaluate how significant the ratio of redundant linearly dependent packets over all received coded packets at each peer is (referred to as linear dependence ratio), in order to evaluate the potential negative effects of network coding in a real-world operational system. Figure 6.6 compares linear dependence ratio between using $GF(2^8)$ and $GF(2^{16})$. As seen, the dependence ratio is smaller than 10^{-6} when using Galois Field (2^{16}). Furthermore, the larger network coding size (the number of packets involved in encoding/decoding process), the smaller dependence ratio. Even with Galois Field size of 2^8, the dependence ratio is smaller than 3×10^{-4}. Which means that the coding redundancy can be negligible in real world application.

Now we want to investigate the performance of the proposed architecture in a real video streaming system. We implement RESnc core applications on a number

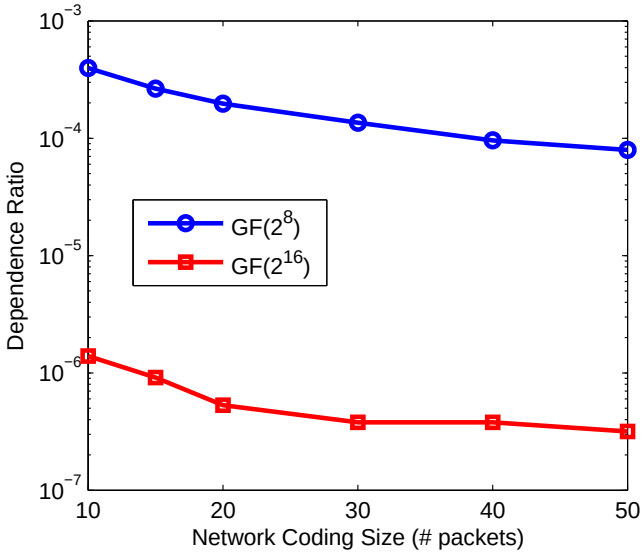


Figure 6.6: Dependence ratio at each peer.

of ordinary computers (Pentium 4 2.0GHz with 1 GB of memory and gigabit ethernet). The source distributes network coded packets to a number of storage peers with a given redundancy level. A client peer connects to a fixed number of storage peers to get video stream simultaneously. In the next section, we show the effect of network coding on the streaming performance.

6.6.3 Throughput versus network coding size

In this experiment, a client connects to 3 storage peers to download a video stream of 34.07MB. The total redundancy level of the three senders is 20%. The chunk size is set at 100KB and the sending rate is 200KBps. Figure 6.7 shows the actual throughput (include all transmission delay and processing delay) at the receiver as

a function of the network coding size i.e. the number of packets involved in each network coding operation. As seen, the throughput reduces significantly with small network coding size due to the dependent packets. Also, when the network coding size increases to a certain number, i.e., 50 in this experiment, the throughput reduces significantly due to computation cost of decoding. It clearly shows that the cost of transmitting a dependent packet is much higher than the decoding cost at the receiver side. This is well matched with the theoretical analysis. The system designer should choose the right network coding size that balances the dependence rate and the coding complexity.

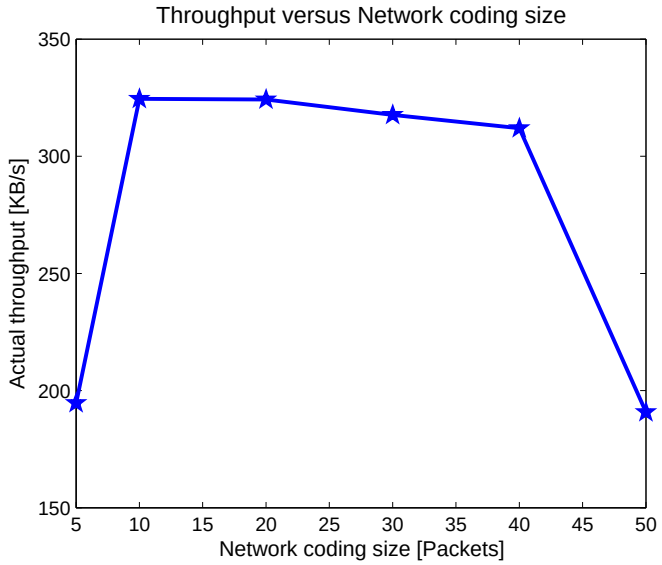


Figure 6.7: Actual throughput as function of Network coding size

6.6.4 Throughput versus chunk size

As it takes time for the braking messages (to tell senders move to the next chunk) to reach the senders, a receiver peer may receive additional packets after a chunk is completed. To reduce download redundancy within the constraints of coding complexity, one should use small packet size. However, to better utilize the TCP traffic, the packet size must be large enough to fill the TCP frame. We want to evaluate the effect of download redundancy to system performance in real setting. We use the same setting as with previous experiment. In which, a client connects to 3 storage peers to download a video stream of 34.07MB. The total redundancy level of the three senders is 20%. The the sending rate is 200KBps and the network coding size is 20 packets. Figure 6.8 shows the actual throughput at the receiver as a function of the chunk size. As seen, small chunk size results in larger throughput due to smaller amount of download redundancy. When we increase the chunk size, the actual throughput reduces significantly since the larger package size creates much more download redundancy.

6.6.5 Playback quality

We now want to characterize the performance of our network coding scheme versus the variation of download links' qualities. Specifically, the link capacities are varied with time, according to the Gilbert's model. In this model, there are two states: good and bad. In the good state, data can be sent, and in the bad state, congestion happens, and none of the data can get through. The transition proba-

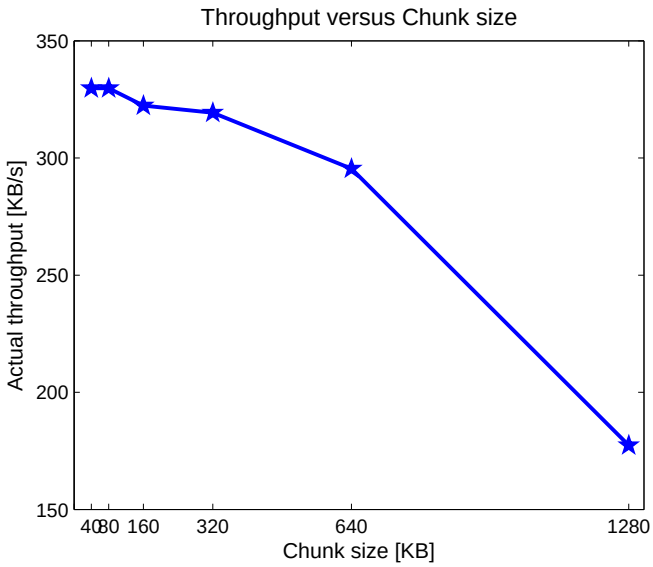


Figure 6.8: Actual throughput as function of chunk size

bilities between the states are shown in Fig. 6.9. We run this experiment with a video file of 10MB which has been split into chunks of 12.5 KB. The ideal sending rate is 128KB/s.

Figure 6.9 shows the number of buffer underflows with a fixed buffer size of 12.5KB and varied consumption rate. The consumption rate is the rate at which data is read from the buffer compared to the rate at which data is being sent when all senders are in a good state. Every underflow indicates that the receiver had to wait for one time step before resuming playback. We see that the multisender streaming schemes outperform single sender scheme. The more senders involved, the better playback quality. Please note that our multisender streaming scheme requires no coordination among senders. And the number of underflows seems

high due to a very small buffer size of 12.5KB. When we increase the buffer size, the playback quality increases significantly as shown in figure 6.10. As seen, the 3-sender scheme is consistently better than 1-sender and 2-sender schemes. Furthermore, with the buffer size of 125KB, the 3-sender scheme provides smooth playback without any glitches and adapts well to the variation in link capacities.

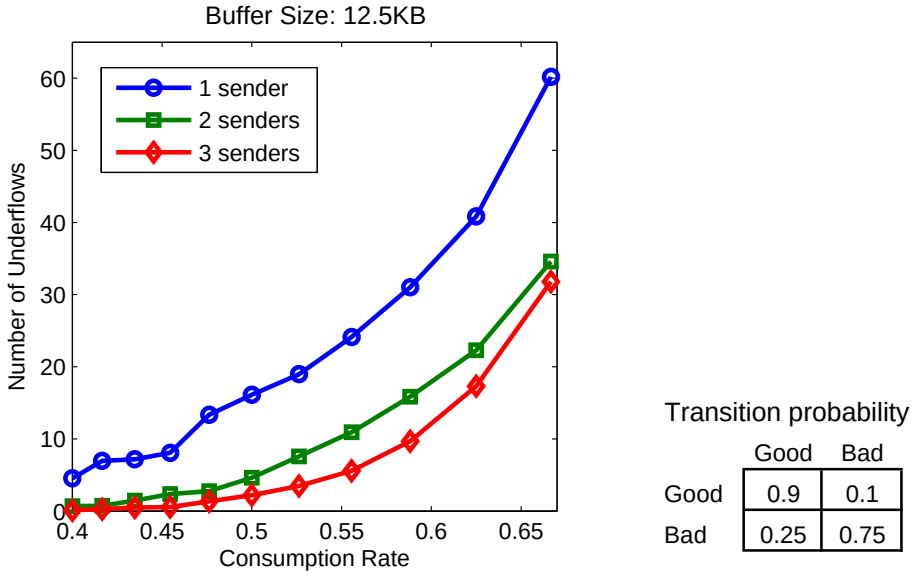


Figure 6.9: Buffer underflow versus Consumption rate.

6.7 Conclusions

We have proposed a RESnc that provides both performance improvement and scalability. It does so by employing data dispersion, data replenishment, a path diversity streaming protocol. Experimental results demonstrate that our system adapts well with link's variation and provides smooth playback quality.

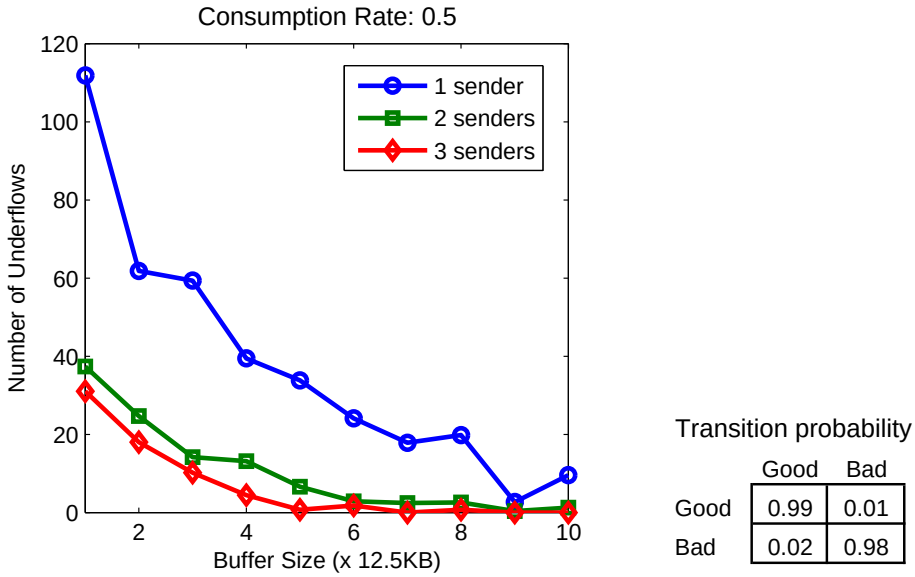


Figure 6.10: Buffer underflow versus Buffer size.

Chapter 7 – Conclusions and Future work

7.1 Conclusions

In this dissertation, we focus on exploiting network coding, an elegant transmission paradigm which is alternative to the traditional routing algorithms, to increase the energy efficiency, bandwidth utilization and network robustness, as well as reduce storage redundancy in real-world scenarios. We have shown our main research results on design, analysis and implementation of network coding for some practical applications.

Chapter 3 is devoted to address the uneven energy consumption problem in data gathering sensor networks where the nodes closer to the sink tend to consume more energy than those of the farther nodes. Consequently, the lifetime of a network is significantly shortened. We have designed and analyzed a novel energy-efficient network coding technique that maximizes the life-time of a sensor network using On-Off Keying. This cross-sensor coding scheme exploits (a) the trade-off between delay and bandwidth and (b) the network topology in order to alleviate the overall energy consumption of the network. Cross-sensor coding can significantly extend the network lifetime as compared with traditional (binary) coding by solving the energy-consumption unfairness problem. We have presented the theoretical and experimental results to show that transmission energy can be reduced substantially (e.g., a factor of 15) and the unequal energy consumption among nodes can be practically eliminated. The results presented in this chapter had been published in [56] [57].

In Chapter 4, we describe a rate distortion aware hierarchical NC technique and

transport protocol for Internet video streaming. We have proposed a NC-based multi-sender streaming framework for efficient media streaming in P2P networks in which, multiple servers/peers are employed to simultaneously stream a video to a single receiver. Our framework reduces the storage redundancy, simplifies the tight synchronization between the senders and receivers, and is integrated easily with TCP. Furthermore, we have proposed an hierarchical network coding technique to be used with scalable video bit stream to enable a receiver to adapt to the available bandwidth. Simulation results have demonstrated that under certain scenarios, our proposed schemes can result in bandwidth saving up to 60% over the traditional schemes. The results had been published in [59] [58].

Performance scalability and robustness against single-point failures are the hallmarks of a distributed system. If not properly designed however, the complexity of a distributed system, especially the complex coordinations among its many subsystems could incur a significant overhead that renders itself much less useful in practice. In Chapter 5, we investigate a class of randomized peer-to-peer (P2P) approach to Internet-wide distributed data storage systems that promises to reduce the coordination complexity and increases performance scalability. The core of these randomized P2P data storage systems is the data replenishment mechanism. The data replenishment automates the process of maintaining a sufficient level of data redundancy to ensure the availability of data in presence of peer departures and failures. We have proposed a novel analytical time-backward technique to bound the expected time, the longer the better, for a piece of data to remain in P2P systems. Both theoretical and simulation results are in agreement, indicating

that a proposed data replenishment via random linear network coding (RLNC) outperforms other popular strategies that employ repetition and channel coding techniques. Specifically, we have shown that expected time for a piece of data to remain in a P2P system is exponential in the number of peers used to store the data for the RLNC-based strategy, while they are quadratic for other strategies. Furthermore, the time-backward technique can be applied to problems in other disciplines such as gene population modeling in theoretical biology, which is intractable to directly model an exponentially large number of states using Markov chain representations.

Finally in Chapter 6, we present the architecture, design, and experimental results of an actual NC-based distributed video streaming system. We first implement random linear network coding (RLNC) library and show the feasibility of using RLNC in P2P video streaming applications. Then we design, implement and analyze RESnc - a resilient video storage and streaming over the Internet using network coding. RESnc increases the streaming throughput and data resiliency against peer departures and failures using peer diversity. These improvements are based on three architectural elements: (1) The random network coding scheme that codes and disperses video stream; (2) The scalable mechanism for automating the data replenishment process; (3) The path-diversity streaming protocol with minimal coordination. Experimental results demonstrated that our system adapts well with bandwidth fluctuation, provides significant playback quality improvement and bandwidth saving. The main results has been partially published in [60] [62].

7.2 Future work

We have advocated a new network coding approach to designing a distributed storage and streaming system in which participated peers contribute not only bandwidth but also storage resources. However, by nature, peers join and leave the network frequently. As a result, data might be lost temporarily or permanently. Thus, there is a need to develop techniques and policies for providing sufficient redundancy to ensure that, when a video is requested, it is available in the network.

It has been shown that random linear network coding can replenish data with less network traffic than replica and erasure codes. We have designed and analyzed a data replenishment mechanism using random network coding that automates the process of maintaining a sufficient redundancy to ensure the availability of data in presence of peer departures and failures. It stores network coded data distributedly at different nodes. The replenishment mechanism provides high data reliability by using redundancy and the regeneration of new redundant data at the newcomer whenever lost happen. Replenishment process is expected to be finished as soon as possible, because the replenishment time can influence the data reliability and availability of the systems.

However, our current replenishment mechanism mainly focused on how to replenish data to reduce the regeneration traffic, and to increase the lifetime of a piece of data in the system. We analyzed it based on the number of replenishments, not the replenishment time itself. Thus, designing a fast replenishment mechanism that take into account the advantage of distributed routes is an interesting prob-

lem. In future work, we want to investigate more on that direction.

Fast data replenishment structure

To ensure data reliability and availability, we expect the replenishment time to be as little as possible. The less replenishment time, the more replenished data can be preserved in the system with data loss. Less replenishment time can also result in higher probability that the replenishment process is successfully finished before any involving nodes leaves the system. One possible solution to reduce the replenishment time is to reduce the network traffic in the replenishment. We have already integrated random linear network coding in our current replenishment mechanism, which helps incur less replenishment traffic than repetition codes and erasure codes.

However, in the current replenishment scheme, replenished data are transferred directly from existing storage nodes (providers) to the newcomer and the new data is generated at the newcomer only. Therefore, the replenishment time is limited by the slowest link between newcomer and providers. The problem can be clearly shown via following example.

Consider the scenario that a newcomer C connects to two existing providers E_1 and E_2 to download the stored data. It then generate its own data by linear network coding the downloaded parts from E_1 and E_2. Assuming link bandwidths between providers and newcomer are as in Figure 7.1. In our current replenishment scheme, the newcomer receives data from each provider directly as in Figure 7.1(a). In this scheme, the replenishment speed depends on the minimal edge connecting to the newcomer, which is 30KB/s on link E_1-C. Thus, the bandwidth bottleneck

on link E_1-C limits the actual transmission rate during the replenishment process to 30KB/s only. Now if we allow the provider E_2 download data from provider E_1, and combine it with its own stored data to produce a replenished data then send to the newcomer C. Using this updated scheme, the bottleneck link is E_1-E_2 with 40KB/s. It increases the actual transmission rate during the replenishment process to 40KB/s. We can clearly see the improvement in replenishment speed if we utilize the available bandwidth between providers, which will reduce replenishment time as well.

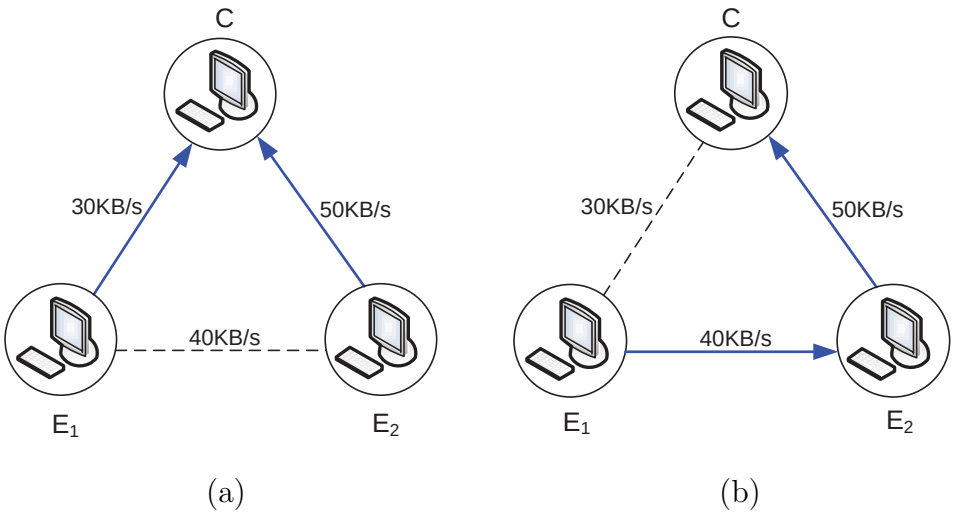


Figure 7.1: Data replenishment scheme: (a) Directed; (b) Optimal

In future work, we want to exploit the bandwidth between providers and find an optimal replenishment scheme with shortest time. In this scheme, data can be transferred from providers to the newcomer through a number of providers. A provider can receive data from other providers, then encode the received data with

the data this provider stores, and finally send the encoded data to another provider or to the newcomer. We believe that this scheme will reduce the replenishment time as it better utilizes the distributed bandwidth among providers and newcomer.

Bibliography

[1] http://www.bittorrents.com.

[2] http://www.kazaa.com.

[3] S. Acendanski, S. Deb, M. Medard, and R. Koetter. How good is random linear coding based distributed networked storage? In *Workshop on Network Coding, Theory, and Applications (NetCod'05)*, 2005.

[4] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46:1204–1216, July 2000.

[5] I Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, pages 102–106, 2002.

[6] D.G. Andersen. *Resilient overlay networks, Master Thesis.* Massachusetts Institute of Technology, 2001.

[7] D.G. Andersen, H. Balakrishnan, M.F. Kaashoek, and R. Morris. The case for resilient overlay networks. In *Proceeding of HotOS VIII*, May 2001.

[8] J. Apostolopoulos. On multiple description streaming with content delivery networks. In *Infocom*, volume 4310, June 2002.

[9] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *ACM SIGCOMM'02*, August 2002.

[10] Phillip G. Bradford, Mordecai J. Golin, Lawrence L. Larmore, and Wojciech Rytter. Optimal prefix-free codes for unequal letter costs: Dynamic programming with the monge property. In *European Symposium on Algorithms*, 1998.

[11] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost. Informed content delivery across adaptive overlay networks. *IEEE/ACM Transactions on Networking*, 12(5), October 2004.

[12] Ning Cai and Raymond. W. Yeung. Secure network coding. In *IEEE International Symposium on Information Theory (ISIT)*, 2002.

[13] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in a cooperative environment. In *SOSP*, October 2003.

[14] P.A. Chou, Y. Wu, and K. Jain. Practical network coding. In *41st Annual Allerton Conference on Communication Control and Computing*, Monticello, IL, USA, October 2003.

[15] Y. Chu, A. Ganjam, T. S. E. Ng, S. Rao, K. Sripanidkulchai, J. Zhan, and H. Zhang. Early experience with an internet broadcast system based on overlay multicast. In *USENIX'04*, 2004.

[16] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 1991.

[17] R. Cristescu and B. Beferull-Lozano. Lossy network correlated data gathering with high-resolution coding. *IEEE/ACM Transactions on Network and IEEE Transaction on Information Theory, Special Issue on Networking and Information Theory*, March 2005.

[18] R. Cristescu, B. Beferull-Lozano, and M. Vetterli. On network correlated data gathering. In *Proceeding of INFOCOM*, 2004.

[19] R. Cristescu, B. Beferull-Lozano, M. Vetterli, and R. Wattenhofer. Network correlated data gathering with explicit communication: Np-completeness and algorithms. *IEEE/ACM Transactions on Networking*, 2004.

[20] J. Heidemann D. Estrin, R. Govindan and S. Kumar. Next century challenges: scalable coordination in sensor networks. In *Proceedings of the fifth annual ACM/IEEE international conference on Mobile computing and networking*, pages 263–270, Seattle, WA USA, 1999.

[21] Supratim Deb, Michelle Effros, Tracey Ho, David R. Karger, Ralf Koetter, Desmond S. Lun, Muriel Medard, and Niranjan Ratnakar. Network coding for wireless applications: A brief tutorial. In *International Workshop on Wireless Ad-hoc Networks (IWWAN)*, May 2005.

[22] Supratim Deb, Muriel Medard, and C. Choute. On random network coding based information dissemination. In *IEEE International Symposium on Information Theory*, pages 278–282, Adelaide, Australia, September 2005.

[23] A. Dhulipala, C. Fragouli, and A. Orlitsky. Silence based communication for sensor networks. In *IEEE International Symposium on Information Theory*, July 2006.

[24] A. G. Dimakis. Codes on graphs for distributed storage in wireless networks. Master's thesis, University of California at Berkeley, 2005.

[25] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran. Network coding for distributed storage systems. *IEEE Transactions on Information Theory (to appear)*, 2010.

[26] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran. Decentralized erasure codes for distributed networked storage. *IEEE Transactions on Information Theory*, June 2006.

[27] M. Enachescu, A. Goel, R. Govindan, and R. Motwani. Scale free aggregation in sensor networks. In *Proceedings of the First International Workshop on Algorithmic Aspects of Wireless Sensor Networks*, 2004.

[28] C. Gkantsidis and J. Miller P. Rodriguez. Comprehensive view of a live network coding p2p system. In *IMC*, 2006.

[29] C. Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. In *IEEE Infocom*, 2005.

[30] Mordecai J. Golin, Claire Kenyon, and Neal E. Young. Huffman coding with unequal letter costs. In *Proceedings of the 34th annual ACM symposium on Theory of computing*, May 2002.

[31] A. Gupta, D. Agrawal, and A. Abbadi. Approximate range selection queries in peer-to-peer systems. In *VLDB Conference on Innovative Data Research*, 2003.

[32] Anwar Al Hamra, Chadi Barakat, and Thierry Turletti. Network coding for wireless mesh networks: A case study. In *International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'06)*, 2006.

[33] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *33rd International Conference on System Sciences*, January 2000.

[34] Wendi Rabiner Heinzelman, Amit Sinha, Alice Wang, and Anantha P. Chandrakasan. Energy-scalable algorithms and protocols for wireless sensor networks. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP '00)*, June 2000.

[35] T. Ho, R. Koetter, M. Mdard, M. Effors, J. Shi, and D. Karger. A random linear network coding approach to multicast. *IEEE/ACM Transtions on Information Theory*, 10, Oct 2006.

[36] T. Ho, R. Koetter, M. Medard, D. Karger, and M. Effros. The benefits of network coding over routing in randomized setting. In *International Symposium on Information Theory (ISIT)*, January 2003.

[37] T. Ho, M. Mdard, J. Shi, M. Effros, and D. R. Karger. On randomized network coding. In *41st Annual Allerton Conference on Communication Control and Computing*, January 2003.

[38] Tracey Ho, Muriel Medard, Michelle Effros, and Ralf Koetter. Network coding for correlated sources. In *Conference of Information Science and Systems*, 2004.

[39] Information Sciences Institute, http://www.isi.edu/nsnam/ns. *Network simulator*.

[40] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of the sixth annual international conference on Mobile computing and networking*, pages 56–67, Boston, MA USA, 2000.

[41] V. Jacobson. Building scalable and robust peer-to-peer overlay networks for broadcasting using network coding. In *ACM symposium on Principles of distributed computing*, pages 51 – 59, Las Vegas, NV, USA, July 2005.

[42] J. Kahn, R. Katz, and K. Pister. Mobile networking for smart dust. In *ACM/IEEE International Conference on Mobile Computing and Networking*, August 1999.

[43] Richard M. Karp. Minimum-redundancy coding for the discrete noiseless channel. *IRE Transactions on Information Theory*, pages 27–38, January 1961.

[44] S. Katti, D. Katabi, Wenjun Hu, Rahul Hariharan, and Muriel Medard. The importance of being opportunistic: Practical network coding for wireless environments. In *43rd Allerton Conference on Communication, Control, and Computing*, September 2005.

[45] J. Kim and J.G. Andrews. An energy efficient source coding and modulation scheme for wireless sensor networks. In *IEEE 6th Workshop on Signal Processing Advances in Wireless Communications*, June 2005.

[46] R. Koetter and M. Medard. An algebraic approach to network coding. *IEEE/ACM Transactions on Networking*, 11:782 – 795, October 2003.

[47] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *SOSP*, October 2003.

[48] J. Li, P. Chou, and C. Zhang. Mutualcast: An efficient mechanism for one-to-many content distribution. In *ACM Sigcomm Asia Workshop*, April 2005.

[49] S.-Y. R. Li, R. W. Yeung, , and N. Cai. Linear network coding. *IEEE Transactions on Information Theory*, 49:371 – 381, February 2003.

[50] S. Lin and D. Costello. *Error Control Coding (2nd Edition)*. Prentice Hall, 2004.

[51] S. Lindsey, C. Ragavendra, and K. Sivalingam. Data gathering algorithms in sensor networks using energy metrics. *IEEE Transactions on Parallel and Distributed Systems*, 23:776–787, 2002.

[52] C. H. Liu and H. H. Asada. A source coding and modulation method for power saving and interference reduction in ds-cdma sensor network systems. In *American Control Conference*, May 2002.

[53] H. Ma and M. El Zarki. Broadcast/multicast mpeg-2 video over wireless channels using header redundancy fec strategies. In *Proceedings of The International Society for Optical Engineering (SPIE)*, volume 3528, pages 69–80, November 1998.

[54] Microsoft, http://research.microsoft.com/ pablo/avalanche.aspx. *Avalanche: File Swarming with Network Coding.*

[55] Rex Min, Manish Bhardwaj, Seong-Hwan Cho, Amit Sinha, Eugene Shih, Alice Wang, Anantha Chandrakasan, and Eugene Shih Alice Wang. An architecture for a power-aware distributed microsensor node. In *In IEEE Workshop on Signal Processing Systems (SiPS 00*, October 2000.

[56] Kien Nguyen and Thinh Nguyen. Cross-sensor coding techniques for low energy sensor networks. In *The 6th International Wireless Communications and Mobile Computing Conference (IWCMC10)*, Caen, France, June 2010.

[57] Kien Nguyen, Thinh Nguyen, and Senching Cheung. On reducing communication energy using cross-sensor coding technique. *submitted to International Journal of Distributed Sensor Networks*, 2010.

[58] Kien Nguyen, Thinh Nguyen, and Senching Cheung. Video streaming with network coding. *The Journal of Signal Processing Systems*, 57(3):319–333, June 2010.

[59] Kien Nguyen, Thinh Nguyen, and Sen ching Cheung. Peer-to-peer streaming with hierarchical network coding. In *IEEE International Conference on Multmedia and Expo, 2007*, July 2007.

[60] Kien Nguyen, Thinh Nguyen, and Y. Kovchegov. A p2p video delivery network (p2p-vdn). In *Computer Communications and Networks, 2009. ICCCN 2009. Proceedings of 18th Internatonal Conference on*, pages 1 –7, 2009.

[61] Kien Nguyen, Thinh Nguyen, Yevgeniy Kovchegov, and Viet Le. Distributed data replenishment. *submitted to IEEE Transactions on Distributed and Parallel Systems*, 2010.

[62] Kien Nguyen, Thinh Nguyen, Yevgeniy Kovchegov, and Ian Milligan. Resilient p2p video storage and streaming with network coding. *submitted to the Journal of Parallel and Distributed Computing*, 2010.

[63] T. Nguyen and A. Zakhor. Distributed video streaming. In *Proceedings of the SPIE - The International Society for Optical Engineering, Multimedia Computing and Networking (MMCN)*, volume 4673, pages 186–95, San Jose, CA, January 2002.

[64] T. Nguyen and A. Zakhor. Distributed video streaming with forward error correction. In *Packet Video Workshop*, Pittsburg, PA, April 2002.

[65] T. Nguyen and A. Zakhor. Multiple sender distributed video streaming. *IEEE-Transactions on Multimedia and Networking*, 6(2):315–326, April 2004.

[66] V.N. Padmanabhan, H.J. Wang, and P.A. Chou. Resilient peer-to-peer streaming. In *IEEE International Conference on Network Protocols*, page 16, 2003.

[67] V.N. Padmanabhan, H.J. Wang, P.A. Chou, and K. Sripanidkulchai. Distributed streaming media content using cooperative networking. In *ACM NOSSDAV*, Miami, FL, May 2002.

[68] Fabio Pianese, Diego Perino, Joaqun Keller, and Ernst W. Biersack. Pulse: an adaptive, incentive-based, unstructured p2p live streaming system. *IEEE Transactions on Multimedia*, 9(6), 2007.

[69] G. Pottie and W. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5), 2000.

[70] Y. Prakash and S. K. S. Gupta. Energy efficient source coding and modulation for wireless applications. In *Proceedings of IEEE WCNC'03*, March 2003.

[71] J. Rabaey, J. Ammer, J. Silva, and D. Patel. Picoradio:ad-hoc wireless networking of ubquituous low-energy sensor/monitor nodes. In *Proceedings of the IEEE Computer Society Annual Workshop on VLSI*, April 2000.

[72] V. Raghunathan, C. Schurgers, S. Park, and M.B. Srivastava. Energy-aware wireless microsensor networks. *IEEE Signal Processing Magazine*, 19(2), 2002.

[73] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of ACM SIGCOMM*, August 2001.

[74] R. Rejaie and A. Ortega. Pals: Peer to peer adaptive layered streaming. In *International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, June 2003.

[75] R. Rejaie and S. Stafford. A framework for architecting peer-2-peer receiver-driven overlays. In *NOSSDAV*, June 2004.

[76] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms*, November 2001.

[77] A. Rowstron, A-M. Kermarrec, M. Castro, and P. Druschel. Scribe: The design of a large-scale event notification infrastructure. In *NGC*, November 2001.

[78] O. Sahin, A. Gupta, D. Agrawal, and A. Abbadi. A peer-to-peer framework for caching range queries. In *IEEE International Conference on Data Engineering (ICDE)*, 2004.

[79] K. Sankaralingam, S. Sethumadhavan, and J. C. Browne. A peer-to-peer framework for caching range queries. In *IEEE International Symposium on High-Performance Distributed Computing*, June 2003.

[80] C. Schmidt and M. Parashar. Flexible information discovery in decentralized distributed systems. In *IEEE International Symposium on High-Performance Distributed Computing*, June 2003.

[81] J. Selker, L. Thevenaz, H. Huwald, A. Mallet, W. Luxemburg, N. van de Giesen, M. Stejskal, J. Zeman, M. Westhoff, and M. Parlange. Distributed fiber-optic temperature sensing for hydrologic systems. *WATER RESOURCES RESEARCH*, december 2006.

[82] M. Sharaf, J. Beaver, A. Labrinidis, and P. Chrysanthis. Balancing energy efficiency and quality of aggregate data in sensor networks. *VLDB journal*, 13:384–403, 2004.

[83] Eugene Shih, Seong-Hwan Cho, Nathan Ickes, Rex Min, Amit Sinha, Alice Wang, and Anantha Chandrakasan. Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks. In *International Conference on Mobile Computing and Networking*, pages 272–287, 2001.

[84] D. Slepian and J.K. Wolf. Noiseless coding of correlated information sources. *IEEE Transaction on Information Theory*, 19:471–480, July 1973.

[85] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, September 2001.

[86] W. Tan and A. Zakhor. Real-time internet video using error resilient scalable compression and tcp-friendly transport protocol. *IEEE Transactions on Multimedia*, 1:172–186, june 1999.

[87] D. A. Tran, K. A. Hua, and T. T. Do. A peer-to-peer architecture for media streaming. *IEEE Journal on Selected Areas in Communications*, 20(1), january 2004.

[88] D.A. Tran, K.A. Hua, and T. Do. Zigzag: An efficient peer-2-peer scheme for media streaming. In *INFOCOM*, April 2003.

[89] J. Kulik W. R. Heinzelman and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of the fifth annual ACM/IEEE international conference on Mobile computing and networking*, pages 174–185, Seattle, WA USA, 1999.

[90] M. Wang and B. Li. Lava: A reality check of network coding in peer-to-peer live streaming. In *IEEE INFOCOM*, May 2007.

[91] M. Wang and B. Li. R2: Random rush with random network coding in live peer-to-peer streaming. *IEEE Journal on Selected Areas in Communications*, 25(9):16551666, December 2007.

[92] S. Wicker. *Error Control Systems for Digital Communication and Storage*. Prentice Hall, 1994.

[93] A. Woo and D. Culler. A transmission control scheme for media access in sensor networks. In *ACM/IEEE International Conference on Mobile Computing and Networks (Mobicom'01)*, July 2001.

[94] Wei Ye, John Heidemann, and Deborah Estrin. An energy-efficient mac protocol for wireless sensor networks. In *Proceedings of the IEEE Infocom*, pages 1567–1576, June 2002.

[95] X. Zhang, J.C. Liu, B. Li, and T.P. Yum. Coolstreaming/donet: A data-driven overlay network for efficient live media streaming. In *INFOCOM*, March 2005.

[96] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IIEEE Journal on Selected Areas in Communications*, January 2004.

[97] Y. Zhu, B. Li, and J. Guo. Multicast with network coding in application layer overlay networks. 22(1):1–13, January 2004.

[98] Y. Zhu and R. Sivakumar. Challenges - communication through silence in wireless sensor networks. In *Mobicom*, 2005.