ABSTRACT

Towards Efficient and Practical Reliable Bulk Data Transport
for Large Receiver Sets

Andrew E. Cutchin, M.S.

Mentor: Michael J. Donahoo, Ph.D.

Many critical network applications require the transmission of bulk data to a large, heterogeneous, asynchronous receiver set. Standard unicast solutions exhibit poor scaling due to inefficient use of bandwidth over shared links, prompting consideration of multicast and peer-to-peer systems. Unfortunately, these approaches introduce their own problems. In multicast, we must provide transport layer services, such as reliability and congestion/flow control. To deal with these, researchers have proposed the use of several layered multicast scheduling techniques using cyclic transmission and FEC. For peer-to-peer systems, we must address the problem of block location and extinction. Work in network coding provides an elegant solution to these problems; however, a naive implementation of such coding is computationally expensive. We propose a practical implementation of network coding. Next we compare

several layered encoding schemes. Finally, we compare the performance of layered multicast and network coding in peer-to-peer systems.

Towards Efficient and Practical Reliable Bulk Data Transport
for Large Receiver Sets

by

Andrew E. Cutchin, B.S.

A Thesis

Approved by the Department of Computer Science

Donald L. Gaitros, Ph.D., Chairperson

Submitted to the Graduate Faculty of
Baylor University in Partial Fulfillment of the
Requirements for the Degree
of
Master of Science

Approved by the Thesis Committee

Michael J. Donahoo, Ph.D., Chairperson

David B. Sturgill, Ph.D.

Glenn B. Blalock, Ph.D.

Accepted by the Graduate School
December 2007

J. Larry Lyon, Ph.D., Dean

Page bearing signatures is kept on file in the Graduate School.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

The author gratefully acknowledges the advice, assistance, and support of Dr. Jeff Donahoo who primarily directed this research and of Dr. David Sturgill who assisted in its direction at the later stages, both of whom provided many helpful and brilliant ideas. The author further acknowledges the generous assistance of Dr. Greg Hamerly who provided access to his computing resources for many of the network coding experiments presented herein. Additionally, the author is grateful to Mike Hutcheson and Baylor University Academic and Research Computing Services for providing access to the High Performance Computing Cluster which assisted greatly in the performance of the multicast experiments. Finally, the author acknowledges the support of the Department of Computer Science as a whole for providing the funding that made this research possible.

DEDICATION

To my friends and family

with heavy emphasis upon my parents,

Edward and Paula Cutchin,

without whose support this work,

even before such things were considered,

would not have been possible.

CHAPTER ONE

Introduction

The issue of reliably transporting data from a single original source to multiple interested recipients, particularly when the receiver set is large, is one of growing prominence as the Internet and computing in general become more and more ubiquitous. While there are certainly solutions available to the problem – the most obvious of which is the use of unicast to simply transmit a separate copy of the data to every interested recipient – few provide solutions that are both efficient and practical. Accordingly, we endeavor examine this problem and to provide some contribution towards its solution.

1.1 The Unicast Solution

The most obvious, though hardly most efficient, method for transmitting a file from a single source to a large receiver set is to simply transmit a copy of the file to each interested receiver as they request it. This is the solution often favored on the Internet for a great many types of downloads, the most apparent and common of which are downloads initiated by web browsers requesting a web page. This "solution", if it truly can be called such, to the problem of reliably transmitting bulk data to a large receiver set ignores the fact that many of the packets will traverse the same links repeatedly. The source's own link to the network provides the most obvious case of this issue. Since it is likely that the source will have only one link connecting it to

the network, every packet transmitted must pass across that link. Accordingly, this source's link must then carry every packet from the file a minimum of one time *for every member of the receiver set*! Obviously then, for large receiver sets, this is a highly inefficient use of the source's bandwidth as the same packets will traverse the link a great many times. This problem is further compounded by the fact that this situation would be simply an annoyance if the source possessed unlimited bandwidth but the realities of finite bandwidth complicate matters by limiting the rate at which the source can transmit data. Thus, the more simultaneous recipients there are, the slower each of them will receive the file as the source must attempt to distribute its bandwidth between them.

1.1.1 Mirroring

One well-known method for attempting to lessen the impact of the receiver set size upon the data rate and to prevent the source from simply being overwhelmed by requests is the creation of mirror sites. These mirrors are secondary sources of the desired data. We refer to them as secondary as the data must still originate from some primary (or original) source that conveys the data to the mirrors which then transmit it to the receiver set on request. While this technique has the benefit of spreading the receiver set across multiple sources, unless the number of mirrors scales with the size of the receiver set, this technique will still face the same problems experienced by the more naïve single-source solution with a sufficiently large receiver set. Thus, while this technique lessens the impact of the size of the receiver set upon the data

rate experienced at either end, it does not alleviate the problem as the receiver set continues to grow.

1.1.2 Benefits

Despite the concerns enumerated previously, the unicast solution is not wholly without virtue. The process of transmitting bulk data from a source to a recipient using unicast (and by way of the client/server model) is an extremely well-understood area of networking research. While new techniques are often being proposed to further improve the performance of this type of transmission, techniques adequate to the task already exist, particularly in terms of ensuring reliable delivery. Because these transmissions often make use of an application-layer protocol like TCP, special techniques beyond those employed by the protocol itself are rarely required in order to ensure reliable delivery. This ability to guarantee reliable and in-order delivery is the most significant benefit offered by the unicast solution. However, the lack of scalability due to bandwidth inefficiency continues to make this solution impractical for use as the size of the receiver set grows.

1.2 Multicast Solutions

In response to the multitude of problems associated with the use of unicast technologies and techniques for carrying out reliable bulk data transport to a large receiver set, we now consider the alternative technique known as multicast. Multicast involves the source transmitting a limited number of copies of the data (one in the case of network-layer multicast but possibly multiple copies in the case of application-layer

multicast) into the network, which then bears the responsibility for ensuring that the data is replicated as needed and delivered to all interested parties. By allowing the network to make decisions regarding how best to distribute the data, many of the bandwidth inefficiencies experienced with the unicast solution diminish. Under the multicast paradigm, it is no longer necessary to transmit multiple copies of the same packet across the same link; rather it is now sufficient to transmit a single copy. This copy will then be replicated at whatever point it becomes necessary for it to traverse two *different* links. Thus, through the use of multicast, mirrors are no longer required and a source that might previously have been dedicated to simply serving one file to large but size-bound (for reasons of practicality) receiver set is now free to serve as many files as its bandwidth will allow, each of which can be transmitted to receiver sets that are virtually unbounded in size.

1.2.1 Drawbacks

While it is clear that multicast offers significant benefits in terms of bandwidth efficiency and scalability with respect to receiver set size, it achieves these benefits at the cost of reliability, asynchronous behavior, and bandwidth scaling to the needs of individual receivers. Consideration of the techniques used for multicast make it quickly apparent that the mechanisms utilized over unicast by protocols like TCP to achieve reliability and congestion control will not translate directly to the multicast paradigm. While TCP-based unicast (and the application-layer techniques for solving our multiple recipient delivery problem which make use thereof) relies upon congestion control and avoidance algorithms(Jacobson 1995) to find the proper rate

for communicating with the receiver in a manner that both fully utilizes the available bandwidth and does not overwhelm the receiver's connection, network-layer multicast is incapable of using such a mechanism. There are several reasons why this mechanism cannot be applied to the multicast environment: first, it is impractical for multicast to rely upon feedback from the receivers. While it is not unreasonable to expect a multicast source to be able to handle feedback from a handful of receivers, the virtually unbounded nature of the size of a multicast group creates a situation where it is not hard to imagine a group being so large that simply processing acknowledgments from each receiver for each packet would overwhelm the source. Second, even if it were practical to process such a large number of packets, another difficulty presents itself. If we attempt to scale our bandwidth usage to the receivers, we are left with a decision to make: *which* receiver should we scale to? There are several options available to us in that regard: we could send at the rate of the fastest recipient (except everyone slower will be overwhelmed), we could send at the rate of the slowest recipient (except everyone with a faster connection will be able to utilize only a fraction of their available bandwidth), or we could pick some point in between (which is fraught with both of the aforementioned problems). Clearly, attempting to have a single channel which scales to the bandwidth of a single "representative" (we use the term loosely) receiver is untenable.

In addition to the problems of bandwidth scaling associated with the impracticality of relying upon TCP-like feedback techniques, the ability to guarantee delivery (and more specifically, in-order delivery) of the data is also lacking the multicast paradigm. Since TCP relies upon the same feedback method for both reliability and

flow control we are left without a method for ensuring reliable delivery due to the possibility of implosion associated with attempting to process acknowledgments from a large receiver set for every packet transmitted. Attempts to correct this problem through the use of the negative-acknowledgment are still fraught with problems as well. Even if recipients inform the source that they have failed to receive a particular packet, the source is still faced with the problem of attempting to carry out retransmission of that data to the receiver that transmitted the negative-acknowledgment. This problem then becomes much more severe as the receiver set size increases and we are forced to fall back on unicast techniques to replace lost packets for more and more nodes.

Another consideration associated with the use of multicast (particularly in its most basic form) is the lack of support for asynchronous membership in the receiver set. We cannot assume that all users will begin the download at the same moment as this would place a great limitation upon our system. Additionally, it is clear that as the number of recipients grows, the difficulty in bringing about synchronization between them grows as well. As there is no limit to the potential size of the recipient group (theoretically, every node on the entire Internet could choose to participate), this difficulty rapidly becomes a source of major concern and makes our system difficult to use for large-scale applications. Clearly, then, we must have a system which is capable of allowing users to join (and leave and rejoin as many times as desired) the recipient group in an asynchronous fashion.

1.2.2 Layered/Encoded Multicast

In an effort to address many of the shortcomings associated with the use of multicast, a number of techniques (Vicisano 1997; Donahoo, Ammar, and Zegura 1999; Birk and Crupnicoff 2003) have been proposed. Among these are the use of layered channels, cyclic transmission, and Forward Erasure Coding(Rizzo 1997). These techniques attempt to achieve bandwidth scaling by supplying a number of channels to which users are able to subscribe according to their bandwidth capabilities and thus allow those with greater bandwidth to simply subscribe to more channels in and effort to increase the speed of their download. They also make use of erasure coding in order to make retransmission of specific packets unnecessary. The fact that TCP-like retransmission of missed packets is difficult under the multicast paradigm leaves us with the need for a system under which any given packet will be useful to *every* receiver regardless which packets they have already received. The authors of (Byers, Luby, Mitzenmacher, and Rege 1998) term this a "digital fountain" making use of the analogy between filling a glass by collecting water drops from a fountain and the fact that the receiver should not be required to collect specific packets but simply collect a sufficient number of packets to rebuild the original data (through some coding technique). This idea that packets are useful in a wide variety of contexts is of importance if we wish to be able to serve receivers at different rates and with asynchronous group membership. Erasure coding, with its ability to generate a multitude of redundancy packets is seen as a solution to this problem.

Two major problems remain with the use of multicast, however, despite the solutions brought by these innovative techniques. The first is the fact that under any

form of the multicast paradigm, the original source is required in order to drive the transmission. This, in turn, requires that the source remain a part of the network (and willing to transmit the file) until *all* interested receivers have successfully acquired the file. The other drawback, and by far the most serious, is the present and continuing for the foreseeable future unavailability of multicast at the network layer. While there have been attempts at application-layer implementations of multicast in order to overcome this limitation, they have been eclipsed by the rise to prominence of peer-to-peer systems.

1.3 Peer-to-peer

The lack of availability of multicast at the network layer in the Internet as well as a number of other drawbacks and complications associated with the use of multicast has given rise to a number of application-layer techniques which have eventually evolved into peer-to-peer systems. These systems, unlike the multicast paradigm, do not rely upon a single source transmitting the file to the receivers in a defined order. Rather, they consist of a source which disseminates the data in a fractured manner to as many different receivers as possible. Receivers are then responsible for acquiring any missing data from other members in the network (referred to as *peers*), thereby freeing the original source from sole responsibility for ensuring the successful transmission of the file and even allowing the source to eventually leave the network.

1.3.1 Basic Peer-to-Peer

A number of peer-to-peer systems have been developed since the concept first began to gain prominence but the one most commonly accepted as being the most popular is BitTorrent(Cohen 2003). Under this system, peers connect to a number of other peers (this is referred to as a *swarm*) who then share data with one another. A number of mechanisms, such as tit-for-tat, are used in order to ensure that peers contribute to the system as well as benefiting from it. The protocol also requires peers to acquire information from the peers to whom they are connected about the blocks presently available at those peers and to then make intelligent decisions regarding which block to download next. These decisions are expected to favor those blocks that are rare in an effort to prevent the last copy of any block from leaving the network before it can be acquired elsewhere and thus causing the block to become extinct and making it impossible to completely download the file.

1.3.2 Network Coding

In an effort to alleviate problems associated with block extinction and the need to acquire information from every connected peer regarding available blocks, network coding (Chou, Wu, and Jain 2003) has been proposed as an addition to peer-to-peer systems. Network coding provides many of the same benefits as erasure coding in that we are no longer required to acquire specific blocks but, instead, to acquire a specific number of unique blocks[1] . In addition, network coding provides the important benefit

[1] Uniqueness here only applies to erasure coding techniques like FEC as network coding enforces the stronger constraint that blocks must be not only unique but linearly independent of each other as well.

of allowing the peers themselves to generate new encoded packets rather than placing that power with the original source alone. This is extremely important and results in much of the benefit associated with network coding because it results in every block acquired being in some way a linear combination of the original file. Thus, the goal of a peer becomes the finding of blocks which are linearly independent from those already received and thus we no longer required detailed information about the blocks available at the connected peers. This has the added benefit of reducing the possibility of being unable to completely recover the original data due to block extinction since, as long as a sufficient number of linearly independent blocks are available in the network, the file can always be recovered.

Despite the many benefits of using network coding in a peer-to-peer environment, there are a number of concerns that have been raised regarding its use. In particular, the practicality of network coding has been questioned (Wang and Li 2006) because of the computational complexity associated with its encode and decode operations. These are some of the issues that we seek to address in this document.

1.4 Contributions

These systems have provided what is, to date, the most practical solution to the problem of reliably transmitting bulk data to large receiver sets though efficiencies still abound. We seek to make three main contributions to the work in this area. First, we endeavor to examine and differentiate the various encoded and layered multicast schemes which have been proposed. In this regard, we will demonstrate that one of the proposed multicast schemes, namely the Session Organization scheme (Vicisano 1997)

is the most practical of the schemes presently proposed. Second, we propose our own implementation of network coding complete with a number of improvements which we believe will result in a level of performance sufficient to conclude that network coding is in fact practical. These improvements include the use of more efficient forms of matrix multiplication and the introduction of parallelism into the decoding process. Third and finally, we attempt to provide a comparison of the performance of network coding with that of multicast in terms of requisite download time through the use of simulations. We believe that this comparison will prove instructive in helping to chart the course for future work in this area.

We thus begin our discussion by considering the related and previous work in this area in detail in Chapter 2 which is primarily divided between a detailed examination of multicast techniques and an overview of peer-to-peer systems. Next, we provide in depth discussion of naïve network coding and move to propose our own implementation of network coding in Chapter 3. Here we provide basic information regarding our implementation and detail improvements made first to the encoding process and then to the decoding process. Subsequent to this, we present the results of our work in Chapter 4 where we begin by detailing the performance of our network coding implementation. We then present comparisons relating the multicast techniques to one another and conclude by presenting experiments that seek to compare peer-to-peer with network coding and multicast. Finally, we summarize the results of our work and briefly discuss directions of possible future work.

CHAPTER TWO

Related Work

The problem of scaling content distribution to multiple, asynchronous recipients with heterogeneous capabilities has become more and more prominent with the growth of the Internet. The desire of companies to provide software updates and patches to consumers by way of the Internet provides the perfect example of this problem. Indeed, even cursory examination of such a situation demonstrates that attempting to achieve such data transport through the use of traditional unicast solutions leads to gross inefficiency in terms of network resource utilization due to message replication on shared paths.

Over the years, a great deal of effort has been expended in trying to improve the efficiency of such data transport. To this end, a number of solutions have been proposed, from network-layer multicast to application-layer multicast, and, finally, to peer-to-peer networks. Each of these methods offers some benefit but, as is often the case, carries with it drawbacks as well. This chapter is dedicated to presenting past and related work in these areas of research.

2.1 Multicast

One approach for solving the multiple recipient delivery problem beyond simple unicast involves multicast, which is a data transmission technique that leverages points in the network in an effort to make efficient use of shared paths by relying

upon them to replicate packets as needed in order to ensure that only one copy of a given packet traverses a shared path. As an example, in the case of network-layer multicast, the network routers themselves serve as the replication points along the paths. This technique offers a number of benefits because of its tight integration into the network layer but is currently unavailable for network-wide use.

2.1.1 IP Multicast

The set of participants in a multicast session is called the *multicast group*, which is identified by a multicast address. This address is taken from a subset of the larger IP address space which is reserved for use in identifying multicast groups. Each member of the multicast group must identify themselves with this IP address in order to signify their membership in the group. Having now identified the group, each member is able to multicast data to the remainder of the group by addressing data to the group IP address. It is then left to the routers in the network to ensure that packets are transmitted to group members as efficiently as possible. This is accomplished through the use of routing protocols, such as Distance Vector Multicast Routing Protocol (DVMRP)(Waitzman, Partridge, and Deering 1988), by which routers attempt to remain apprised of whether or not there are currently down-stream members of a given multicast group. This knowledge then allows routers to only pass packets on to other nodes interested in receiving the data from the multicast group.

While network-layer multicast performs well with respect to network bandwidth consumption, it is not without problems and drawbacks of its own, particularly from the perspective of the recipients. One of the prime benefits of the more traditional

and common unicast model for data transport is that a protocol, such as TCP, can be utilized which ensures that the recipients receive the data in a way that guarantees delivery and tailors itself to the bandwidth and congestion conditions along the path from the source to that recipient; these benefits are completely lacking from IP multicast (particularly in its purest form as it has been described thus far). TCP requires that receivers transmit acknowledgments for all data received in order to ensure delivery of data even in the face of network loss. This reliance upon receiver feedback alone makes TCP an imperfect match for multicast. Even were we willing to consider allowing a multitude of receivers to all transmit acknowledgments back to the multicast source, we quickly find that this creates further problems. First, allowing multicast receivers to transmit acknowledgments to the source clearly does not scale to very large groups as the links to the source would be overwhelmed by acknowledgments. Additionally, since guaranteeing reliable delivery to every member of the multicast group would require a retransmission of a packet if even a single recipient failed to receive the packet, aggregation of acknowledgments becomes problematic as it must be clear to the source that a particular acknowledgment has failed to arrive. Second, there is the problem of which packets to retransmit. Even if we are able to overcome the acknowledgment aggregation problem or we adjust our protocol to make use of the negative acknowledgment (NACK) paradigm (whereby recipients transmit a NACK indicating that they did not receive a particular packet rather than acknowledging those packets which were received), we still must determine the particular packet to retransmit and how this should be accomplished. Since it is highly unlikely that every member of the group will lose the same packet every time, this

could potentially require the retransmission of a packet to the entire group which is only required by one member; this is a gross inefficiency in bandwidth usage. The alternative is that retransmissions should be transmitted by way of unicast only to those group members who require the packet being retransmitted. However, as loss increases and the source is forced to spend more and more time helping recipients make up for missed packets, this solution clearly degrades to pure unicast. More importantly, whether we retransmit over unicast or multicast, the source itself is still forced to process a potentially large amount of information in order to determine which packets must be retransmitted and to whom they should be sent. All of these problems combine to make the best-understood methods of ensuring reliable delivery, which can react to congestion and loss in the network, unworkable for use over IP multicast.

Another problem relates to the rate at which data is transmitted to the multicast group. In practice, we must be able to serve a large number of recipients with heterogeneous bandwidth capabilities. This creates a significant difficulty, however, as transmitting at a rate suitable for the fastest receivers quickly overwhelms the lower end receivers while transmitting at a rate acceptable to even the slowest receivers leaves the more capable receivers unable to utilize their full bandwidth. Thus we find ourselves in the position of having to intentionally disadvantage one portion of the multicast group in order to better serve another portion. While it could be argued that a compromise approach would be the best solution, the reality (particularly given the difficulties in retransmission already discussed) is that the only workable solution that minimizes packet loss is to scale the transmission rate back to match

that the of the slowest member of the multicast group. Clearly this behavior of scaling transmission rate to match the slowest member is unacceptable if we desire to utilize bandwidth in the most efficient manner possible.

Having examined IP multicast and the problems associated with it, it becomes clear that, while multicast offers a number of benefits in terms of avoiding transmission of duplicate information across a link, it is not without its own set of serious drawbacks that must be addressed before it can be considered a viable solution to the problem we seek to solve. A number of solutions have been proposed to help deal with these issues and the next sections will seek to examine several of them.

2.1.2 Application-layer Multicast

The continuing unavailability of IP multicast in the Internet has been one of the greatest impediments to its widespread use. One attempt to achieve a similar mode of transport for bulk data that researchers have proposed is the concept of application-layer (or end-system) multicast as exemplified by the Narada network (hua Chu, Rao, and Zhang 2000). The Narada authors realized that if the network-level resources (i.e., the routers) could not be leveraged to allow multicast delivery, then, perhaps, end-systems could be. Thus, each node wishing to participate in the multicast session connects to other nodes already present in the session (as opposed to setting an IP address which informs the router network of the node's new membership status) and essentially grafts itself into the tree or mesh of nodes participating in the system. This is made to work by having each node maintain multiple connections to nodes upstream (fewer hops from the source) and downstream (more hops from the

source). In their simplest form, these networks are established as a tree where each node maintains only one upstream connection but multiple downstream connections. The source is then able to transmit the data to the entire tree by transmitting it (by way of unicast) to each of the nodes connected directly to it. These nodes then follow the same procedure, storing the data they have received from the source and likewise transmitting it to all downstream nodes connected directly to them. We note that this process, while multicast-like in nature, does not offer the same benefits in terms of efficient usage of network bandwidth; the source (and, indeed, each node) is still required to transmit multiple copies of the same data across the same link thereby wasting some measure of bandwidth whereas, under network-layer multicast, the network is responsible for ensuring that no more than one copy of the data crosses a given link. We note that further inefficiencies exist due to the fact that network-layer multicast is able to remain largely within the core of the network until final dispersal to the interested nodes (who only make up the leaf-nodes in the network-layer multicast tree) whereas application layer multicast requires data to be transmitted to end-systems (which are generally found at the edges of the network) at every stage of delivery (that is, end-systems are no longer exclusively leaf nodes but now serve as interior nodes as well). Additional difficulties associated with this technique are related to the problems of bootstrapping – that is, how does one find and establish the initial connections to nodes in the network – common to all such networks, as well as handling node failure and allowing/assisting nodes to find the optimal location in the network for them.

2.1.3 Reliable Multicast

Having examined IP and application-layer multicast, we have made it clear that there are a number of drawbacks associated with its use if one wishes to achieve TCP-like reliable transport while utilizing available network bandwidth as efficiently as possible for all parties involved in the transmission. In response to these issues, a number of schemes have been proposed utilizing various techniques in order to improve the effectiveness and efficiency of bulk data transport over IP multicast while at the same time attempting to make some guarantees about the reliability of the transmission. Four of these schemes are the Session Organization (SO) and Channel Organization (CO) schemes proposed in (Vicisano 1997), the Partition Organization (PO) scheme proposed in (Donahoo, Ammar, and Zegura 1999), and another scheme proposed in (Birk and Crupnicoff 2003) which we will refer to henceforth as the Burk-Crupnicoff (BC) scheme. We next present several of the techniques that these schemes all rely upon in order to effect their improvements in multicast performance; we then present each of these schemes in greater detail in Section 2.1.4.

2.1.3.1 *Cyclic Multicast.* Given that selective packet retransmission is impractical over IP multicast for the reasons described previously, it is clear that some alternate mechanism for replacing lost packets must be instituted. Perhaps the most obvious method of attempting to solve this particular problem is to repeatedly transmit the data in a cyclic fashion, a technique known as *cyclic multicast*(Almeroth, Ammar, and Fei 1998). Under ideal (lossless) conditions, each member of the multicast group remains in the group for one complete send cycle. It is important to

note that, from the receiver's perspective, a cycle need not begin with the packet that is actually the first in the file but, rather, that a cycle is measured from receipt of the first packet after joining the group until that packet is received again. The receiver must wait for the packet to be retransmitted during the normal course of any subsequent cycle in order to recover the lost packet.

The effectiveness of this solution is significantly diminished, however, by the fact that this need to receive subsequent cycles makes the recovery of a single packet a potentially expensive process in terms of bandwidth and time required. The reason for this is that if a receiver misses a single packet (for a worst-case situation, let us assume this to be the last packet of their first cycle), they must then wait for the packet to be transmitted again (as the last packet of the second cycle), in the meantime receiving duplicate copies of every packet received during the first cycle. While (Ammar, Almeroth, Clark, and Fei 1998) suggests this technique for use with web pages whose generally small size makes such inconvenience trivial in most cases, the application of this technique to bulk data transport is far more problematic due to the significantly greater file sizes involved. Additionally, there is no guarantee that we will actually receive the missing packet during the second cycle. Thus, in the face of high loss probability it is likely to become necessary to remain for a number of cycles, resulting in gross inefficiency of bandwidth usage.

While cyclic multicast may be promising for use with small files, as file sizes increase the overhead associated with it increases as well making it unreasonable as a solution to the reliable delivery problem for bulk data on its own. Note, however, that this technique is not wholly without virtue; it will be shown in subsequent sections

that the combination of this technique with other techniques can provide a workable solution to reliable delivery. Additionally, the use of cyclic multicast also provides us with the ability to serve groups whose membership is asynchronous.

2.1.3.2 *Erasure Coding.* Having now established a technique by which we can transmit bulk data to multicast groups with asynchronous membership, we again turn our attention to the issue of reliable delivery. Since few practical means for recovering a missing packet through retransmission have been forthcoming, researchers proposed the use of erasure codes to recover from data loss. The most commonly used of these erasure codes is the Forward Erasure Code (FEC)(Rizzo 1997), which allows us to generate some number (chosen *a priori*) of redundancy packets from the original packets making up the file. More formally, we consider a file to be divided into k packets; we then apply a transformation to the k packets which generates $n - k$ additional redundancy packets, giving us a total of $n \gg k$ packets. From the n packets now available, it suffices to receive any subset of k unique packets in order to successfully recover the original data. This, then, provides us with a mechanism for recovering from packet loss during multicast transmissions without the need for retransmission of specific packets. The source merely needs to transmit each of the n packets in turn, and each receiver remains until it has received k packets. The loss of a packet now means that only one additional packet need be received rather than having to request retransmission of that packet or wait for the packet to be sent again as part of another cycle.

While the use of FEC seems to provide a mechanism to allow recovery from packet loss without the need for retransmission, it is suffers from one serious problem: the computational complexity of FEC makes it unsuitable values of k beyond approximately 32(Rizzo 1997). This combined with the limitations placed on packet size by UDP limits the size of the file that can be transmitted to tens of kilobytes rather than the multi-gigabyte files we desire. In answer to this, schemes such as that proposed in (Vicisano 1997) suggest dividing the file into disjoint units (termed *blocks*) upon which coding will be performed independently. Thus, we divide the file into $B = \frac{X}{k}$ disjoint blocks of k packets each, where X is the number of packets in the file; these blocks are then encoded separately from one another. The receiver must then receive k packets from a given block in order to decode that block and must decode B blocks in order to recover the original file. It is important to note that packets from one block are of no use to other blocks. The problem, however, of how to schedule the sending of packets from the various blocks in such a way to ensure the most efficient use of bandwidth is now non-trivial; Section 2.1.4 will examine three systems which present such scheduling techniques.

One other method of encoding which has also received attention is the use of Tornado codes associated with the digital fountain system(Byers, Luby, Mitzenmacher, and Rege 1998). This encoding method, unlike FEC, is capable of generating a virtually limitless number of encoded packets. This extraordinary ability comes with the caveat, however, that slightly more (the exact number varies) than k packets must be received before decoding can commence. Further, as this technique is proprietary, we do not consider its use further.

Finally, it is worth noting that the methods of actually performing the coding as described in (Rizzo 1997) make for one particularly interesting property: the k blocks used to generate the redundancy blocks remain unchanged and unencoded. This occurs because the generation process relies upon multiplying the data by another well-known matrix which generates the encoded blocks. As originally presented, the first k rows of this matrix form the k identity matrix thus leaving the original data unchanged by the encoding process. The performance implications of this fact are that if a receiver is fortunate enough to receive only the k original packets, there will be no need to actually perform the decode operation as the decoding matrix would be merely the identity matrix. As will be seen, this fact is leveraged by the PO scheme(Donahoo, Ammar, and Zegura 1999) in order to attempt to increase the performance of the system.

2.1.4 Layered Multicast Scheduling

We have now largely addressed the issues related to recovering from packet loss and are thus left with the issue of bandwidth scaling remains as one of the most serious challenges to the effective use of IP multicast. The fact that multicast transmissions are intended for multiple recipients makes impossible the matching of bandwidth to any one client without potentially adversely affecting other clients. One solution which has been proposed to solve this problem for the process of multicasting streaming multimedia content, is receiver-driven layered multicast(McCanne, Jacobson, and Vetterli 1996). Under this paradigm, multimedia (such as streaming video) is encoded in a layered manner such that there is a base layer (which contains the

most basic information required to experience even a low quality version of the multimedia) and some number of predetermined additional layers, which cumulatively add quality to the lower layers. Each encoding layer is then transmitted on a separate multicast channel. Receivers then are able to subscribe to a contiguous set of the layered channels (as long as the set contains the "base" channel) up to their bandwidth capabilities. In this way, low-bandwidth receivers may experience a low-quality version of the content by subscribing to only the lower-level channels while high-bandwidth receivers can experience the full quality by subscribing to a larger subset of the available channels.

In order to allow bulk data receivers to experience a similar bandwidth scaling in order to improve the speed of the download (as opposed to the quality of the content), a similar process can be used, as demonstrated in (Vicisano 1997), which presented a technique for accomplishing this known as the Session Organization (SO) scheme. To accomplish this, the authors of the SO scheme define a *session* as a contiguous set of channels that a receiver joins with the requirement that every session must contain the base channel and denote a session as S_i where i is the highest channel in the session. Thus S_0 contains only channel 0 (denoted C_0) and S_3 contains channels C_0, C_1, C_2, and C_3. In order to allow users to receive the file without the need to receive duplicate data, it becomes necessary to schedule the sending of the file's packets in such a way that a receiver who listens to one complete cycle of any given session (assuming no loss) will receive the entire file without receiving any duplicate packets. We now define the *frequency* of a session to be the number of cycles completed on that session relative to S_0 (which is defined to always have frequency of

1). Obviously, the higher channels have high frequencies (and in the case of channels with exponentially distributed bandwidths, as is the case in many of the schemes that have been proposed, including the SO scheme, the higher sessions have exponentially higher frequencies), the process of scheduling the sending of packets in such a way that no session experiences a duplicate within the cycle is non-trivial.

We now move on to considering three proposed schemes for solving the single source/multiple receiver reliable delivery problem making use of IP multicast, cyclic transmission, layering, and FEC. The three schemes that we will consider are the Session Organization (SO) scheme (Vicisano 1997), the Partition Organization (PO) scheme (Donahoo, Ammar, and Zegura 1999), and an unnamed scheme developed by the authors of (Birk and Crupnicoff 2003) which we will henceforth term the "BC" scheme. A fourth scheme, termed the Channel Organization (CO) scheme, was also presented alongside the SO scheme in (Vicisano 1997) but is generally omitted from discussion as the SO scheme tends to show superior performance. The main features of these schemes in which we are presently interested are their methods for scheduling the sending of packets on the layered channels. We note that with the exception of different algorithms for scheduling the sending of packets the SO and BC schemes are essentially identical in that they appear to produce schedules which maintain the same properties (a fact which will be explored further in chapter 4). Both schemes make use of layered channels with exponentially distributed bandwidths (which require subscription as described in Section 2.1.4) transmitting packets FEC encoded packets in a cyclic fashion. Both schemes also approach the scheduling problem with a desire to effect what is alternately termed *group interleaving*(Birk and Crupnicoff 2003) or

block interleaving(Vicisano 1997); we will prefer the latter of these terms as it remains consistent with our earlier use of the term "block". This property of a schedule ensures that after receiving a packet from any block $b_i \in b_0 \ldots b_{B-1}$, the receiver (assuming no loss) will receive $B-1$ packets from other blocks before receiving another packet from b_i. Thus, one packet will be received for every block before a second packet is received for any block. This helps to ensure that a receiver is not forced to wait for an inordinate amount of time before a packet is received to fill in a block from which one is missing (though if a given block is missing two and all others are complete, this can create severe inefficiencies as will be seen in Chapter 4). The PO scheme, by contrast, still makes use of layered channels and FEC but organizes the channels according to block and differentiates between redundancy blocks and original blocks.

Throughout the discussion we will be primarily focused upon the metric for these schemes known as *reception efficiency* which is defined as the number of packets required to perform the decode operation, Bk, divided by the total number of packets actually received(Donahoo 1998). We are particularly interested in this metric in the face of two types of network loss, random and burst. Random loss is the random dropping of individual packets without regard to previous loss of packets and should occur fairly uniformly across the duration of the transmission. Burst loss on the other hand is a single instance of loss lasting across several packet transmissions (thus causing several consecutive packets to be lost) and all channels.

2.1.4.1 *SO Scheme.* The Session Organization scheme (SO) was originally developed by Vicisano in (Vicisano 1997) and reexamined by Donahoo in (Donahoo

1998). In addition to the number of blocks, B, and the FEC parameters k and n, the SO scheme has a number of parameters and variables which must be defined prior to describing it. First, we recall that the channels under this scheme are exponentially distributed in terms of bandwidth. Formally, this is defined as follows: first, let w_i be the bandwidth of channel C_i, then the bandwidth for C_0 is w_0 and the bandwidth for all subsequent channels (C_i) (and where n_c is the number of channels) is given by (from (Donahoo 1998)):

$$w_i = 2^{i-1} w_0 \quad (0 < i \leq n_c - 1) \tag{2.1}$$

Then we can define the ratio of bandwidth between the highest bandwidth channel and the lowest bandwidth channel as $R = 2^{n_c - 1}$(Vicisano 1997). With this, we are now prepared to describe the scheduling algorithm used by the SO scheme. Let us note that $s_j(i)$ denotes the i^{th} packet in the j^{th} session; likewise, $c_j(i)$ denotes the i^{th} packet on the j^{th} channel(Vicisano 1997). We now define the packet ordering for the highest session using the function of the form s_{n_c-1} = (block, packet) (excerpted from (Donahoo 1998) and (Vicisano 1997)):

$$s_{n_c-1} = \left(\left| i + \left\lfloor \frac{i}{\max(B, R)} \right\rfloor \right|_B, \left| |i|_R + \left\lfloor \frac{i}{BR} \right\rfloor R + \left\lfloor \frac{i}{nB} \right\rfloor \right|_n \right) \tag{2.2}$$

These block-packet pairs are then divided up among the sessions as follows (again, excerpted from (Vicisano 1997)):

$$s_0(i) = c_0(i) \tag{2.3}$$

$$s_j(i) = \begin{cases} c_j\left(\left\lfloor \frac{i}{2} \right\rfloor\right) & |i|_2 = 0 \\ s_{j-1}\left(\left\lfloor \frac{i}{2} \right\rfloor\right) & \text{otherwise} \end{cases} \tag{2.4}$$

From this, we can see that every even numbered (that is, i is an even number) packet is transmitted on the highest channel whereas the odd packets are passed onto the next session down. At that session, every other packet (that is, packets 1, 5, 9, etc.) is transmitted on that the second highest channel while the rest are again passed down.

2.1.4.2 *PO Scheme.* In response to a number of potential drawbacks perceived in the SO scheme, Donahoo proposed the PO scheme(Donahoo, Ammar, and Zegura 1999). The PO scheme, while still making use of layered channels, cyclic transmission, and FEC, takes a radically different approach to the problem of scheduling the sending of packets on multicast channels from that used by the SO scheme (and the BC scheme as will be seen in Section 2.1.4.3). To understand the approach taken by the PO scheme it is first necessary to recall from our discussion of FEC in Section 2.1.3.2 that one useful property of FEC is that the matrix used to generate the encoded blocks consists of the I_k identity matrix in its first k rows. This causes the first k of the n generated blocks to be the unencoded k blocks which make up the original file. This fact is then leveraged with the realization that receiving those k blocks may be in many ways more beneficial than receiving the generated redundancy blocks (that is, not all blocks are created equal). Certainly it is true that any subset of k unique blocks from the n available will suffice to recover the original file but those file types which may be accessed prior to the completion of their download (web pages, some forms of multimedia, downloads consisting of multiple uncompressed files) can derive greater benefit from blocks 0 through $k-1$ since these blocks can be read and used

immediately upon receipt. To this end, the author proposes to create two sets of layered channels, original channels and redundancy channels, each set consisting of n_c channels and using uniform bandwidths rather than the exponentially distributed bandwidths common to other techniques. The set of original channels (denoted as channels $C_0^O, \ \ldots \ , \ C_{n_c-1}^O$) is used exclusively for the transmission of k original packets (that is, packets with index i such that $0 \leq i < k$) from the various blocks. More specifically, the set of total blocks making up the file being transmitted are divided into a set of $\left\lceil \frac{B}{n_c} \right\rceil$ groups $G = \left\{ G_0, \ \ldots \ , \ G_{\lceil \frac{B}{n_c} \rceil - 1} \right\}$ such that each group is a set of blocks (each of which are, in turn, a set of packets) satisfying

$$G_i = \left\{ B_{i\lceil \frac{B}{n_c} \rceil}, \ B_{i\lceil \frac{B}{n_c} \rceil + 1}, \ \ldots \ , \ B_{(i+1)\lceil \frac{B}{n_c} \rceil - 1} \right\} \tag{2.5}$$

for each $G_i \in G$ and noting that, ideally, B should be a multiple of n_c. The PO scheme then transmits all of the original packets for each block contiguously (that is k packets from a given block will be transmitted on a single channel before packets from any other block are transmitted on that channel). Receivers experiencing no loss will then be able to complete their download by subscribing to all of the original channels for one complete cycle (note, however, that it is not required to join all of the channels simultaneously). Those receivers who do experience loss, however, will then leave the original channels and join the associated redundancy channels (the redundancy channel C_i^R is associated with original channel C_i^O for some i such that $0 \leq i < n_c$). Unlike the original channels which transmit k (or, in some versions, $k+\delta$ where δ is a small integer intended to provide some redundancy packets in order to account for the nearly inevitable loss of some data) packets of a single block before

transmitting packets of a different block, the redundancy channels, much like the previously discussed SO scheme, transmit one packet from each block for which they are responsible before transmitting a second packet for any block thus maintaining the block interleaving property discussed previously. We note that this mechanism is designed so as to maximize the reception efficiency in the face of *random* loss as such loss should effect all blocks and channels equally. The manner in which this serves to address random loss is best illustrated through a counter example. If we consider a system which attempts to provide some number of packets (say y for simplicity) from a single block before providing packets of a different block in much the same way as the original channels, then we are faced with the question of what value is appropriate for y. Clearly $y = 1$ is the block interleaving approach so we must choose some greater value for y. We quickly discover, however, that the value of y is dependent upon how many packets we need to replace from a given block, a value which is receiver-dependent. Thus, any value we choose for y may be too great for some receivers and far too small for others. Additionally, some receivers may not lack any packets for a particular block thus making large values of y extremely costly for them. If instead we choose the block interleaving method, we are able to provide packets from the entire block-space for which the channel is responsible as quickly as possible and thus, hopefully, provide something useful to all recipients in a reasonably efficient manner. The observant, however, will note that if this method is targeted towards improving reception efficiency in the face of random loss it may, then, be somewhat deficient in the face of burst loss. This does, in fact, prove to be the case. The reason for this, however is related to the properties of both the original channels and the

redundancy channels. Since the original channels transmit k packets of a given block consecutively, an instance of burst loss is able to cause the receiver to miss a number of packets from the same block. This is in contrast to the same situation under a scheme which ensures block interleaving where such an instance of loss would cause a number of blocks (specifically, a number of blocks equal to the number of packets lost) to miss a packet. Since all blocks are then equally deficient, the block interleaving property of the redundancy channels proves just as useful for recovering from lost packets as in the case of random loss. Thus, as demonstrated in (Donahoo, Ammar, and Zegura 1999), the PO scheme performs quite well (particularly compared to the SO scheme) under random loss but suffers severely when faced with even small burst losses.

The second major drawback to this scheme is related to the fact that it does not require cumulative subscription. The original premise of layered multicast as described in (McCanne, Jacobson, and Vetterli 1996) was that by requiring cumulative subscription, recipients were prevented from interfering with one another by join disjoint sets of channels. The reasons for requiring cumulative subscription become quiet obvious if one considers the implications of not doing so. Let us consider the simple network shown in Figure 2.1. In this we see that the endpoints, N_0 and N_1, receive data from the source, S, by way of dedicated links and a shared link. N_0's dedicated link from the router (R) has 10Mbps of available bandwidth while N_1's link has only 5Mbps available. The two nodes share a link between the router and the source which also has an available bandwidth of 10Mbps. Let us now assume that N_0 joins a set of channels from the source such that its bandwidth capability

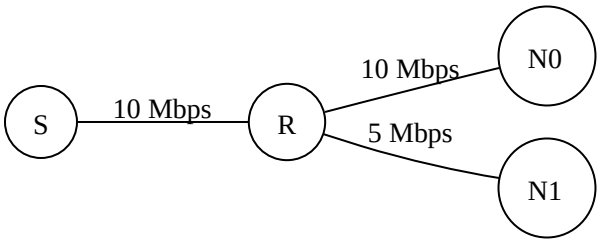


Figure 2.1: A simple network made up of a source (S), a router (R), and two receiver nodes.

is completely utilized (with minimal congestion). Now, let us assume that N_1 also which to join channels to fully utilize its bandwidth capabilities. If the source requires cumulative subscription, then N_1 will join a subset of those channels to which N_0 belongs. However, if the source does not require cumulative subscription, then N_1 may some join channels which N_0 has not joined. In the worst case, the sets of channels joined by N_0 and N_1 may be completely disjoint leading to an attempt to utilize a data rate of 15Mbps across the shared link and, thus, dramatically increasing the loss rates for both nodes. It is this ability to interfere with other receivers that makes the cumulative subscription requirement necessary. This drawback taken in concert with the problems associated with recovering from burst loss when using the PO scheme makes it unlikely to be a viable solution to our problem and thus we do not consider it further in our discussion.

2.1.4.3 *BC Scheme.* Another layered multicast scheme with FEC is the Birk-Crupnicoff (BC) scheme which was proposed in (Birk and Crupnicoff 2003). The authors of this scheme also claim to maintain the property of block interleaving (and, in fact, provide formal proof of this fact). Their only criticism of the SO scheme seems

to be that the author of (Vicisano 1997) does not provide formal proof of properties such as block interleaving though they provide no evidence that this is impossible to do. The authors of this scheme handle their packet scheduling in a somewhat different manner using two 2-parameter functions to define the schedule: $g(j,i)$ denotes the block number of the i^{th} packet on the j^{th} channel and $p(j,i)$ denotes the packet number (within the block) of the i^{th} packet on the j^{th} channel. We now present the scheduling functions (excerpted from (Birk and Crupnicoff 2003)) as contrast to those presented for the SO scheme:

$$g(j,i) = \left| \left\lfloor \frac{i}{2^{\max(0,j-1)}} \right\rfloor + \left\lfloor \frac{B}{2^j} \right\rfloor + \max\left(1, \left\lfloor \frac{B}{2^{\max(0,j-1)}} \right\rfloor\right) |i|_{2^{\max(0,j-1)}} \right|_B \quad (2.6)$$

$$p(j,i) = \left| \left\lfloor \frac{i}{2^{\max(0,j-1)}B} \right\rfloor + \left\lfloor \frac{n}{2^j} \right\rfloor + \max\left(1, \left\lfloor \frac{n}{2^{\max(0,j-1)}} \right\rfloor\right) \left| \left\lfloor \frac{i}{B} \right\rfloor \right|_{2^{\max(0,j-1)}} \right|_n \quad (2.7)$$

From this we can see that the scheduling techniques used by the SO and BC schemes generate schedules which are quite similar (though obviously not identical). It is for these reasons that we are initially skeptical of the originality of the BC scheme, a view which the results presented in chapter 4 will serve to confirm.

2.2 Peer-to-Peer

Despite the progress made in terms of developing a system capable of reliable and efficient bulk data transport over multicast, the widespread deployment of such solutions has not occurred. The primary reason for this, as well as the primary reason multicast is rarely leveraged in general, is the fact that network-layer multicast is presently unavailable in the Internet and will almost certainly remain so for the foreseeable future. We note that this is not an implementation failing as virtually

all technologies are capable of multicast, rather it is a business decision to simply disable that functionality. In response to this, a number of other ideas have been proposed. These systems rely upon the end-users in the network to store, serve, and disseminate the file. Nodes joining such networks are encouraged to help others in addition to themselves through fate sharing mechanisms which lead to misbehaving nodes be ostracized from the rest of the download community. These systems are generally termed *peer-to-peer* networks.

2.2.1 Workings of a Peer-to-Peer System

As stated previously, application-layer multicast networks consist of trees of nodes in the network in their most simplistic form. More advanced forms of these systems seek to create meshes whereby nodes establish not only multiple downstream connections but multiple upstream connections as well. This leads, of course, to the formation of cycles within the network, making it now possible to receive multiple copies of the same packet. This has in turn led to the development of the more well-known form of peer-to-peer networks which rely upon the fate-sharing model while at the same time providing peers greater freedom in terms of the manner and order in which their download will proceed. Under these systems, peers (the term for nodes in a peer-to-peer network) chose to download specific pieces of the file from the other peers to whom they are connected rather than having the data simply transmitted to them in an order defined by the original source. This conversion from a push to a pull model also allows a number of other improvements. Peers are now free to establish as many connections as their individual bandwidth capabilities allow, thus eliminating

one of the major issues facing single-source/multiple-receiver data delivery. Likewise, many networks, such as BitTorrent (Cohen 2003), allow peers to request information from the peers they are connected to regarding which packets those peers possess and thus avoid the problem of downloading duplicate data. Additionally, these networks are now no longer completely dependent upon the source node to disseminate the data; as long as the source node successfully places one complete copy of the data into the network (spread across any number of peers), it is theoretically possible for the peers to still successfully complete their download even if the original source leaves the network permanently.

2.2.2 Concerns with Peer-to-peer

It must, of course, be noted that peer-to-peer networks are not without their own drawbacks and deficiencies. One of the most glaring the problem of users being unable to complete their download because no source peers exist and the network as a whole no long maintains a complete copy. While stated previously that it should be theoretically possible to recover the file even if the source is unavailable, this is not in practice always the case. Since peers are free to come and go within the network at anytime, it is quite possible for a peer which holds the only copy of a given packet to leave the network and not return. In this event, the data is simply lost and those peers attempting to download the file will find that they are incapable of completing the download. Accordingly, most peer-to-peer systems implement some mechanism to combat this problem even if they cannot eliminate the possibility entirely. Under the BitTorrent peer-to-peer protocol (Cohen 2003) which has risen to the forefront

as the most popular such protocol, users attempt to discern (based upon information from those they are currently connected to, known as their *neighborhood*) the rarest block (or, at the very least, the rarest block within their neighborhood known as the *local rarest*) and then seek to download that block. This is intended to have the effect of increasing the number of copies of rare blocks in order to ensure that they do not become extinct.

Another problem which is related to the block extinction problem is the well-known *last block problem*. This situation, which is known to be fairly common on peer-to-peer networks (Tian, Wu, and Ng 2006), occurs when a peer has successfully downloaded all of the required packets for a given download except one. The reasons this occurs may vary, for example, it may be the case that the packet remaining is one which is rather rare across the entire network (that is, it does not occur in very many neighborhoods and is rare in those in which it occurs). Regardless of the reason, however, it has been shown that the last block often requires a significant time expenditure (particularly when compared to the per-packet time required for the previously received packets). The solutions to this problem are not well known though it is generally hoped that the same mechanisms which are intended to prevent block extinction will work to lessen the effects of the last block problem as well.

Peer-to-peer systems, despite their drawbacks, have thus far emerged as the most practical and efficient solution readily available to the single-source/multiple receiver bulk data delivery problem. We next move to introduce the information theoretic concept of network coding which is believed by some to be a technique that

can be applied to peer-to-peer networks in order to address many of the drawbacks associated with their use (Gkantsidis and Rodriguez 2005).

Having now examined the history of the efficient single-source/multiple-receiver reliable bulk data transport problem as well as a number proposed solutions to it, we move on to consider our own implementation of network coding, an information theoretic concept that has been proposed as a method to possibly alleviate many of concerns related to the use of peer-to-peer networks. The reasons for favoring network coding over the multicast solutions relate to the greater availability of network coding at present (that is, network coding can be used on any network whereas the use of network-layer multicast is highly restricted). We will, however, return to the question of which method is superior when we present the results of our work in chapter 4.

CHAPTER THREE

Efficient Network Coding

In theory, network coding possesses many desirable characteristics. In practice, however, the operations involved are computationally complex leading to concerns that, while potentially solving many of the problematic issues found on peer-to-peer networks, network coding may not be practical for use on end-user machines. In this chapter, we focus on creating an implementation of network coding which is practical for use on an end-user system. Accordingly, we seek to present those design decisions which impact the computationally intensive aspects of network coding (i.e., encoding and decoding) rather than the data transmission aspects of network coding in a peer-to-peer environment. We begin by examining naïve network coding in detail and then present a description of improvements we have made in order to counteract the effects of many of the previously discussed concerns. Finally, we present an overview of our actual implementation.

3.1 Naïve Network Coding

In this section, we examine the information and coding theoretic concept of *network coding*(Fragouli, Boudec, and Widmer 2006) by which intermediate nodes in a network (not just the original source) are able to produce encoded blocks of data for dissemination to other nodes with a high probability of such data being useful. This is suspected to provide many of the same benefits to concept of peer-to-peer networks

as did the introduction of FEC and other techniques described in Section 2.1.3 to network-layer multicast. Next we consider prior work that has been done regarding the integration of peer-to-peer networks and network coding in an effort to better approach a solution to the aforementioned efficient delivery problem. Finally, we examine some of the concerns and criticisms which have been leveled against network coding as a technique for use in peer-to-peer networks.

3.1.1 Description

Network coding is a powerful technique for encoding information in such a way as to allow interior nodes in the network upon which the information is being transmitted to produce newly coded packets (Fragouli, Boudec, and Widmer 2006). Specifically, network coding accomplishes this by producing and transmitting *random linear combinations* of the original, unencoded packets. The process behind this is fairly straightforward even if it is not exactly obvious. We consider now a file which is divided into n block vectors (be they encoded or not), $B_i = \begin{bmatrix} b_{i_0} & \dots & b_{i_{m-1}} \end{bmatrix}$ (where $0 \leq i < n$), of a predetermined block size, m. To encode a block, we begin by picking a coefficient vector (made up of random elements), $V = \begin{bmatrix} v_0 & \dots & v_{r-1} \end{bmatrix}$ (where r is the number of blocks that have been received so far; note that for the original source $r = n$). Next we compute a linear combination of all of the blocks received so far (which are unencoded blocks for the source or other encoded blocks for the intermediate nodes) to produce a coded block, E, as follows:

$$E = \sum_{i=0}^{r-1} v_i B_i \tag{3.1}$$

It is important to note (as it is required for the final step of the generation process) that it is necessary for recipients to store the coefficient vectors they have received as well as the encoded blocks with which they are associated in order to be able to successfully decode the file and we must now create a linear combination of those coefficient vectors to transmit along with the encoded block that has been created. The previously received coefficient vectors are stored as the row vectors of our coefficient matrix, C. We create the new coefficient vector by computing V', the linear combination of the coefficient vectors received so far as follows (note that for the source, these coefficient vectors are simply the row vectors making up the identity matrix I_n):

$$V' = \sum_{i=0}^{r-1} v_i C_i \tag{3.2}$$

It is this coded block/coefficient vector pair, (E, V') that is then transmitted to a recipient node (Fragouli, Boudec, and Widmer 2006). It is also worth noting at this point that, in order to avoid the problems of overflow inevitable when repeatedly adding and multiplying, these calculations are, in practice, carried out in a finite field. For our experiments, we have chosen the sixteen-bit finite field $GF\,(2^{16})$.

Once a node has received a coded block/coefficient vector pair, it still must verify that the coded block is useful (usually referred to as *innovative* in the network coding literature (Gkantsidis and Rodriguez 2005)). This verification is performed by determining if the newly received coefficient vector is linearly independent from those which have already been received; if it is, then the encoded information is innovative, if it is not then the coded block may be discarded. It is for this reason (as well as

the requirements of the decode operation to be discussed shortly) that the received coefficient vectors must be retained in the form of a coefficient matrix (note, however, that only those coefficient vectors which are associated with innovative blocks need be kept). Also note that it is possible to choose our coefficient vectors during the encoding process in such a way as to cause a block that is, in fact, innovative to appear useless. The obvious (and trivial to prevent) example of this is in the event where the coefficient vector chosen is the zero vector which will, quite obviously, result in a coded block/coefficient vector pair which is useless. While this case is easy to prevent (and is prevented in our implementation), other coefficient choices may (in the context of the other coefficients which have been previously chosen by other peers) generate a false-negative result and cause an innovative block to be discarded. It has been noted that such instances should be rare with a sufficiently large finite field (Gkantsidis and Rodriguez 2005).

After a node has received n linearly independent blocks, it can begin the process of decoding the file. The first step in this process is to compute the inverse of the coefficient matrix C. It is worth briefly noting that since we have previously only accepted those encoded blocks (and, more importantly, coefficient vectors) whose associated coefficient vectors were linearly independent with respect to the previously received vectors and since the matrix C is a square n-by-n matrix, it is guaranteed to have an inverse. We now must also adjust our view of data from a series of blocks to a matrix, B, whose row vectors are the received blocks. Having inverted the coefficient matrix and established the existence of the "data matrix", B, we now compute the decoded matrix, $D = C^{-1}B$ and note that for the original source in

which the blocks were never encoded, $C = C^{-1} = I_n$ and $B = D$. Finally, we observe that concatenating the row vectors of the decoded matrix, D, together, one after another, forms the original, decoded, file.

3.1.2 Network Coding in a P2P Environment

As described in Section 2.2, there are a number of concerns associated with the use of peer-to-peer networks for reliable bulk data transport. One solution intended to address some of these concerns (particularly the last block problem) is the introduction of erasure coding at the source. This concept, borrowed from similar ideas proposed for use on layered-multicast channels in (Vicisano 1997), (Donahoo, Ammar, and Zegura 1999), and (Birk and Crupnicoff 2003), has been previously described in Section 2.1.3.2. These solutions, which rely upon coding at the source alone, do not, however, address the issue of duplicated blocks (it is still quite possible for a few blocks to be exchanged so often as to become vastly more common than the other blocks) nor the associated problem of having to determine which blocks are available in the neighborhood that the node does not already possess. Additionally, FEC comes with its own set of drawbacks associated with its computational complexity that require the block size to be kept small (on the order of 1 KB) and require n to be a small number (generally not more than 32)(Rizzo 1997). These two requirements taken together mean that only the smallest files may be coded in their entirety with the remainder being divided into a multitude of partitions each of which is coded separately from the others (reintroducing a form of the last block problem, albeit a probably far less common form).

The introduction of network coding into the peer-to-peer environment seeks to solve or, at least, simplify a number of the problems left unresolved by encoding at the source alone. Like digital fountain (Byers, Luby, Mitzenmacher, and Rege 1998), network coding requires no choice regarding the number of encoded blocks that can be generated to be made *a priori* (though this number can be affected by the choice of finite field size). Far more importantly, however, is the fact that, as described in Section 3.1.1, network coding eliminates the need to be aware of every block possessed by every member of a given node's neighborhood, the need for selectively choosing a block from a neighboring peer, and the need for selection mechanisms such as local rarest. Since every node produces a linear combination of all of the blocks which that node has previously received, there is only a single block available to choose from so we need no longer be concerned with attempting to avoid duplicates of other blocks. It is true, however, that network coding does not eliminate completely the need for nodes in the network to require *some* information about blocks available within their neighborhood. The required information is, however, greatly reduced as it is now necessary to know only whether or not a peer possesses at least one block which is linearly independent from those the receiving peer has already acquired. As described in (Gkantsidis and Rodriguez 2005), this can be accomplished by having the sending peer transmit its coefficient matrix to the receiving peer or, in a far more efficient solution, the sending peer may simply generate a new coefficient vector (just as it would during the block generation process) which the receiving peer may then use to check for linear independence. This, quite obviously, reduces the amount of information that must be transmitted in order for a peer to decide that its neighbors

possess innovative information and removes the need for complicated block selection algorithms and heuristics.

3.1.3 Performance Concerns

Despite the many benefits of network coding associated with improved efficiency of block delivery and a simplified peer-to-peer network protocol, a number of concerns have been raised regarding the practicality of this technique, especially in terms of the computational complexity associated with the encoding and decoding of the file(Wang and Li 2006). In regards to the generation of an encoded block, we must create a linear combination of every block we have so far received. For the original source, this means that every block in the entire file must be read, multiplied by a coefficient, and added together in order to generate a single encoded block. For very large files, this clearly becomes a potential bottleneck and may have the effect of limiting the rate at which encoded blocks can be produced, thereby increasing the time required to complete the download as compared to a non-coding solution if the time to generate the encoded block is greater than the time required to transmit it onto the link.

One solution which some authors have proposed in order to improve the performance of the encode operation (and thereby the rate at which encoded blocks can be produced) is the introduction of a new parameter which they term *density*(Wang and Li 2006). The density parameter is a measure of how many blocks from the file will be used in the generation of a new block. We think of this as some positive integer $q \leq n$ (in fact, density is actually the ratio of q to n in the form $\frac{q}{n}$ but for ease of discussion we will often take it to mean merely the parameter q). Having chosen q,

we now pick a set of random indices $Q = \{Q_0, \ \ldots, \ Q_{q-1}\}$ such that $0 \leq Q_i < n$ for $Q_i \in Q$. Next we pick a set of data blocks, B', from those blocks currently available such that $B' = \{B_{Q_0}, \ \ldots, \ B_{Q_{q-1}}\}$ (and noting that $|B'| = q$). We then choose our random coefficient vector, V, such that it contains q elements and then calculate the coded data in a fashion similar to that described in Equation 3.1:

$$E = \sum_{i=0}^{q-1} v_i B'_i = \sum_{i=0}^{q-1} v_i B_{Q_i} \tag{3.3}$$

Likewise, we similarly randomly select a set of coefficient vectors, C', from our coefficient matrix, C, and then compute it in a manner similar to Equation 3.2:

$$V' = \sum_{i=0}^{q-1} v_i C'_i = \sum_{i=0}^{q-1} v_i C_{Q_i} \tag{3.4}$$

This technique, then, has the potential to decrease the time required to generate a new encoded block by relaxing the requirement that such encoded data blocks be linear combinations of every data block in the file. The drawback to this, however, is that research shows that as density decreases, the number of linearly dependent blocks generated increases(Wang and Li 2006). In chapter 3, we will demonstrate how we are able to eliminate the need for this parameter through the use of a technique we term *folded-block encoding*.

A far more frequently discussed problem is the complexity associated with the decode operation. Recall that the process of decoding the file requires the inversion of the coefficient matrix, a calculation with a computational complexity of $O\left(n^3\right)$(Gkantsidis and Rodriguez 2005) (recalling that n is the number of blocks in the original file). Following this computation, the new inverse coefficient matrix must be multiplied by the encoded data requiring a single matrix-matrix multiply of the

n-by-n inverse coefficient matrix, C^{-1}, with the encoded data matrix, B, which is n-by-m. This results in a computational complexity of $O(mn^2)$. It is important to note, however, that m is usually far larger than n, in which case the matrix multiplication step computationally dominates the matrix inversion. We will demonstrate how this fact may be leveraged to improve this performance in Section 3.3. Clearly, however, the computational complexity of the decode operation (particularly as described thus far) has the potential to increase the download time (measured from the time a peer first begins attempting to download blocks for a file until the time when the file is completely decoded) significantly, possibly even to the point of consuming any performance increases achieved at the network level.

One proposed solution to the potential difficulties posed by the complexity of the decode operation (and, in its naïve form, the encode operation as well) is to partition the file into small groups known as *generations*(Gkantsidis, Miller, and Rodriguez 2006) which will be encoded and decoded independently of one-another. This has the effect of essentially performing network coding on a number of small, independent files and then simply concatenating them together after each has been decoded. If the same block size as would be used without generations is maintained, then this process effectively reduces the block count used in the proceeding computational complexity analysis. The drawback to this method, however, is that as the block count per generation decreases (that is, as the number of generations used increases), the number of coded blocks that can be generated with potential for innovation decreases to the point that having a number of generations equal to the block count is essentially a non-coding solution.

3.2 Encoding Improvements

The requirement that every block received thus far, which is potentially the entire file (as well as the entire coefficient matrix), be processed every time a new encoded block is generated has the potential, particularly when using large generations, to make the block generation process a serious bottleneck to the performance of a peer-to-peer network coding system. This performance degradation occurs if we are unable to produce blocks at a rate at least equal to the available upload capacity. Consider the process of encoding a new block at the source, which, by definition, must possess n blocks. In order to accomplish this, we must multiply every element of the first block by a random scalar, an operation that is $O(n)$ and then add these elements to the second block, another $O(n)$ operation, after similarly multiplying its elements. This process must then continue for every block in the file resulting in a total complexity of $O(n^2)$. Thus the complexity of the encoding operation quickly grows as n gets large.

Recall from Section 3.1.3 that some researchers have proposed a technique based upon a density parameter in an effort to decrease the number of blocks that must be used for encoding and, therefore, achieve a higher encoding rate. We employ a similar method for generating encoded blocks that is effectively a hybrid between the use of every block received thus far and a random subset of those blocks. Our system maintains a coefficient vector register in which we accumulate the coefficient to send. The value of this register at any given time is the coefficient vector V_i where the subscript i refers to the last vector from the coefficient matrix, C, to be "folded" into the coefficient vector register. Similarly, we maintain an encoded data block

register whose value is the encoded block vector E_i. Thus, the first block/vector pair that we create is (E_0, V_0) which is created by picking a random scalar value, s_0, and then computing $E_0 = s_0 B_0$ and $V_0 = s_0 C_0$. We then maintain both of these values and increment the subscript (from 0 to 1). The next time we are asked to supply a block/vector pair, we compute the encoded block E_1 and the coefficient vector V_1 by picking another random scalar value, s_1, and then calculating $E_1 = E_0 + s_1 B_1$ and $V_1 = V_0 + s_1 C_1$. We are then able to transmit the block/vector pair (E_1, V_1) and increment our subscript, i, to 2. It is this process of maintaining a "running" block and coefficient vector that we refer to as *folded block encoding*. This technique offers the clear benefit of allowing us to process the file one block at a time while eventually including every block in the file as a linear combination within a single encoded block.

More formally, we recursively define V_i using the randomly chosen scalar s_i:

$$V_i = \begin{cases} \vec{0}, & r = 0 \\ s_i C_i, & i = 0 \text{ and } r \neq 0 \\ V_{i-1} + s_i C_{|i|_r}, & \text{otherwise} \end{cases} \tag{3.5}$$

Similarly, the encoded data block is defined recursively as follows (relying upon the same random scalar s_i as was used in Equation 3.5):

$$E_i = \begin{cases} \vec{0}, & r = 0 \\ s_i E_i, & i = 0 \text{ and } r \neq 0 \\ E_{i-1} + s_i B_{|i|_r}, & \text{otherwise} \end{cases} \tag{3.6}$$

It is worth noting that, unlike (Wang and Li 2006), we do not require the introduction of the density parameter to our system. Recall from Section 3.1.3 that some have suggested the introduction of a parameter known as *density* which is essentially a percentage of the number of blocks that will be chosen randomly to be linearly combined into the encoded block. It could be argued that our choice of utilizing only a single additional packet from the file for any given encoded block would correspond to a density value of $\frac{1}{n}$. This, however, is incorrect as such a density setting would, in fact, not be network coding at all but would consist simply of selecting a block from the file, multiplying it by some random scalar value and then transmitting the "encoded" packet. However, since our system maintains the previously generated block/vector pair and combines (or folds) an *additional* packet (or coefficient vector, as the case may be) into this previously generated pair, this is not the case. Indeed, if any claim regarding the density used in our implementation is to be made, it is that the density in our system is variable rather than fixed with each subsequently generated encoded block using successively higher densities. Since this is implicit in the encoding technique definitions which have already been presented and as this definition changes density from an independent variable or parameter to a variable property of the system dependent upon the index of the packet currently being encoded, we do not consider the potential need for this parameter further.

3.3 Decoding Improvements

We now turn our attention to the decode process, which is known to be computationally intensive and, as such, can require a significant portion of time to complete.

This effectively increases the required download time of a file as, until the decode process has completed, the file is of no practical use to the user. In an effort to lessen the overhead in terms of the time required for the decode operation, we have considered a number of improvements that are now detailed. Recall that the decode operation as described in Section 3.1.1 requires two main steps: first we must invert the coefficient matrix and, second, we must multiply the encoded data matrix by the inverted coefficient matrix. It is upon this second step which we focus our efforts towards reducing the decode time and, accordingly, we now detail the various improvements we have attempted for this process.

3.3.1 Decompositional Matrix Multiplication

Recall from the naïve implementation described in Section 3.1.1 that the decode operation requires the multiplication of the encoded data matrix by the inverted coefficient matrix. We also note that the naïve implementation for multiplying two square matrices is known to have an asymptotic computational complexity of $O(n^3)$. Accordingly, the first improvement which we consider is the use of a technique adapted from Strassen's algorithm(Strassen 1969)[1] , which we term *Decompositional Matrix Multiplication* (DMM), to decrease the complexity and, therefore, the running time, of the process of multiplying the inverted coefficient matrix and the encoded data matrix. This works by recognizing, like Strassen's work, that matrix multiplication is made up of a series of repeated adds and multiplies of both the elements and

[1] We choose Strassen's algorithm as our starting point because while faster algorithms exist, such as Coppersmith-Winograd(Coppersmith and Winograd 1987), which is $O\left(n^{2.376}\right)$, they are considered impractical for use due to extremely large constants hidden by the asymptotic notation.

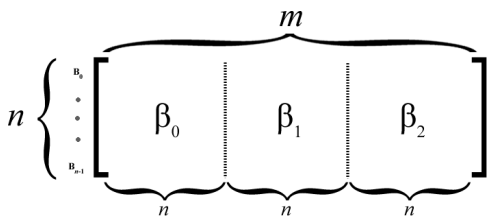


Figure 3.1: An example of how the encoded data matrix, B, is divided into $\lceil \frac{m}{n} \rceil$ n-by-n matrices labeled β_i for $0 \leq i < \lceil \frac{m}{n} \rceil$

the submatrices that make up the matrix. Accordingly, we are able to recursively subdivide the matrix and make use of this process to precompute a number of the values that will be used repeatedly throughout the multiplication. We are able to gain further benefit by recognizing that we will be performing multiplication using the same left-hand matrix (the inverted coefficient matrix) repeatedly. In this, we are able to achieve a computational complexity of approximately $O\left(n^{2.81}\right)$. This does have the potential drawback, however, of requiring that both of the matrices used for multiplication be square.

We recognize that since our encoded data matrix, B, is an n-by-m matrix, it may not be suitable for use with DMM. However, instead of considering this as a single n-by-m matrix, we see that it is also possible to treat it as several ($\lceil \frac{m}{n} \rceil$ to be specific) n-by-n matrices with the last matrix possibly requiring one or more columns of padding to handle the situation where m is not a perfect multiple of n. We can now leverage this fact to allow us to make use of DMM for multiplying square matrices together and, hopefully, achieve some performance gain by doing so. As show in Figure 3.1, we will divide B into several n-by-n matrices, β_i (where $0 \leq i < \lceil \frac{m}{n} \rceil$). We are now able to compute the decoded data matrix, D, by recognizing that D

can be similarly divided (and padded as necessary) resulting in $D = [\delta_0| \cdots |\delta_{\lceil \frac{m}{n} \rceil - 1}]$ (we refer to the matrices δ_i and β_i as *decoded submatrices* and *encoded submatrices*, respectively). This will then allow us to compute the decoded submatrices as follows:

$$\delta_i = C^{-1}\beta_i \tag{3.7}$$

Since we have now reduced the decode operation to a series of multiplications of square matrices, we are able to leverage DMM.

It is important to note, however, that this improvement is not without some overhead of its own. The use of DMM places some additional constraints upon the parameters chosen for our file, namely the block count. In order for the recursive implementation to successfully divide the submatrix, it is necessary for the dimension of the matrix to be a power of 2 multiplied by some number less than or equal to the "base" matrix size[2] ; for our implementation, the base matrix size has been selected to be 16. Thus, for some positive integer $x \leq 16$, the block count, n, must satisfy $n = x2^k$ for some non-negative integer k. This additional restriction may, at times, require that additional padding be added to the file beyond what would otherwise be required if any positive integer were acceptable for the block count. Furthermore, this padding, of course, brings additional potential for computational overhead. However, we believe that the speed-up experienced due to the use of this technique will more than make up for any extra overhead required as a result of this additional constraint for reasonable choices of block count and block size.

[2] the size below which we simply perform the naïve version of matrix multiplication

3.3.2 Buffering

One of the more serious drawbacks to use of DMM on encoded submatrices is the fact that it results in a significant amount of additional overhead for disk I/O. Since the file is stored in block order (that is, one block after another), the fact that each submatrix contains only a portion of each block (specifically, n times the size of the field bits), the process of reading a submatrix into memory can require up to n seeks to be performed on the disk: one for each row of the submatrix. The entire decode process can then require up to $\lceil \frac{m}{n} \rceil (n)$ seek operations to be performed on the disk. We can compare this to the number of seeks that would be required if we instead read and decoded a single block at a time in its entirety would require n seeks (one per block). We have already established that the process of reading a single submatrix requires n disk seeks and therefore, the decode process as modified to use encoded submatrices and as described thus far can never perform fewer seeks than the more naïve implementation.

In order to address this problem, we propose the rather obvious solution of reading multiple submatrices at a time in order to decrease the number of times this process is required. Quite obviously, this still will not allow us to perform fewer disk seeks than required under the naïve implementation but it does potentially offer some performance advantage over the unbuffered solution. Accordingly, our implementation is designed to accept a user-specified parameter indicating a maximum amount of memory that may be used to buffer encoded submatrices in order to improve the speed of the decode operation.

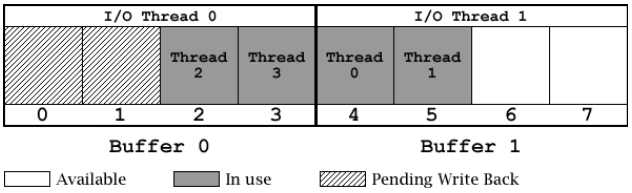


Figure 3.2: An example of the setup of the buffers and how the various threads interact with them during intra-generation parallel decoding (assuming four computational threads).

3.3.3 Parallel Decode

Having implemented both the improved matrix multiplication technique and submatrix buffering, we now turn our attention to parallelization of the decode process. Given the growing prevalence of multi-core systems, the ability to take advantage of true parallelism and its obvious potential for performance increase cannot be ignored. Therefore, we have implemented in our network coding system a parallel decoding process. We believe that the decoding process lends itself to a high degree of parallelization as, once the inverse coefficient matrix has been calculated, each submatrix can be decoded completely independently. Further, the matrix multiplication operation is known in its naïve form to have a computational complexity of $O(n^3)$. We believe that this complexity will dominate the I/O operations for all but the most trivial values of n and thus allow us to glean a significant performance increase from the use of parallel decoding. To this end, we choose to perform parallel decoding within a single generation (intra-generation) rather than across multiple generations as it is our belief that this will promote a number of desirable properties such as locality of reference as we will be able to use the same, precomputed, inverse coefficient matrix across all threads of execution. Accordingly, the parallel decode is accom-

plished by creating a multi-partition buffer[3] , each partition of which is capable of holding one submatrix for each thread that will be used. Thus, in total, the number of submatrices that will be buffered will be twice the number of threads we intend to make use of. Associated with each partition is also a thread which is dedicated to performing disk I/O operations for that partition and is, thus, referred to throughout our discussion as an *I/O thread*. These threads are in contrast to the threads that will actually perform the matrix multiplication associated with the decoding process and are correspondingly referred hereafter as *computational threads*. Our system begins the decode process for a given generation by inverting that generation's coefficient matrix and then spawning a predetermined number of computational threads. Each thread then requests a submatrix from the buffer and decodes the matrix. Once all of the submatrices in a given partition have been decoded, the partition is written back to the disk and, unless the end of the file has been reached, it is refilled with encoded submatrices. This allows each computational thread to continually decode submatrices (as long as they are available) without the need to pause for disk I/O. An example of the workings of this process is shown in Figure 3.3.3 which illustrates a potential (and typical) situation from a decode process involving four computational threads.

3.4 Overview of Implementation

The system is designed with a relatively simple interface that permits the retrieving and inserting of encoded blocks, the retrieving and checking of coefficient

[3] in our implementation, we use a two partition buffer, the partitions of which will be referred to as "buffer 0" and "buffer 1"

vectors for the purposes of determining if a peer possesses innovative blocks, and the decoding of the file. It is also able to supply basic information such as the block count but, as these methods are trivial, we do not discuss them in detail. Further, our implementation is designed to support the use of generations as described in Section 3.1.3. Recognizing that the introduction of generations may also introduce the need for the file to be padded in order for every generation to be of equal size and recognizing that said equal size must be $n \times m$ where n is the block count and m is the block size in bytes, we have chosen to place all of the padding at the end of the last block of the last generation in order to prevent the overhead associated with removing padding from the middle of the file. The handling of generations is designed in such a way as to ensure that, at the coding layer, the generations are treated completely independently of one another. This is achieved through the use of a generation network coder which is responsible for performing all coding operations on a particular generation. Thus each generation maintains its own coefficient matrix and concept of which blocks have been processed in a manner independent from the other generations in the file. This technique also lends itself to parallelization at the application level. Thus, the system is designed so that it is possible for an application to read/write from/to one generation without regard to what (if any) operations are being performed upon the other generations in the file.

File storage, in contrast to the independent view of generations used by the coding logic, is handled somewhat differently. The concept of a file is not, in fact, present within the network coding logic itself; rather, the network coding logic is aware only that it has a "data-store" provided to it capable of reading and writing

data blocks (encoded or not) to whatever underlying system is desired. Thus, while the coding of the generations is carried out in a completely independent manner at the level of the network coding logic (and may, for all that the system knows, be stored in completely different locations), the data-store is aware of each generation and maintains a single, unified view of the data. This allows each generation to be written to different positions within the same file, thereby eliminating the need to concatenate multiple small files together after the decode process, and thus hopefully reducing the number of costly seek operations that must be performed on the disk.

Another aspect of network coding implementations which has received some discussion is the choice regarding the size of the finite field. Some past authors have chosen the sixteen-bit finite field $GF(2^{16})$(Gkantsidis, Miller, and Rodriguez 2006) while others have favored the eight-bit finite field $GF(2^8)$(Wang and Li 2006). We have implemented our system in such a way as to make either option available. However, the sixteen-bit finite field offers a significantly decreased chance of generating dependent blocks through bad coefficient choices and thus we favor the sixteen-bit field. Accordingly, all of our experiments, presented in Chapter 4, have been carried out using the sixteen-bit version of the system unless otherwise noted.

While the implementation just described is capable of performing network coding as described in Chapter 2, it is also hindered by the concerns regarding computational complexity enumerated in Section 3.1.3. In order to alleviate these difficulties, we have made a number of improvements, especially with regard to the encode and decode operations, that we will now detail beginning with the improvements made

to the encoded block generation process. The results of experiments performed upon these improvements will then be presented in Chapter 4.

3.5 Summary

Thus, we have now presented our implementation for network coding which we believe to be highly optimized and efficient. We have attempted to describe a mechanism for generating encoded blocks which we term *folded-block encoding*, and believe to be less computationally intensive and, correspondingly, less expensive in terms of time required than the naïve method originally proposed for use with network coding. More importantly, we have paid a great deal of attention to the need to improve the decode rate of the system through several improvements. The realization that the encoded data matrix can be converted to the concatenation of several square (specifically n-by-n) matrices combines with the adapted technique for improved matrix multiplication to create a system which should perform significantly better than the more traditional approach, particularly as the block count grows large. Additionally, the techniques for buffering and parallel decoding are expected to further improve the performance of the system (in the case of parallel decoding, we would expect to see linear improvement in performance with respect to the number of threads utilized and the number of processors available for use on the executing system). The next chapter will focus on detailing the experiments we have performed in order to verify (or discount) the anticipated effects of our improvements. The results of these experiments will be presented where we will describe first the effects of folded-block encoding and then examine the effects of each of the decoding improvements as they

are successively added to the system. We will also seek to perform a comparison between network coding in a peer-to-peer environment and the layered and encoded multicast systems described in Chapter 2.

CHAPTER FOUR

Experimental Results

We begin by considering the performance of our implementation of network coding described in Chapter 3. In this regard, we examine both the effects of our optimizations and improvements upon the encoding performance as well as the decoding performance and draw comparisons between the results of our work and the results presented in previous works. We then prepare for a comparison of the required download times between a network coding peer-to-peer system and a multicast system by returning to the issue of layered and encoded multicast at which point we seek to determine the question left unanswered in Chapter 4 of what, if any, difference exists between the SO and BC schemes and, thus, which scheme is more appropriate for use in our comparison with network coding or, put more simply, which of the two schemes performs better. Finally, we present a description of our network simulation system, including a detailed explanation of the protocol we have implemented for testing network coding on a peer-to-peer system (modeled after live Internet tests presented in (Gkantsidis, Miller, and Rodriguez 2006)) and then present the results of the simulations and compare these to results from comparable simulations performed using multicast.

4.1 Network Coding Performance

Having examined the work that led to the original proposal of network coding for use over peer-to-peer systems and the associated concerns with such use as well as our own ideas for alleviating some of the difficulties associated with network coding, we now present the results of our performance experiments upon the network coding implementation we have previously described. We begin by examining the rates at which encoded/decoded blocks can be produced in Sections 4.1.1 and 4.1.2 respectively. Accordingly, we have performed experiments to measure the time required for these operations under a variety of choices for the block size as well as for the block count (and thus, the file size) using the network coding implementation as detailed in Chapter 3 noting that some features (such as buffering and parallel decoding) will be introduced into the implementation later in this chapter in order to make their effects clear. The network coding implementation used in these experiments has been implemented in C++ (with the parallel portions making use of the pthreads package) and compiled with full optimization (compiler option `-O3`) and carrying out network coding-related computation in the sixteen-bit finite field $GF\left(2^{16}\right)$ (unless noted otherwise). We have performed the timed portions of these tests on a 32-bit dual CPU system using dual core 3.0 GHz Intel Xeon processors (for a total of four cores) with a 4 MB cache per processor running the Linux operating system. Finally, unless otherwise noted, all times presented (recognizing that times not explicitly stated are still implicit in the presentation of metrics such as bandwidth) refer to the wall-clock time of the operation being tested.

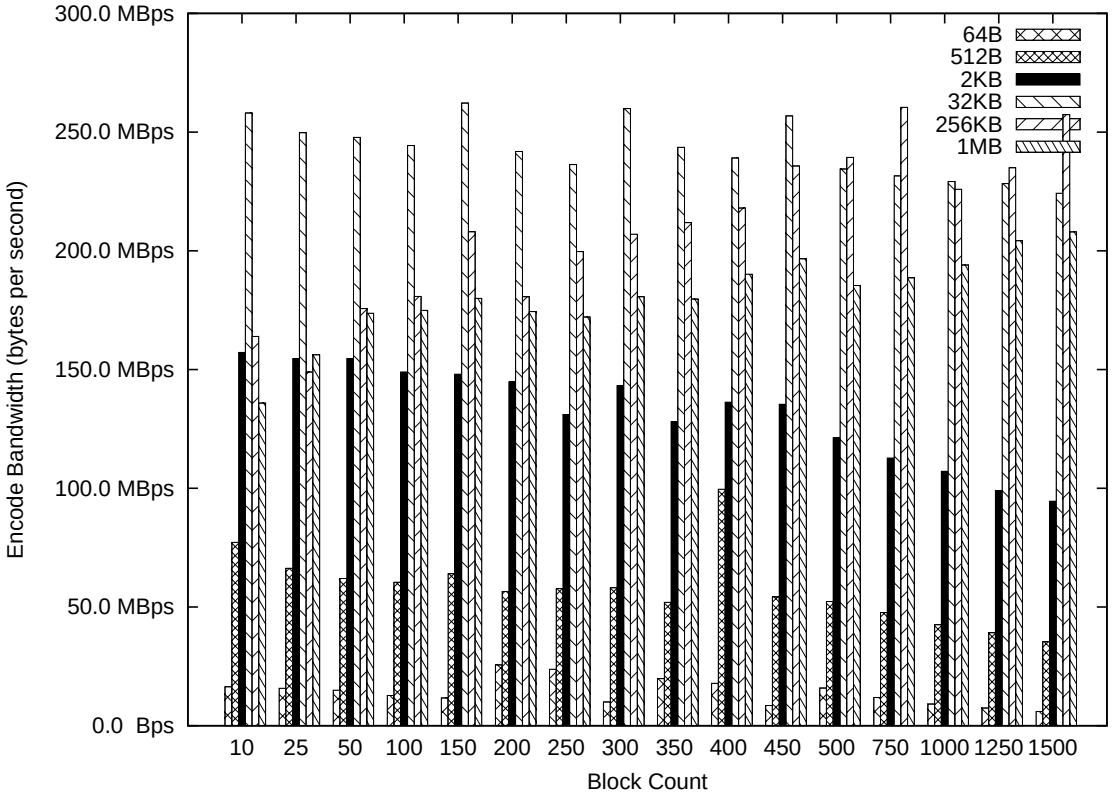


Figure 4.1. Average bandwidth to encode using *folded-block encoding*

4.1.1 Encoding

We begin by evaluating the folded-block encoding scheme described in Section 3.2, which would seem to allow a high rate of encoded block generation due to the fact that it is only required to process a single block from the file in order to generate an encoded block. In order to examine whether or not this is indeed the case, we create a single original source node and then generate 2,000 encoded blocks for block counts taken from 10 to 1500 and block sizes ranging between 64 bytes and 1MB originally chosen from work performed in (Wang and Li 2006). We compute the time required to complete the encode operation for each encoded block generate and compute the average of these times for each of the possible block count/size pairings. From this,

we then compute the rate (bytes per second) at which encoded data can be produced. The results of these experiments are presented in Figure 4.1.

In this graph, each bar represents a particular block size in bytes while the x-axis lists the chosen block counts. The y-axis presents the encoding rate, measured in bytes per second. Note that regardless of block count, the encoding bandwidth for a given block size is relatively constant. While there are some variations seen within a given block size, we believe these to be caused by a combination of random variation and caching behaviors. Further, we see that several block sizes (namely the smallest) show particularly low encoding rates with respect to the other block sizes; we believe that this is due to the fact that the cost of reading a block from the file is spread over the size of the block during computation and is thus much more apparent in small block sizes.

It is worth noting that for all but the smallest block sizes, the encoding bandwidth exceeds 50Mbps, well above the average bandwidth of DSL and cable connections which are the most common types of connections found in these networks(Saroiu, Gummadi, and Gribble 2003). Thus, these results indicate that the typical user on peer-to-peer networks can generate encoded blocks faster, using this method, than the speed at which they can transmit them onto the network. This is in stark contrast to the naïve network coding generation method which experiences a significant drop in encoding rate as the block count increases(Wang and Li 2006). This then serves to demonstrate that it is, in fact, possible to generate encoded blocks in a manner which is practical for use in a peer-to-peer environment.

4.1.2 Decoding

Next we consider the performance of our optimized and improved decode process. For these experiments, we create a simple network consisting of a single, unencoded source and a single client, which receives and, eventually, decodes encoded blocks from the source. We consider only the time required to actually perform the decode operation and present this in the form of the rate decoded data is produced (measured in bytes per second); the time to receive the file is not considered as we wish, at this time, to evaluate the performance of the decode phase only. We evaluate performance for the same block count/size pairings as were used for the encoding experiments in Section 4.1.1 and present the average of conducting each experiment five times. We evaluate the efficiency of each decode optimization beginning with DMM. We then consider the addition of buffering and, finally, parallelization to the system. Finally, we present the effects of parameter choice on decode rate and demonstrate that block size, rather than block count, is the parameter to explicitly choose‘ in order to maximize the performance of the system.

4.1.2.1 *Effects of Optimized Matrix Multiplication.* We begin by evaluating our implementation of network coding using only the optimized matrix multiplication technique described in Section 3.3.1; thus, we perform no buffering and use an entirely serial version of the system. Figure 4.2 shows the bandwidth of the decode operation under these circumstances. As can be seen, the decoding bandwidth remains relatively high for the larger block size values. Small block sizes perform so poorly at high block counts because our matrix multiplication algorithm's requirement for square

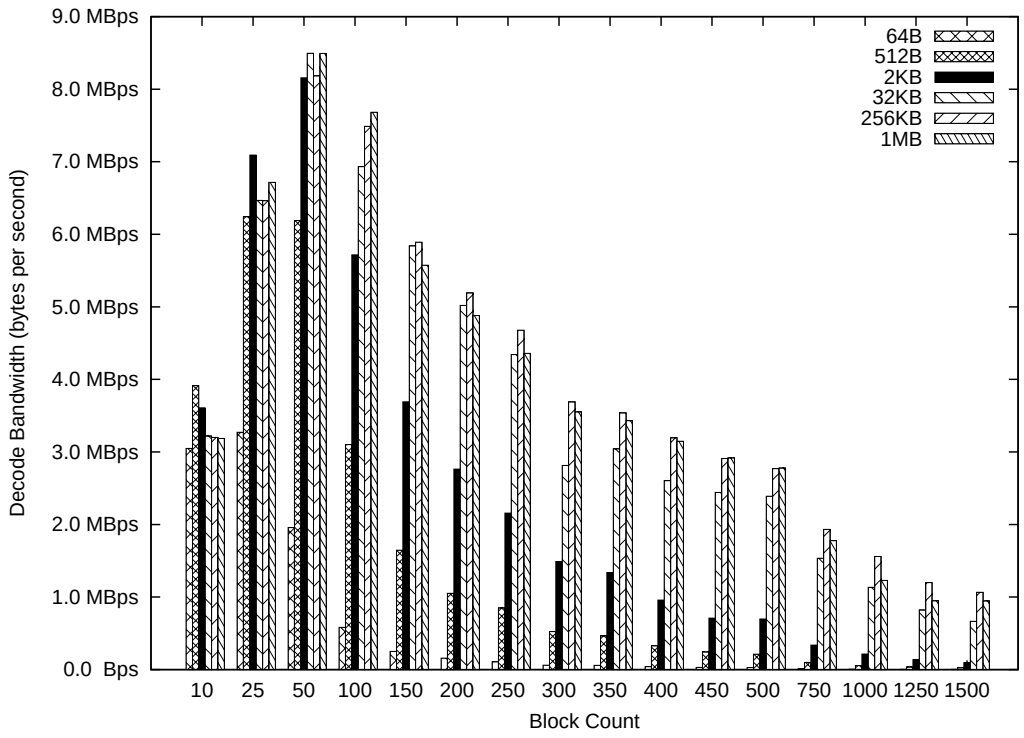


Figure 4.2: Average bandwidth to decode in $GF\,(2^{16})$ using only DMM as described in Section 3.3.1 (no buffering or parallelization) for a variety of configurations and file sizes

matrices. In those situations where the block size is less than the square of the block count (multiplied by two in the case of a sixteen-bit finite field), a significant amount of padding may required to make the matrix square. For example, using a block count of 1500 with a block size of 64 bytes results in a matrix which is 1500×32 when in fact we require a matrix which is 1500×1500 elements in size[1] . Thus, the actual data only fills approximately 2% of the decode matrix. We do not, however, consider these to be reasonable choices for the parameters and assert that the block size should be much larger than the block count for all cases for which network coding is to be considered practical. In fact, we note that under these circumstances, the overhead

[1] because our field is 2 bytes in width, each element within the matrix is 2 bytes in size resulting in 32 elements

associated with network coding, such the transmission of the coefficient vector, is actually greater than the data block size itself.

We next consider the effect of reducing the finite field size on the decode rate by repeating the previous experiments using the eight-bit finite field $GF(2^8)$. These experiments have been performed under exactly the same circumstances (in terms of parameter choices and methodology) as those in Figure 4.2 with the exception of the differing choice for finite field size (Figure 4.2 makes use of $GF(2^{16})$). Note that in all cases the block size is given in bytes as opposed to units relative to the size of the finite field; therefore, the block sizes in these experiments are the same as those in Figure 4.2. However, due to the differences in field size, the blocks in these experiments contain twice as many *elements* as the blocks in the latter. We consider this to be a reasonable comparison because we maintain the same parameters between the experiments thus providing a comparison of the performance of sixteen and eight-bit finite fields upon the same files. Figure 4.3 presents the results of decoding experiments performed in $GF(2^8)$.

We see from comparing the two figures that the decision to cut the size of the finite field in half similarly decreased the decode rate by approximately half in virtually all cases. This is unsurprising as the choice of block count is known to dominate our computational cost due to the one-time matrix inversion, which is $O(n^3)$, and the multiple matrix multiplications, which each have computational complexity of $O(n^{2.81})$. Since our matrix multiplication relies upon dividing the file into $\lceil \frac{m}{n} \rceil$ n-by-n matrices that are then multiplied by the n-by-n coefficient matrix, the only aspect affected by the choice for field size is the number of these matrices created. This then

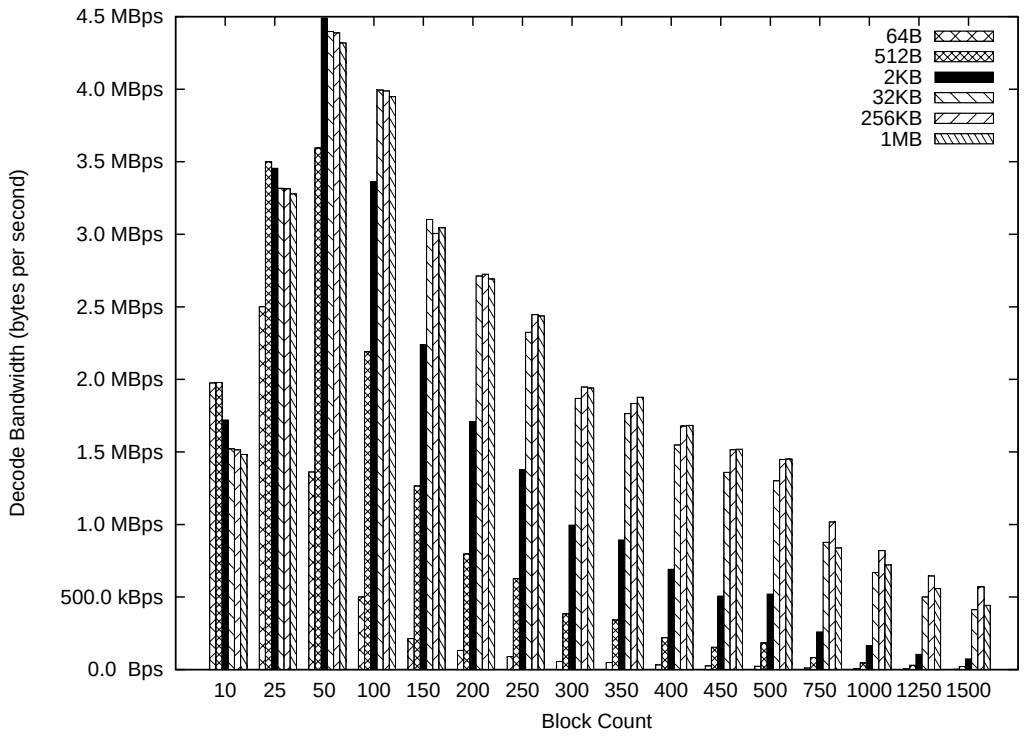
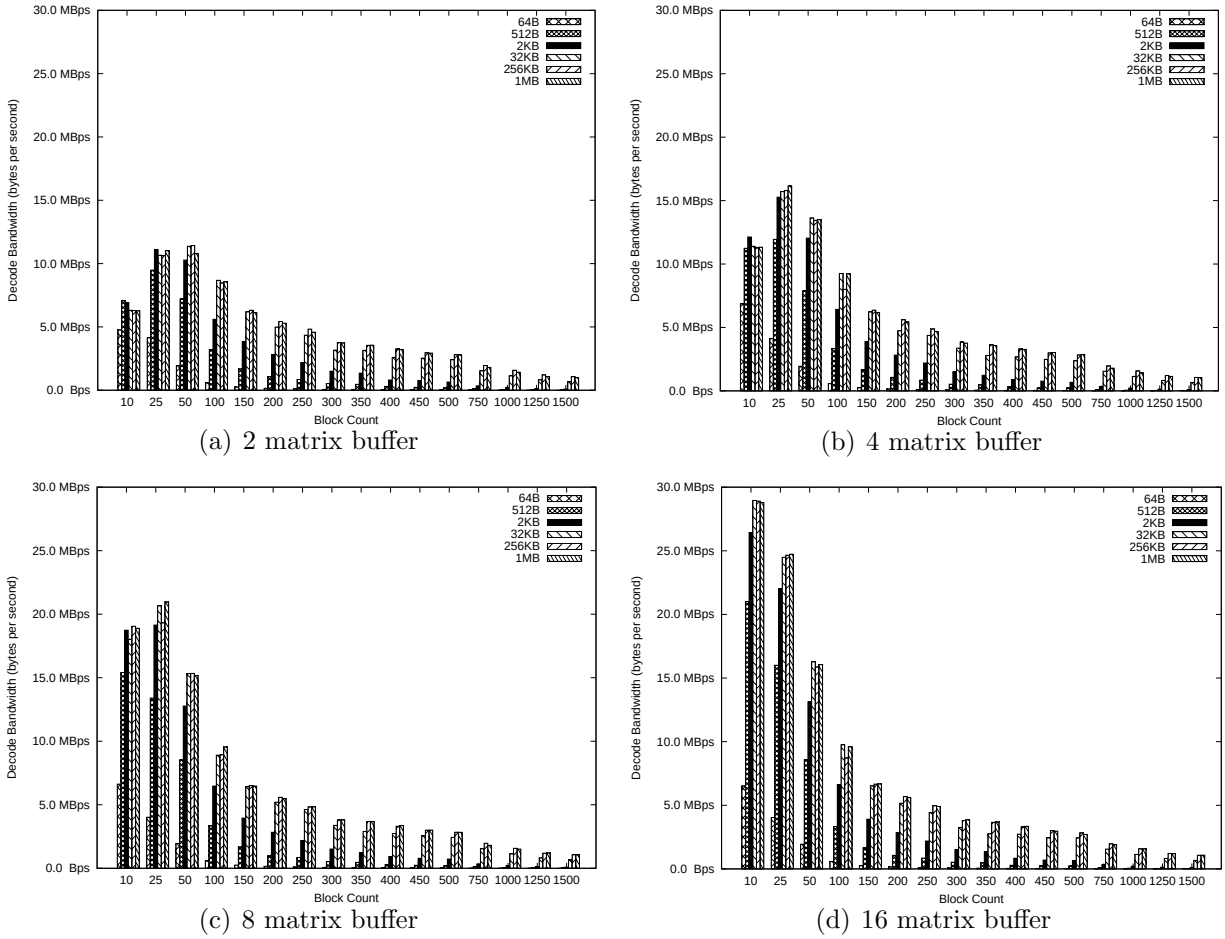


Figure 4.3: Average bandwidth to decode in $GF(2^8)$ using only DMM as described in Section 3.3.1 (no buffering or parallelization) for a variety of configurations and file sizes

has a *linear* effect upon the decode complexity, which, as has already been noted, is dominated by the actual costs of multiplication, and accordingly, the decoding rate. This analysis, then, supports the results from comparing Figures 4.2 and 4.3, which show a linear decrease in decode rate for the decrease in finite field size. We note that this is only one of the benefits we receive by favoring the use of a sixteen-bit field. Additional benefits are associated with the large field also include a greater potential for innovation due to a decreased chance of collision as a result of the field's greater size.

4.1.2.2 *Effects of Buffering.* Next we consider the effects of adding buffering to our implementation as described in Section 3.3.2. Accordingly, we rerun the set of experiments for Figure 4.2 with the addition of a parameter for buffer size chosen to be 2, 4, 8, or 16 matrices. Here we specify the size of the buffer in terms of matrices (in order to ensure that useful and relevant buffer sizes are used for every configuration). Specifying a buffer size in bytes for the experiments may result in buffers that were too large for some configurations (entire file held in the buffer) or were too small for other configurations (buffer too small to hold even a single submatrix). Figure 4.4 presents the results of the buffer experiments. From the graphs, we can see that as the buffer size increases, the rate at which decoded data is produced increases dramatically for small files (or, more importantly, small block counts) with some experiencing a rate over 7 times the base rate when a buffer of 16 matrices is used. However (and far more importantly), we see that beyond the relatively small block count of 50, the dramatic rate increases disappear almost entirely; at this point, the size of the buffer appears to have virtually no effect on the rate at which decoded data is produced. We believe this to be the point at which the computational costs of the decode operation dominate the I/O costs so thoroughly as to make the I/O costs a negligible expense in comparison to the actual act of decoding. Accordingly, the choice of buffer size has no discernible effect as the buffering of the data is intended only to increase performance in the face of I/O bottlenecks and does little to reduce the computational cost of the decode operation.

In effort, however, to alleviate the I/O bottleneck for small block counts, we have implemented another improvement to the buffering system. Recognizing that a



(a) 2 matrix buffer

(b) 4 matrix buffer

(c) 8 matrix buffer

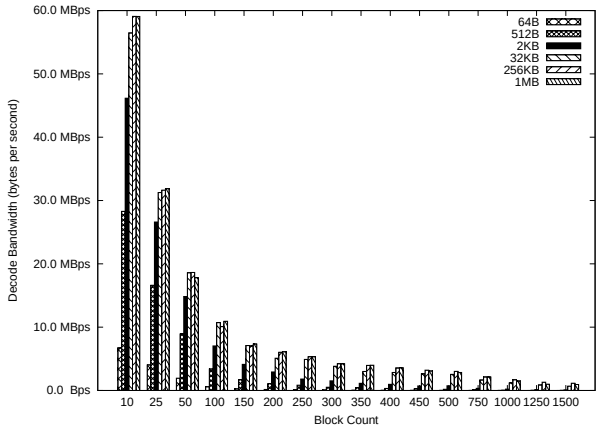
(d) 16 matrix buffer

Figure 4.4: Average bandwidth to decode using DMM and buffer sizes of 2, 4, 8, and 16 matrices (no parallelization).

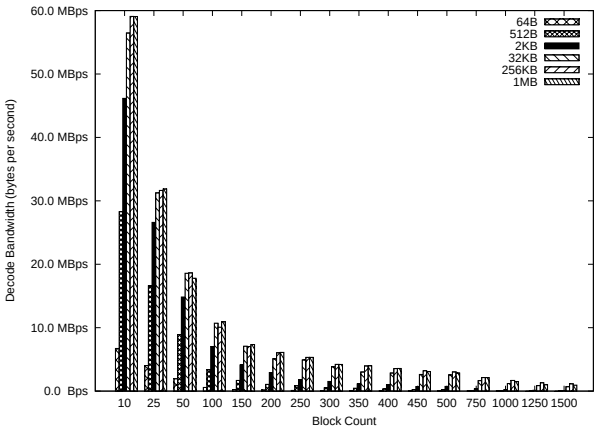
block count of 10 results in 10 reads of 20 bytes each in order retrieve a single matrix and that this is a highly inefficient use of the disk, we have reimplemented the data-store to favor reads of a size sufficient to more fully utilize the disk. Accordingly, the buffer parameter is now ignored by our system unless the length of a single row of the buffer (that is, the length of a row in the matrix times the number of matrices to be buffered) exceeds a predetermined value which in our case is 4096 bytes. If the proposed buffer size it too small to result in a read of this size, then the buffer size is increased to allow reads of this length or the reading of the entire file

into the buffer; whichever is smaller. The results of running the system under this configuration are shown in Figure 4.5. We note that all of the trials now result in virtually indistinguishable performance. The reason for this is that for small block counts, the disk utilization mechanism is suppressing explicit selection of buffer size and thus utilizing the same buffer size across all experiments while at the higher block counts, the computational bottleneck is again preventing any difference between the buffer sizes from becoming apparent. However, given the vastly superior performance experienced through the use of this technique, this is the implementation that we will continue to utilize throughout the remainder of our discussion (with the addition of parallelism in the next section).

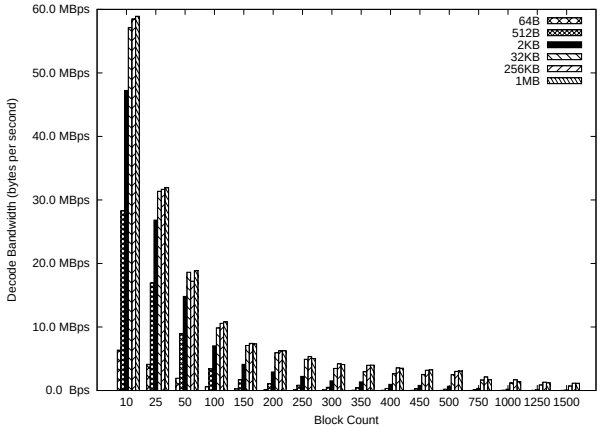
4.1.2.3 *Effects of Parallel Decoding.* Clearly the decode process is dominated by computational costs, rather than I/O costs; therefore, we explore the gain from parallelization. Thus, we repeat our experiments with a parallelized decode process as described in Section 3.3.3. For these experiments, the parameter previously introduced for buffer size has been replaced by a new parameter which indicates the number of computational threads that should be utilized. For this parameter, we have chosen to use values of 1, 2, 4, and 8 threads. Recall that, in every circumstance, we have two threads devoted to processing I/O; therefore, these configurations reflect the use of 3, 4, 6, and 10 threads in total. We present the results of these experiments in Figure 4.6. Examination of Figures 4.6(b) and 4.6(c) reveal that, for block counts greater than 50, the addition of threads linearly improves the performance. We also note that smaller block counts at times experience performance increases but these



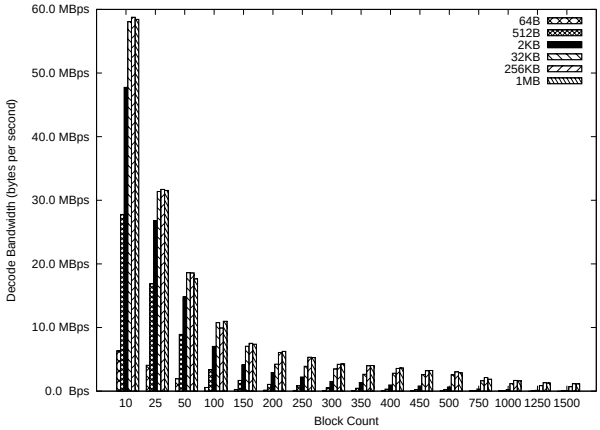
(a) 1 matrix buffer



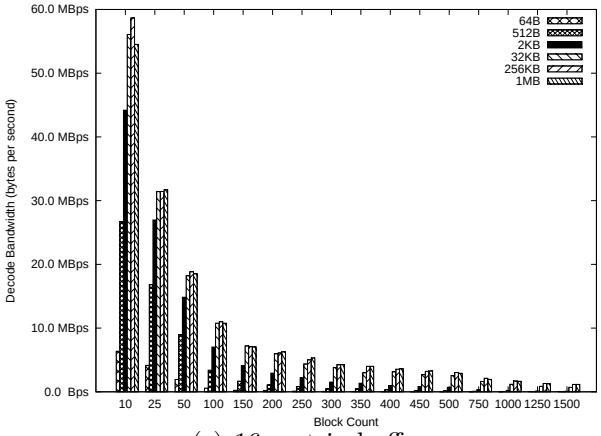
(b) 2 matrix buffer



(c) 4 matrix buffer

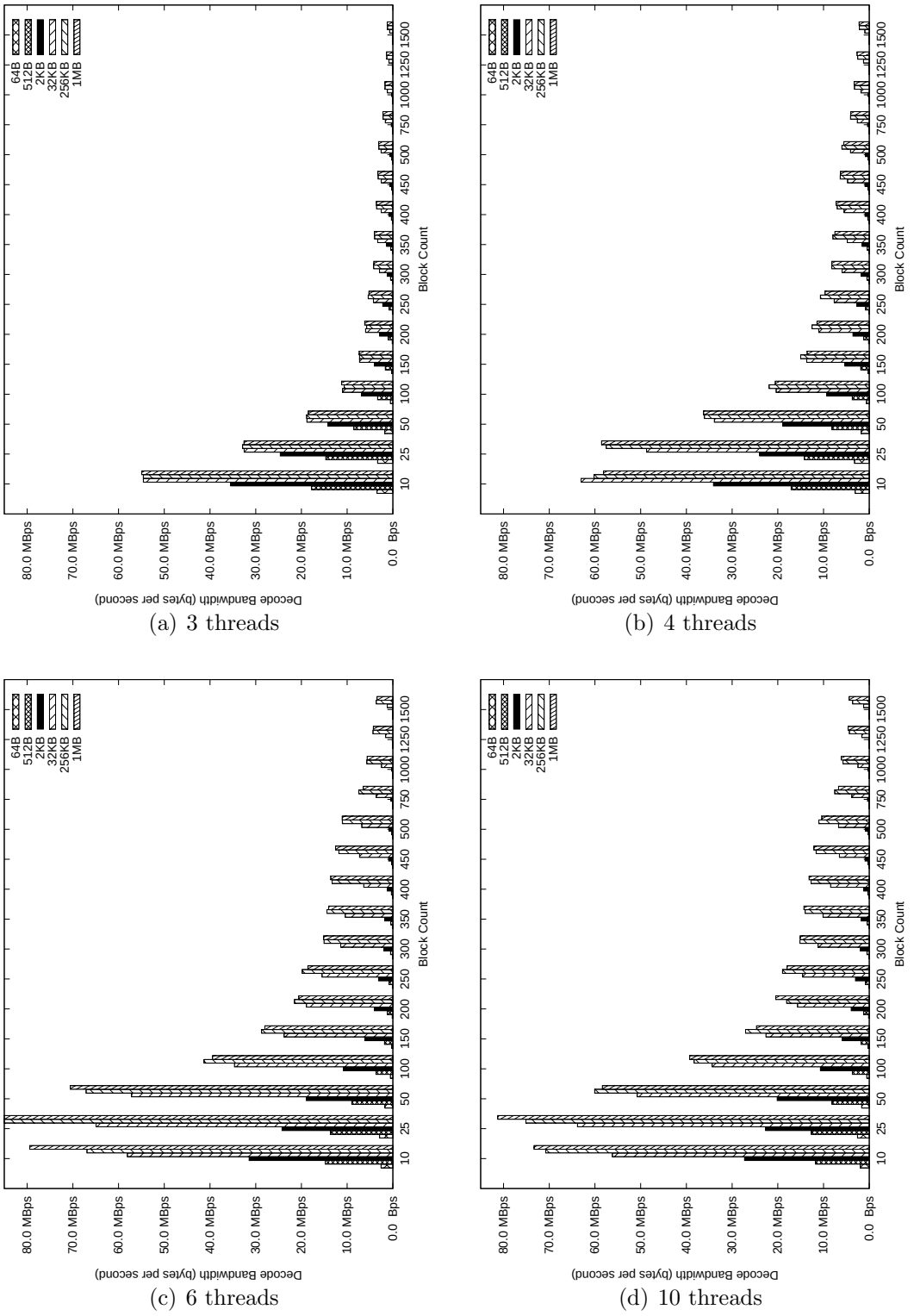


(d) 8 matrix buffer



(e) 16 matrix buffer

Figure 4.5: Average bandwidth to decode using DMM and buffer sizes of 1, 2, 4, 8, and 16 matrices (no parallelization).



(a) 3 threads

(b) 4 threads

(c) 6 threads

(d) 10 threads

Figure 4.6: Average bandwidth to decode using DMM and parallel decoding with 3, 4, 6, and 10 threads (two of which are always responsible for I/O, the remainder are used for computation).

are often less than the performance increases experienced under serial computation and buffering; we believe this to be a two-fold result. First, the use of the parallel decode requires some buffering from which the performance increase arises while, second, the performance is decreased slightly by the overhead associated with creating, starting, and stopping threads and other overhead attached to parallel decoding. Finally, we note that for virtually all interesting cases, the results of Figure 4.6(d) closely match those found in Figure 4.6(c); that is, we do not see the continuation of the linear speedup experienced with the increase from 4 to 6 threads. The reason for this is that, as described previously, our experiments were performed on a quad-core system. Thus, beyond 6 threads (that is, 4 computational threads), all of the cores are already being fully utilized and no extra gain is seen from the use of additional threads. This suggests that the computational costs vastly outweigh the I/O costs, allowing the I/O threads to write and refill their buffers at a rate which ensures that the computational threads are never forced to wait for I/O operations. These results indicate that a properly designed and optimized implementation of network coding can prove practical, even with very large file sizes using high block counts, in spite of suggestions by others that such decoding can only be performed at rates closer to 20KBps (Wang and Li 2006). We further note that the addition of parallelism to the decode process yields a linear increase in performance over serial versions and consider this to be another strong argument in favor of the practicality of network coding given the current level of availability of dual-core systems and the growing availability of quad-core systems.

4.1.2.4 *Parameter Choice.* Thus far we have presented our decoding rate results across a multitude of different block counts for different block sizes, a situation which leaves implicit the size of the file being used. It is clear that for any reasonable content distribution system, the size of the file to be transmitted is not a dependent variable determined by the parametric constraints of the algorithm in use. Accordingly, we now present the results of simulations to determine decode rate using a more reasonable paradigm for the selection of network coding parameters.

Here, we decode a range of file sizes using a number of block sizes chosen *a priori*. While the block count is the most important and dominate factor in determining the performance of the decode operation within our system, we believe that is it far more reasonable to base our choices of parameters upon block size and file size; the reasons for this are fairly straightforward. It is obvious that the file size *must* be an independent variable as any useful content distribution system accepts a file of any size. The choice then is left between block count and block size as to which is more appropriately dependent. From the network coding performance perspective, it is clear that the choice of block size should be subordinate to the choice of block count. However, if we consider the networking concerns associated with using such a system on a peer-to-peer network, the reverse becomes apparent. The reason for this is that favoring block count in such a way as to minimize the computational complexity of the decode operation leads to two results since the optimal block size for decode time minimization is 1: first, a block count of $n = 1$ is essentially clear-text data transmission (that is, unencoded transmission) and thus not network coding at all while, second, attempting to place the entire file into a single block clearly results

in an unreasonably large block size for all but the smallest files. If we move from this extreme and place some minimum on block size so as to ensure that we are, in fact, performing network coding and maintain a reasonable probability of generating innovative blocks, we are still faced with a conundrum. Let us consider merely as a example that we choose $n = 10$ as the block size. While this may appear reasonable for small files where, for example, a file consisting of 10^6 bytes would have blocks of 10^5 bytes, or around 100KB, it leads to impossible situations for larger files where, following from our previous example, a file of 10^9 bytes would require block sizes of 10^8 bytes (approximately 100MB)! While this is a non-issue from a theoretical perspective, it is untenable in the network. The reason for this is that a block must be received in its entirety to be of any use to the recipient. This means that if a recipient has received 99% of a block and then the sending peer suddenly leaves the network, the data received thus far for that block is now useless. This potential problem is due to the encoded nature of the data and the requirement that such encoded data be paired with the linear combination of the coefficient vectors used to generate it. Since we must pair these two parts together in order to derive any useful meaning from the encoded data and since the encoded data was generated by a peer with a unique set of encoded blocks (we note that all sets of encoded blocks are unique in that they are all generated using different random coefficients though this clearly does not imply that they are all innovative when taken as a whole), it is impossible to recover the remainder of the encoded block from any other peer as they would be unable to generate the same encoded block as we had previously partially received. It is also impractical to ask the peer who transmitted the partial block

originally (assuming the peer eventually reappears in the network) to regenerate the block and transmit the remainder as the peer may (and is even likely to) have received and encoded additional blocks since the encoding of the block we partially received. This, then, indicates that while large block sizes help to decrease the complexity of the decode operation by causing a corresponding decrease in the block count, they become impractical as the rate of churn in the network increases and it becomes harder to completely download an encoded block before the sending peer leaves the network or simply disconnects from us. It is this need to ensure that the blocks transmitted are likely to be received in their entirety born of the issues associated with network churn that leads us to favor block size as the independent parameter despite the potentially negative effects this may have upon the decode rate.

Having now established the need to consider block size and file size as independent variables, we now move to present the results of experiments designed to showcase the effects of these parameters upon the decode rate. For these experiments, we have chosen to test four different maximum block sizes: 32KB, 128KB, 512KB, and 2MB. It is important to note that these are *maximum* block sizes. The reason there is no guarantee made that these block sizes will be used exactly is related to the parametric requirements of our optimized matrix multiplication operation. Recall from Section 3.3.1 that our block size is required to be a multiple of some value less than or equal to our base matrix size (the point below which naïve matrix multiplication is performed), s, such that $n = s2^k$ for some non-negative integer k and recalling that for our implementation, we have chosen to use $s = 16$. Thus, due to this constraint, we may be required to adjust our block size down by calculating a preliminary block

count based upon the exact maximum block size, then applying a correction factor to this block count to determine our true value of n by increasing it so that the constraint is met, and, finally, by dividing our file size by the new block count to arrive at a new (and slightly lower) block size value. Additionally, we have chosen to perform these tests on a set of files with logarithmically distributed sizes between 10^6 bytes and 10^9 bytes (inclusive). The exact file sizes chosen are multiples of the values 1, 1.6, 2.5, 4, and 6.3. Finally, it is worth noting that the experiments have been run using the parallel version of the system with four computational threads (six threads overall) and that to remain at least partially faithful to the desire to consider only those cases which are reasonable both for use in the network and in terms of decode complexity, we have capped our experiments to only test those instances where the block count satisfies the condition $10 \leq n \leq 2500$. We now present the results of these experiments in terms of decode rate (or bandwidth) in Figure 4.7.

We see from the graph shown in Figure 4.7 that beyond a certain point, the rate at which decoding can occur drops significantly though we also note that the rates often remain on track with those presented in Figure 4.6(c). Additionally, we note that the curves presented in Figure 4.7 are similar to a graph of $\frac{1}{n^3}$ which is as would be expected given that the decoding rate is known to be $O(n^3)$. Finally, it is interesting to note that at the larger block sizes, the slope of the curve appears far more gradual then for the small block sizes. The reason for this is that at larger block sizes, the corresponding block count grows much more slowly than at the smaller block sizes. Thus, the graphs for the larger block sizes resemble a graph of $\frac{1}{n^3}$ which has been stretched along the x-axis.

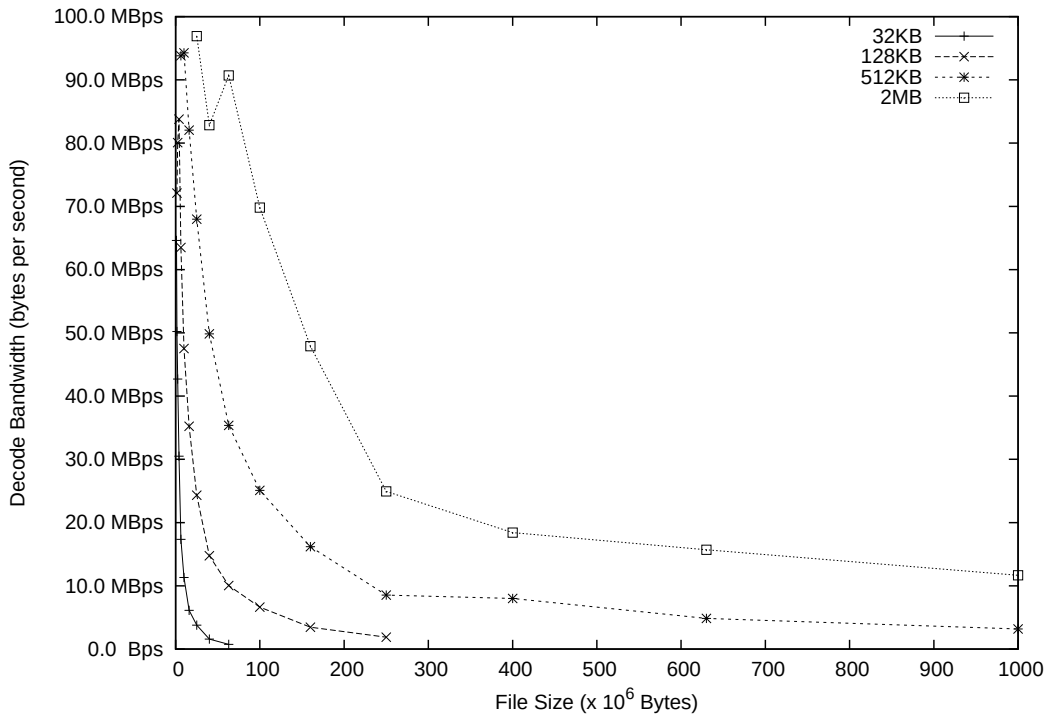


Figure 4.7: Average bandwidth to decode a variety of file sizes (shown as the x-axis) using block counts of 32KB, 128KB, 512Kb, and 2MB and four computational threads.

Finally, in order to explicitly demonstrate the practicality of this approach, both to network coding in general and parameter choice in specific, we present the average of the actual decoding times required during these experiments in Figure 4.8. The most valuable result apparent from this graph is the fact that even for a large file (nearly 1 GB) we can decode the entire file (viewed as a single generation) in approximately a minute and a half for a block size of 2MB while decreasing the block size by a factor of 4 to 512KB still yields a decode time of slightly over five minutes. It can also be seen from this graph that if we wish to maintain a particular decoding bandwidth, then this can be done by linearly increasing the block size for every corresponding increase in file size. We believe these results serve to demonstrate that

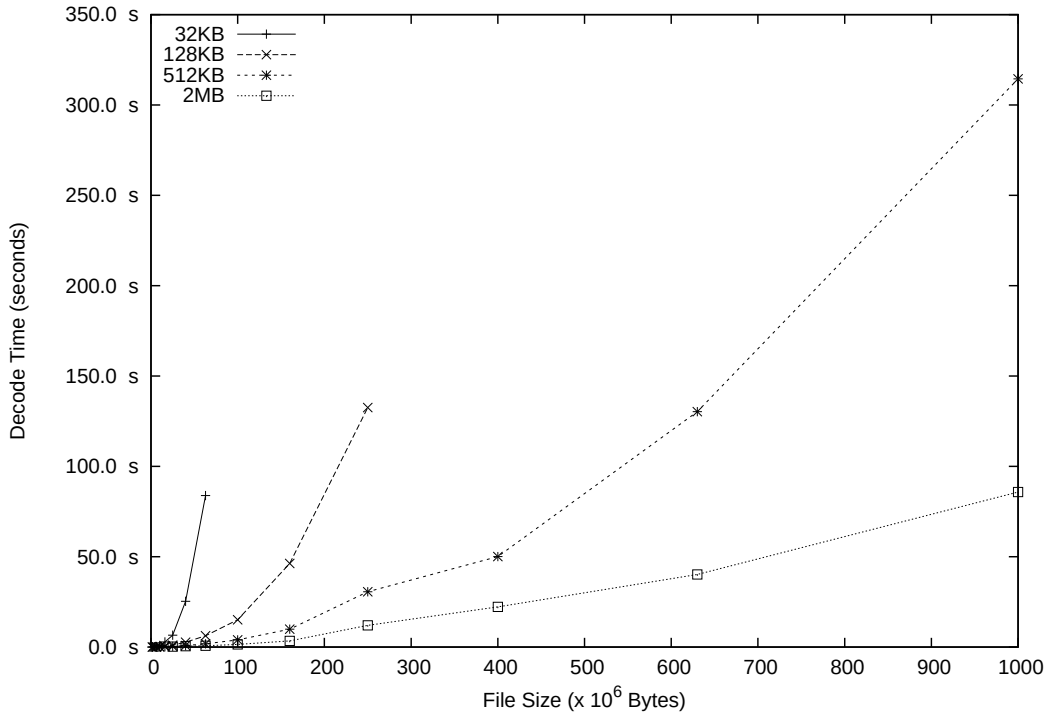


Figure 4.8: Average time to decode a variety of file sizes (shown as the x-axis) using block counts of 32KB, 128KB, 512Kb, and 2MB and four computational threads.

network coding is indeed practical if properly implemented and with proper parameter choices.

4.2 Multicast Performance Results

We now turn our attention to comparison of layered scheduling of multicast. Recall from Chapter 2 that two layered and encoded multicast schemes remain under consideration as solutions to our reliable bulk data transport problem: the SO scheme (Vicisano 1997) and the BC scheme (Birk and Crupnicoff 2003) and we recognize that both schemes attempt to provide many of the same properties. As such we now evaluate results of simulations performed upon implementations of both schemes, using the research standard network simulator NS-2, which seek to determine what,

if any, differences exist between the two. Both schemes were simulated carrying out the transfer of a $X = 4096$KB file using 1KB packets with $k = 16$ and $n = 2048$. From these parameters we can also calculate that the number of blocks required to complete the download is $B = 256$. Finally, for both schemes we choose $n_c = 8$ and simulate clients which subscribe to all eight channels. As in (Vicisano 1997) and (Donahoo, Ammar, and Zegura 1999), we perform two sets of experiments: one set using increasing levels of random packet loss and another using increasing levels of burst loss. We define random loss of percent p as being loss such that each packet transmitted is lost with probability p. In contrast to this, burst loss of percent p is defined as loss such that at some point during the transfer p percent of the total file (that is, a continuous set of packets totaling to p percent of the total file size) will be lost in a single burst(Donahoo 1998). For these experiments, we consider the reception efficiency metric introduced in Section 2.1.4.

We chose to measure and present two metrics for the schemes: reception efficiency and download time. We define reception efficiency as the size of the original file divided by the actual amount of data received(Vicisano 1997). The results of the reception efficiency measurements are presented in Figure 4.9. From this figure we can see that the two schemes perform virtually identically (in fact, there is often no statistically significant difference in their performance). Figure 4.10, which presents the time required to complete the download, presents much the same picture with the schemes performing with statistically identical results. It is based upon this evidence (as well as the multitude of other similarities between the two schemes) that we conclude that there is no significant difference between the two and it is for this

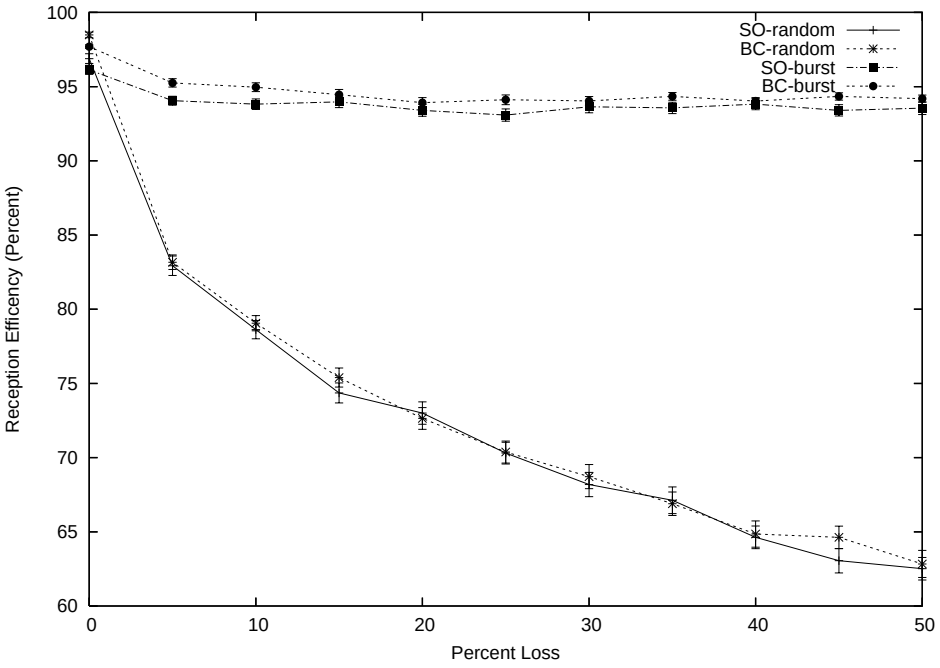


Figure 4.9: Average reception efficiency (with 95% confidence interval) of SO and BC schemes under both random and burst loss using 100 clients, $B = 256$, $k = 16$, $n = 2048$, and $n_c = 8$

reason that for the balance of this discussion we consider only the SO scheme (as it was the first to be developed).

4.3 Comparing Multicast and Network Coding

Next, having established that network coding can, in fact, prove practical in terms of its encoding and decoding rates, we now seek to compare the performance in terms of download time required between a network coding peer-to-peer system and the Session Organization scheme for layered, FEC-encoded, cyclic multicast. For this comparison, we have crafted simulations of both implementations for use with NS-2. It is important to note that the implementation of network coding used in this simulation does not actually perform any coding upon actual data; simulated

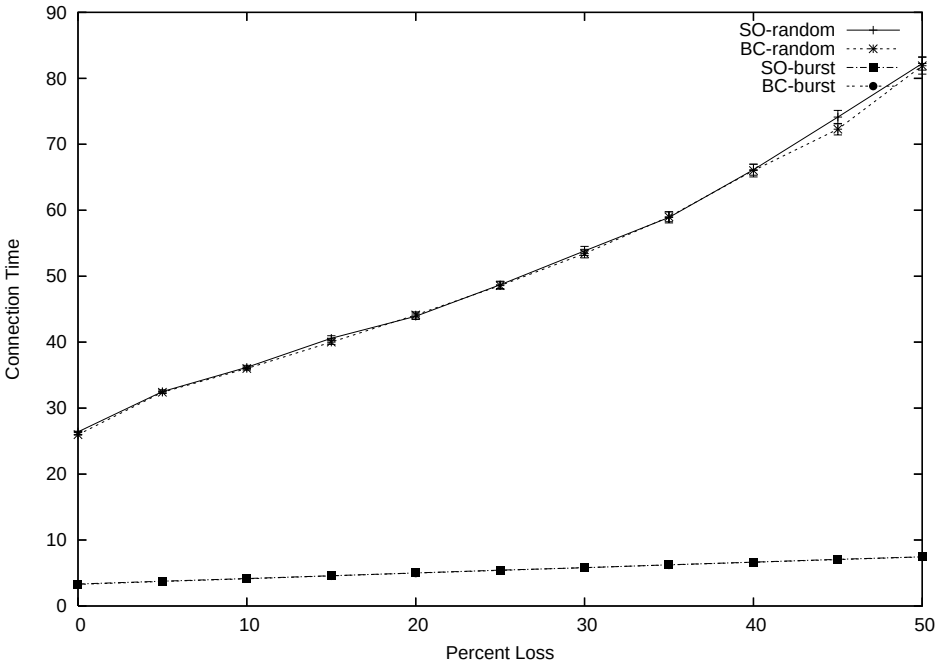


Figure 4.10: Average download time (with 95% confidence interval) of SO and BC schemes under both random and burst loss using 100 clients, $B = 256$, $k = 16$, $n = 2048$, and $n_c = 8$

data only is used. This is not considered significant as, for the moment, we are only concerned with the time to actually receive all of the data required to decode the file and not the time required to perform the decode operation itself. Thus, we concern ourselves only with those aspects of network coding, such as checks for linear independence to ensure that packets are innovative, which have an impact upon the network performance of the system (such as the number of packets one must receive and, thus, the time required to complete the download). Before presenting the actual results, however, we first present explanations of the simulated network used and the peer-to-peer protocol simulated for the network coding tests.

Table 4.1: List of link types, their associated bandwidths, and the percentage of the network they make up(Saroiu, Gummadi, and Gribble 2003).

Link Type	Probability
T3	2%
T1	5%
Cable	44%
DSL	14%
Dual ISDN (128 Kbps)	3%
ISDN (64 Kbps)	3%
Dialup - 56 Kbps	23%
Dialup - 33.6 Kbps	1%
Dialup - 28.8 Kbps	1%
Dialup - 14.4Kbps	4%

4.3.1 Network Design

The network coding simulations were carried out using a randomly generated topology created with the BRITE(Medina, Lakhina, Matta, and Byers 2001) topology generation tool. The generated topology contains 1000 nodes with relatively high-bandwidth links (compared to the links the peers themselves are directly connected to) between them, approximately 200 of which are leaf nodes; these 1000 nodes are meant to simulate the router network which forms the backbone of the Internet. From the approximately 200 leaf nodes, 72 are chosen at random to have peer-to-peer client nodes attached to them using a link with bandwidth chosen randomly from the choices shown (along with their associated probability of selection) in Table 4.1. This design is intended to recognize that end users are generally found at the edges of the network (it would be rare indeed to find an end-user directly connected to a core router). It is also at these edges that the lowest capacity links are likely to be

found as the interior of the network must handle extremely high volumes of data thus making low capacity links a non-option.

4.3.2 Simulation Design

In order to test and evaluate the network coding concept in a manner conducive to comparison with the previously described multicast schemes, we chose to implement a simulated version of it using NS-2. As stated previously, it should be noted that this implementation, as it is intended for use in a simulated environment which can be completely controlled, is not designed to handle random network failure, misbehaving peers, and/or other random or unexpected errors that a real-world implementation would be required to deal with in order to perform successfully. While these conditions are certainly possible and important to consider, their effect is outside the scope of our present research. This is not to say that the simulation is completely naïve; rather, the simulation simply does not consider error conditions. Thus, simulated peers are completely free to join and leave the network (and, in fact, do); they are simply required to do so in an orderly manner (i.e. by sending disconnection notices to all connected peers) rather than simply failing.

4.3.3 Protocol

The successful operation of any peer-to-peer system is dependent upon a well defined communications protocol to govern interaction between the peers. The protocol which has been implemented for our simulations of network coding can be divided

into several stages: connecting, testing, transferring, and disconnected. Several message types are associated with each of these stages and will now be described.

Connecting

- `INVITE` - informs the recipient that the sender would like to establish a connection with them in order that the sender might supply the recipient with data

- `CONNECT` - informs the recipient that the sender would like to establish a connection in order that the recipient might request and receive data from the sender; may be sent as a response to the `INVITE` message if the recipient of that message chooses to accept the invitation; contains the generation number for which the sender would like to receive data

- `ACCEPT` - informs the recipient that the sender has decided to accept their connection request and moves the connection to the *testing* phase

- `REJECT` - informs the recipient that the sender has decided not to accept their connection request and moves the connection to the *disconnected* phase

Testing

- `REQ_TEST` - informs the recipient that the sender would like the recipient to transmit a coefficient vector (as would be transmitted during normal data exchange) so that the sender might verify that the recipient has innovative packets to send

- `TEST` - sent in response to a `REQ_TEST` message; includes a test coefficient vector

- `REQ` - informs the recipient that the sender would like the recipient to begin sending actual data packets; sent after verifying that the test coefficient vector included with the `TEST` message is linearly independent from the vector already present at the sender; moves the connection to the *transferring* phase

- `DISCONNECT` - informs the recipient that the sender wishes to cancel the connection attempt and disconnect; usually sent after the test coefficient vector is found to be linearly dependent with the vector already present at the sender; moves the connection to the *disconnected* phase

Transferring

- `DATA` - a packet containing a coefficient vector and linearly combined data

- `STOP` - informs the recipient to stop sending data but does not close the connection; usually used when both peers are transferring data to each other and one no longer needs data (i.e. has completed their download) but is willing to continue supplying data

- `DISCONNECT` - informs the recipient that the connection is being terminated (if the recipient is sending data to the sender, it should cease immediately); moves the connection to the *disconnected* phase

Disconnected

- `DISCONNECT` - sent in response to any message received while the connection is in the *disconnected* phase except `INVITE` or `CONNECT` messages

While this protocol is certainly simplistic in nature and makes no effort to handle the multitude of failures and error conditions possible in network communications,

Table 4.2: Average download statistics for a 3.5GB file with 15 generations containing 100 packets each over the simulated network coding network

Statistic	Value
Mean	14.9 hr.
Standard Deviation	1.9 hr.
95% Confidence Interval	14.4 hr. - 15.4 hr.

we believe that it is capable of providing us with an accurate simulation of a reasonable peer-to-peer protocol. We also believe that this makes a reasonable effort to perform as would an actual protocol for use with network coding in a network upon which reliability (that is, the fact that nodes will never fail) was guaranteed.

4.3.4 Simulation Results

We now present the results of our simulations. For the network coding simulation, 72 peers participated in the network at various times (peers joined and left the network several times) over a period of 69 simulated hours (though peers do not begin arriving until hour 10 for an effective time of 59 hours). In all simulations (both network coding and multicast), the size of the file being downloaded was 3.5GB and the bandwidth of the original source is 2.5MBps. In terms of network coding, this simulation used 15 generations made up of 100 packets each. We present the average time required to download the file (measured as the time spent in the network before completing the download) for those peers who were able to successfully complete the download in the allotted time. Table 4.2 contains the results of this simulation.

Next we compare these results with the results of the multicast simulation. For the multicast simulation, we have simulated 72 multicast clients (again with

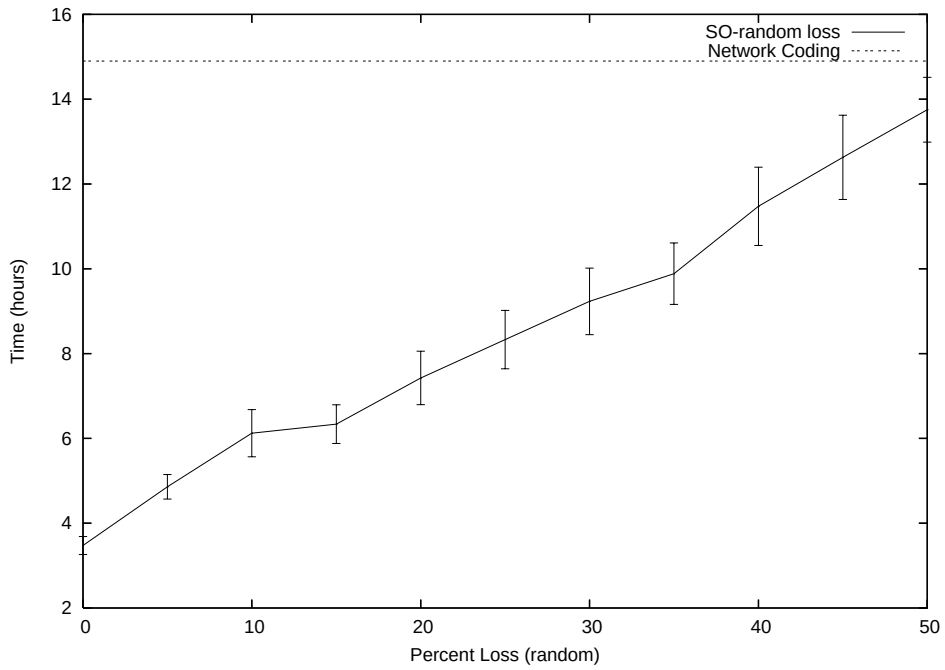


Figure 4.11: Average time in the network to receive a 3.5GB file for both the SO multicast scheme and network coding

probabilistically determined bandwidths from Table 4.1) using the SO scheme. For this we have used $k = 16$ and $n = 2048$. As usual, the packet size for these simulations is 1KB and the simulation time is capped at 59 hours. The simulations were carried out with varying levels of random packet loss on the network. The results of this simulation are presented in Figure 4.11. The figure also contains a reference line for the previously described network coding simulation (though, it should be noted that this is only for reference and the random loss was not used in the network coding simulation). As can be seen from the figure, the multicast scheme always outperforms network coding. We believe this to be partially related to the overhead associated with peer-to-peer networks in terms of locating suitable nodes to peer with where a suitable node is defined to be a peer who: one, is willing to accept an additional

connection and supply data to the connecting node and, two, has innovative data to supply.

These results, then, seem to clearly indicate that the multicast solution offers vastly superior performance over the peer-to-peer solution. This is not overly surprising (though just how much better multicast is may be) given that multicast has virtually no overhead associated with it while peer-to-peer networks are plagued by such problems. We believe that this indicates that, while we feel we have presented a practical and efficient implementation of network coding, further research into how to make network-layer multicast practical and widely available is clearly warranted given the performance benefits associated with its use.

CHAPTER FIVE

Summary

We have now presented a number of techniques for crafting an efficient and practical solution to the problem of reliably transporting bulk data from a single original source to multiple recipients with asynchronous interest in receiving the data and heterogeneous bandwidth capabilities. We have established a number of requirements that any system seeking to achieve a solution this problem must meet and have discussed both the benefits and the shortcomings of previously proposed systems. Having determined that network-layer multicast is, at present, not practical due to its lack of widespread availability in the network, we have examined in detail the technique of utilizing the information theoretic concept of network coding, which allows the interior nodes of the network to actually generate new encoded packets by linearly combining those which they have already received, on peer-to-peer networks. We have also noted that the practicality of this technique has been called into question by some due to the apparently high computational costs associated with its use.

In response to the criticisms of the network coding technique, we have developed and presented our own implementation of network coding which makes use of a number of improvements and optimizations over the original, theoretical concepts. Probably most important among these is the improvement in the decoding operation to utilize an algorithm for performing matrix multiplication which is adapted from

Strassen's algorithm (Strassen 1969). Combined with this is the realization that the decoding operation is computationally bounded for virtually all but the most trivial of block counts; a fact which, since after inversion coefficient matrix is only read (and not written to) during the decode process, allows for a high degree of parallelization in the decoding operation and thus results in a linear speed-up in terms of decode time with respect to the number of threads used for the decoding operation (and assuming a sufficient number of cores available for each thread to make use of a dedicated cores).

With our network coding system implemented, we have then presented the results of experiments to determine whether or not the system demonstrates performance that is sufficient to make it practical on end-user machines. As stated in Chapter 4, we believe that this is indeed the case. Additionally, the performance of our network coding system will continue to improve as systems with more and more cores become available due to our use of intra-generation parallel decoding.

We have also examined several multicast techniques for solving this same reliable bulk data transport issue. After examining past work regarding the use of layering, cyclic transmission, and forward erasure coding (FEC), we have presented three schemes which attempt to provide efficient and practical solutions: the SO scheme (Vicisano 1997), the PO scheme (Donahoo, Ammar, and Zegura 1999), and the BC scheme (Birk and Crupnicoff 2003). Of these schemes, we have recognized that the lack of a requirement for cumulative subscription creates a number of potential difficulties for the PO scheme and have thus, dropped it from consideration. Left to consider the SO and the BC schemes, we have recognized that there is no

significant difference between the schemes neither in terms of guaranteed/expected properties nor in terms of actual performance. For this reason, we have continued considerations of multicast schemes by considering the SO scheme only.

Finally, we have sought to compare the performance of the SO scheme with that of a network coding-based peer-to-peer network through the use of simulations. In doing so, we have discovered that the multicast solution is far more efficient, a fact which clearly indicates that work into making multicast a practical reality on the Internet should not be abandoned and, if anything, intensified. Despite this, however, we believe that network coding provides a technique for reliably transporting bulk data to a large receiver set in an efficient manner and believe that our implementation and results demonstrate that is also a practical solution.

This work does not, however, address all of the practicality concerns associated with the use of network coding. One of the most glaring holes remaining is with regard to issues of security and integrity. Due to the encoded nature of the data transmitted on a network coding-based peer-to-peer network, it is difficult to detect such problems as poisoning of the data (either intentional or accidental). Before network coding can truly be considered a practical solution, questions regarding how to guard against such issues must be thoroughly explored and answered as we have made no attempt to do so here.

The end result, however, is three-fold: first, we have established that the SO scheme for multicast transmission remains among the most efficient and practical discovered thus far. Second, we have established that it is possible to craft a practical implementation of network coding for use on a peer-to-peer network. Third and

finally, we have demonstrated that the multicast approach remains the most efficient approach presently known and thus remains worthy of active investigation. That being said, it is worth noting that there are a number of situations which lend themselves to the peer-to-peer approach even were multicast readily available. Among these are situations where it is necessary for recipients to be able to download the data long after the original source have left the network or otherwise become unavailable. As multicast provides no mechanism for accomplishing this, peer-to-peer is the only viable solution and we believe that network coding will increase the efficiency of such transmissions and, at the same time, believe that our work has provided a demonstrably practical implementation of such a system.

BIBLIOGRAPHY

Almeroth, K. C., M. H. Ammar, and Z. Fei (1998). Scalable delivery of web pages using cyclic best-effort multicast. In *INFOCOM '98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, Volume 3, pp. 1214–1221 vol.3.

Ammar, M., K. Almeroth, R. Clark, and Z. Fei (1998). Multicast delivery of web pages or how to make web servers pushy.

Birk, Y. and D. Crupnicoff (2003). A multicast transmission schedule for scalable multirate distribution of bulk data using nonscalable erasure-correcting codes. *INFOCOM 2003.Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies.IEEE 2*, 1033–1043 vol.2. IS:.

Byers, J. W., M. Luby, M. Mitzenmacher, and A. Rege (1998). A digital fountain approach to reliable distribution of bulk data. *SIGCOMM Comput.Commun.Rev. 28*(4), 56–67.

Chou, P., Y. Wu, and K. Jain (2003). Practical network coding.

Cohen, B. (2003). Incentives build robustness in bittorrent.

Coppersmith, D. and S. Winograd (1987). Matrix multiplication via arithmetic progressions. In *STOC '87: Proceedings of the nineteenth annual ACM conference on Theory of computing*, New York, NY, USA, pp. 1–6. ACM.

Donahoo, M. J. (1998). *Application-based enhancement to network-layer multicast.* by Michael J. Donahoo.; xvi, 268 leaves : ill. ; 29 cm; Thesis (Ph.D.)–College of Computing, Georgia Institute of Technology, 1999. Directed by Ellen W. Zegura.; Vita.; Bibliography: leaves 254-268.; 690: Computer science$xTheses$y1999.

Donahoo, M. J., M. H. Ammar, and E. W. Zegura (1999). Multiple-channel multicast scheduling for scalable bulk-data transport. *INFOCOM '99.Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies.Proceedings.IEEE 2*, 847–855 vol.2. IS:.

Fragouli, C., J.-Y. L. Boudec, and J. Widmer (2006). Network coding: an instant primer. *SIGCOMM Comput. Commun. Rev. 36*(1), 63–68.

Gkantsidis, C., J. Miller, and P. Rodriguez (2006). Comprehensive view of a live network coding p2p system. In *IMC '06: Proceedings of the 6th ACM SIGCOMM on Internet measurement*, New York, NY, USA, pp. 177–188. ACM Press.

Gkantsidis, C. and P. R. Rodriguez (2005). Network coding for large scale content distribution. *INFOCOM 2005.24th Annual Joint Conference of the IEEE Computer and Communications Societies.Proceedings IEEE 4*, 2235–2245 vol. 4. IS:.

hua Chu, Y., S. G. Rao, and H. Zhang (2000). A case for end system multicast (keynote address). In *SIGMETRICS '00: Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, New York, NY, USA, pp. 1–12. ACM Press.

Jacobson, V. (1995). Congestion avoidance and control. *SIGCOMM Comput. Commun. Rev. 25*(1), 157–187.

McCanne, S., V. Jacobson, and M. Vetterli (1996). Receiver-driven layered multicast. In *SIGCOMM '96: Conference proceedings on Applications, technologies, architectures, and protocols for computer communications* (Palo Alto, California, United States ed.)., New York, NY, USA, pp. 117–130. ACM Press.

Medina, A., A. Lakhina, I. Matta, and J. Byers (2001). Brite: An approach to universal topology generation. In *MASCOTS '01: Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'01)*, Washington, DC, USA, pp. 346. IEEE Computer Society.

ns (2006). The Network Simulator NS-2. http://www.isi.edu/nsnam/ns/.

Rizzo, L. (1997). Effective erasure codes for reliable computer communication protocols. *SIGCOMM Comput.Commun.Rev. 27*(2), 24–36.

Saroiu, S., K. Gummadi, and S. Gribble (2003). Measuring and analyzing the characteristics of napster and gnutella hosts.

Strassen, V. (1969). Gaussian elimination is not optimal. *Numerische Mathematik 14*(3), 354–356.

Tian, Y., D. Wu, and K. W. Ng (2006). Modeling, analysis and improvement for bittorrent-like file sharing networks. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pp. 1–11.

Vicisano, L. (1997). Notes on a cumulative organization of data packets across multiple stream with different rates.

Waitzman, D., C. Partridge, and S. E. Deering (1988, November). RFC 1075: Distance vector multicast routing protocol. Status: EXPERIMENTAL.

Wang, M. and B. Li (2006). How practical is network coding? In *Quality of Service, 2006. IWQoS 2006. 14th IEEE International Workshop on*, pp. 274–278.