August 2015

A Survey of Network Coding and Applications

Jonny L. Winger
University of Nevada, Las Vegas, wingerj@unlv.nevada.edu

A SURVEY OF NETWORK CODING AND APPLICATIONS

by

Jonny L. Winger

Bachelor of Science (B.Sc.) University of Nevada, Las Vegas 2011

A thesis submitted in partial fulfillment of
the requirements for the

Master of Science in Computer Science

Department of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
August 2015

UNLV | GRADUATE COLLEGE

Thesis Approval

The Graduate College
The University of Nevada, Las Vegas

June 4, 2015

This thesis prepared by

Jonny L. Winger

entitled

A Survey of Network Coding and Applications

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science
Department of Computer Science

Ajoy Datta, Ph.D.
Examination Committee Chair

Kathryn Hausbeck Korgan, Ph.D.
Graduate College Interim Dean

Juyeon Jo, Ph.D.
Examination Committee Member

Lawrence Larmore, Ph.D.
Examination Committee Member

Venkatesan Muthukumar, Ph.D.
Graduate College Faculty Representative

Abstract

Common networks with source, internal, and destination nodes put data packets in queues for forwarding.Network coding aims to improve network throughput and energy consumption by combining received data packets before forwarding. In this survey, we will explore various network coding schemes, along with the behavior of network coding in applications. Sensor, wireless routing, and distributed storage networks can benefit greatly from network coding implementations. Flooding is a procedure in distributed systems which broadcasts a message to all nodes in the network. NC-Flooding is introduced, which uses network coding to possibly decrease the message complexity and/or time complexity of flooding.

Acknowledgements

Thank you to my advisor and chair Dr. Datta, my family, and friends for their support and understanding.

Jonny L. Winger

University of Nevada, Las Vegas
August 2015

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

[5] shows network coding to be an alternative method to traditional store-and-forward routing; encoding and decoding schemes are implemented to increase the amount of data in each packet. Thus, higher throughput, greater efficiency, and less bandwidth usage are expected from network coding schemes. The basis of these schemes stem from analysis Dr. Ahlswede, Li, Cai, and Yeung first contributed [1].

1.1 Contribution

This report contributes theoretical analysis to the read, write, and repair operations in Cloud and Distributed Storage with Network Coding. RAID-6 and Simple regenerating codes include individual analysis of their respective basic operations which apply to storage systems. In addition, network coding is applied to flooding, and theoretical analysis is included as a contribution from this report.

Chapter 2

Foundations of Network Coding

Network coding, often times abbreviated to NC, is a fairly recent field of interest starting in 2000 [12]. Ahlswede is seen as the father of network coding. He and peers combined network topology, algorithms, and mathematical computations to construct what is now the foundation of NC. A staple in NC research articles is the exploration of benefits found within a butterfly network topology. And to prove NC has the theoretical capabilities to improve the performance of networks in any form, the max flow-min cut and Ford Fulkerson algorithm are applied. The Galois field is often used in most network coding schemes, and will be explained in section 2.2.

2.1 A Brief History of Network Coding

According to [12], network coding surfaced in 2000 by Ahlswede and peers during work on multi-casting. This in turn led to studies of network coding benefits in specific network topologies. Ahlswede's article *Network Information Flow* was the first article to use the term *network coding* [1, p. 1204]. In it, intermediate nodes, or routers, were viewed as locations which could provide better throughput for data through additional computational tasks. [12, 5, 28] and other sources have discussed the benefits of network coding by demonstration via a butterfly network, similar to figure 2.1.

A few years later, studies of wireless network coding emerged [12]. Up until 2006, network coding remained theoretical, but Sachin Katti and peers created [12], which aimed to deploy COPE: wireless routing with network coding interleaved. Then in 2007, [6] discussed Avalanche, also known as Microsoft Secure Content Distribution (MSCD). This was a deployed peer-to-peer file distribution network which also utilized network coding. A common visual display for the benefits NC has to offer is the butterfly network. In figure 2.1, two source nodes are to broadcast their packets to

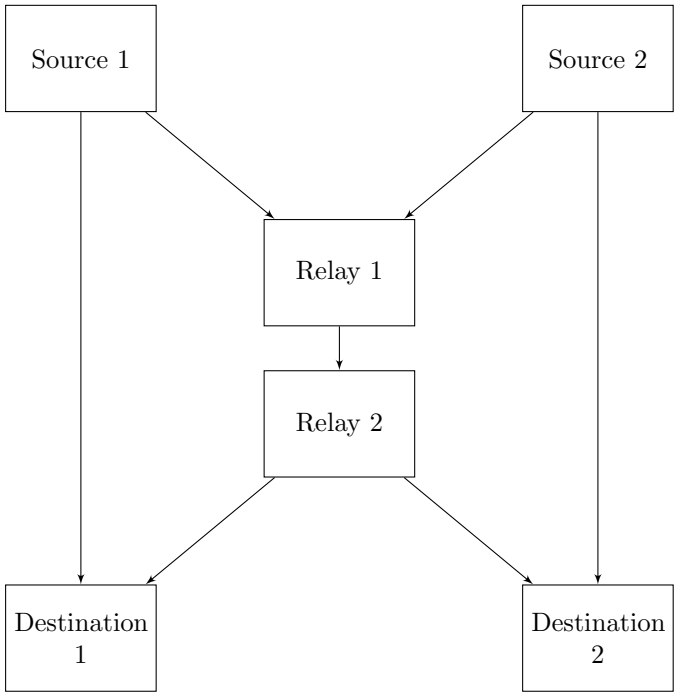


Figure 2.1: A Classic Butterfly Network

both destination nodes. According to [19], with multi-casting implemented, NC performs better at relaying data to both destination nodes. In both NC and traditional, *Source 1* will send its packet to *Destination 1*, and *Source 2* to *Destination 2*, in one round respectively. However, the difference in performance occurs at *relay node 1*. Without NC, *Relay 1* will need two rounds in order to deliver all received packets to *Relay 2*. *Relay 2* will broadcast to the destination nodes for two rounds. Thus it will take four rounds for all packets to be delivered to *Destination 1* and *Destination 2*.

With NC, however, both incoming packets can be combined and sent to relay node 2 in one round. *Relay 2* will broadcast only one packet to *Destination 1 & 2*. Then, *Destination 1 & 2* will decode based on the decoding scheme specified for the particular network. Thus, all destination nodes will receive all packets in three rounds. Thus with network coding, a 25% throughput improvement can be noted in a butterfly network. However, depending on the specified field size of the network, this may not be the case. This network coding benefit in a butterfly network can be explained using the max-flow min-cut theorem.

2.2 Galois Field

A Galois Field, more commonly referred to as a finite field, is a set of numbers represented as *GF(n)* or F_n. The parameter n is a prime number in which the set contains all whole numbers {0,

1, ... n-1}. [24]. For the purpose of practical network coding in sets of hardware, operating systems, and software, the most efficient galois field set is {0, 1}, which is represented as GF(2) [21]. And according to [21], $GF(2^t)$ generates a vector of t entries from the set GF(2). For example, a node in a network will need to generate coding vectors, which will be discussed further starting in chapter 3. These coding vectors are the resulting set from GF(2^t), and each element is multiplied with a respective incoming packet, and all the packets multiplied over the vector are summed to form an outgoing packet [5]. Given a node generates $GF(2^4) = \{0, 1, 1, 1\}$, the outgoing packet at that node is $\{0B_1 + 1B_2 + 1B_3 + 1B4\}$ [21]. According to [19], an ideal finite field size, which will work efficiently with most general computers, is from 2^8 to 2^{16}.

2.3 Max-Flow Min-Cut Theorem

[1, p. 1205-1207] discusses max-flow min-cut theorem and how it relates to the theoretical analysis of network coding. Given a network with one source node, several sink nodes internal nodes, the edges connecting all of the nodes have a certain integral capacity. Flow is the usage of an edge to its capacity at most. F is the symbol for a flow in the network from the source to any of the sink nodes. A general network flow rule is that the total incoming flow should equal to the total outgoing. This applies from the scope of a singular node to the network as a whole, which is referred to as Kirchloff's Current Laws [24]. The maximum flow is the flow of the path from the source to any of the sink nodes which uses as much of the edge capacities as possible to achieve the highest throughput.

[24] discusses min cut: Nodes are put into two sets: S and S_c. The sink will always be in the set S_c, and the source will always be in the set S. The capacity of a cut is the total edges from S to S_c. The cut occurs at edges in the path from the source to the sink that do not utilize the edge at full capacity. When that cut is met, all nodes in the same side as the source are in set S, and all nodes on the side of sink are in the set S_c. The cut capacity is the total capacity of edges separating sets S and S_c. According to [24], max flow and min cut are proven to be equal by the Fulkerson algorithm, when all capacities are of an integer value. And accoding to [24, p. 609], Menger's Theorem states that "the number of edge-disjoint paths from the source to the sink is equal to the max-flow of the network".

Chapter 3

Network Coding Schemes

The purpose of network coding is to increase the throughput and efficiency of transmitting data within a network. With this goal in view, encoding and decoding schemes are created for designated nodes in a network. Simple network coding involves a quick encoding and decoding scheme using $\oplus$, also referred to as XOR or exclusive-or, which can also be referred to as bit-wise addition [5]. Linear network coding involves a more complex encoding and decoding scheme which includes a coefficient matrix that contains important information used at decoding nodes [5]. Erasure coding is an encoding and decoding scheme primarily used for information redundancy, and generally performed only at source and destination nodes [25]. In simple and linear network coding, any node in the network, that is not the destination node, may encode two or more packets together [10, 5]. The following sub-sections will discuss simple, linear, and erasure network coding in greater detail.

3.1 Simple Network Coding

Simple network coding uses bit-wise addition between packet data, also referred to as the XOR operation, and is one of the most easily understood network coding schemes. When a node receives packets from two different sources, simply XOR them together, and send them to outgoing nodes [28]. Encoding involves adding zeros for padding, if needed, and then performing bit-wise addition in any manner the receiving node is aware. Additionally, a receiving node must maintain a record of at least n - 1 original packets encoded in the simple packet received [5]. This scheme works well for wireless communication. To decode, the destination node simply XORs the data received, with the packet it has kept in memory, in the appropriate manner to get the full packet. Thus, network coding has an added benefit of cryptography, since an interceptor would not easily recognize the file's manipulation process [5].

With all network coding schemes, receiving nodes must have some data to use in order to decode packet information received. There are several more network coding schemes mentioned in great detail in [28]. Simple network coding is efficient, in terms of overhead, and is acceptable for multicasting, broadcasting, or two-way communication.

3.2 Linear Network Coding

Linear coding extends simple network coding by incorporating additional information and including it in the packet payload. A node can multiply some coefficient with a fixed size of data in the original message, and combine its own data with a coefficient [5].

The following encoding and decoding process is generalized from [5]. For an incoming packet to be combined with a packet the node has generated, first the node must add padding, if necessary, to match the size of the encoded information, excluding the coefficient matrix. At the same position in each packet, multiply the desired size by a coefficient. Normally, a coefficient applies to a whole packet. But if a network wants more security, applying coefficients to some partition of a packet payload may be possible. Then the node adds all of the encoded segments at that position to the same position in the incoming packet. And finally, the finished coefficient vector used in the new packet will be added to the incoming coefficient matrix, if available, and included in the payload. There are several ways to decode, which generally varies for each network coding scheme, but the most simple implementation is to perform Gaussian elimination to solve for the original data by each position [5]. In order to decode, the coefficient matrix must have a total rank greater than or equal to the number of packets encoded.

One drawback of linear network coding is the increase in overhead of encoding and decoding data. As will be seen in later sections, computational overhead makes some network coded cloud storage systems inefficient in comparison to more widely known redundancy configurations. Also, [5] notes that the coefficients must be strategically chosen so as to maintain linear independence in each row of data.

3.2.1 Example of Random Linear Network Coding

With linear network coding, the coefficient matrix that the destination nodes will receive comprise of coefficient vectors with values chosen over a Galois Field [5]. Within round 1 of figure 3.3a, source nodes 1 and 2 will send the set $\langle c, P \rangle$, where P is the packet generated from the source, and c is the coefficient vector. If a packet is native, meaning it is not decoded, generally a native flag is raised in the packet header.

In the second time slot of linear network coding, as shown in figure 3.3b, *Destination 1 & 2* receive packets from *Source 1 & 2* respectively. *Relay 1* computes its outgoing file by receiving two random coefficients from Galois Field 2, multiplying them by the two incoming packets, and summing them up to receive an encoded packet. The generated coefficient vector $\begin{bmatrix} 1 & 1 \end{bmatrix}$ is appended to one of the incoming coefficient vectors, since they are equal. A zero is added to the incoming coefficient vector for alignment at the destination node solely, but this is not required practice at relay nodes.

In the final round, as shown in figure 3.3c, *Relay 2* simply forwards the incoming packet to the destination nodes. This is simply for a more simple decoding process, but all relay nodes can further encode incoming packets of any degree if they so choose.

The decoding process at *Destination 1 & 2* are similar in practice. The coefficient matrix for *Destination 1* is $\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$. According to [5], this coefficient matrix needs to be linearly independent amongst at least n rows in order to decode the packets successfully. This coefficient matrix is put into Gaussian elimination format with the encoded and native packets they arrived with. In the end, destination 1 and 2 decode $\begin{bmatrix} 4 & 0 \\ 2 & 8 \end{bmatrix}$ and $\begin{bmatrix} 2 & 1 \\ 1 & 0 \end{bmatrix}$.

3.3 Erasure Coding

Erasure coding is in the class of fountain codes [5]. When a packet is erasure encoded, only the destination node may manipulate it, for sake of decreasing overhead. Fountain codes are used for cryptography. The primary purpose of erasure coding is to ensure information redundancy in a network where dropped packets can occur frequently [5]. One property in erasure coding is called maximum distance separable, which has the form *MDS(n,k)* [23]. *MDS(n, k)* ensures that out of n packets being delivered to a destination, up to n-k can be lost, and the receiving node can rebuild the lost information from remaining packets [23]. In terms of cloud storage, up to n - k repositories may fail at a given time without loss of data. Another form of erasure coding involves encoding based on the Vandermonde matrix [25]. This matrix shows linear independence in any subset of the matrix; linear equations remains solvable up to a certain threshold in loss of rows. This matrix is used in Reed-Solomon codes, which are also used in RAID-6 construction [25, 23].

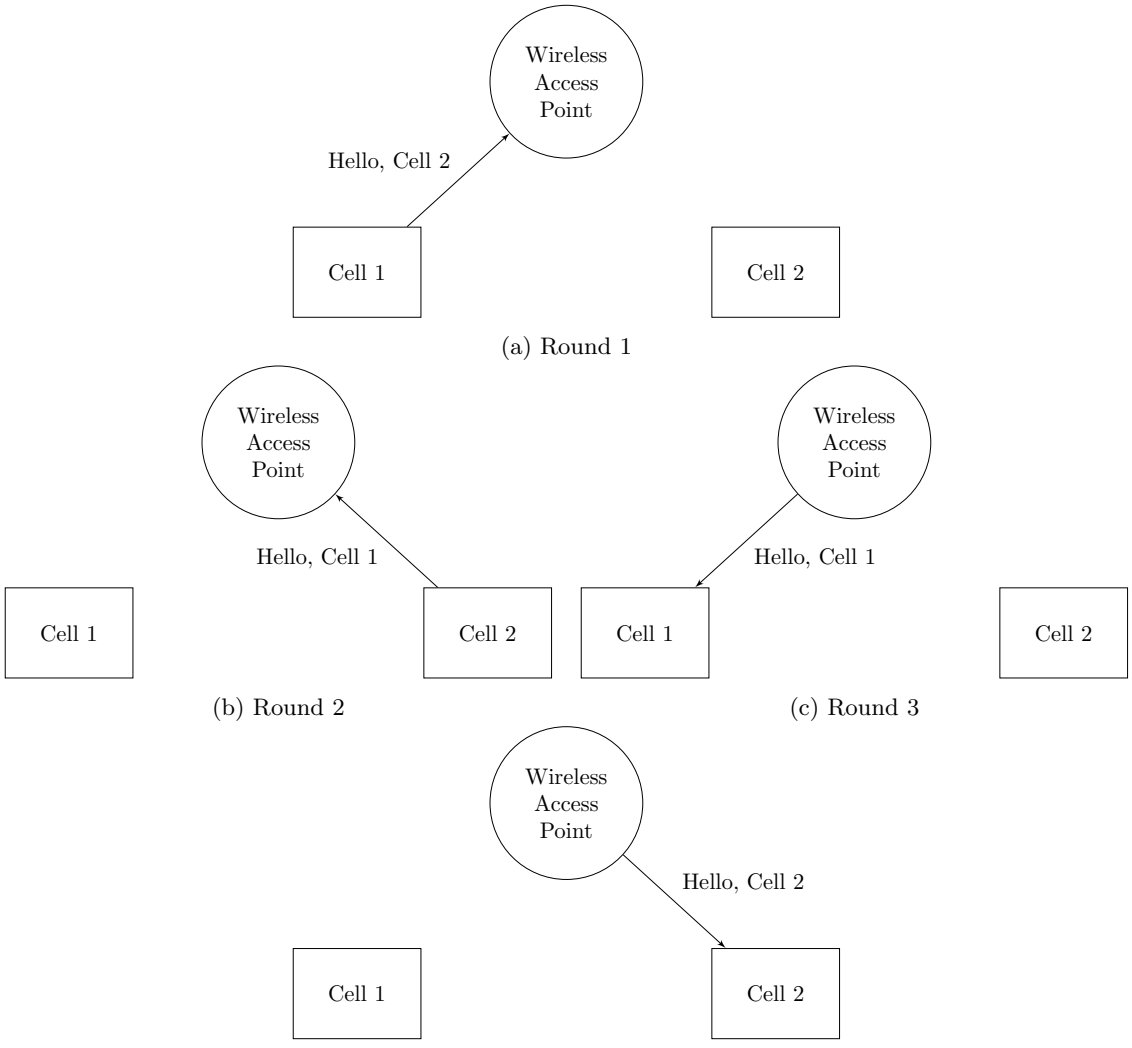
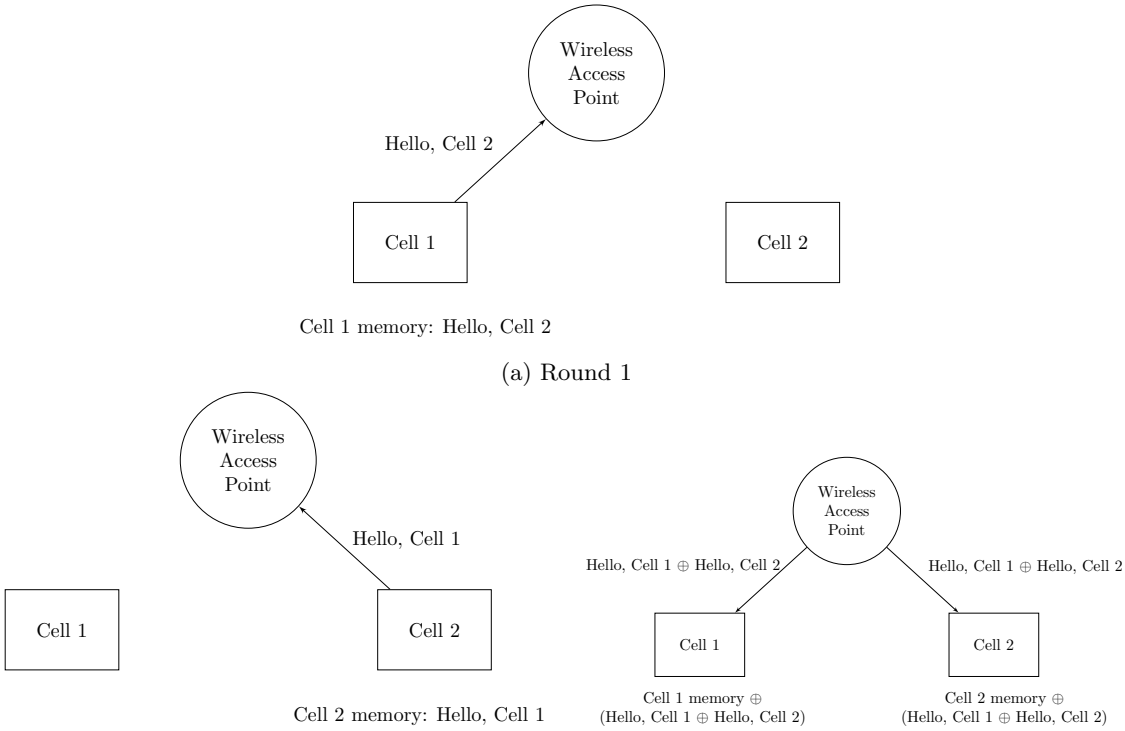


Figure 3.1: Traditional Routing in Mobile Network



(a) Round 1

(b) Round 2

(c) Round 3

Figure 3.2: Simple Routing in Mobile Network

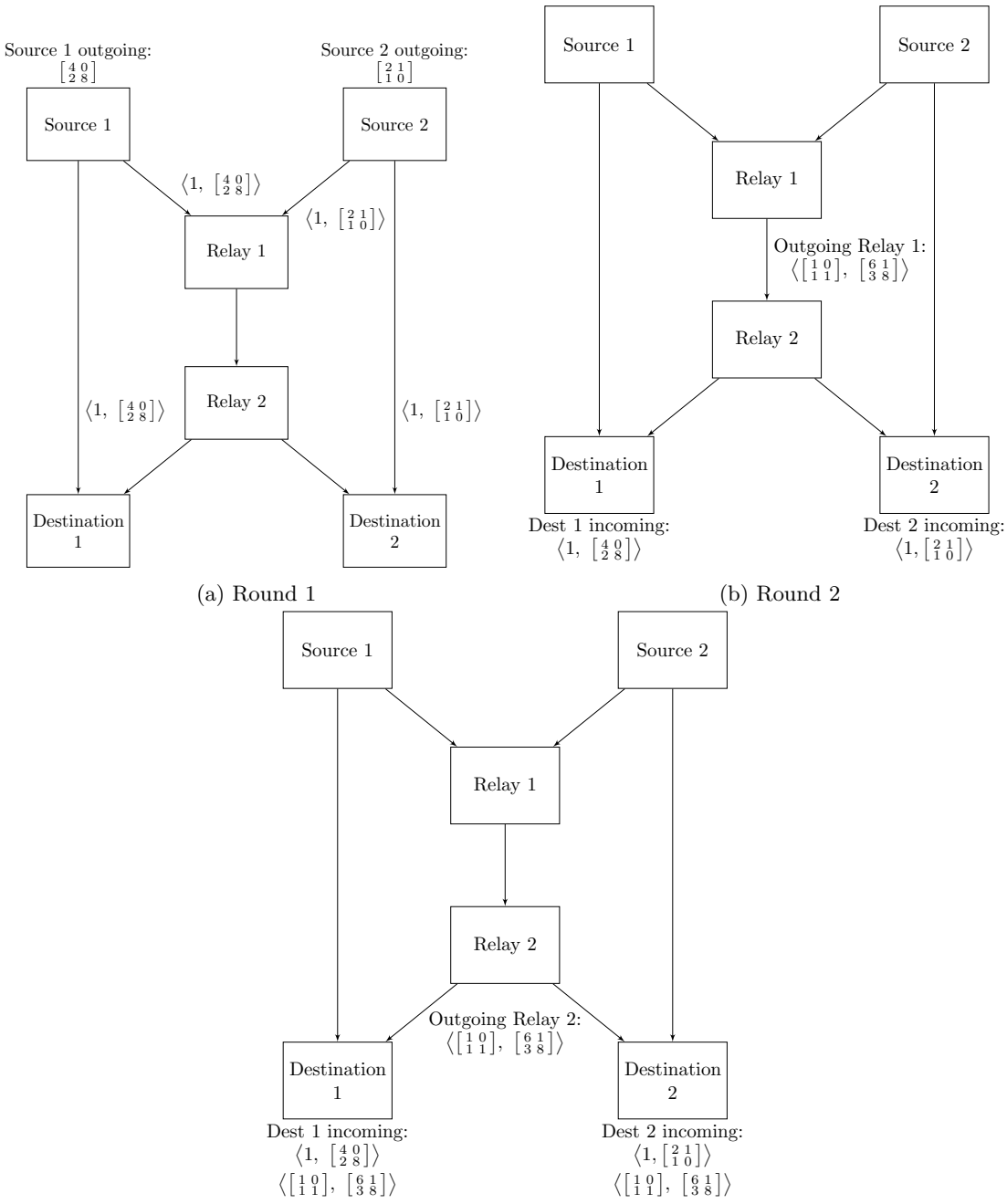


Figure 3.3: Random Linear Network Coding in Butterfly Network

Chapter 4

NC-Flooding: Flooding Using Network Coding

In this chapter, the flooding algorithm, commonly used in distributed systems, is modified to incorporate network coding principles. The routing packet information and protocols are mentioned. A sample runthrough is provided in section 4.2, where a node count network operation is applied to the NC-enhanced flooding protocol, and theoretical analysis is provided at the end of the chapter.

Flooding in distributed systems broadcasts a message across all nodes in a network. The distributed system must be strongly connected, meaning there must be at least one path from and to all other nodes in the system. In most distributed systems, a singular message or message type floods the network for a finite number of rounds. A time unit, or a round, contains the computation of all incoming messages within a brief timeframe, and the transmitting of a message to one or more neighbors [18, 3]. For instance, alpha synchronizers tell nodes to broadcast a $READY$ message in order to signify that node is ready to start a new synchronous round in an asynchronous system [18, 3]. According to [18, 3], the time complexity to flood a network with a message is $\mathrm{O}(diam)$ and the message complexity is $\mathrm{O}(|E|)$, where $diam$ is the diameter of the network, and E is the total edges found in the network. The diameter is the maximum shortest path between any two nodes in the network [18, 3].

However, nodes that want to flood individual messages into the network increases this complexity in a traditional store and forward distributed system. If all nodes wish to flood the network with unique messages, in a traditional routing system this is equivalent to each node broadcasting in a network of the same size where only that node's packet has the highest priority. Thus, in a distributed system with N nodes and E edges, the complexity to flood N individual packets is as

follows:

1. Message Complexity = $\mathrm{O}(N \times |E|)$
2. Time Complexity = $\mathrm{O}(N \times diam)$

Network coding can potentially reduce the message or time complexity of this unique flooding situation. NC-flooding will be used to name network coding enhancements for flooding in an asynchronous distributed system. Simple network coding, which uses exclusive-or to combine packets, is the recommended coding scheme for packets with no particular source, destination, or number of hops from source to destination. [11] introduced a simple network coding simulation which specified the packet structure and encoding and decoding algorithms needed for successful transmission, which will be applied to flooding.

4.1 NC-Flooding Protocol Variables

4.1.1 Variables

Each edge has a *LINK_ID*, and are bidirectional; each edge can handle incoming and outgoing traffic simulataneously. However in packets, *LINK_ID* can be a list of neighboring link IDs in which the packet should be delivered onto. Each node will be assigned a unique ID, *NODE_ID*, which will be used for creating packet IDs (*PACKET_ID*). For each node in the network, a data pool, incoming queue, and outgoing queue will be used to keep track of the packets for encoding or decoding purposes. Some packet header data from [11] is also useful for simple network coding implementations; *IS_NATIVE*, *PACKET_SEQUENCE*, *NO_OF_PACKETS*, and *DATA_POOL* were adapted to the NC-Flooding implementation. Table 4.1 lists the packet data used during the decoding and encoding processes. Table 4.2 lists the objects and variables each node will use locally for the construction and deconstruction of packets.

LINK_ID	List of edge IDs
PACKET_ID	Outgoing Packet ID
IS_NATIVE	Flag if the packet is native
PACKET_SEQUENCE	List of PACKET_IDs for decoding
NO_OF_PACKETS	Number of encoded packets
PAYLOAD	The data to be transmitted
ACK	Acknowledgement of PACKET_ID on LINK_ID
NACK	Cannot acknowledge PACKET_ID on LINK_ID

Table 4.1: NC-Flooding packet data

NODE_ID	Unique ID of the node
INCOMING_QUEUE	Queue holding all incoming packets
OUTGOING_QUEUE	Outgoing queue of packets
TOTAL_LINK_IDS_CONNECTED	Number of LINK_IDs connected to NODE_ID
DATA_POOL	Native packet data stored: ⟨PACKET_ID, LINK_ID, PAYLOAD⟩
PACKET_COUNTER	Returns and increments the next packet count for the PACKET_ID, starting at 0

Table 4.2: NC-Flooding node data

4.1.2 NC-Flooding Protocols

The decoding protocol pseudo-code is shown in algorithm 1.

Algorithm 1 NC-Flooding Decoding Protocol

```
1: while true do
2:     while INCOMING_QUEUE.isEmpty() do ▷ Ensure there is an incoming message to process
3:         trigger NC-Flooding encoding protocol
4:     P = INCOMING_QUEUE.pop()
5:     if P.IS_NATIVE then
6:         if not already in DATA_POOL then
7:             DATA_POOL.put(⟨P.PACKET_SEQUENCE, P.LINK_ID, P.PAYLOAD⟩)
8:             if NODE_ID.LINK_ID - P.LINK_ID > 0 then
9:                 OUTGOING_QUEUE.put(⟨P.PACKET_SEQUENCE,NODE_ID.LINK_ID -
   P.LINK_ID,P.PAYLOAD⟩)
10:        send ACK of P.PACKET_ID on P.LINK_ID
11:    else if P.ACK then                                          ▷ process for ACK
12:    else
13:        Decode using P.PACKET_SEQUENCE and DATA_POOL
14:        if node cannot decode then
15:            send NACK with P.PACKET_ID on P.LINK_ID
16:            discard P
17:        store all new native packets into DATA_POOL and OUTGOING_QUEUE
18:        put ACK containing P.PACKET_ID and P.LINK_ID on OUTGOING_QUEUE
```

The encoding protocol is outlined in algorithm 2. Each node will start decoding and then encoding once all links connected to the node have sent their respective messages. The decoding process will trigger the encoding process once finished decoding all in the *INCOMING_QUEUE*. This is to ensure that no errors occur further into the protocols as well as give an opportunity for network coding to enable. The assumption is that there are no hardware-related issues which prevent the protocol or NC-Flooding algorithm to execute successfully.

During the encoding process, at most two packets can be encoded at a time in order to ensure that the packet will be decoded at all neighboring nodes. [11] discusses an encoding policy which

Algorithm 2 NC-Flooding Encoding Protocol

```
1: while TRIGGERED do
2:     if OUTGOING_QUEUE.isEmpty() then          ▷ Want to ensure there is data to process
3:         TRIGGERED = FALSE
4:     Process and send all ACK and NACK packets from OUTGOING_QUEUE on appropriate
       LINK_IDs first
5:     if !OUTGOING_QUEUE.isEmpty() then
6:         Packet P = OUTGOING_QUEUE.pop()
7:         Packet N = null
8:         P_COUNTER = PACKET_COUNTER
9:         if TOTAL_LINK_IDS_CONNECTED == 2 then          ▷ NODE_ID can combine and
       send another packet in OUTGOING_QUEUE if it only communicates with two other nodes in
       the network.
10:            Packet N =OUTGOING_QUEUE.pop()
11:            if N != null && P_COUNTER != 0 then
12:                P.IS_NATIVE = FALSE
13:            else
14:                P.IS_NATIVE = TRUE
15:        else if OUTGOING_QUEUE.isEmpty() then          ▷ If only packet P is outgoing
16:            P.IS_NATIVE = TRUE
17:        else                                  ▷ Can only send out one native packet at a time
18:            N = DATA_POOL.get(NODE_IDs Native Packet)
19:            P.IS_NATIVE = FALSE
20:        if ! P.IS_NATIVE then
21:            P.PAYLOAD = P.PAYLOAD ⊕ N.PAYLOAD
22:            P.PACKET_SEQUENCE = ⟨P.PACKET_ID, N.PACKET_ID⟩
23:            P.LINK_ID = ⟨P.LINK_ID, N.LINK_ID⟩
24:            P.NO_OF_PACKETS = 2
25:        else
26:            P.PACKET_SEQUENCE = ⟨P.PACKET_ID⟩
27:            P.NO_OF_PACKETS = 1
28:        P.PACKET_ID = NODE_ID-P_COUNTER
29:        send P on P.LINK_ID
```

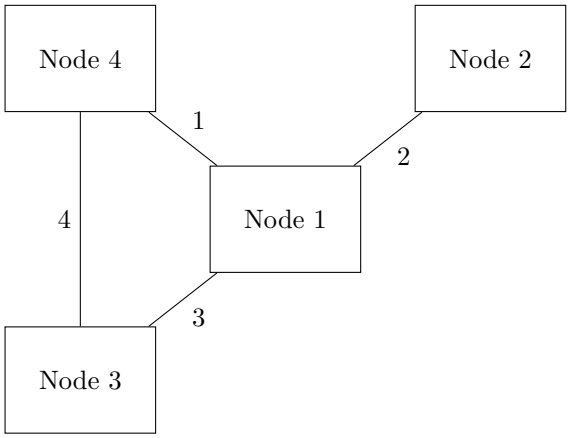


Figure 4.1: Network Topology for NC-Flooding

is adapted to NC-Flooding; a node may only be encoded into a packet if the possibility of all destination nodes to decode the packet is significantly high. In a distributed system, nodes have a low visibility of their neighbors and only perform the operations instructed. Therefore, when a node has *TOTAL_LINK_IDS_CONNECTED* greater than two, it is not guaranteed that a neighboring node will be able to decode more than two encoded packets. Note that these protocols do not have a finite amount of steps or terminate within the parameters because it is up to which distributed algorithm applied that determines when these protocols terminate. Any acknowledgement packets from a distributed algorithm can be passed within the payload of the *ACK* packet, hence the comment to process an *ACK* in algorithm 1.

4.2 NC-Flooding Example

NC-Flooding can be applied to any distributed algorithm. One example algorithm that will be applied to the NC-Flooding protocol is allowing each node in a network to determine how many other nodes are in the network. Figure 4.1 is the network topology that will be used for application and analysis of the distributed algorithm and NC-flooding protocol. The *NODE_ID* is simply the number appended to *Node*, and the *LINK_ID* is the number on each edge. There will be three phases to this application. In the first phase, each node sends out their *NODE_ID* in the payload to all neighboring nodes. All neighboring nodes will put an *ACK* in their *OUTGOING_QUEUE* to each node that has sent their packet and propagate that data to all other neighboring nodes. Receiving an *ACK* from all neighboring nodes signifies the end of phase one. In the second phase, each node will send out a ready packet, called *READY* contained in the *PAYLOAD*, once all neighboring nodes have sent an *ACK* back to the node and there are no further packets in the *OUTGOING_QUEUE*.

A node will signify it knows its ID is being propagated by sending out a *READY* packet, but can only proceed to the third phase once it receives a *READY* from all neighbors. This packet does not have to be unique; each node may simply store and forward a *READY* packet. Note that a node must still accept any incoming messages and possibly send out messages during this phase. In the final phase, each node will send out a special acknowledgment packet, the *FINAL_ACK* packet, which signifies a node is ready to terminate. These example phase packets can be found in table 4.3. Once all neighboring nodes send a *FINAL_ACK* packet, a node can safely terminate. To determine if a node can transition to a different phase, *ACK_POOL* and *FINACK_POOL* will be used in order to Right before termination, each node will count how many native packets are in its *DATA_POOL* to determine how many nodes are in the network.

NODE_COUNT	Number of nodes detected in the network, including self
FINAL_ACK	Signal in which a node wants to terminate. Found within the ACK payload
READY	Signal in which a node wants to move to the final phase. Found within a regular packet payload
PHASE	Used to determine which phase a node is in: 1, 2, or 3
ACK_POOL	A list of acknowledgement packets from LINK_ID only related to the native packet sent in the first PHASE
READY_POOL	A list of acknowledgement packets from LINK_ID sent in the second PHASE
FINACK_POOL	A list of final acknowledgement packets from LINK_ID

Table 4.3: NC-Flooding Example Phase Packets

The algorithm for Node Network Count is provided in 3.

In addition, algorithms 1 and 2 need to be modified to perform well with this Node Network Count algorithm. In algorithm 1, processing of *ACK* packets will involve reading and storing $\langle PACKET\_ID, LINK\_ID \rangle$ in the appropriate acknowledgement pool based on the message in the *PAYLOAD*. For the algorithm 2, *FINAL_ACK* packets should not be sent out of order, like other acknowledgement packets are allowed in the protocol.

4.3 Node Network Count Analysis

A sample run displaying the input and output queues using figure 4.1 is shown in tables 4.4 and 4.5. For sake of brevity, *ACK* rounds were combined with the next important round. However, this does not compromise the integrity of the sample run or example, because acknowledgement packets will only double the time and message complexity. Since the theoretical analysis following will be theoretical worst-case, doubling does not alter a theoretical mathematical derivation. Thus acknowledgements, except the *FINAL_ACK* in this case since it is significant, can be compressed and/or omitted.

Algorithm 3 Node Network Count

```
 1: PHASE = 1
 2: NODE_ID_SENT = FALSE
 3: while TRUE do
 4:     process all incoming and outgoing using NC-Flooding decoding protocol
 5:     allow NC-Flooding to update ACK_POOL and FINACK_POOL
 6:     if PHASE == 1 then
 7:         if NODE_ID_SENT then
 8:             if ACK_POOL.size() == TOTAL_LINK_IDS_CONNECTED then
 9:                 send READY packet to all neighbors with IS_NATIVE = TRUE
10:                 PHASE = 2
11:         else
12:             send NODE_ID to all neighbors
13:             NODE_ID_SENT = TRUE
14:     else if PHASE == 2 then
15:         if READY_POOL.size() == TOTAL_LINK_IDS_CONNECTED then
16:             send FINAL_ACK to all neighbors
17:             PHASE = 3
18:     else
19:         if FINACK_POOL.size() == TOTAL_LINK_IDS_CONNECTED then
20:             NODE_COUNT = DATA_POOL.size()
21:             terminate algorithm
```

The first five rounds in 4.4 show the hindrance *NODE 1* puts on the entire network to advance to the final phase. *READY* is not sent until all native packets from other nodes connected to *NODE 1* have been propagated. However, this also demonstrates the safety of the provided Node Network Count algorithm and NC-flooding protocols; since the network is strongly connected, no node will be able to terminate if a node has a large queue because this node has a path to all nodes in the network.

Node Network Count Theoretical Analysis

As mentioned in chapter 4, the message and time complexity of flooding N messages in a traditional distributed system are $\mathrm{O}(N \times |E|)$ and $\mathrm{O}(N \times diam)$ respectively. In all phases, each node is originating the following packets: *NODE_ID*, *READY*, *ACK*, and *FINAL_ACK*. The *ACK* packets add an additional *diam* of message complexity and an additional $|E|$ per message sent.

Assume that in a traditional flooding implementation, each node can send differing packets on each connected link, just as in the NC-Flooding protocol. In both traditional and NC-protocols, acknowledgements will still take at most one time unit to send to their respective neighbors. To

explain the complexity of ACK, these packets are essentially transmitting in the opposite direction the packet it is acknowledging in the network. Since the distance a message travels is *diam*, the ACK packet also takes a *diam* in the network.

The $READY$ and $FINAL\_ACK$ messages only need to be transmitted once; the $FINAL\_ACK$ is in the acknowledgement payload for the $READY$ packet. Thus they will be counted once in the complexity. The total message and time complexity in a traditional distributed system, without network coding involved, is calculated in equations 4.2 and 4.1.

$$
\begin{aligned}
&Time\ Complexity(Node\ Detection\ Without\ NC) \\
&= ((NODE\_ID + ACK) \times diam + READY + FINAL\_ACK)N \times diam \\
&= (2 \times diam + 1 + 1)N \times diam \\
&= (2 + 2 \times diam)N \times diam \\
&= (2N + 2N \times diam) \times diam \\
&= 2N \times diam + 2N \times diam^2
\end{aligned}
\tag{4.1}
$$

$$
\begin{aligned}
&Message\ Complexity(Node\ Detection\ Without\ NC) \\
&= (NODE\_ID \times 2 + READY + FINAL\_ACK)N \times |E| \\
&= (2 + 1 + 1)N \times |E| \\
&= 4N \times |E|
\end{aligned}
\tag{4.2}
$$

$$
\begin{aligned}
&Time\ Complexity(Node\ Detection\ With\ NC) \\
&= ((NODE\_ID + ACK) \times diam + READY + FINAL\_ACK)N \times diam \\
&= (2 \times diam + 1 + 1)N \times diam \\
&= (2 + 2 \times diam)N \times diam \\
&= (2N + 2N \times diam) \times diam \\
&= 2N \times diam + 2N \times diam^2
\end{aligned}
\tag{4.3}
$$

$$
\begin{aligned}
&Message\ Complexity(Node\ Detection\ With\ NC) \\
&\quad = (NODE\_ID \times 2 + READY + FINAL\_ACK)N \times |E| \\
&\quad = (2 + 1 + 1)N \times |E| \\
&\quad = 4N \times |E|
\end{aligned}
\tag{4.4}
$$

With NC-Flooding protocols enabled, the worst case scenario must be observed, as without NC-Flooding the worst case scenario is calculated as well. In this scenario, the worst case is when all nodes in the network have more than two neighboring nodes. In this case, a time unit must be consumed in order to propagate each incoming packet that is not an ACK. Thus, it behaves as the traditional routing protocol. As seen in equations 4.4 and 4.3, the message and time complexities are $4N \times |E|$ and $2N \times diam + 2N \times diam^2$ respectively.

The best case scenario for NC-Flooding is when all nodes in the network have exactly two links to neighboring nodes. This topology can best be represented as a ring network, where each node can only have two neighbors. Each message will still have a time and message complexity of *diam* and $|E|$, where $|E|$ is N, the amount of nodes in the network, and *diam* is $\frac{N}{2}$. This network behaves in a similar manner as seen in the wireless butterfly figure 3.2. It is observed that the time unit is reduced for each NODE_ID being transmitted in the network. Thus the best case message and time scenario for NC-Flooding are $5N$ and $N + \frac{N^2}{2}$ respectively. The best case scenario in this same ring topology using traditional flooding means that at each round, a node may only broadcast one message to at most all neighbors. Thus, the time and message complexities are applied to the equations 4.2 and 4.1. Simplifying the diameter and amount of edges, the message and time complexity of traditional flooding in Node Network Count are $5N$ and $\frac{1}{2}N^3 + N^2$. Network coding, in this application, improved the time complexity in comparison to traditional routing. A final comparison chart is provided in table 4.6.

NODE_ID	INCOMING QUEUE: $\langle PAYLOAD, (LINK\_ID\ LIST)\rangle$	OUTGOING QUEUE: $\langle PAYLOAD, (LINK\_ID\ LIST)\rangle$
1	-	<1, (1,2, 3)>
2	-	<2, 2>
3	-	<3,(4,3)>
4	-	<4, (4,1) >

(a) Round 1

NODE_ID	INCOMING QUEUE: $\langle PAYLOAD, (LINK\_ID\ LIST)\rangle$	OUTGOING QUEUE: $\langle PAYLOAD, (LINK\_ID\ LIST)\rangle$
1	<4,1><3,3><2,2>	<4, (2, 3)><3, (1,2)><2, (1, 3)><ACK,(1,2,3) >
2	<1,2>	<ACK,(2) >
3	<1, 3><4,4>	<(1⊕3), (3,4)><ACK,(3,4) >
4	<1,4><1,1>	<(1⊕3), (3,4)><ACK,(1,4) >

(b) Round 2

NODE_ID	INCOMING QUEUE: $\langle PAYLOAD, (LINK\_ID\ LIST)\rangle$	OUTGOING QUEUE: $\langle PAYLOAD, (LINK\_ID\ LIST)\rangle$
1	<(1⊕3),(3,1) ><ACK,(3,1,2)>	<3, (1, 2)><2, (1,3)><ACK,(1,2,3) ><READY,(1,2, 3)>
2	<4,(2,3)><ACK,2 >	<READY,2 >
3	<ACK,(3,4) ><4,3 ><(1⊕3),4 >	<ACK, (3,4)><READY,(3,4) >
4	<(1⊕3),4><ACK,(4,1) >	<ACK,4 ><READY,(1,4) >

(c) Round 3

NODE_ID	INCOMING QUEUE: $\langle PAYLOAD, (LINK\_ID\ LIST)\rangle$	OUTGOING QUEUE: $\langle PAYLOAD, (LINK\_ID\ LIST)\rangle$
1	<READY,(1,2,3) ><ACK,1>	<2, (1,3)><READY,(1,2, 3)>
2	<3,2>	<ACK,2 >
3	<READY,4 >	-
4	<READY,4><3,1 >	<ACK,1 >

(d) Round 4

NODE_ID	INCOMING QUEUE: $\langle PAYLOAD, (LINK\_ID\ LIST)\rangle$	OUTGOING QUEUE: $\langle PAYLOAD, (LINK\_ID\ LIST)\rangle$
1	-	<READY,(1,2, 3)>
2	-	-
3	<2,3 >	<2,4 >
4	<2,1><3,1 >	<2,4 >

(e) Round 5

NODE_ID	INCOMING QUEUE: $\langle PAYLOAD, (LINK\_ID\ LIST)\rangle$	OUTGOING QUEUE: $\langle PAYLOAD, (LINK\_ID\ LIST)\rangle$
1	<ACK,(1,3) >	-
2	<READY,2 >	-
3	<2,4 ><READY,3 >	<ACK,4 >
4	<2,4><READY,1 >	<ACK,4 >

(f) Round 6

Table 4.4: Node Network Count Phases 1 and 2

NODE_ID	INCOMING QUEUE: $\langle PAYLOAD, (LINK\_ID\ LIST)\rangle$	OUTGOING QUEUE: $\langle PAYLOAD, (LINK\_ID\ LIST)\rangle$
1	-	<FINAL_ACK,(1,2,3) >
2	-	<FINAL_ACK,2 >
3	<ACK,4 >	<FINAL_ACK,(3,4) >
4	<ACK,4 >	<FINAL_ACK,(1,4) >

(a) Round 7

NODE_ID	INCOMING QUEUE: $\langle PAYLOAD, (LINK\_ID\ LIST)\rangle$	OUTGOING QUEUE: $\langle PAYLOAD, (LINK\_ID\ LIST)\rangle$
1	<FINAL_ACK,(1,2,3) >	NODE_COUNT & terminate
2	<FINAL_ACK,2 >	NODE_COUNT & terminate
3	<FINAL_ACK,(3,4) >	NODE_COUNT & terminate
4	<FINAL_ACK,(1,4) >	NODE_COUNT & terminate

(b) Round 8

Table 4.5: Node Network Count Phase 3

	O ($TimeComplexity$)	O ($MessageComplexity$)	ΩTime Complexity	Ω Message Complexity
Node Network Count Without NC-Flooding	$2N \times diam + 2N \times diam^2$	$4N \times \|E\|$	$\frac{1}{2}N^3 + N^2$	$5N$
Node Network Count With NC-Flooding	$2N \times diam + 2N \times diam^2$	$4N \times \|E\|$	$N + \frac{N^2}{2}$	$5N$

Table 4.6: Node Network Count Comparison

Chapter 5

Network Coding in Peer-to-Peer Networks

The most popular peer-to-peer network is called Bit Torrent. The network model for Bit Torrent, and most other peer-to-peer networks, involves one or more central server locations and several peers which connect to the server. Once connected, central servers can share data blocks with peers, or allow peers to exchange data between one another directly. Peers are simply consumers which are allowed to leave the network at any time. Most likely, a peer will leave once they receive all data blocks needed to construct the entire data, be it an application, media, etc. However, peers that stay longer than expected, to contribute to the network as a whole, may receive perks.

In this chapter, the Coupon Collector's problem will be used to show how network coding can improve performance in peer-to-peer networks. A brief introduction to tree coding will be presented. Microsoft's network-coding enhanced peer-to-peer network "Avalanche", and a media streaming service with NC implementation, called "UUSee" are briefly discussed to show that real life applications are being created with decent performance outcomes.

5.1 Benefits of Network Coding in Peer-to-peer Networks

The main theoretical benefit NC can bring to peer-to-peer networks is derived from the coupon collector problem [5, 19]. The coupon collector's problem is the probability of discovering a new coupon from a pile of coupons that are put back into the pile after viewing them [19]. As the amount of pulls occur, the probability of choosing a new coupon diminishes. Network coding can eliminate this issue with its redundancy in packet transmissions [19]. If a person picked four coupons at a time, that person would be more likely to discover all coupons in the pile more quickly than a person

picking one coupon at a time.

Theoretically, it would take $\theta(n)$ rounds in a centralized system for all nodes to receive all coupons, and it would take $\theta(n \log(n))$ in a decentralized system [5]. However with NC, it takes $\theta(n)$ rounds in a decentralized and centralized system [5]. Therefore, optimal performance is achieved with network coding. In each round, every packet could contain several new coupons, which increases the probability of a new coupon to be discovered. The coupon collector problem is applicable to notable peer-to-peer network issues. With NC, peers are able to receive unique datablocks more frequently as opposed to traditional store and forwarding [5, 19].

5.2 Tree Coding

[26] presented a tree-structured network-coded peer-to-peer system. A tree structure consist of leaves, internal nodes (which contain leaves or other internal nodes as children), and a root node. All nodes not leaves must contain at most two child nodes. The leaves of the tree are original data blocks multipled by a coefficient [26].

At the second layer, which are parents of the leaves, these nodes are the XOR operation between both child nodes. According to [26], data should be spread across multiple trees in which each tree should use different coefficients. All trees consist of sub-trees, which any internal node can be temporarily considered a root node, and all successors underneath it create the tree [26]. According to theorem 6 in [26, p. 119], tree coding with k trees each peer has a worst case read/write complexity of $\mathrm{O}\left(kn \log^2(n) + ns\right)$ and average complexity $\mathrm{O}\left(kn \log^2(n)\right)$. In comparison to traditional peer-to-peer network read/write complexity, which is simply the number of blocks in the system, an additional complexity of $\mathrm{O}\left(k \log^2(n)\right)$ theoretically is a small increase when the perks of NC are taken into consideration.

Tree coding performs perfectly for two types of communication topologies. The first topology has a depth 1 in the online model with one tree with a policy with worst case read/write complexity nr + ns", and the second has a depth [of] 2 in the online model with $k = 2pt \log(n)$ trees where p is the number of peers in depth 1" [26, p. 119]. Note that t is the number of rounds, n is the number of blocks [26]. The average read/write complexity for tree coding is $\mathrm{O}\left(ptn \log^3(n)\right)$ [26, p. 119].

5.3 Avalanche

Avalanche is Microsoft's contribution of peer-to-peer networks [5, 19]. Avalanche uses NC to solve the issue of premature peer exits from the network. The mechanics of Avalanche are as follows, from [19]: files are divided into r blocks. All peers send random linear encoded blocks of received blocks.

Neighbors accept datablocks until they can reconstruct from r linearly independent coded blocks. A global encoding vector sent in the header of each packet, for decoding purposes.

5.4 UUSee

UUSee is a streaming system which uses a network coding system called R^2 in order to propagate streaming data [19]. The results are less buffering delays and bandwidth costs. According to [19, loc. 2230] UUSee is satisfactory for "normal-quality videos".

Chapter 6

Wireless Routing Networks

In this chapter, COPE and MORE, two wireless networks which utilize network coding, are explained. Each of these inject a layer between the IP and MAC layers which contain important information used for encoding and decoding packets. Their performance gains are mentioned, and each has the potential to outperform traditional wireless routing networks. The chapter ends with Analog NC, which is network coding at a physical level, which encourages packet transmission collisions.

6.1 COPE

COPE is a wireless routing network protocol which takes advantage of network coding [12, 19]. According to [12], the two prominent design principles for COPE are to exploit broadcasting and network coding.

6.1.1 COPE Overview

In addition to broadcasting and network coding, COPE seizes as many opportunities as possible to propagate data in a network as quickly as possible. Between the *IP & MAC* packet header is where COPE network coding data is injected [12]. Each node has a pool, or a buffer, to store native packet data retrieved. Opportunistic listening can be included, which allows nodes in vicinity to overhear packets being transmitted. If an overheard packet contains new data, store it in the local data pool [12].

[12, p. 3] discusses the importance of reception reports in COPE. At some specified time interval, all nodes send out a list of packet IDs that are in their buffer. Empty buffer nodes send out an empty report. In addition to opportunistic listening, opportunistic coding can be utilized [12, 19].

A node should "maximize the number of native packets delivered in a single transmission, while ensuring that each intended nexthop has enough information to decode its native packet" [12, p. 3]. A general rule is that packets can be combined only if the node is certain all next-hop neighbors have all but one of the packets being combined [12, 19].

How nodes learn the state of each neighbor, so as to determine which packets to combine, etc., is by analyzing incoming reception reports [12, 19]. However, if the network is congested, the second best solution is to use guessing. According to [12], the process is as follows: compute the delivery probability of all paths, broadcast the results, and use the results in link-state [stored data about neighboring nodes]. A guess may be incorrect, but NC is resilient against erroneous data; in the event of a wrong guess, encode more native packets for next retransmission [12]. Packets contain three blocks of data. They are the IDs of the coded native packets, reception reports, and an accumulation of acknowledgments that should be delivered [12]. Upon receiving a new packet, process any acknowledgments and reception reports, and update the local neighbor status [12]. If the packet contains encoded data, decode and store locally any new native packets and then send out an acknowledgment and reception report for such packets [12].

6.1.2 COPE Performance Gains

According to [19], COPE halves the amount of transmissions needed to propagate data through a network. The main reason is due to broadcasting encoded packets; network coding frees up bandwidth by condensing packets as well as the queue of relay nodes, but broadcasting encoded packets also creates redundancy in the network. The positive traits of encoding and redundancy and bandwidth reduction, in theory and practice, create a more efficient network.

Coding + MAC gains in COPE benefit stressed networks, i.e., networks with severe bottleneck. [12, p. 4] states that "MAC divides the bandwidth equally" between nodes in a network. Consider again the wireless butterfly network in figure 3.2. The bottleneck is through the singular relay node in the network. Without NC, the router would need to send more packets with limited bandwidth, if the end users are constantly transmitting data. This creates bottleneck. With NC utilized, the relay node can double the data it transmits to the end users, which, in theory, doubles throughput in the bottleneck network .

According to [12, p. 4], the maximum coding gains without opportunistic listening, but with coding + MAC, is 2 and it is achievable. However, with opportunistic listening included, there is no bound to the maximum coding gain [12, p. 5]. To prove this, a wheel topology was observed in [12]. A single relay node is in the middle of several users in a circular arrangement. In order for any two nodes to communicate, it takes two time slots; send data to the relay node, and the relay node will

forward to the target user. However, users which are not directly opposite from one another in the network topology can overhear transmissions from the eligible users.

Without NC, communication will always require two timeslots; N transmissions will be needed for the middle node, where N is the number of units within the network [12]. This means that there is a bandwidth of $\frac{1}{N}$ per node [12]. However, with coding + MAC without opportunistic listening, the relay node can encode all N packets received and simply broadcast the data outward; broadcasting is only one packet [12]. Thus, the coding + MAC gain, in comparison to not using NC, is N [12]. With opportunistic listening, users can overhear packets from other users not directly opposite in the topology. Thus, the use of the relay node may not be necessary to complete most transmissions. As N, the number of users, grows, there is no bound to the gains a wheel topology network can experience with coding + MAC and opportunistic listening [12].

6.2 MORE

MORE is a wireless routing implementation which uses intra-flow network coding [19]. In other words, packets which have the same destination are converged, or encoded, together. Opportunistic routing protocols can be utilized to reduce the probability of packet loss [19]. MORE works best routing general files of common size.

6.2.1 MORE Protocol

As with COPE, MORE is a routing protocol data is located between the IP and MAC layer of packets. However, MORE uses a multicast protocol. The source node performs several tasks [19]: The source node, which may be known as the central network server, must partition the file into K native packets. Then a random linear combination of the K packets must be performed, and the resulting packet broadcasted to all neighbors. In addition to the encoded packet, the coefficient vector used to encode, the batch of packet IDs, source and destination addresses, and eligible intermediate nodes are all transmitted with the packet. The source will continue to broadcast coded packets until receiving nodes send an acknowledgment packet.

The intermediate, or relay, nodes then perform the following tasks upon receiving incoming data [19]: Relay nodes listen for packets and check if they are eligible to forward the packet. Decode any packets an eligible node can forward. If an innovative packet is present from the decoded packet, then the relay node will store that native packet data within its data pool, and create a new random linear combination of all packets in its data pool. Note that the data pool with MORE protocol can consist of native and encoded packets, but during the encoding process, all packets in the data pool

are treated the same way.

The destination node will decode the incoming packets, by using matrix inversion on the incoming encoded packet, discard all redundant packets while accepting innovative packets, and send an acknowledgment packet [19].

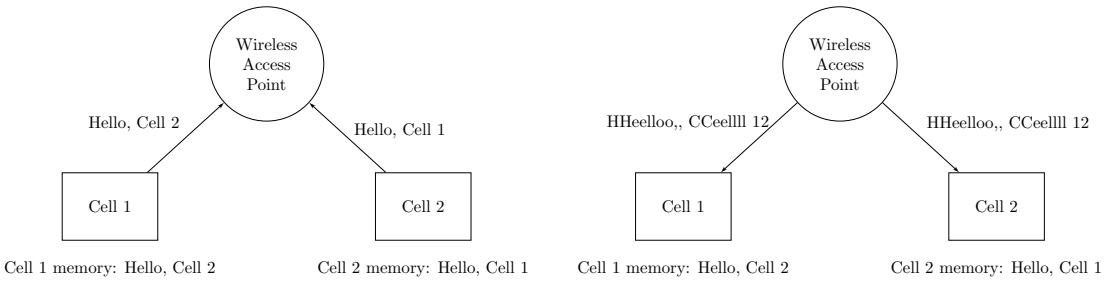
6.2.2 MORE Gains

MORE was compared against two other multicast protocols: Srcr, a multicast tree protocol, and ExOR, which "exploits the broadcast nature of the medium to deliver a packet to multiple nodes simultaneously" [19, loc. 1636]. The results from [19] state that MORE had 35-200% more throughput over ExOR, and 100-300% throughput over Srcr. As the number of destination nodes increased during testing, MORE was reported to have more gains due to the fact that network-coding yielded higher throughput over the other protocols [19].

6.3 Wireless Networks with Collisions

Collisions in networks can occur when two or more units transmit data to the same destination at the same time, and their signals entwine. Network coding can take advantage of two collided signals, which creates a branch of study known as Analog Network Coding [19]. For example, a wireless butterfly network can demonstrate the benefit of analog NC over simple or linear NC [19]. Each user will keep a copy of their transmitted data in local memory as they broadcast, as seen in figure 6.1a. In figure 6.1b, it is shown that the wireless access point will simply broadcast what is received; it will not perform any encoding or decoding operations as relay nodes do in other network coding implementations. Instead, the receiving users will subtract the signal they transmitted from the siganl they received so that the new message may reveal itself. In comparison to simple NC, which takes three timeslots to complete in a wireless butterfly network, Analog NC takes only two steps; the users unicast simultaneously, and the wireless access point broadcasts anything received to both users.

Two recovery schemes for Analog NC are explained in [29] and go into great deal the mechanics of separating signals, which mainly occurs at the physical layer of packet transmissions. These recovery schemes are called BPSK and $\frac{\pi}{4}$ - QPSK [29]. BPSK recovers packets from a network with links that can only send and receive [29]. However with $\frac{\pi}{4}$ - QPSK, a link consists of two parallel data streams; one stream is dedicated for incoming data traffic, and the other for outgoing data traffic [29]. With $\frac{\pi}{4}$ - QPSK, the signal from the other user will be computed based on the incoming and outgoing data streaming from the parallel data links.



(a) Round 1: Simultaneous sending

(b) Round 2: Analog NC Forwarding

Figure 6.1: Butterfly Collision Example

Chapter 7

Network Coding in Mobile and Wireless Sensor Networks

Mobile networks present an overt but rewarding hurdle for network coding implementations to overcome. In comparison to a desktop machine or laptop, cell phones have significantly smaller battery life and computational power, which are two resources network coding needs in order to better the performance of a system. However, broadcasting data to several mobile devices could prove to be a compelling reason to choose network coding with the propagation of data.

Mobile networks in developing countries can also benefit from network coding, as will be shown in section 7.1.1. And the chapter ends with RDTS, a wireless sensor network created to reduce energy consumption during the transmission of data from the sensors to the central servers the network reports to.

7.1 Network Coding in Mobile Networks

Multiple cell phones can be located in the same vicinity at any given time. If more than one cell phone is streaming the same data, given how costly NC can be, it would be best to avert the usage of an overlay network and simply share data directly. But for mobile devices at a reasonable distance apart, unicast or broadcast content exchange is an option. There are perks for using either, depending on the amount of receivers. A unicast is ideal if the amount of target mobile devices are small, and can save on energy consumption in comparison to a broadcast implementation [19]. However, broadcasting is an effective solution, but it needs error correcting code implemented upon arrival to destination. Forwarding to close neighbors reduces server load for broadcasting.

7.1.1 Mobile Network Coding in Developing Countries

Developing countries may not have the means, at the moment, to install several wireless access points across the land. Therefore, routers and data sharing are imperative for wireless data to be propagated to all network users. Mesh networks are a common practice in developing countries; by relying upon nearby routers to forward data from wireless access points, data can reach several users at a reasonable cost. Network coding can help increase efficiency and throughput in these special networks.

Users near a wireless access point, or a hot-spot, are able to broadcast the data received to neighboring users [19]. Receiving users collect, and can forward, the extra data received. With network coding, the receiving neighboring users are able to receive more data more frequently. Multi-path reception is an addition that improves data transmission [19]. Air interfaces are added for more ways a node can retrieve wireless data. However, a wi-fi link could be erroneous at times. But using NC to duplicate and disperse packets reduces the amount of errors that could occur. Another benefit mobile networks receive from using NC is explained by the coupon collector's problem, in section 5.1.

7.1.2 Problems of Mobile Network Coding

Random linear network coding is generally considered for mobile network coding. This requires a random coefficient be generated at relay nodes when combining packets. However, according to [19], built-in random number generators are not reliable. The best remedy for this is to use a trusted random number generator algorithm [19].

Another issue arises from performing arithmetic operations. These can be resolved by using the Euclidean algorithm for division, and Gauss-Jordan algorithm for decoding [19]. Varying mobile device specifications can cause additional strugles with incorporating NC. These can change the efficiency of data processing and computational speed. From a benchmark study mentioned in [19], the highest amount of throughput (50%) was achieved with a data block size of 10, and a field size of 2^{16}. However, the larger the block size, the greater the possibility of partitioning to occur during transmission between access points. If any partition becomes erroneous, the whole data block will be useless to the targeted user [19].

7.2 RDTS Implementation for A Wireless Sensor Network

[25] proposed an erasure coding implementation for wireless sensor data networks called Reliable Data Transfer Scheme. It uses the Vandermonde matrix to encode data, but each node in transit is

able to encode and decode data, if necessary.

The following generalized process is from [25]: At the source, which is a sensor, n packets encoded through the Vandermonde matrix to create $n+k$ packets, and then transmitted. At each node, the first n transmissions received are accepted, and all others from that source are dropped until completion of the current batch. A tally of the amount of original packets, labeled o in Illustration 4: RDTS transmission example, is performed over the n packets, and compared to the amount of packets the next node needs to receive, labeled t. If the amount of original packets are at least t - n, then forward all packets received. If not, then decode all packets, and re-encode to produce t packets needed for the next hop.

This implementation of RDTS was compared to end-to-end erasure coding, or EEEC for short [25]. The simulations involved wireless sensor networks in increasing size. EEEC needed to transmit data more frequently as the size of the network grew. Even though RDTS had more coding overhead, it was negligible, and performed better overall than EEEC in all reported size networks. Also, RDTS needed to transmit data less often than EEEC, which conserved more energy; a difference of total transmission size of 300 mega-bytes was noted in one simulation run between RDTS and EEEC [25]. Network coding shows the most promise in networks where relatively small data is transferred frequently, such as in wireless sensor networks. However there are some, albeit smaller, benefits to network coding in more dense data transmission, such as cloud storage repairs.

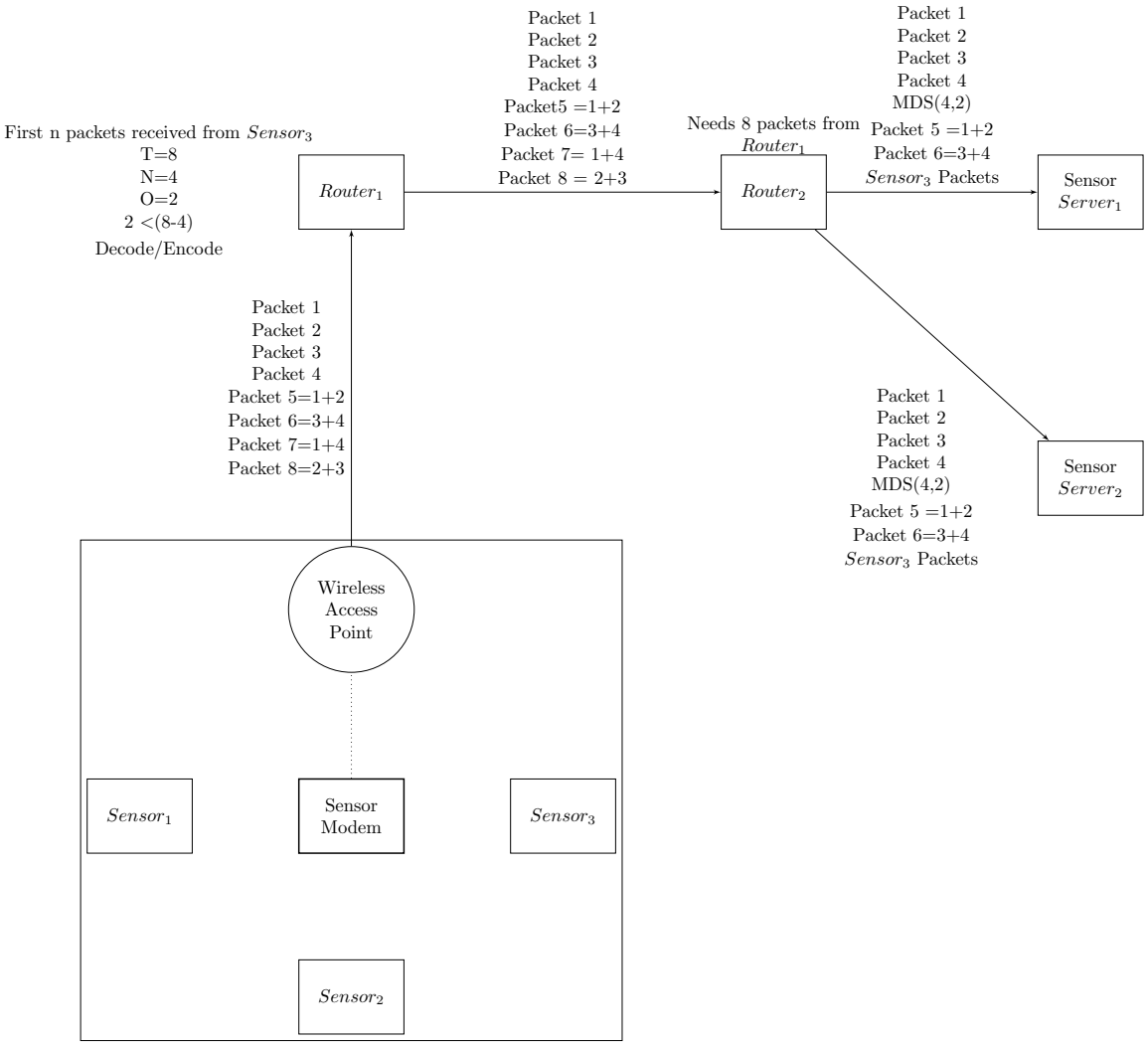


Figure 7.1: An RDTS Transmission Example

Chapter 8

Network Coding in Cloud and Distributed Storage

In this chapter, cloud storage systems with network coding are explained in detail. Traditional storage networks implement RAID-5 or RAID-6, which both disperse incoming data, and encoded partitins created, onto several repositories. RAID-6 will be the basis of comparison to Simple Regenerating Codes, also abbreviated to SRC, starting in sub-section 8.3.1. In addition, NC-Cloud, another form of network-coding storage which performs similarly to a linear network coding scheme, is briefly discussed.

Several articles discuss concepts or include implementations which could improve the performance of network coding in Cloud and distributed storage networks, which is discussed in more detail in section 8.5. For added security during storage, a key could be provided during transmission of data from a user to the repository. NC-Audit, a network coding repository system which emphasizes secure data access, is discussed in more detail in section 8.6.

8.1 Cloud Storage Overview

Storage systems are a fundamental part of storage as a service (SaaS), and a multitude of services provided by Clouds [13]. Since one characteristic of Cloud services is location transparency, Cloud storage can be referred to as cloud repositories, or simply part of the cloud. Currently there are three types of cloud models, which cater to the consumer's needs; there are public, private, and hybrid clouds [13]. Public clouds cater to a general population, private clouds are accessed by a specific group, like a company or organization, and hybrid clouds have flexible data capabilities which are able to move between clouds [13]. In all models, the storage system must correctly store and quickly

retrieve data upon consumer request.

8.1.1 Various Architectures

Cloud storage architectures vary; besides the necessity of repositories, there does not exist a standard cloud storage architecture [13]. However, a common feature amongst all clouds is the existence of storage management. This can come in the form of middle-ware [13] or a lead server [14]. These communicate with Cloud storage repositories, and would be the location of network coding implementations to the system. [27] also mentions the importance of primary and backup storage; primary cloud storage can be used for more frequent and smaller reads and writes, while backup storage is used to store larger, less used data; primary storage can be compared to Read Access Memory, and back storage to magnetic disks on a computer. Understanding these two types of cloud storage may be key for optimum performance in a specific cloud system. For added security, some Clouds also contain a hyper-visor, which monitors access requests and state of data in the Cloud [13].

8.2 RAID overview

In several articles, comparisons to replication or triplication are included in evaluations of the speed and efficiency of their network coding implementations. Replication is RAID-1, which involves duplicating the entire contents of one storage node, and storing it in the other. Triplication has data contents copied into three storage nodes. This allows for two storage node failures to be tolerated without loss of data.

8.2.1 RAID in Cloud

According to [8] and [23], RAID-6 is used as a standard comparison to network coding implementations in Cloud. This is due to RAID-6's efficient redundancy. RAID-6 is similar to RAID-5, but with the addition of another parity block [8]. This additional parity block allows for up to two failed storage nodes to be tolerated without loss of data.

RAID-6 Repair Analysis

[23] explains the concept of maximum distance separable. *MDS(n, k)* is a property which states that out of n storage nodes, n-k of the lot contain enough content to repair all other nodes [23]. According to [9] and [27], RAID-5 refers to *MDS(n, n-1)*, $n \geq 3$, which implies that this system can

only tolerate *n-(n-1) = 1* storage failures at a time. RAID-6, however, has *MDS(n, n-2)*, $n \geq 4$; RAID-6 can handle up to two repository failures at a given time without loss of data.

To repair up to two failed nodes, all other available nodes must send copies of the appropriate blocks to replacing nodes for reconstruction [7]. Personal analysis of the entire theoretical process was conducted. A table of variables mentioned throughout the duration of this article is in table 8.1.

b	Total number of blocks to repair
n_{failed}	Number of failed nodes
$n_{available}$	Number of alive nodes
d	Decode cost
e	Encode cost
div	File division for mds(n, k) cost
F_{total}	Total number of files in a storage node
s	Sparse code cost

Table 8.1: Repair Variables

$$\begin{aligned} total\ number\ of\ data\ sent &= bn_{failed}(n - n_{failed}) \\ &= bn_{failed}(n_{available}) \\ total\ repair &= bn_{failed}(n_{available} + n_{available}d) \end{aligned}$$

In section 8.3.1, the total repair of RAID-6 will be compared to total repair cost of SRC.

RAID-6 Read/ Write Operation Analysis

To write files, according to [23], divide file into k partitions, adding padding if necessary. Next, run the partitions through *MDS(n, k)*. Finally, store the resulting bits circularly in the cloud repository in parallel. The cost to store f files is:

$$f(div + e + n)$$

Reading files in [7] is relatively simple; collect all relevant blocks in $n_{available}$ repositories and send them to the central control. If $n_{available} = n$, then reading f files takes $\mathrm{O}(fn)$ time. If there is a storage node failure, parity blocks must be decoded before sending to the receiver. Thus, reading during storage node failure takes $f(n_{available} + d)$.

RAID-6 Benefits

According to [7] and [23], RAID-5 and RAID-6 offer fast parallel reads and writes. Also, there is less computational overhead in comparison to network coding implementations.

Problems with RAID / Triplication

Replication and triplication have a standard, reliable outcome [23]. However, the primary deterrent from their usage in most Clouds is cost; duplicating data on multiple storage nodes fills space quickly, and thus requires more storage nodes to handle any further storage.

Pertaining to RAID-6, repairing a single node alone is about $\frac{3}{4}$ the efficiency of an SRC implementation [9]. SRC will be explained in further detail shortly. In [27], several issues with cloud RAID implementations are discussed. One issue mentioned is the need for handling multiple writes; files that are only written to the system once is a rare occurrence [27]. Also, storing and transmitting large blocks of data could be an issue, depending on the purposes of the cloud; enterprises and organizations may need to store and update large sets of data repeatedly.

8.3 Simple Regenerating Codes

SRC stands for simple regenerating codes [23]. It builds upon RAID-6 in the sense that *MDS(n, k)* is used. *MDS(n, k)* has a process which uses linear algebra to divide a file into several pieces, and XOR certain sections to create redundancy. However, an additional parameter is added, to specify how many incoming files can be stored together, which is referred to as *f*, to create *SRC(n,k,f)* [23]. The f files are each divided into k equal pieces and ran through separate *MDS(n,k)* operations per file, and then the output is ran through a sparse encoder, part of the erasure coding process, which performs an XOR operation between output blocks to create parity blocks for all *f*. This is similar to the encoding process found in [25].

The storage nodes in figure 8.1 are labeled from 0 to n-1. These pieces are then circularly dispersed into block slots designated for all n storage nodes. [22] discusses the various manners in which the parity blocks may be placed in a storage system for redundancy. In [23], however, it is a standard rotation placement method. For each block, with f files in it that need to be reconstructed, the appropriate chunks must be received from $\min(n-1, 2f)$ nodes [23], and perform $(f+1)d$ operations to get the appropriate chunk. Note, that a chunk is $\frac{1}{f+1}$ of a block.

The total repair cost is: $\frac{f_{total}}{b} n_{failed}(2f + (f+1)d)$ In comparison to repair costs of RAID-6 (left-hand side taken from results in RAID repair analysis):

$$bn_{failed}(n_{available} + n_{available}d) \equiv \frac{ftotal}{b} n_{failed}(2f + (f+1)d)$$

assume decode is 1 unit:

$$bn_{failed}n_{available}2 \equiv \frac{f_{total}}{b}n_{failed}(2f + f + 1)$$

assume $n_{available} = 2f$ which happens when network coded packet compression is done:

$$bn_{failed}n_{available}2 \equiv \frac{f_{total}}{b}n_{failed}(n_{available} + (f + 1))$$

$f + 1$ is the bulk of computational overhead in SRC repairs. From these calculations, RAID-6 may appear less computationally taxing, but according to [23] and [8], SRC is proven to have better repair cost over RAID-6.

8.3.1 Simple Regenerating Code Read / Write Operation Analysis

Theoretically, reading data in SRC is relatively the same as in RAID-6 [23, 7]. The process involves collecting all relevant partitions from the active repositories, in parallel, and forwarding to the receiver. This takes O (fn) time without storage node failure. With storage node failure, parity bits are taken from available nodes instead, as in RAID-6. Thus, the read cost with storage node failure is:

$$f(n - n_{failed} + d)$$

The simulation results in [23] mimic theoretical calculations, generally.

Taken from [23], writing first divides files into k partitions, adding padding if necessary. Then run each file's partitions through their own *MDS(n,k)* in parallel. Upon completion, run all output through a sparse coder, which outputs $2n$ chunks to be stored circularly upon cloud repositories. Each node gets a total of $f + 1$ chunks. The cost to store f files is:

$$div + e + s + n(f + 1)$$

Comparing SRC write cost (right-hand side) to RAID-6 write cost:

$$f(div + e + n) \equiv div + e + s + n(f + 1)$$

assume d, e, and s are 1 unit:

$$f(1 + 1 + n) \equiv 1 + 1 + 1 + n(f + 1)$$

$$f + f + fn \equiv 1 + 1 + 1 + nf + n$$

$$2f + fn \equiv 3 + nf + n$$

$$2f \equiv 3 + n$$

O (f) vs O (n) The number of files being stored may increase at a faster rate than the amount of cloud repositories. And, with the option of uploading multiple files at once as in [4], multiple file uploading may increase as the years progress. Also, *3 + n* increases at a much slower rate than *2f*, which means SRC may perform repairs more swiftly in real cloud networks.

8.3.2 Simple Regenerating Codes Benefits

In terms of the articles reviewed, SRC appears to have faster repair times over RAID-6, and a nearly identical read performance. And from personal theoretical analysis, the results are almost in parallel; SRC appears to have worse repair times than RAID-6, primarily due to the computational overhead. But RAID-6 and SRC have nearly identical theoretical read and write costs. SRC utilizes erasure coding to store multiple files at once, while keeping a storage method similar to RAID-6.

8.3.3 Problems with Simple Regenerating Codes

[27] reports bottleneck can occur during writes in SRC. Table 8.2 summarizes the theoretical performance gathered from RAID-6 and SRC in cloud:

	Simple Regenerating Codes	RAID-6
Read Performance	$(f(n - n_{failed} + d)$	$f(n - n_{failed} + d)$
Write Performance	div + e + s + n(f+1)	f(div + e + n)
Repair Performance	$\frac{f_{total}}{b} n_{failed}(2f + (f + 1) * d)$	$bn_{failed}(n_{available} + n_{available}d)$

Table 8.2: Theoretical Summary of Performance

8.4 NC-Cloud

[2] proposes a linear network-coded cloud system called NC-Cloud. The storage process is slightly different from SRC; only encoded blocks are stored in all repositories. However, all blocks are stored, retrieved, and repaired in the same manner as SRC [2]. The NC-Cloud implementation has 3 layers [2]. The file system layer, which includes a mounted drive. The coding layer contains encoding and decoding functions as well as metadata. And finally, the storage layer consists of read / write requests between cloud repositories.

The process of storing data in the NC-Cloud implementation is explained in [2]. To store a file, *k(n - k)* partitions are made of the original file. For each partition, an encoding vector with k(n - k) entries is created. Per encoding vector, each element is multiplied by its respective file partition.

All multiplied operations are then added together into one file that is approximately the size of one file partition. At the end of this process, each cloud repository will receive *n(n - k)* of these encoded blocks, along with the encoding vectors. Repairs, file upload, and file downloads, all perform linear encoding and decoding to store and restore data. [2] reports a 25 - 50% greater repair performance to RAID-6. However, computational costs are driven in the process. Simulation results in [2] also show slightly higher read and write times in comparison to RAID-6; at one time up to a two second difference. The article admits the slight increase in response time is due to overhead from linearly combining and deconstructing blocks of data. The deployment of NC-Cloud depends on the amount of reads and writes are expected in the network.

To study the importance of read / write performance, [17] was reviewed. In this article, a college cloud repository network was monitored during a busy school semester in hopes of reporting typical cloud usage. While there was no known mention of cloud storage failure, read / write performance was shown to be important in certain situations. Most file upload and downloads were small, and thus little reading and writing was performed overall. There were two types of accounts which provided access to the cloud: personal, and group [17]. Personal accounts overall did not utilize reading and writing as much as group accounts. This can be equivalent to a public cloud with a small set of personal users.

However, the group accounts used significantly more storage, and performed a higher amount of reads and writes. The ratio of group to personal reads, in Gigabytes, was 40.322, and the ratio of group to personal writes to the cloud, also in Gigabytes, was 18.97 [17]. A group account can thus be reflective of a typical private cloud for some enterprise. Extrapolating from the article, the importance of read / write performance may depend on the model of the cloud; NC-Cloud may function best in a small public cloud model.

8.5 Proposed Improvements and Implementations to Network Coding in Cloud and Distributed Storage Networks

8.5.1 Cooperative Recovery Codes

Cooperative recovery codes, or CRC, are a slight improvement in the recovery process SRC presents [8]. In multiple storage loss, these nodes may be repaired in parallel in both SRC and CRC, in the same process. However, the two repairing repositories may exchange decoded chunks the other may need [8]. This reduces the amount of repair bandwidth necessary; only 2 calls between the repairing nodes is favorable over 4 calls between repairing nodes and active nodes.

While [8] and [9] are for distributed systems and not necessarily clouds, an adaptation is easy. The

central control mechanism has a connection to all cloud repositories. While they are all connected as well, the central control mechanism may update living repositories more quickly when a storage node has failed. When a replacement is installed, the central control can update active repositories that a new storage unit is in place. At which time, the repositories may open any direct connections to the new node, or forward all data through central controls to the replacement node. In which case, this configuration is similar to a distributed storage system, where all repositories may act independently of one another.

8.5.2 Tree Formation in a Network-Coding-based Repository

In [16], a tree formation in a distributed file system was implemented and compared to a star network, which is a typical generalized view of a repository network like Cloud. Given a set of repositories, construct a tree formation; nodes can receive data from any other repository, but must send data only to one node, which is their parent. Upon arrival, a new node is given a parent which will send it network-encoded packets to decode and store. The process is as follows: a newcomer request will bubble up to the root. And then the root will send an encoding scheme of data chunks for the newcomer to store. [16] reports improvements over a star network.

8.6 Cryptographic Benefits of Network Coding in Cloud Storage

As mentioned in earlier sections, network coding has an added benefit of encryption; packets of data intercepted are not in the original format and require some effort to properly decode. NC-Audit implements stricter encryption on data stored with in cloud repositories [15].

8.6.1 NC-Audit

In [15], NC-Audit is a symmetric key-based cryptographic protocol. In addition to the cloud repositories and control mechanism, an auditor is assigned to authorizing cloud storage access and retrieval requests. In a general overview of NC-Audit, an encryption key is created based on the consumer's MAC address [15]. This encoding key, along with linearly encoded data destined for storage, are sent to the cloud repository [15]. Data is processed and stored in the same manner as the NC-Cloud.

In addition, the MAC address of the user is sent to the cloud auditor. If data needs to be checked, or requests need to be authorized, then an auditor may use the MAC key to re-create the encryption key and access user data stored in the cloud[15]. Some properties of NC-audit, as

proven by evaluations in [15], include efficient integrity checking, efficient support for repair and data dynamics, and efficient privacy protection.

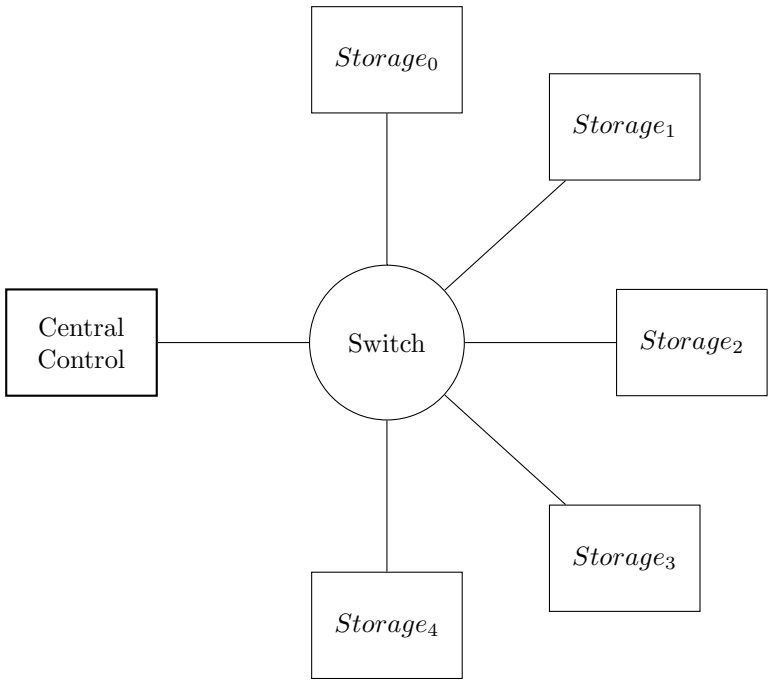


Figure 8.1: Cloud Storage Layout Example

Chapter 9

Network Security

Network coding has a naturally added measure of security against malicious attacks from interceptors. [13] reports that even if an attacker were to modify a packet in a network, the attack alone would not alter correctness of the network as a whole. Attackers would alter messages from at least one edge in a network, but not all edges are expected to be tampered by a single attacker. Since redundancy is a key trait associated with network coding, duplicates of native packets propagate throughout. If a node cannot properly decode an incoming packet, be it due to linear dependency in the data or malicious tampering, an incoming packet from a later session and a different neighbor is expected to have correct data.

[13] goes on to discuss another instance of malicious attack that NC protects against. If an attacker wishes to read data from a network with NC implemented, the attacker would need to have all previous native packets received by the intercepted router. Even if the attacker were to receive all native packets needed to decode, they would need to guess the last combination in order to decode all data successfully. However, the possibility of a successful attack is not negligible. One solution against link tapping is to combine outgoing data with useless data. However, this may increase chances of an erroneous packet upon decoding if a receiving node cannot discern between useful and useless data successfully.

Chapter 10

Limitations of Network Coding

While network coding has been shown to provide many benefits, such as greater energy efficiency, faster download times, there are specific flaws which currently prohibit network coding for competing with traditional store-and-forward schemes. The network topology itself could be a hindrance to the performance of network coding; nodes in such networks that do not have the opportunity to encode data from multiple incoming neighbors are probably better using a traditional routing protocol.

NC-Flooding, the network-coding-enhanced protocol provided in chapter 4, cannot perform well in networks where nodes have more than two neighbors. Pertaining to network coding in peer-to-peer networks, the primary reason for its lack of popularity is due to the high computational costs for encoding and decoding data; since the computational power of peers is unknown, a peer may perform worse in an NC-peer-to-peer network as opposed to a traditional peer-to-peer network.

10.1 Network Topology

Network coding excels primarily in network topologies which have little bottleneck, as can be inferred from section 2.3. Expanding upon Menger's Theorem mentioned in [24], a topology with minimal links or edges from source to sink nodes may not have a proper flow to see a significant performance increase. For instance, a chain topology only has *n-1* edges, as seen in figure 10.1. If *node 1* were to send a message to *node 4*, no network coding would increase the throughput; the message must only pass through *node 2* and then *node 3* in order to reach its destination. Only erasure coding, which is encoding and decoding only at the source and destination respectively, can increase the throughput of this network topology in comparison to a traditional store and forward routing method. Thus, network coding is of little use in topologies with minimal edges between any pair of nodes.

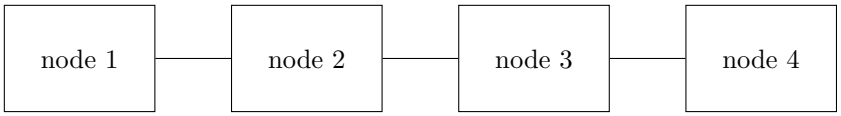


Figure 10.1: Chain Topology

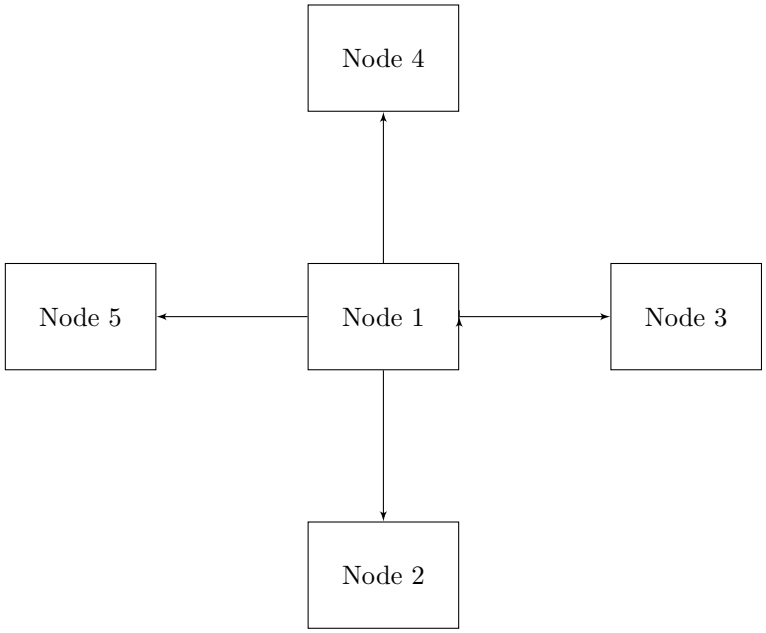


Figure 10.2: Four-way Bottleneck

10.2 Limitations of Network Coding in NC-Flooding

NC-Flooding does not perform well with network topologies in which more than two links per node are common. This is due to a rule for simple network coding which must be adhered to in order to ensure successful decoding of packets. [11] states that when encoding an outgoing packet, a node must have a high probability that all nodes receiving the packet will be able to decode successfully. Consider figure 10.2. *Node 1* has just received *M2, M3, M4*, and *M5*, which are messages from the corresponding nodes in the network. *Node 1*'s next outgoing packet must be decodeable by all nodes. If any two of the incoming messages were combined and broadcasted, only two nodes would be able to decode, while the other nodes will drop the packet since it is not decodeable. Thus, only one message per time unit can ge broadcasted to all of the other nodes. An enhancement to reduce the time is to send differing encoded or native packets on each link.

10.3 Limitations of Network Coding in Peer-To-Peer Networks

Bit Torrent remains more notable and prominent peer-to-peer network over Avalanche, Microsoft's NC-infused answer [26]. Computational overhead is perhaps the main reason NC peer-to-peer networks are not popular. The target computer specifications may vary from peer to peer. Thus, slower computers may affect the performance of said peer in the network, since it would take longer to decode and encode data blocks. Theorem 5 from [26, p. 117] states that the read/write complexity of Network Coding of each peer is in the worst case at most $n_r n_s$ and $\mathrm{O}\left(n^2\right)$ in the average". This means peers must read $\mathrm{O}\left(n^2\right)$ data blocks in order to send n message blocks, as opposed to the traditional store-and-forward implementation Bit Torrent uses.

Chapter 11

Open Issues

Open issues are briefly provided in hopes of exciting more exploration of network coding. [23] reports there are currently no practical code constructions of efficiently repairable codes with data rates above $\frac{1}{2}$. [8] reports three open issues. The first issue is discovering the optimal scheme for repair links if the repair bandwidth varies, as in the real world. The second issue is creating an encoding and decoding scheme which allows the file system to expand, as opposed to maintaining a fixed amount of repositories. And the final open problem mentioned in [8] is creating a coding scheme with less overhead. [26, p. 122] mentions that an "efficient network coding with optimal performance for all communication graphs" is not known. And in relation to communication, an "... NC [implementation] for multiple concurrent communication session[s]" is needed [20, p. 242]. Research on the benefits of COPE is ongoing; [12] reports that maximum achievable coding gains for unicast transmissions, and for opportunistic listening, are yet to be determined.

Chapter 12

Conclusion and Future Work

The practical applications of network coding have been shown to have the potential to decrease download times, increase data throughput in networks, and improve the overall theoretical performance of a distributed algorithm. A primary issue with network coding is the amount of computational power needed in order to analyze these condensed packet payloads from senders in a network.

This is perhaps one of the main reasons why Bit Torrent is still more successful than Microsoft's Avalanche project. When mobile devices have very little resources dedicated to computation, and a small battery life, NC implementations become very challenging. However, since NC is still a relatively new research topic, there is much progress to be made in enhancing the practical NC application to common network-related issues. The more studies and articles which come out on findings of new techniques and shortcuts in which network coding implementations can use, the better the likelihood of more popular software applications incorporating network coding. Thus, future work will be to remain current with new articles published on network coding and new mathematical techniques in which network coding could be enhanced by its application.

Bibliography

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

[2] H.C.H. Chen, Yuchong Hu, P.P.C. Lee, and Yang Tang. Nccloud: A network-coding-based storage system in a cloud-of-clouds. *Computers, IEEE Transactions on*, 63(1):31–44, Jan 2014.

[3] Ajoy Datta. Lecture notes in distributive computing, March 2013.

[4] Dropbox. HTML. www.dropbox.com.

[5] Christina Fragouli, Jean-Yves Le Boudec, and Jörg Widmer. Network coding: An instant primer. *SIGCOMM Comput. Commun. Rev.*, 36(1):63–68, January 2006.

[6] Christina Fragouli and Emina Soljanin. Network coding applications. *Found. Trends Netw.*, 2(2):135–269, January 2007.

[7] Fujitsu. Raid. HTML Flash, 1995-2015. www.fujitsu.com/global/services/computing/storage/eternus/glossary/raid/.

[8] Yuchong Hu. Cooperative recovery of distributed storage systems from multiple losses with network coding. Powerpoint, 2010. http://www.inc.cuhk.edu.hk/sites/default/files/seminars/slides/Demo - presentation (Yuchong Hu dd201010).ppt.

[9] Yuchong Hu, Yinlong Xu, Xiaozhao Wang, Cheng Zhan, and Pei Li. Cooperative recovery of distributed storage systems from multiple losses with network coding. *Selected Areas in Communications, IEEE Journal on*, 28(2):268–276, February 2010.

[10] A.A. Kadhim, T.A. Sarab, and H. Al-Raweshidy. Improving throughput using simple network coding. In *Developments in E-systems Engineering (DeSE), 2011*, pages 454–459, Dec 2011.

[11] A.A. Kadhim, T.A. Sarab, and H. Al-Raweshidy. Improving throughput using simple network coding. In *Developments in E-systems Engineering (DeSE), 2011*, pages 454–459, Dec 2011.

[12] S. Katti, H. Rahul, Wenjun Hu, D. Katabi, M. Medard, and J. Crowcroft. Xors in the air: Practical wireless network coding. *Networking, IEEE/ACM Transactions on*, 16(3):497–510, June 2008.

[13] Gurudatt Kulkarni, Rani Waghmare, Rajnikant Palwe, Vidya Waykule, Hemant Bankar, and Kundlik Koli. Cloud storage architecture. In *Telecommunication Systems, Services, and Applications (TSSA), 2012 7th International Conference on*, pages 76–81. IEEE, 2012.

[14] A. Kumar, Byung Gook Lee, HoonJae Lee, and A. Kumari. Secure storage and access of data in cloud computing. In *ICT Convergence (ICTC), 2012 International Conference on*, pages 336–339, Oct 2012.

[15] Anh Le and A. Markopoulou. Nc-audit: Auditing for network coding storage. In *Network Coding (NetCod), 2012 International Symposium on*, pages 155–160, June 2012.

[16] Jun Li, Shuang Yang, Xin Wang, Xiangyang Xue, and Baochun Li. Tree-structured data regeneration with network coding in distributed storage systems. In *Quality of Service, 2009. IWQoS. 17th International Workshop on*, pages 1–9, July 2009.

[17] Songbin Liu, Xiaomeng Huang, Haohuan Fu, and Guangwen Yang. Understanding data characteristics and access patterns in a cloud storage system. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 327–334, May 2013.

[18] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[19] Muriel Médard and Alex Sprintson. *Network coding: fundamentals and applications*. Academic Press, 2012.

[20] Zoran Miličević and Zoran Bojković. Goals and perspectives of the network coding. *XXIX Simpozijum o novim tehnologijama u potanskom i telekomunikacionom*, 2011.

[21] Aishwarya Nagarajan, Michael J. Schulte, and Parameswaran Ramanathan. Galois field hardware architectures for network coding. In *Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '10, pages 35:1–35:9, New York, NY, USA, 2010. ACM.

[22] E. Pakhomova and J. Pakhomova et al. Raid 6 recovery. HTML, 2009. http://www.freeraidrecovery.com/library/raid6-recovery.aspx.

[23] D.S. Papailiopoulos, Jianqiang Luo, A.G. Dimakis, Cheng Huang, and Jin Li. Simple regenerating codes: Network coding for cloud storage. In *INFOCOM, 2012 Proceedings IEEE*, pages 2801–2805, March 2012.

[24] K.V. Rashmi, NiharB. Shah, and P. Vijay Kumar. Network coding. *Resonance*, 15(7):604–621, 2010.

[25] M.S. Srouji, Zhonglei Wang, and J. Henkel. Rdts: A reliable erasure-coding based data transfer scheme for wireless sensor networks. In *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*, pages 481–488, Dec 2011.

[26] Arne Vater, Christian Schindelhauer, and Christian Ortolf. Tree network coding for peer-to-peer networks. In *Proceedings of the Twenty-second Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '10, pages 114–123, New York, NY, USA, 2010. ACM.

[27] Jianzong Wang, Weijiao Gong, P. Varman, and Changsheng Xie. Reducing storage overhead with small write bottleneck avoiding in cloud raid system. In *Grid Computing (GRID), 2012 ACM/IEEE 13th International Conference on*, pages 174–183, Sept 2012.

[28] Raymond W. Yeung, S-y Li, and N. Cai. *Network Coding Theory (Foundations and Trends(R) in Communications and Information Theory)*. Now Publishers Inc., Hanover, MA, USA, 2006.

[29] Jingyao Zhang, Kai Cai, K.B. Letaief, and Pingyi Fan. A network coding unicast strategy for wireless multi-hop networks. In *Wireless Communications and Networking Conference, 2007.WCNC 2007. IEEE*, pages 4221–4226, March 2007.

Vita

Graduate College University of Nevada, Las Vegas Jonny L. Winger

Degrees:

Bachelor of Science in Computer Science 2011 University of Nevada Las Vegas

Thesis Title: A Survey of Network Coding and Applications

Thesis Examination Committee:

Chairperson, Dr. Ajoy Datta, Ph.D.

Committee Member, Dr. Ju-Yeon Jo, Ph.D.

Committee Member, Dr. Lawrence L. Larmore, Ph.D.

Graduate Faculty Representative, Dr. Venkatesan Muthukumar, Ph.D.