This is to certify that the
dissertation entitled

NETWORK-EMBEDDED FEC FOR OVERLAY MULTICAST:
ANALYSIS AND OPTIMIZATION

presented by

MINGQUAN WU

has been accepted towards fulfillment
of the requirements for the

Ph. D. degree in Electrical and Computer Engineering

Major Professor's Signature

October 25, 2005

Date

# NETWORK-EMBEDDED FEC FOR OVERLAY MULTICAST: ANALYSIS AND OPTIMIZATION

By

Mingquan Wu

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Electrical and Computer Engineering

College of Engineering

2005

## ABSTRACT

NETWORK-EMBEEDED FEC FOR OVERLAY MULTICAST:
ANALYSIS AND OPTIMIZATION

By

Mingquan Wu

Forward Error Correction (FEC) can only be implemented on an end-to-end basis between the sender and the multicast clients in traditional multicast systems (such as IP multicast). Emerging overlay networks, however, open the door for new paradigms of network FEC. This thesis presents a new framework, which we refer to as Network-Embedded FEC (NEF) for overlay multicast networks. Under NEF, we place FEC encoders and decoders (codecs) in selected intermediate nodes of an overlay network. The NEF codecs detect and recover lost packets within FEC blocks at earlier stages before these blocks arrive at deeper intermediate nodes or at the final leaf nodes. This approach significantly reduces the probability of receiving undecodable FEC blocks. In essence, the proposed NEF codecs work as signal regenerators in a communication system and can reconstruct most of the lost data packets without requiring retransmission. The packet-loss model of NEF over random multicast trees that exhibits packet losses with memory over its links were developed and analyzed. Both centralized and distributed algorithms for the placement of NEF codecs within random multicast trees were designed and implemented. NEF can be used independently to achieve a desired level of reliability for certain applications; it can also be integrated with ARQ to achieve complete reliability. Our theoretical analysis and simulation results show that a relatively small number of NEF codecs placed in sub-optimally selected intermediate nodes of a network can improve the goodput and overall reliability dramatically.

DEDICATION

To my wife, Yingzi, and my two daughters, Leilei and Shanshan

ACKNOWLEDGMENTS

I am sincerely grateful to my advisor Dr. Hayder Radha for his guidance and advice over the last four years. Without his support, this work would not be possible. I would like to thank my committee members, Dr. Subir Biswas, Dr. TongTong Li and Dr. Philip. K. McKinley for their constructive comments during the development of this thesis. I also owe my thanks to my colleagues in the WAVES lab for their valuable inputs and help. Last but not least, I would like to thank my wife for her constant support during this long period of Ph.D study.

TABLE OF CONTENTS

LIST OF FIGURES

# LIST OF TABLES

# Chapter 1 Introduction

Providing reliable multicast services over the Internet is driven by the increasing popularity of applications that deliver realtime and non-realtime multimedia content to groups of users. Consequently, reliable multicast protocols have received a great deal of attention and have been studied extensively, and the area of reliable multicast continues to present many challenging problems. Generally, there are two schemes that provide reliable communication, Automatic Repeat Request (ARQ) and Forward Error Correction (FEC). ARQ based multicast protocols may suffer from the well-known feedback implosion problem, and hence, a variety of techniques have been proposed to improve their scalability [30][31][32]. Meanwhile, FEC-based approaches do not, in general, guarantee reliable multicast, and therefore, these approaches are often integrated with some form of an ARQ scheme [33] [34].

Under FEC, redundant parity packets are sent together with (or sometimes after) the original data packets. A popular scheme of linear-block FEC codes that have been proposed and used extensively for packet loss recovery is the family of Reed Solomon (RS) codes. A $RS(n,k)$ encoder takes $k$ message packets and produces $n-k$ parity packets. A receiver that receives any $k$ of the $n$ packets can reconstruct the original data. The $k$ message packets are referred to as a transmission group or TG, and the $n$ packets are referred to as an FEC block.

This thesis presents a new FEC-based packet-networking framework, which we refer to as Network-Embedded FEC (NEF). The proposed NEF approach exploits new and emerging overlay network paradigms by placing FEC codecs at selected nodes within an overlay multicast network. In an overlay multicast network, multicast functions such as

membership management and data replication are promoted to the application layer [49] [52]. Thus, our proposed NEF framework can also be supported at the application layer within an overlay network. Application-level functions, in general, and the proposed network-embedded FEC, in particular, could add a rather significant burden on the resources of the overlay network due (in this case) to the complexity of channel codecs. Therefore, only a small percentage of total nodes in the network should be assigned as codecs. In essence, the proposed NEF codecs work as signal regenerators in a communication system, and hence, they can reconstruct the vast majority (and sometimes all) of the lost data packets without requiring retransmission. This can be illustrated in a simple example as shown in Figure 1.

Figure 1 shows two multicast trees, (a) is a traditional IP multicast tree and (b) is an overlay multicast tree. Assume that the source sends a $RS(20,15)$ block; each link has a loss rate of 10%. For a traditional end-to-end route (e.g., under IP multicast) in Figure 1(a), the probability that the receiver can decode the FEC block is 26% (This is the same as the probability that the receiver receives 15 or more packets within a 20-packet FEC block). For the overlay multicast case (Figure 1(b)), if the codec receives more than 15 packets, it can reproduce the original data and parity packets and send 20 packets to the receiver. Since the codec is closer to the source than the receiver, it has a much higher probability to decode a FEC block than the receiver. Using this scheme, the probability that the receiver can decode FEC blocks increases from 26% to 69%. This improvement can be achieved while the re-generated packets (by the codec) experience the same loss probability as other packets.

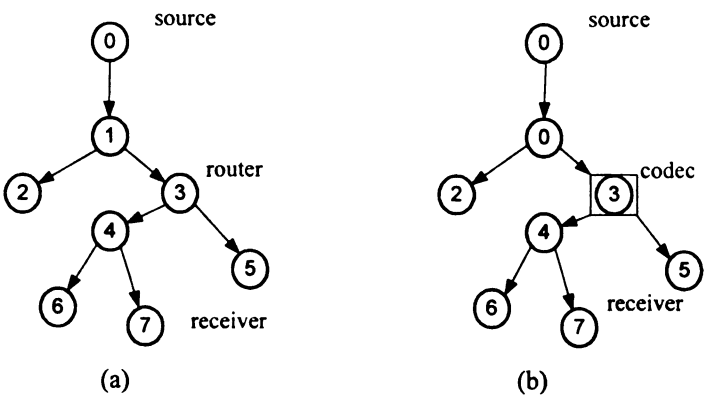



**Figure 1 (a) IP multicast, router does not participate in FEC (b) A NEF codec in a multicast tree can recover lost data and parity packets and send these packets downstream.**

When integrated with ARQ, NEF codecs can share the burden of the source to process feedbacks and retransmissions. When a receiver sends a feedback message along the reverse path of the multicast tree toward the source, the feedback message can be intercepted by the first codec it reaches. The codec will then send the repair packets to the receiver. As feedback and retransmission packets do not have to travel all the way to and back from the source, bandwidth usage and delay penalty decrease.

We show through extensive analysis and simulations that a small number of NEF codecs can significantly improve the overall goodput and decodable probability of a given multicast network. We believe that the proposed NEF approach can open the door for the development of new highly reliable multicast networks. These NEF-based networks can be designed with the desired level of reliability for the delivery of realtime (e.g., video and audio) or non-realtime content to a large number of users.

The main contributions of this work are:

- A Network-embedded FEC (NEF) framework has been proposed and developed. When compared with end-to-end FEC, NEF can greatly improve the playback

quality of a real time application using a relatively high code rate. NEF can also be integrated with ARQ to achieve complete reliability.

- The packet-loss model of NEF over random multicast trees that exhibits packet losses with memory over its links has been developed and analyzed. The model takes into consideration the Markov-chain nature of losses and the impact of linear-block based FEC on such losses.
- Both centralized and distributed codec placement algorithms have been designed and implemented. New Network Simulator (*ns2*) [82] components have been developed and integrated within *ns2* to conduct proper network simulations. The performance of both algorithms and their impacts on NEF-based network have been analyzed and evaluated.
- A hybrid ARQ and NEF (HANEF) reliable multicast protocol has been designed and implemented. The performance of HANEF has been evaluated and compared with other types if reliable multicast protocols.
- The use of NEF for multihop wireless multicast applications has been studied, comparison with other MAC layer multicast/broadcast protocols has been performed

The remainder of the thesis is organized as follows. Chapter 2 provides a brief overview of related work. Chapter 3 presents an analytical model for Network-Embedded FEC routes within an overlay multicast network. Chapter 4 describes and analyzes a centralized NEF codec placement algorithm. In Chapter 5, a distributed codec placement algorithm is presented. Chapter 6 gives the details of the design and implementation of the hybrid ARQ and NEF reliable multicast algorithm. In chapter 7, we study the use of

NEF in multihop wireless networks. Summary and future works are presented in Chapter 8.

# Chapter 2 Related work

In this chapter, basic concepts and background material related to the proposed NEF framework are briefly covered. We first review the IP multicast model and its recent development; we then give a short introduction to overlay and peer-to-peer networks. Reliable issues for real-time multicast applications are discussed in the last part of this chapter.

## 2.1 Multicast

Multicast is an efficient way to delivery data from one or more sources to a group of users. It is efficient because: 1) it does not put any additional burden on the multicast source or receivers; 2) it saves bandwidth in that only one copy of data is transmitted on each branch of a multicast tree. In contrast to unicast, which provide point-to-point communication, multicast provides one-to-many or many-to-many communications. Typical applications for multicast include video conferencing, remote learning, stock quotes and real-time broadcasting. Figure 2 shows a simple network with four end systems and two routers, where source A needs to send data to three receivers B, C and D. In Figure 2a, source A sends to each receiver a copy of the data using unicast. In this case, three copies of the data are transmitted on the link between source A and router1 and two copies of the data are transmitted between router1 and router2. For IP multicast (shown in Figure 2b), data packets are replicated at each router, only one copy of the data is transmitted on each link of the forwarding path, which means a significant saving on bandwidth resources.

The IP multicast model was first introduced by Stephen Deering in 1989 [1]. The model is based on the concept of group. A multicast group consists of users who are interested in receiving a particular data stream. The group is open; sources only need to know a multicast address[1], they do not need to know group membership; sources do not need to be group members to which they are sending. The group is dynamic; users can join or leave a multicast group at will. The group does not have any physical or geographical boundaries—every host connected to the internet can join the group. IP multicast is based on UDP; packets are delivered using a best-effort policy.

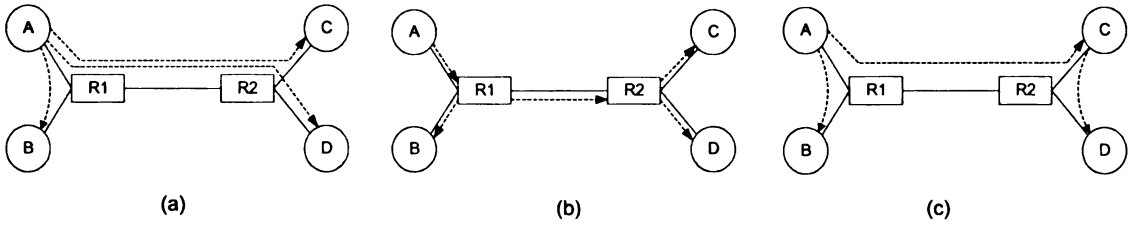



**Figure 2 (a) multiple unicast, (b) IP multicast (c) overlay multicast**

The first network that provided multicast service was the Multicast Backbone (Mbone) [2], which was created in 1992. The Mbone uses Distance Vector Multicast Routing Protocol (DVMRP) [3] to build and maintain a multicast distribution tree. Several intra-domain multicast routing protocols were then developed and became standards. These include Multicast Extension of OSPF (MOSPF) [4], Protocol Independent Multicast dense mode (PIM-DM) [5], Protocol Independent Multicast sparse mode (PIM-SM) [6] and Core Based Trees (CBT) [7]. Inter-domain multicast is much more complex, a temporary solution is to use three protocols working together, these

[1] The Internet Assigned Authority (IANA) has assigned the class D address space for IP multicast, which falls in the range of 224.0.0.0 to 239.255.255.255.

protocols are Multiprotocol extensions for BGP (MBGP) [9], Multicast source discovery protocol (MSDP) [10] and PIM-SM[6].

The difficulties in multicast address management [12][13] and inter-domain multicast implementation have lead researchers to make some fundamental changes to the IP multicast model. Express Multicast [14] brings up the logical channel model. In Express, the source and multicast group address (S, G) pair forms a unique multicast channel, group members must send explicit join message along the reverse unicast path to the source. Express solves the address management problem; it also reduces the routing complexity and makes the collection of information about subscribers much easier. Express is designed for single source multicast applications, Simple Multicast [15] is similar to Express multicast but it allows multiple sources per group.

Another issue is that it is very difficult to implement congestion control in IP multicast. In unicast, congestion control can be achieved by transport layer protocols like TCP; for multicast, things become much more complicated; the paths lead from the source to different users may have different channel capacities; different end users may have different memory and computation capability and may also have different quality requirements. Much effort has been put into designing multicast congestion control algorithms [16]-[28], yet congestion control for multicast remains a challenging problem.

## 2.2 Overlay and Peer-to-Peer Networks

For reasons we have described above, the implementation of IP multicast is very slow. This has lead research effort to overlay multicast [49]-[58]. In overlay multicast, multicast related functionality such as membership management and packet replication

are implemented in end systems rather than intermediate routers. End systems self-organize into an overlay network upon the underlying unicast infrastructure, multicast are actually implemented on top of the IP unicast service.

There two kinds of overlay networks, proxy-based and peer-to-peer (P2P) [60][61][62]. In P2P overlay multicast network, each node in the multicast tree (or mesh) are also multicast clients. At each node in the network, data packets will go all the way up to the application level and then be replicated and forwarded. In the proxy based overlay networks, proxies provide application level services to multicast clients. Nodes between the sources and proxies and between proxies and clients are still running conventional multicast (IP multicast) protocols.

When compared with IP multicast, overlay multicast does not need to add any additional burden on intermediate routers. This adheres to the long standing belief that intelligence should be pushed to the edge of the network. In IP multicast, routers need to keep the status of the multicast tree, replicate packets, and running complicated multicast routing protocols. The implementation of these functions requires fundamental changes to the existing internet infrastructure. Though most routers nowadays implement native IP multicast, they introduce much complexity and serious scale problems at the IP layer. By shifting the multicast functionalities from routers to end systems, a major obstacle of implementing multicast is removed.

Another advantage of overlay multicast is that it makes higher layer functionalities such as congestion control and error correction much easier. As overlay multicast is built upon unicast, it can resort to transport layers protocols such as TCP for congestion control. Overlay multicast also give more flexibility for error correction. The proposed Network-

Embedded FEC exploits the flexibility of the overlay multicast to provide reliable transmissions for multicast applications.

There are also drawbacks for overlay multicast. Figure 2c shows an example for overlay multicast. When compared to IP multicast in Figure 2b, overlay multicast may introduce multiple copies of packets on some of the branches of the multicast tree such as the hop from node A to router1 and the hop from router2 to node C. Packets may not always follow the optimum shortest path from the source to each receiver; in this example packets from A to D does not follow the shortest path. This may introduce extra delay for some users. In [59], Castro et al showed that, as compared to IP multicast, overlay multicast increases average delay penalty by 1.5-2.5 times and average link stress by 1.3-1.5 times.

When compared to unicast in Figure 2a, overlay multicast can still save bandwidth. This is especially true if the overlay network is carefully designed, and only one copy of data packets is on expensive and critical paths. In Figure 2c, if the path from router1 to router2 is an inter-continent hop, the bandwidth saving is still significant.

## 2.3 Realtime Reliable Multicast

Realtime applications such as video/audio multicast can often tolerate limited packet losses. Some of these applications, such as multimedia streaming, could also tolerate some initial delay. For such applications, "reliable realtime" multicast is somewhat different from the conventional end-to-end reliable protocols: "Reliability" in this case does not necessarily imply a none-fault (perfect) delivery of every packet to every user in the group. "Realtime reliability" is rather still a best effort protocol in the sense that it

enables each receiver to recover as many lost or corrupted packets as possible before the playback deadline expires. Further, for these applications, ARQ schemes may add very large round trip time (RTT) delay to recover a lost packet; this makes FEC more appealing for many realtime applications. Some proactive FEC protocols were designed for realtime and non-realtime services. These protocols require no receiver feedback and retransmission, like RMDP [35] and Digital Fountain [36]; other implementations explore the possibility of integrating FEC and ARQ [30][38]. In this proposal, we advocate the employment of NEF-based multicast *independently* (without ARQ) with certain realtime applications (e.g., streaming) in mind.

# Chapter 3 Analysis of Network Embedded FEC Based Routes

In this chapter, we first review the packet loss pattern in the internet which, in general, can be captured by the Gilbert Model. In the second section of this chapter we give a short introduction to system analysis of Markov process; we then use this powerful tool to perform probabilistic analyze of the Gilbert Model in the context of FEC. In the last part of this chapter, we describe the analysis of network-embedded FEC based routes, which will be used in our codec placement algorithm in the following chapter.

## 3.1 The Packet Loss Model

Packet loss patterns have a crucial impact on the performance of the channel coding used over multicast trees. This in turn impacts the message-packet goodput of both realtime and non-realtime multicast applications. In particular, the message goodput (or conversely the effective packet loss) experienced by a realtime application provides a direct measure for the quality of the received played-back multimedia content. The packet loss patterns are influenced by the underlying random process that induces these patterns. Analysis in [40] and [41] have used a binomial loss model, where the only parameter used to capture the loss process is the average loss rate. In reality, this is not enough as packet loss often occurs in bursts. For a particular receiver, if one packet is lost, the packet following this packet is more likely to get lost, in other words, the packets loss are temporary correlated. Two reealtime applications experienced the same average loss rate but different length of loss bursts may show complete different play out quality.

A class of random process that can well capture the correlations between lost packets and, at the same time, not difficult to analysis, is Markov Chain.

Consider a random process $\{X_n, n = 0,1,2,....\}$, where $X_n$ can take on a finite set of possible values. This set of possible values can be denoted by a set of nonnegative integers $\{0,1,2,....\}$, we call this set of nonnegative integers as the states of the random process. If $X_n = i$, the process is said to be in state $i$ at time $n$. A Markov chain has the property that, given the present state, the conditional distribution of the future state is independent of the past state, in other words,

$$P\{X_{n+1} = j \mid X_n = i, X_{n-1} = i_{n-1}, ...., X_1 = i_1, X_0 = i_0\} = P\{X_{n+1} = j \mid X_n = i\}$$

This probability is also defined as the transition probability of the Markov process, denoted as $p_{ij}$, that is,

$$p_{ij} = P\{X_{n+1} = j \mid X_n = i\}$$

$p_{ij}$ represents the probability that the process, when in state $i$, will next make a transition into state $j$. For any state $i$ and $j$, we have

$$p_{ij} \geq 0, \quad i, j \geq 0; \qquad \sum_{j=0}^{\infty} p_{ij} = 1, \quad i = 0,1,....$$

If we use **P** to denote the matrix of one-step transition probability of the process, we have

$$\mathbf{P} = \begin{vmatrix} p_{00} & p_{01} & \cdots & p_{0j} & \cdots \\ p_{10} & p_{11} & \cdots & p_{1j} & \cdots \\ \vdots & \vdots & & \vdots & \\ p_{i0} & p_{i1} & \cdots & p_{ij} & \cdots \\ \vdots & \vdots & & \vdots & \end{vmatrix}$$

The Gilbert model is the most widely used model to simulate the packet loss process [76][77]. Gilbert model is represented by a two-state Markov chain. When the process is in good state, all packets are received correctly; when the process is in bad state, all packets are lost or corrupted. The state diagram of the Gilbert model is shown in Figure 3, where state 0 and state 1 represent the Good state and Bad state, respectively.

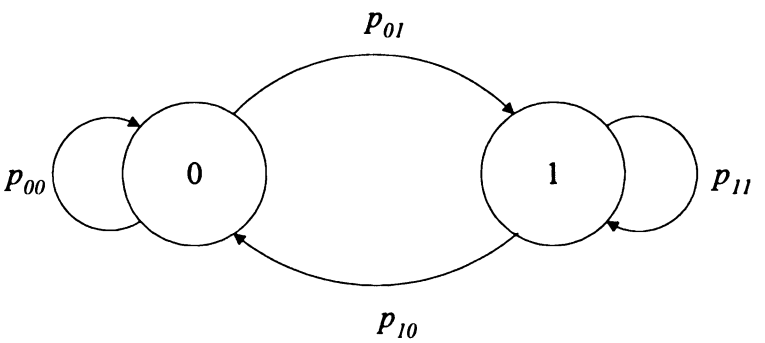



**Figure 3 State Diagram of Gilbert Model**

the one-step transition probability matrix for this process is

$$P = \begin{bmatrix} p_{00} & p_{01} \\ p_{10} & p_{11} \end{bmatrix}$$

We are often interested in multiple step transition probability. Let $\phi_{ij}(n)$ be the probability that the process will occupy state $j$ at time $n$ given that it occupied state $i$ at time 0,

$$\phi_{ij}(n) = P\{X_n = j \mid X_0 = i\} \quad 0 \le i, j \le 1, n = 0,1,2,....$$

the quantity $\phi_{ij}(n)$ is called the $n$-step transition probability of the Markov process from state $i$ to state $j$. Using matrix notation, the $n$-step transition probability matrix

$$\phi(n) = \{\phi_{ij}(n)\} = \begin{bmatrix} \phi_{00}(n) & \phi_{01}(n) \\ \phi_{10}(n) & \phi_{11}(n) \end{bmatrix} \quad n = 0,1,2,....$$

in general, we have

$$\phi(n) = P^n \quad n = 0,1,2,....$$

where $P^0 = I$, $I$ is the identity matrix.

The steady state probability of this process is

$$\pi(0) = \frac{p_{10}}{p_{01} + p_{10}}$$

and

$$\pi(1) = \frac{p_{01}}{p_{01} + p_{10}}$$

respectively.

In [37], Yajnik have used a general $k$ order Markov chain to model the packet loss process. Here, the packet loss events observed by a particular receiver is mapped to a series of random variables that take on two values, 0 and 1. When the receiver correctly receives a packet, the random variable is 0, when a packet is lost, the random variable is 1. The probability of the current packet will be received or lost will depend upon the status of the last $k$ events. This process can be characterized by a conditional probability matrix $\mathbf{P_k}$ where each element in the matrix can be obtained by the following definition:

$$P_k(x_i \mid x_{i-1},....x_{i-k}) = \Pr ob(X_i = x_i \mid X_{i-1} = x_{i-1},....X_{i-k} = x_{i-k})$$

For an example, in a 3 order Markov chain, $P\{X_i = 0 \mid X_{i-1} = 0, X_{i-2} = 1, X_{i-3} = 0\}$ represent the probability that at time *i,* the receiver will receive a packet correctly given that at time *i-1*, a packet is received; at time *i-2*, a packet is lost and at time *i-3*, a packet

is received. Apparently, for a three order Markov chain, we need 8 states to describe the process. For a general $k$ order Markov chain, we need $2^k$ states. In [37], the authors have used the entropy analysis to show that the 3 order (8 states) Markov chain is the best to simulate the packet loss in Mbone. The general $k$ order Markov chain requires much more states than the simple Gilbert model, from the statistic analysis in [37], its accuracy improvement over the Gilbert model in simulating the loss process is limited.

The simple Gilbert model may not capture the various loss burst length accurately. The general $k$ order Markov Chain requires too many states, and thus incurs significant analysis complexity. Sanneck and Carle in [77] advocate the use of loss run-length to define the states of a Markov chain. In the loss run-length model, a random variable $X$ is defined as follows: $X = 0$: "no packet lost", $X = k$: "exactly $k$ consecutive packets lost", $X \geq k$: "at least $k$ consecutive packets lost". The transition graph of a $m+1$ states loss run-length model is showed in Figure 4:

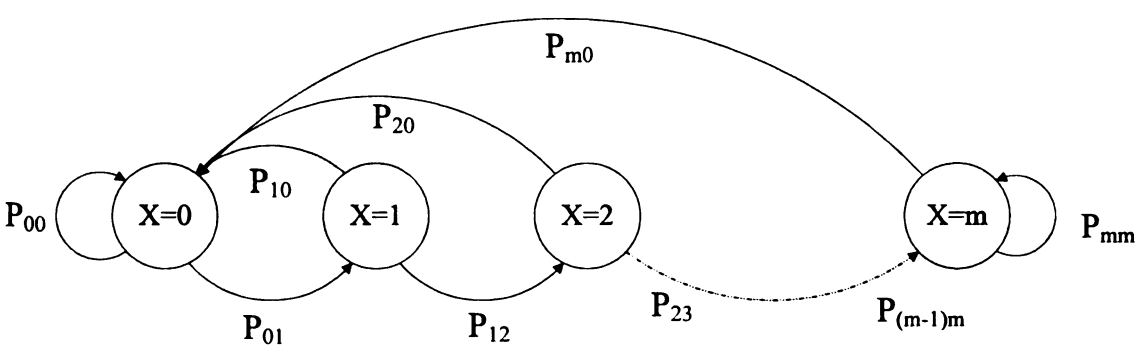



**Figure 4 loss run-length model with (m+1) states [77]**

In this model, when $k = 0$ (no packet lost), if the next packet is correctly received, the process will stay in state $X = 0$, if the next packet is lost, the process will transit into state $X \geq 1$; for $0 < k < m$, every additional lost packet will cause the process transit into the next state, every successfully received packet will cause the process return to state 0;

when the process reaches $X = m$, if the next packet is correctly received, the process returns to state $X = 0$, if the next packet is lost, the process will remain in state $X = m$. The transition matrix of this model is as following:

$$\mathbf{P} = \begin{vmatrix} p_{00} & p_{01} & 0 & \cdots & 0 \\ p_{10} & 0 & p_{12} & & \vdots \\ p_{20} & 0 & 0 & \ddots & \vdots \\ \vdots & \vdots & \vdots & & 0 \\ p_{(m-1)0} & 0 & \cdots & 0 & p_{(m-1)m} \\ p_{m0} & 0 & 0 & \cdots & p_{mm} \end{vmatrix}$$

From the tractability point of view, the Gilbert model provides the best fit. Authors in [37] compared the third order general Markov model and the Gilbert model using entropy analysis. Jian in [76] compared the Gilbert model with the loss run-length model. Both have shown that the Gilbert model provided adequate accuracy in simulating the loss process. In the following, we will use the Gilbert model in our analysis.

## 3.2 System Analysis of Markov Process

For FEC-based real-time applications, one key measure that will determine the quality of playback is the message goodput; another key performance measure, which directly influences the effective goodput, is the probability of recovering all message packets (e.g., $k$ packets) within an FEC block of a given size (e.g., $n$ packets). We refer to this measure as the *decodable probability*. This probability measure and the message goodput are rather straightforward to evaluate if the channel is memory-less. However, for channels with memory, such as Markov channels, these performance measures are not trivial to evaluate.

In [78], Yee and Weldon provided an approach for evaluating the Bit Error Probability (BEP) of the Gilbert-Elliott channel using a combinatorial method. This method, however, is rather complicated to evaluate and does not lend itself to simple analysis of complex networks such as large multicast trees. In [79], the author provides a powerful tool for the analysis of general Markov processes using a system analysis approach. Below, we describe, extend, and employ this approach for the analysis of Markov channels and routes that (1) exhibit lost packets and (2) employ FEC codecs.

The author in [79] developed an approach to study the dynamic probabilistic systems based on the techniques of linear system analysis. A system is defined as an operator that takes one input signal and produces one output signal. Here we are only concerned with discrete systems, where the input and output signals are discrete functions; we use $f(n)$ and $g(n)$ to represent the input signal and output signal, respectively. We further restrict our analysis to linear time-invariant systems. The characteristic of a linear time-invariant discrete system can be uniquely specified by its unit impulse response, $h(n)$. For linear time-invariant discrete systems, we have the following

$$g(n) = \sum_{k=0}^{n} f(k)h(n-k) \quad n = 0, 1, 2, ....$$

We can use vectors to represent the input and output discrete series. The input series $f(0), f(1), f(2), .....$ is represented by $\vec{f}$; similarly, $g(n)$ and $h(n)$ are represented by $\vec{g}$ and $\vec{h}$. The above equation is the well-known convolution of the input and the unit impulse response of the system. Using vector representation, we have the following:

$$\vec{g} = \vec{f} * \vec{h}$$

For large and complicated systems, time domain analysis can become very difficult; system analysis for large systems is often accomplished in the transform domain. The geometric transformation of a sequence $f(n)$ is defined as following:

$$f(z) = \sum_{n=0}^{\infty} f(n) z^n$$

In the transform domain, the relationship between the input and the output becomes:

$$g(z) = f(z)h(z)$$

In the transform domain, the system can be represented by a flow graph:

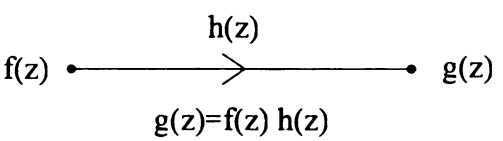



**Figure 5 the basic flow graph**

In the system flow graph, joining the input and output nodes is a line segment, called a branch, directed from the input to the output. $h(z)$ is also called the transmission of the branch.

Feedback system is a common component in complex systems, especially in dynamic probabilistic systems. The following figure gives the block diagram and the flow graph of a simple feedback system. For this feedback system, the relationship between the input and output can be represented by the following equation:

$$\vec{g} = \vec{f} + \vec{g} * \vec{h}$$

In the transform domain, this becomes:

$$\frac{g(z)}{f(z)} = \frac{1}{1-h(z)}$$

This implies that a feedback loop has a transfer function that equals to $[1-h(z)]^{-1}$.

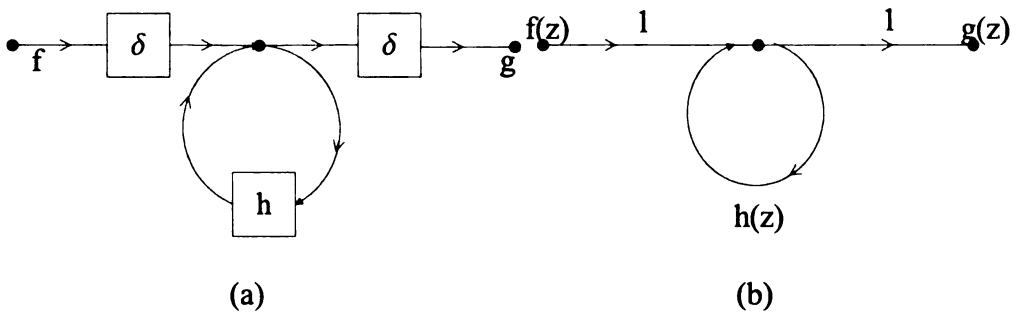



**Figure 6 (a) block diagram of a feedback system. (b) flow graph of a feedback system**

It is trivial to see that two branch systems connected in parallel is equivalent to a single branch system whose transfer function is the sum of the transfer functions of the individual branches; two systems connected in series have an equivalent transfer function that is the product of the transfer function of each system. Using these equivalent relations, very complex systems can be reduced to very simple systems.

Systems with multiple inputs and multiple outputs can be described using matrix notation. For systems with $N$ inputs and $M$ outputs, their input and output relation can be represented by

$$\mathbf{g}(n) = \sum_{k=0}^{n} \mathbf{f}(n)H(n-k)$$

Here, **f**(n) and **g**(n) are row vectors each has N and M components respectively. H is N by M matrix. In the transform domain, the relationship becomes:

$$\mathbf{g}(z) = \mathbf{f}(z)\, H(z)$$

Here, **f**(z) is the N-dimensional vector of input signal transforms, **g**(z) is the M-dimensional vector of output signal transforms and H(z) is the N by M matrix of transfer functions that completely describe the system.

The author in [79] has shown that a Markov process is in fact a time-invariant, physically realizable linear process, and the system analysis methods we described above can be used to analysis Markov process. As it is well known, the $n$-step transition probability of a Markov process can be represented by:

$$\phi(n) = P^n \quad n = 0,1,2,....$$

where $P$ is the one-step transition probability matrix of the Markov process. The geometric transform of $\phi(n)$ is

$$\phi(z) = [I - Pz]^{-1}$$

This is equal to the transmission matrix of a feedback matrix system whose feedback transmission is $Pz$, as shown below:

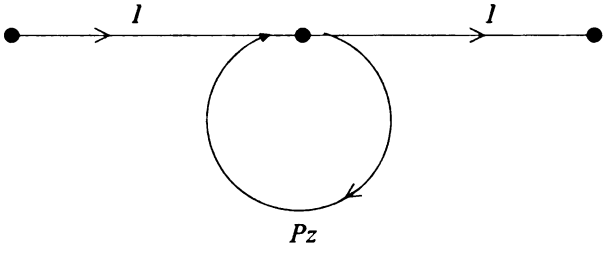



**Figure 7 flow graph of a feedback matrix system**

The flow graph corresponding to the Matrix system is the Markov process transition diagram with each branch labeled with $p_{ij}z$ instead of $p_{ij}$. Figure 8 is the flow graph of a two state Markov process.

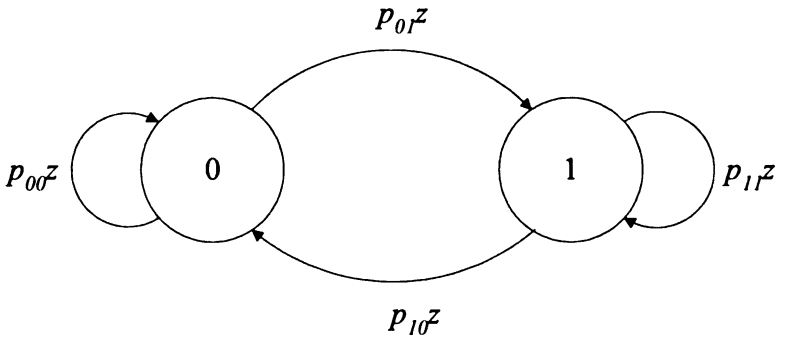



**Figure 8 flow graph of the Markov process**

## 3.3 Probability Analysis of the Gilbert Model[2]

Based on the framework described above, we present a new method for evaluating any desired loss/recovery probability measure for the Gilbert model. In particular, we present a rather elegant and simple approach for evaluating the probability of receiving *i* packets among *n* packets transmitted over Gilbert erasure channels.

To evaluate the desired probability measure for the Gilbert model, we construct another Markov process by extending the two-state Gilbert model. We use *G* and *B* to indicate that the process is in Good state or Bad state respectively; we use the number of correctly received packets as the indexes for the states of the extended Markov chain. For example, if the receiver correctly receives *i* packets and the channel is in a good state, then the process is in state $G_i$ ; on the other hand, if the receiver correctly receives *i* packets and the channel is in a bad state, then the process is in state $B_i$. The state transition of the new (extended) Markov process is shown in Figure 9.

[2] Part of this work has been published in [92]

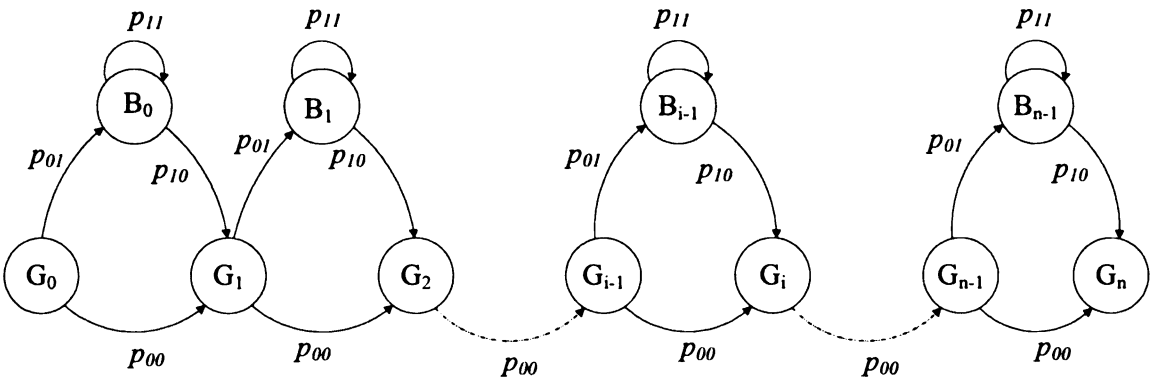



**Figure 9 Extended state transition diagram for the Gilbert model**

The probability that the sender transmits $n$ packets and the receiver correctly receives $i$ packets is the probability that the process starts at $G_0$ or $B_0$ and is in state $G_i$ or $B_i$ after $n$ stages. We use $\phi_{G_0G_i}(n)$ to represent the multistep transition probability from $G_0$ to $G_i$:

$$\phi_{G_0G_i}(n) = P\{s(n) = G_i \mid s(0) = G_0\} \qquad (1)$$

where $s(j)$ is the state of the extended Markov process at time index $j$. Using the method described in the above section, the flow graph of the extended Gilbert model is:

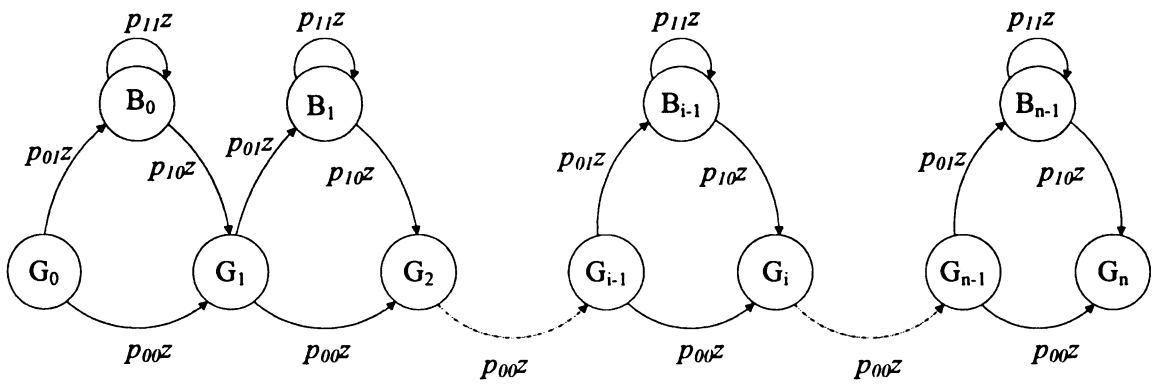



**Figure 10 flow graph of the extended Gilbert model**

Using the flow graph reduction methods,the flow from $G_0$ to $G_k$ can be simplified as:

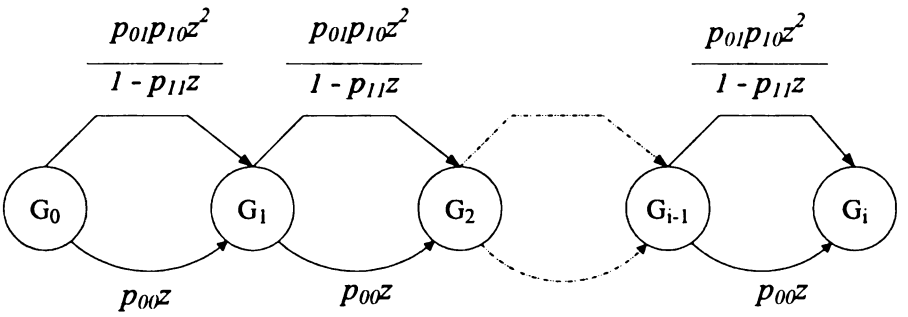



**Figure 11 simplified flow graph of the Gilbert model**

Let $\phi_{G_0G_i}(z)$ be the geometric transform of $\phi_{G_0G_i}(n)$, then $\phi_{G_0G_i}(z)$ is just the transmission from $G_0$ to $G_k$, from the flow graph above, $\phi_{G_0G_i}(z)$ can be obtained as:

$$\phi_{G_0G_i}(z) = (p_{00}z + \frac{p_{01}p_{10}z^2}{1-p_{11}z})^i \quad 0 < i \le n \qquad (2)$$

Now, taking the inverse $Z$ transform, we have:

$$\phi_{G_0G_i}(z) = p_{00}^i z^i (1 + \frac{p_{01}p_{10}z}{p_{00}(1-p_{11}z)})^i$$

$$= p_{00}^i z^i \sum_{m=0}^{i} \binom{i}{m} (\frac{p_{01}p_{10}z}{p_{00}(1-p_{11}z)})^m$$

$$= p_{00}^i z^i \sum_{m=0}^{i} \binom{i}{m} (\frac{p_{01}p_{10}}{p_{00}})^m z^m (\frac{1}{1-p_{11}z})^m$$

$$= p_{00}^i z^i + \sum_{m=1}^{i} \binom{i}{m} p_{01}^m p_{10}^m p_{00}^{i-m} z^{m+i} (\frac{1}{1-p_{11}z})^m$$

We know that for $m \ge 1$ the inverse transform of $(\frac{1}{1-p_{11}z})^m$ is

$$\frac{1}{(m-1)!}(n+1)(n+2)....(n+m-1)p_{11}^n$$

then the inverse transform of $z^{m+i}(\frac{1}{1-p_{11}z})^m$ is

$$\frac{(n-m-i+1)(n-m-i+2)....(n-i-1)p_{11}^{n-m-i}}{(m-1)!}$$

$$=\binom{n-i-1}{m-1}p_{11}^{n-m-i}$$

From above we can get:

$$\phi_{G_0G_i}(n)=p_{00}^{i}\delta(n-i)+\sum_{m=1}^{i}\binom{i}{m}\binom{n-i-1}{m-1}p_{01}^{m}p_{10}^{m}p_{00}^{i-m}p_{11}^{n-i-m}$$

When $k=0$, it is trivial to get $\phi_{G_0G_k}(n)=0$. Summery the above results, we get

$$\phi_{G_0G_i}(n)=\begin{cases}\sum_{m=1}^{i}\binom{i}{m}\binom{n-i-1}{m-1}p_{01}^{m}p_{10}^{m}p_{00}^{i-m}p_{11}^{n-i-m} & 0<i<n\\ 0 & i=0\\ p_{00}^{n} & i=n\end{cases} \quad (3)$$

Similarly, it can be shown that:

$$\phi_{G_0B_i}(n)=\begin{cases}\sum_{m=0}^{i}\binom{i}{m}\binom{n-i-1}{m}p_{01}^{m+1}p_{10}^{m}p_{00}^{i-m}p_{11}^{n-i-m-1} & 0\le i<n\\ 0 & i=n\end{cases} \quad (4)$$

$$\phi_{B_0G_i}(n)=\begin{cases}\sum_{m=0}^{i-1}\binom{i-1}{m}\binom{n-i}{m}p_{01}^{m}p_{10}^{m+1}p_{00}^{i-m-1}p_{11}^{n-i-m} & 0<i\le n\\ 0 & i=0\end{cases} \quad (5)$$

$$\phi_{B_0B_i}(n)=\begin{cases}\sum_{m=0}^{i-1}\binom{i-1}{m}\binom{n-i}{m+1}p_{01}^{m+1}p_{10}^{m+1}p_{00}^{i-m-1}p_{11}^{n-m-i-1} & 0<i<n\\ p_{11}^{n} & i=0\\ 0 & i=n\end{cases} \quad (6)$$

It is worth noting that [80] derives a similar expression, however using a very different approach and different notations.

If we use $\phi(n,i)$ to represent the probability that the sender transmits $n$ packets and the receiver correctly receives $i$ packets, then:

$$\phi(n,i)=\pi(0)(\phi_{G_0G_i}(n)+\phi_{G_0B_i}(n))+\pi(1)(\phi_{B_0G_i}(n)+\phi_{B_0B_i}(n)) \quad (7)$$

where $\pi(0)=\frac{p_{10}}{p_{01}+p_{10}}$ and $\pi(1)=\frac{p_{01}}{p_{01}+p_{10}}$ are the steady state probabilities in good state and bad state respectively.

The desired probability measure $\phi(n,i)$ can be completely evaluated using any two parameters that characterize the underlying Gilbert erasure channel. Traditionally, the transitional probabilities $p_{01}$ and $p_{10}$ (or $p_{00}=1-p_{01}$ and $p_{11}=1-p_{10}$) are used for such characterization. A more useful insight and analysis can be gained by considering other parameter pairs. In particular, in [78] the authors have used the average loss rate $p$ and the *packet correlation* $\rho$ to represent the state transition probabilities. The average loss rate of the Gilbert channel is

$$p=\frac{p_{01}}{p_{10}+p_{01}}$$

The correlation between to consecutive error packets is:

$$\rho = p_{01} + p_{10} - 1$$

The transition probabilities can be represented by $p$ and $\rho$ as:

$$p_{00} = 1 - p(1-\rho);\ p_{01} = p(1-\rho)$$

$$p_{10} = (1-p)(1-\rho);\ p_{11} = 1 - (1-p)(1-\rho)$$

The steady state probabilities are directly related to the loss rate $p$: $\pi(0) = 1 - p$ and $\pi(1) = p$. The packet erasure correlation $\rho$ provides an average measure of how the states of two consecutive packets are correlated to each other. In particular, when $\rho = 0$, $p_{01} + p_{10} = 1$, the loss process is memory-less, and the above probability measures reduce to the special case of a memory-less Binary Erasure Channel (BEC). On the other hand, as the value of $\rho$ increases, then the states of two consecutive packets become more and more correlated. Hence, we find the parameters $p$ and $\rho$ provide an intuitive, insightful, and broad characterization for the impact of channel coding on networks with losses. Later in the proposal, we will present our analysis and simulation results for the centralized and distributed NEF algorithms in terms of the two parameters, average loss rate $p$ and the packet correlation $\rho$, instead of the traditional transitional probability parameters $p_{01}$ and $p_{10}$.

Figure 12 plots the probability of a receiver correctly receives *i* packets when the source sends *n* packets over a Gilbert channel. Here, *n* is set to 30, the average loss rate $p$ is set to 1% and the packet correlation $\rho$ is changed from 0 to 0.9. When compared with the Binomial model (where $\rho = 0$ ), if the number of packets sent is set unchanged, we can see that as $\rho$ increases, the probability of receiving a smaller number of packets increases. For a given $\rho$, as $i$ increases, $\phi(n,i)$ increases exponentially; this increase

slows down as $\rho$ increases. When $\rho = 0.9$, we can see that $\phi(n,i)$ has a small spike at $i = 0$, and a big spike at $i = 30$. This observation is in accordance with common sense; when the correlation is very strong, once the process initially starts in bad or good state, it has the inertia to stay in that state. For $p = 0.01$ and $\rho = 0.9$, the transition probabilities are $p_{00} = 0.999$, $p_{01} = 0.001$, $p_{10} = 0.099$, $p_{11} = 0.901$ respectively.

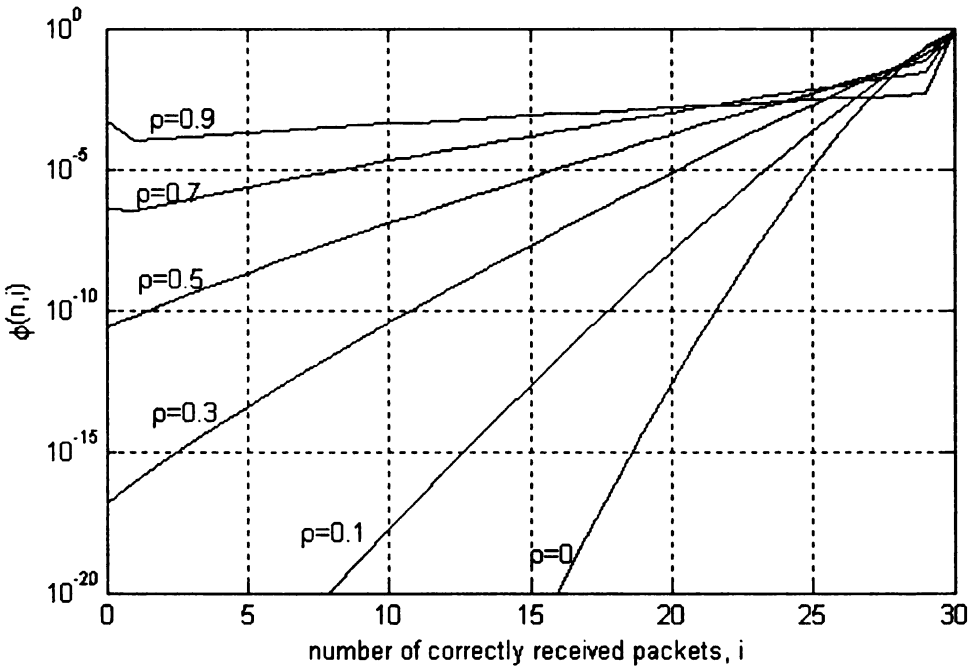



**Figure 12 probability of a receiver to receive *i* packets**

In FEC codes, we are often concerned with the probability that a node can receive enough packets to decode an FEC block. For a $(n,k)$ FEC code, this probability is:

$$P(i \geq k) = \sum_{i=k}^{n} \phi(n,i)$$

Figure 13 shows the average decodable probability of a receiver when the sender transmits FEC blocks through a Gilbert channel. The block size $n$ is set to 30, the

average loss rate of the channel is set to 1%; $k$ is changed from 20 to 30, and the packet correlation $\rho$ is changed from 0 to 0.9.

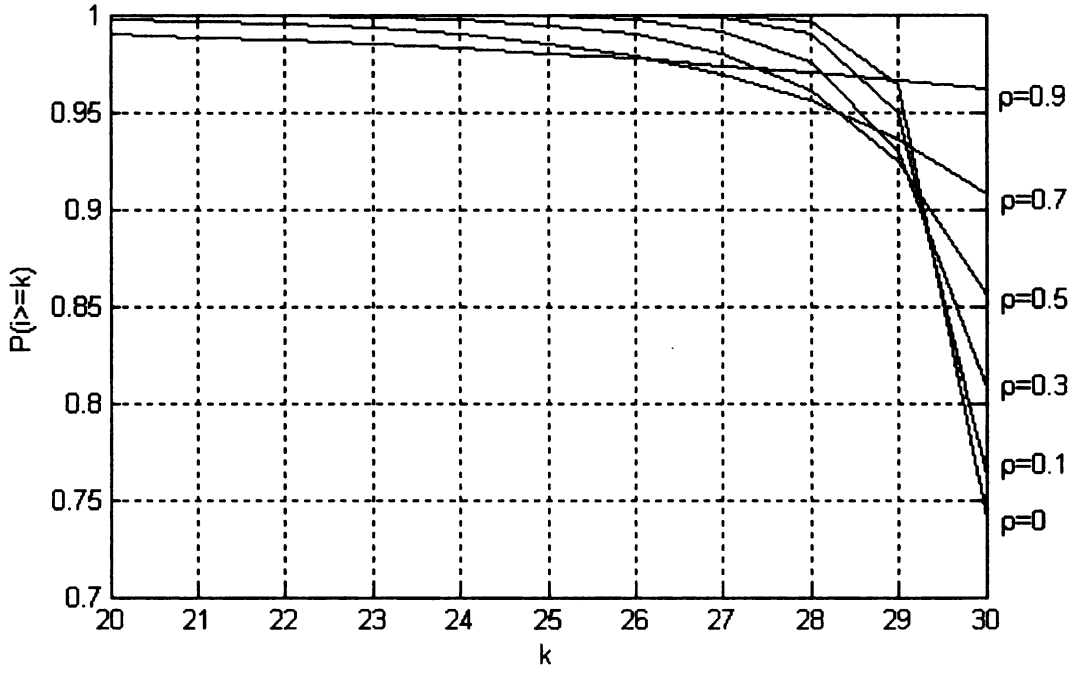



**Figure 13 decodable probability for a receiver to receive packets through a Gilbert Channel**

From the plot, we can see that for a given average loss rate (1% here), when the code rate is low, the higher the packet correlation, the lower the decodable probability; however this phenomenon changes when the code rate increases. For example, when the code rate $k/n \geq 27/30$, the decodable probability for $\rho = 0.9$ is higher than that when $\rho = 0.7$; when the code rate $k/n = 29/30$, the decodable probability for $\rho = 0.7$ is higher than that when $\rho = 0.5$. In the Gilbert model, the average burst length is $1/p_{10} = 1/(1-p)(1-\rho)$, when $\rho$ increases, the burst length increases. For high code rate FEC codes, any FEC block suffer a few packet loss may not be able to be decoded. Given the average loss rate unchanged, longer loss burst means fewer FEC blocks that suffer

from packet losses; however, these fewer FEC blocks with packet losses exhibit long bursts of losses (due to the high correlation). Therefore, having fewer FEC blocks with losses (although bursty) may result in a higher decodable probability.

When designing FEC codes such as $RS(n,k)$ for a Gilbert channel, the code rate must be chosen carefully. Code rate can be adjusted by keeping one of the two variables ($n$ or $k$) constant and change the other. Keeping $n$ unchanged, the number of packets the receiver will receive is a random variable represented by $K$; on the other hand, if we want the receiver side to receive at least $k$ packets, where $k$ is a constant, the number of packets we should send at the source is a random variable represented by $N$. We want to find the mean and variance of $K$ and $N$. For the binomial loss pattern, this is trivial. It is informative if we could also know these parameters for the Gilbert model. Later in our distributed codec placement algorithm, we need to estimate the variance of the number of packets that a codec requires in order to decode a FEC block. The characteristic of these parameters will help us in designing a more efficient distributed codec placement algorithm.

The mean and variance of $K$ is just the mean and variance of state occupancy in Good state when the Markov process is in stage $n$. Let $E_{GG}[k]$ and $E_{BG}[k]$ be the average occupancy in good state when the initial channel state is in Good and Bad states, respectively. We then readily have [79] :

$$E_{GG}[k] = (1-p)(n+1) + \frac{p(1-\rho^{n+1})}{1-\rho} - 1 \qquad (9)$$

$$E_{BG}[k] = (1-p)(n+1) - \frac{(1-p)(1-\rho^{n+1})}{1-\rho} \qquad (10)$$

The average of $K$ is then:

$$E[k] = \pi(0)E_{GG}[k] + \pi(1)E_{BG}[k]$$

$$= (1-p)n \qquad (11)$$

We see that the average received packets are the same as that in the binomial model and it does not depend on the correlation coefficient $\rho$.

Unlike the mean, the variance does depend on the correlation coefficient $\rho$. A closed form of the variance of state occupancy is rather complicated, an asymptotic form of the variance is:

$$Var[k] = \pi(0)Var_{GG}[k] + \pi(1)Var_{BG}[k]$$

$$= np(1-p)\frac{1+\rho}{1-\rho} - \frac{\rho p(1-p)(\rho+2)}{(1-\rho)^2} \qquad (12)$$

Figure 14 plots the impact of packet correlation $\rho$ on the variance of received packet.

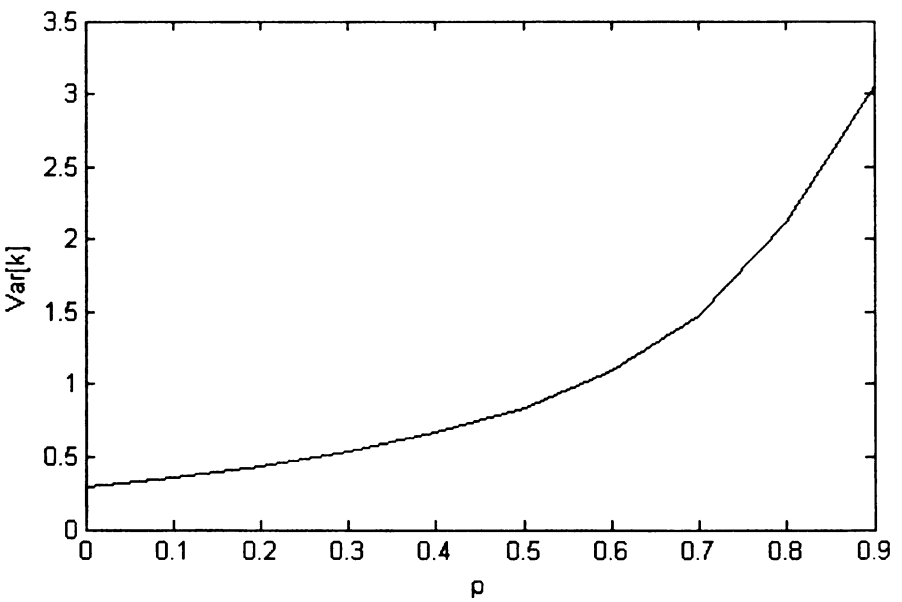



**Figure 14 variance of k versus packet correlation ρ**

Here $n$ is set to 30 and the average loss rate is set to be 1%. We can see that when $\rho = 0$, the variance is the same as that of the binomial model; as $\rho$ increases, the variance of $k$ increases; the slop of the increase also grows with $\rho$ increases, and the variance approaches infinity as $\rho$ approaches 1.

To calculate the mean and variance of $N$ (in order for the receiver to receive more than $k$ packets), we extend the state transition diagram similar to the one we used when we derived the expression for $\phi(n,i)$. The difference is that the process is now trapped at state $G_k$. This is shown in Figure 15. Now the mean and variance of $N$ is just the mean and variance of the stages for the process to be trapped into state $G_k$. Using the similar approach we have used before, we have:

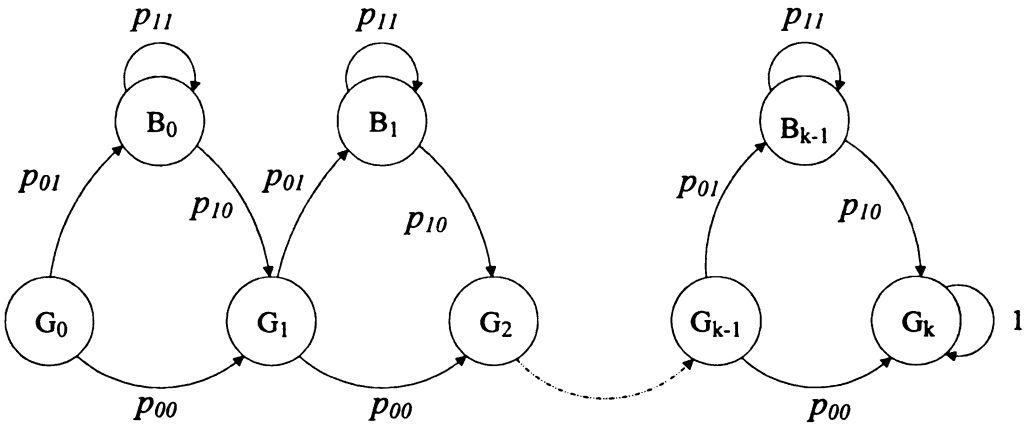



**Figure 15 extended Markov model for analysis of mean and variance of $N$**

$$\phi_{G_0G_k}(z) = \frac{1}{1-z}\left(p_{00}z + \frac{p_{01}p_{10}z^2}{1-p_{11}z}\right)^k \qquad (13)$$

Let $P_{G_0G_k}(n)$ be the probability that the process starts at state $G_0$ and will be trapped in state $G_k$ exactly at stage $n$, that is

$$\phi_{G_0G_k}(n) = \sum_{m=0}^{n} P_{G_0G_k}(m) \tag{14}$$

We have

$$P_{G_0G_k}(z) = (1-z)\phi_{G_0G_k}(z) = (p_{00}z + \frac{p_{01}p_{10}z^2}{1-p_{11}z})^k \tag{15}$$

The mean number of steps from $G_0$ to $G_k$ would be:

$$E_{G_0G_k}[n] = P'_{G_0G_k}(z)|_{z=1} = \frac{k}{1-p} \tag{16}$$

while the variance is

$$Var_{G_0G_k}[n] = \{P''_{G_0G_k}(z) + P'_{G_0G_k}(z) - (P'_{G_0G_k}(z))^2\}|_{z=1}$$

$$= \frac{kp(1+\rho)}{(1-p)(1-\rho)^2} \tag{17}$$

For the same reason, we can get

$$E_{B_0G_k}[n] = \frac{k}{1-p} + \frac{\rho}{(1-p)(1-\rho)} \tag{18}$$

$$Var_{B_0G_k}[n] = \frac{kp(1+\rho)}{(1-p)(1-\rho)^2} + \frac{\rho(1-p(1-\rho))}{(1-p)^2(1-\rho)^2} \tag{19}$$

thus

$$E[n] = \pi(0)E_{G_0G_k}[n] + \pi(1)E_{B_0G_k}[n]$$

$$= \frac{k}{1-p} + \frac{p\rho}{(1-p)(1-\rho)} \tag{20}$$

$$Var[n] = \pi(0)Var_{G_0G_k}[n] + \pi(1)Var_{B_0G_k}[n]$$

$$= \frac{kp(1+\rho)}{(1-p)(1-\rho)^2} + \frac{p\rho(1-p(1-\rho))}{(1-p)^2(1-\rho)^2} \quad (21)$$

Again, we can see that the correlation coefficient $\rho$ has little effect on the mean of $N$ but will affect the variance of $N$ significantly. Figure 16 plots the relationship between the variance of $N$ and $\rho$ when $k = 20$ and $p = 1\%$. As we can see, as $\rho$ approaches 0, the variance of $N$ approaches to the case when the packet loss follow the binomial distribution; as $\rho$ approaches to 1, the variance of $N$ approaches infinity. The increase of the variance is slow as $\rho$ increases from 0 to 0.7, but becomes dramatic as $\rho$ increases from 0.7 to 0.9. In our simulation (that we will show later), we will see that the efficiency of FEC decreases more dramatically at the higher end of $\rho$. Various techniques such as interleaving can be used to decrease the packet correlation.

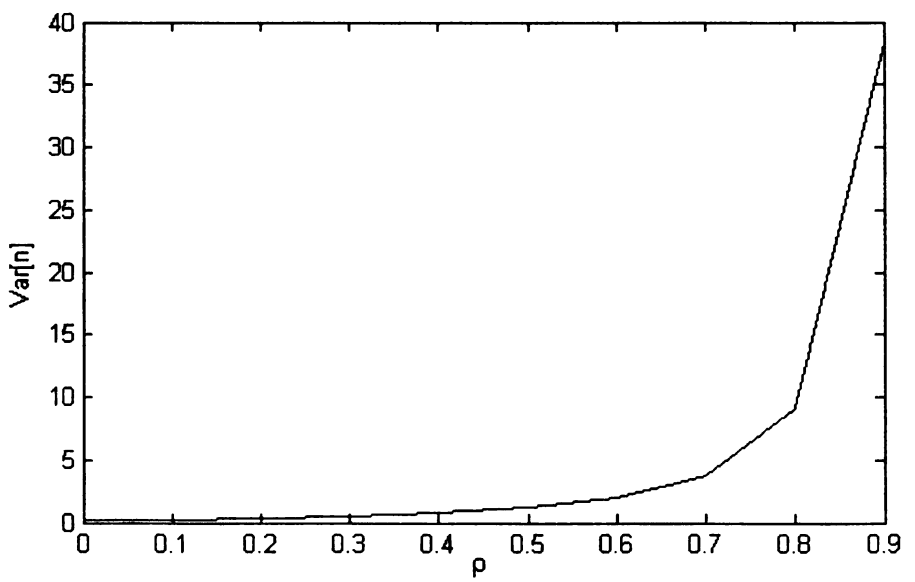



**Figure 16 Variance of N versus packet correlation ρ**

## 3.4 Analysis of Network Embedded FEC Based Routs

Previous studies analyzed the packet-loss model for FEC-enhanced multicast trees (e.g., [41][43]). These studies are based on the IP multicast model, in which intermediate nodes do not participate in FEC. These studies also assume the binomial distribution of packet losses and hence do not take into account the Markov nature of packet losses (as was done in our analysis presented above). Here, we study the packet-loss model of a multicast tree when FEC codecs[3] are placed in the intermediate nodes of a tree.

In our analysis, we use the following notations:

| | |
|---|---|
| $T$ | A multicast tree with a root node $r$ |
| $\|T\|$ | The size (in terms of the total number of nodes) of a multicast tree $T$ |
| $T^c$ | A sub-tree rooted at some node $c \in T$ but does not include the node $c$. |
| $T_l^c$ | The set of leaf nodes of $T^c$. |
| $\|T_l^c\|$ | The total number of leaf nodes of a the sub-tree $T^c$ |
| $p_v(i)$ | Probability that node $v \in T$ receives exactly $i$ packets. |
| $p_{v\|v-1}(i,j)$ | Probability that node $v$ receives $i$ packets given that its parent $v-1$ sends $j$ packets. |
| $p$ | The packet loss probability between the link from $v-1$ to $v$ |
| $\rho$ | The correlation between the states of consecutive packets |
| $(n,k)$ | The desired FEC block parameter pair used by the system. $n$ is the FEC block size, and $k$ is the number of message packets. |
| $RS(n,k)$ | Reed Solomon code with $k$ message packets and $n$-$k$ parity packets. |

[3] In the sequel, we present our analysis in the context of Reed-Solomon (RS) codes which have been among the most popular FEC schemes for packet loss recovery of realtime multicast data. Nevertheless, many aspects of our analysis and the key conclusions of this work are applicable to any other form of linear block codes that are based on successful FEC-block recovery when the number of received packets is equal to or high than the number of message packets in the block.

We use $p_{v|v-1}(i,j)$ to represent the probability that node $v$ receives $i$ packets given that its parent $v-1$ sends $j$ packets. For links with memory-less losses, this probability is simply a binomial distribution:

$$p_{v|v-1}(i,j) = \binom{j}{i}(1-p)^i \qquad (8)$$

On the other hand, if the packet losses follow the Gilbert model, then

$$P_{v|v-1}(i,j) = \phi(j,i) \qquad (9)$$

where from (7):

$$\phi(j,i) = \pi(0)(\phi_{G_0G_i}(j) + \phi_{G_0B_i}(j)) + \pi(1)(\phi_{B_0G_i}(j) + \phi_{B_0B_i}(j))$$

When computing the probability $P_v(i)$ that a node $v$ receives exactly $i$ packets, we need to consider two cases; first, we consider the case when the parent node $v-1$ has no codec; second, we consider the case when the parent node $v-1$ has a NEF codec. If node $v$'s parent does not have a codec, the probability that node $v$ receives $i$ packets is:

$$p_v(i) = \sum_{j=i}^{n} p_{v-1}(j)\, p_{v|v-1}(i,j) \qquad (10)$$

Note that $p_{v|v-1}(i,j) = 0 \quad \forall j < i$. In other words, node $v$ can receive $i$ packets only when its parent $v-1$ sends at least $i$ packets. For the root node ($r$) of the tree, we define

$$p_r(i) = \begin{cases} 0 & 0 \le i \le n-1 \\ 1 & i = n \end{cases} \qquad (11)$$

Equation (10) is a recursive function, and hence with the initial condition from (11), we can calculate the probability $p_v(i)$, for any node $v$ in the multicast tree, that it receives exactly $i$ packets. When a node has a codec for a $RS(n,k)$ block, and if that node receives

less than $k$ packets and cannot decode the FEC block, it will just forward the received packets as usual; if it receives $k$ or more packets, the node can decode the block and reconstruct the original data. It can also reproduce the lost parity packets. In fact, a codec can produce more or less than $n-k$ parity packets if desired; however, in our analysis, we assume that the NEF codecs reconstruct the original data and reproduce the lost parity packets using the same $RS(n,k)$ code. These packets are then multicasted downstream. (In our simulation for the distributed algorithm, we will simulate the scenario where a codec produces a number of parity packets that is required by its children).

A node that has a NEF codec and which receives $k \le j \le n$ packets will send $n$ packets. If $v$ is the immediate child of a codec, the probability that it receives $i$ packets becomes

$$P_v'(i) = \begin{cases} \sum_{j=k}^{n} P_{v-1}(j) P_{v|v-1}(i,n) & k \le i \le n \\ \sum_{j=k}^{n} P_{v-1}(j) P_{v|v-1}(i,n) + \sum_{j=i}^{k-1} P_{v-1}(j) P_{v|v-1}(i,j) & 0 \le i < k \end{cases} \quad (12)$$

Once a node $c$ is assigned a NEF codec, the probability $P_v(i)$ or all $v \in T^c$ will change and need to be recomputed. We use (12) to calculate $P_v(i)$ for the immediate children of the codec. For nodes that are not immediate children of a codec, the calculation of $P_v(i)$ is the same as equation (10).

Here we use $P_v^{dec}$ to represent the probability that node $v$ can decode a $RS(n,k)$ block:

$$P_v^{dec} = P_v(i \ge k) = \sum_{i=k}^{n} P_v(i) \quad (13)$$

We define the average decodable probability of a tree $T$ for *p2p* and proxy-based overlay networks, respectively, as:

$$P_{avg}^{dec} = \frac{\sum_{v \in T-r} P_v^{dec}}{|T|-1} \quad (14)$$

$$P_{avg-leaf}^{dec} = \frac{\sum_{v \in T_l^r} P_v^{dec}}{|T_l^r|} \quad (15)$$

If we use $r_d(v)$ to represent the number of received data packets (not including the parity packets received) of a FEC block at node $v$, then

$$E[r_d(v)] = \sum_{i=k}^{n} kP_v(i) + \frac{k}{n}\sum_{i=0}^{k-1} iP_v(i) \quad (16)$$

For a $RS(n,k)$ block, if a node receives $i \geq k$ packets, only $k$ packets are message packets; if a node receives $i < k$ packets, we assume on average only $(k/n)i$ are message packets. For a *p2p* and overly networks, we define the message goodput as:

$$g = \frac{\sum_{v \in T-r} E[r_d(v)]}{(|T|-1)k} \quad (17);$$

$$g_{leaf} = \frac{\sum_{v \in T_l^r} E[r_d(v)]}{|T_l^r|k} \quad (18)$$

## 3.5 A Cascaded Peer-to-Peer Channel

In a peer-to-peer overlay network, each node in the network is an end system. We define the path between each pair of these p2p nodes a p2p channel. A p2p channel may consist of multiple underlying routers and physical links, and can be modeled as a Gilbert channel. As we have analyzed before, a p2p channel thus can be fully characterized by its

average loss rate $p$ and its packet correlation $\rho$. In a p2p overlay multicast tree, from the sender to each receiver there is a forwarding path consisting of a series of intermediate p2p nodes and a cascaded p2p channels. Before we go forward to analysis the NEF performance on a general multicast tree, we first analysis a cascaded peer-to-peer channel.

Figure 17 shows a cascaded channel consists of ten p2p nodes and nine p2p channels that connect these nodes. The p2p sender channel-codes a set of $k$ message packets into a block of $n$ FEC packets using a systematic Reed-Solomon (RS) erasure recovery code.

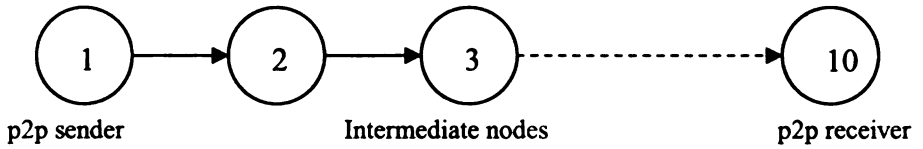
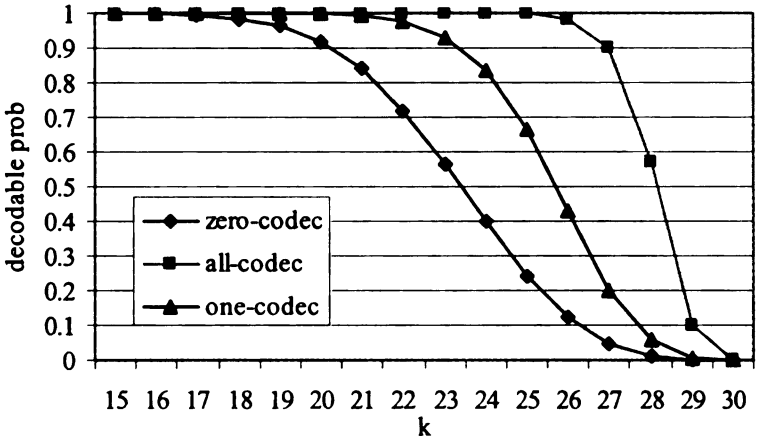



**Figure 17 a cascaded p2p channel**

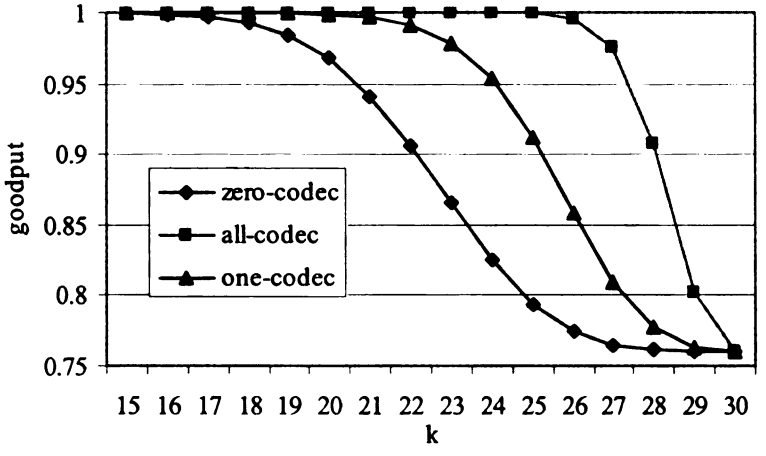
The p2p receiver performs erasure–channel decoding to recover any lost message packets. For a $RS(n,k)$ code, the receiver could recover all $k$ message packets if it receives any $k$ or more packets out of the original $n$ packets transmitted by the sender. Here, the block size $n$ is kept constant, and the number of message packets in a block $k$ is changed, so that the code rate $k/n$ will change according to $k$. The average loss rate of the p2p channels is set to 3%; the packet correlation of the p2p channels will change so that we can see how loss burst length will affect the performance of the NEF. We will use the analysis presented in the last section to calculate the decodable probability and the message goodput at the p2p receiver. While every intermediate node can act as codec, we present only two cases: "all intermediate nodes act as codec" and "a single intermediate node act as codec". Apparently, these two cases represent a measure of "upper" and

"lower" performance bounds in respective to decodable probability and message goodput.

Figure 18 shows the decodable probability and goodput of the p2p receiver when packet correlation is set to 0. Here, $n$ is set to be 30 and $k$ is changed from 15 to 30. We show the results when no intermediate nodes act as codec, one intermediate node act as codec




(a)




(b)

**Figure 18 decodable probability and message goodput of a cascaded p2p channel when ρ=0**

and all intermediate act as codecs. When the code rate is low, all three cases have the decodable probability of close to 1; when the code rate is very high, all the decodable probabilities approaches to 0. Both very high and very low code rate are rarely used in practice; very low code rate causes very high overhead; when the code rate goes so high that the receiver can not decode almost any FEC block, its better not to use FEC at all. For a wide range of code rate between these two limits, we can see that both "all intermediate nodes act as codecs" and "a single intermediate node act as codec" out perform the case when there no codecs at all. For an example, when $k = 24$, if no intermediate node act as codec, the p2p receiver has a decodable probability of 40%; if one intermediate node act as codec, the decodable probability increases to 82%; if all intermediate nodes act as codec, the decodable probability is 100%. The same pattern can be observed on the message goodput of the p2p receiver.

Adding codecs in the intermediate nodes of a network may incur computation complexity and delay penalty. The time needed for encoding and decoding depend on the particular code in use and computation power of the end system. Rizzo in [44] showed the time needed to encode and decode different $(n,k)$ linear erasure codes using software implementation on a Pentium 133 system running FreeBSD. For a (30,16) code, the average encoding time is 1.5ms per packet; the average decoding time is about 1.7ms per packet. The encoding and decoding time will increase as $n$ or $k$ increases. For example, the average times needed to encode and decode a (60,32) code on the same system are 3.0ms and 3.5ms per packet respectively. Often, dedicated hardware is used to speedup the encoding and decoding operations, but this will add extra costs on end systems. Though all intermediate nodes act as codec may cause too much delay penalty, if only

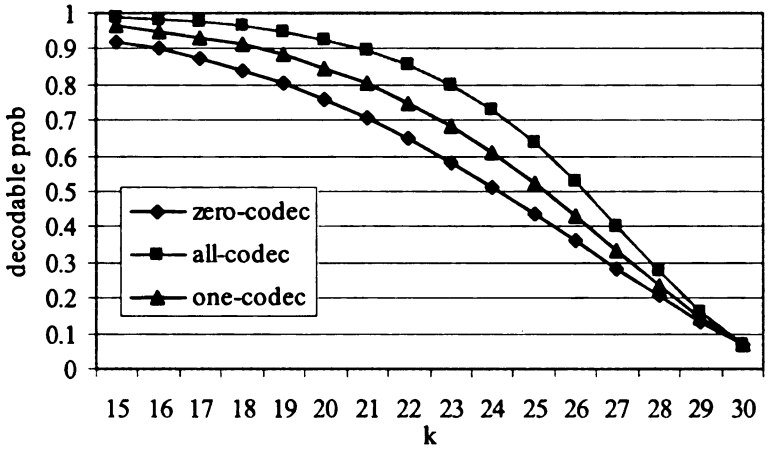
one or two codecs added to the intermediate nodes, the performance improvement may out-weight the delay penalty incurred; this is especially true for multicast applications, where one intermediate codec may serve hundreds or thousands of end users downstream.

Figure 19 shows the decodable probability and message goodput when packet correlation equals to 0.7. We can see that even when one codec (optimally chosen as will be discussed further below), the decodable probability and goodput is clealry better than the case when there are no codecs.

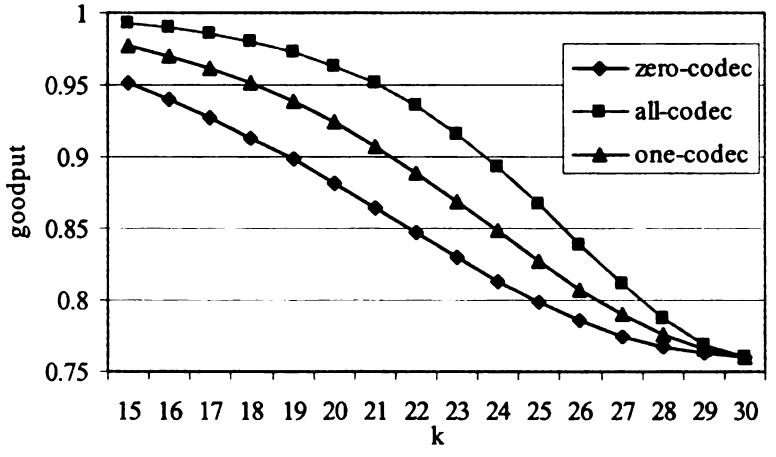
It is important to note also that the improvements due to a NEF codec may decrease at higher values of correlation when compared to the case when the packet correlation equals zero. This is because as the loss-burst length increases, the probability that an intermediate node can decode a FEC block decreases. If an intermediate codec can not decode a FEC block, it can not reproduce the lost message or parity packets, and will just act as a normal node. For an example, When the average loss rate $p = 3\%$ and packet correlation $\rho = 0.9$, the average loss burst length is $1/(1-p)(1-\rho) = 14.3$; even for very low code rate codes such as $RS(30,16)$, a FEC block suffer from such a long burst of loss can not decode the FEC block, so an intermediate codec cannot recover lost packets.

One way to overcome such long loss bursts is to use interleaving. As Yee has shown in [78], an interleave degree of $I$ will decrease the effective packet correlation to $\rho^I$. Another way is to use longer block codes. Figure 20 shows the decodable probability and message goodput when block size $n$ is set to 255 and the number of message per block is changed from 210 to 230. Here the average packet loss is 3% and the packet correlation is 0.9. We see that with very strong packet correlation and average loss burst of 14.3

packets, even with only one intermediate codec, the performance is much better than end-to-end FEC.

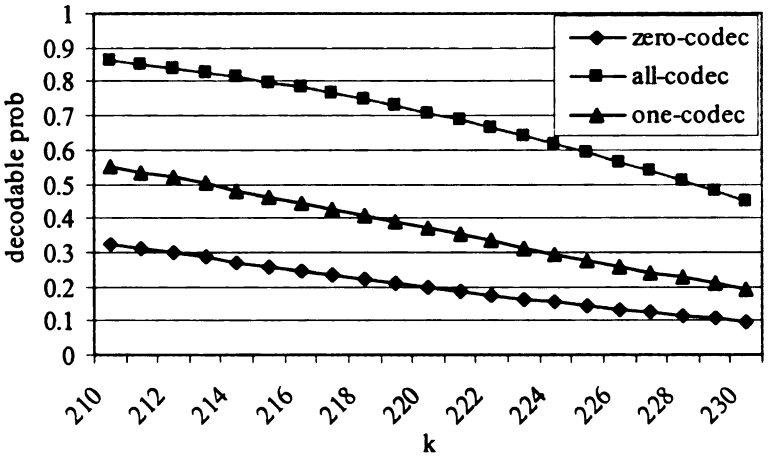



(a)

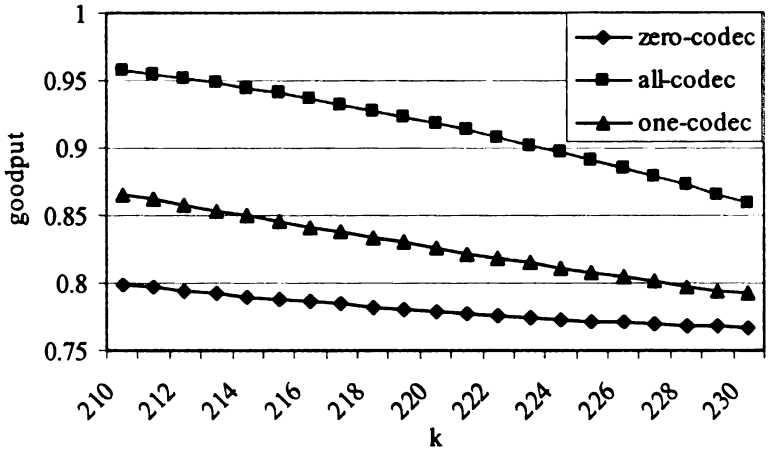



(b)

**Figure 19 decodable probability and message goodput of a cascaded p2p channel when ρ=0.7**




(a)




(b)

**Figure 20 decodable probability and message goodput of a cascaded p2p channel using long block size, $p = 3\%$ , $\rho = 0.9$.**

# Chapter 4 Centralized Codec Placement Algorithm

The analysis presented in the previous chapter did not make any assumption regarding the placement of the NEF codecs within a multicast network. In this chapter, we develop a mechanism for placing NEF codecs within a given network topology. The algorithm is then implemented on random multicast trees to evaluate the impact of intermediate codecs on multicast applications [90][91].

## 4.1 Previous work

In a large topology, identifying the optimum locations for the NEF codecs is not an easy task. One objective is to place codecs in the intermediate nodes of a topology to maximize the average decodable probability. Assuming that the loss rate for each link in the topology and the number of codecs to be placed are known beforehand, the problem is similar to (but different from) the well-known *P*-median problem[63][71]. A *P*-median problem is to find *P* locations in the network to place facilities in order to minimize the overall cost for servicing all of the nodes. Generally, in a *P*-median problem, the cost to serve a node is determined by the weight at the node and the distance between the node and its nearest available facility. It has nothing to do with other facilities placed in the network. As we have seen in the previous section, in order to calculate the decodable probability, we need to know the loss rate of the links on the path between the node to the source (root node); we also need to know the locations of the codecs that have been placed on that path, not just the immediate codec that serves the node.

There are several studies that address the *P*-median problem on a tree. In[66], Goldman found a $O(n)$ algorithm for the 1-median problem. Kariv and Hakimi gave a

$O(n^2p^2)$ solution for a $P$-median problem on a tree with $n$ nodes [67]. More recently, Li et al find a $O(n^3k^2)$ time algorithm to place $k$ web caches on an $n$-node directed tree[71]. In[72], Tamir provides a $O(pn^2)$ algorithm for the $P$-median problem on a tree.

## 4.2 Centralized Codec Placement Algorithm

Because the decodable probability at a node in a NEF network is impacted by all the codecs placed along the path from that node to the source (root), the dynamic programming approaches that have been used in previous network-placement problems (e.g., [71]) cannot be simply adapted to solve the NEF codec placement problem. In the following, we use a greedy algorithm to place $m$ codecs in the multicast tree.

The greedy algorithm finds the best location for the first codec, then the next best location for the second one, and so on. Once a node is selected, an FEC codec is added to regenerate any lost data or parity packets. Let $T^c \subset T$ be the sub tree rooted at node $c \in T$ not including $c$. If $c$ is set as a "codec node", only those nodes $v \in T^c$ will benefit from this selection; meanwhile, the "codec node" $c$ itself will not be affected. For nodes $v' \in T - T^c$, everything remains unchanged. Let $P_v^{dec}$ and $P_v^{'dec}$ denote the average decodable probability for node $v \in T^c$ before and after node $c$ is set as a codec node, respectively. We need to find $c \in T$ that maximizes the following:

$$\max_{c \in T} \left( \sum_{v \in T^c} (P_v^{'dec} - P_v^{dec}) \right) \qquad (19)$$

A similar optimization objective function can be expressed for proxy-based overlay networks, except here the summation takes place over the leaf nodes only. In this case, we need to find $c \in T$ that maximizes

$$\max_{c \in T}\left(\sum_{v \in T_l^c}\left(P_v^{'dec} - P_v^{dec}\right)\right)$$

Under the proposed greedy algorithm, we use an exhaustive search to find the best place for the first codec, after we find the optimum $c \in T$ node, we place the codec at that node. We use the same method to place the next codec; this process continues until all of the $m$ codecs are placed.

## 4.3 Analysis Results

### 4.3.1 Analysis Results for 100 Nodes Network

We applied the performance analysis presented above to a general tree structure. We use the popular Georgia Tech gt-itm [81] network topology generator to produce a set of ten 100-node transit-stub graphs. For each graph, we use Dijkstra's Shortest Path First (SPF) algorithm to produce a tree rooted at a randomly selected node. We used the greedy algorithm described in the previous subsection to place the NEF codecs in the multicast tree. The number of codecs was increased from 0 to 10. After each codec is placed, we calculate the improvement on average decodable probability and goodput.

As mentioned above, in a *p2p* overlay multicast network, nodes in the multicast tree are also end users, which often are placed at the edge of the Internet. Each hop in the overlay network often consists of several underlying physical hops. This implies that the loss rate of each hop could be higher than the loss rate of a backbone link in an IP multicast model. Here, we show results when the loss rate per-link is set to 3%, 4%, and 5%. These loss rates are in accordance with previous studies [37]. We studied the performance improvement under each of these loss rates for a variety of RS codes. Here, we present

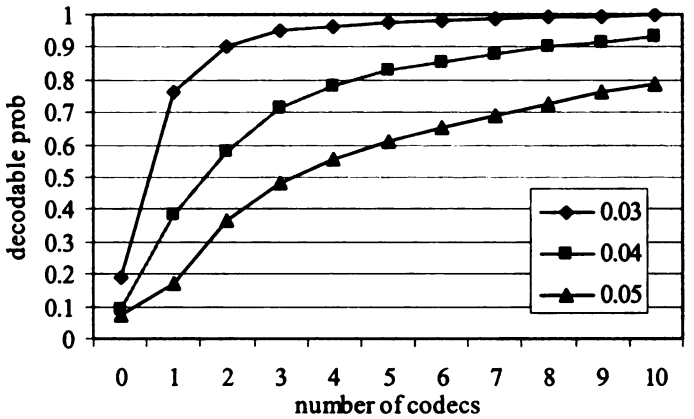
the results for $RS(255,223)$ , which is a popular FEC code that has hardware and software implementations. (The channel coding rate[4] for $RS(255,223)$ is 87.5%.)

The average FEC block decodable probability and data goodput for each tree were evaluated. The results are the averages over all of the ten random trees that were analyzed.

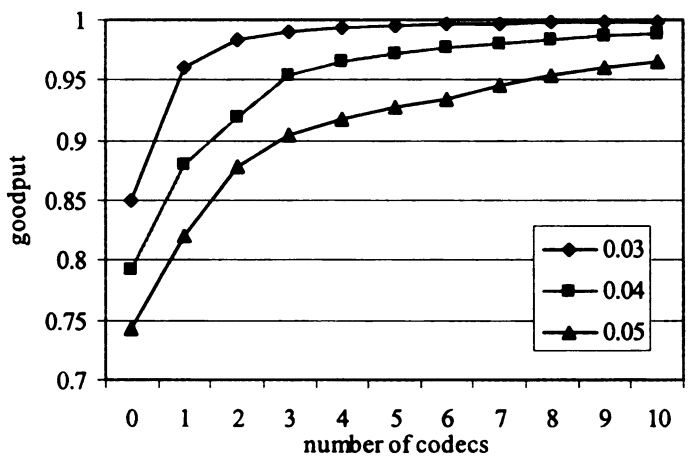
Figure 21 shows the decodable probability and goodput when error packet correlation are set to 0 and 0.8 respectively. In both cases we can see as the number of codecs increases, the decodable probability and goodput increase dramatically. This is specially true when the first few codecs are placed. For example, in Figure 21(a), for a 3% per-link loss rate, the first codec increase the decodable probability from 18.6% to 76%; the first 3 codecs increase the decodable probability from 18.6% to above 95%. In Figure 21(b) the first 3 codecs increase the goodput from 85% to 99% For a typical realtime (e.g., video) application, reducing the effective packet losses from 15% (85% goodput) to less than 1% (higher than 99% goodput) will naturally have dramatic improvements in the decoded video quality, both in terms of PSNR and visual perception. Under high losses, traditional end-to-end FEC could resort to a significantly lower FEC coding rate (to lower the packet losses and achieve high reliability). However, this reduces the effective source rate significantly. In this case, NEF could be used to maintain the high reliability performance

[4] This code rate may be high for some of the loss rates that are evaluated in this paper. However, it is important to note that the main conclusions of our study are valid regardless of the specific RS codes used. In particular, the proposed NEF framework can be used in one of two ways. Under one approach, a *given* RS code is already being used (on an end-to-end basis) prior to adding any NEF codecs. In this case, NEF can significantly improve the overall goodput as shown extensively by our analysis and simulations in this paper. Under another approach, a reliable communication infrastructure is already in place. This reliable infrastructure would be normally based on using very conservative (low) FEC rates (i.e., much lower than the effective end-to-end channel capacity). In this case, NEF can be used to significantly improve the efficiency of the RS codes by increasing its rate while maintaining the same level of reliability provided by the original infrastructure. In this paper, we focused on the first scenario to illustrate the benefits of the proposed NEF-based framework.

while increasing the FEC rate significantly (i.e., increasing the effective source bitrate). Either way, NEF provides salient and dramatic improvements in the delivery of realtime over multicast networks.




(a)

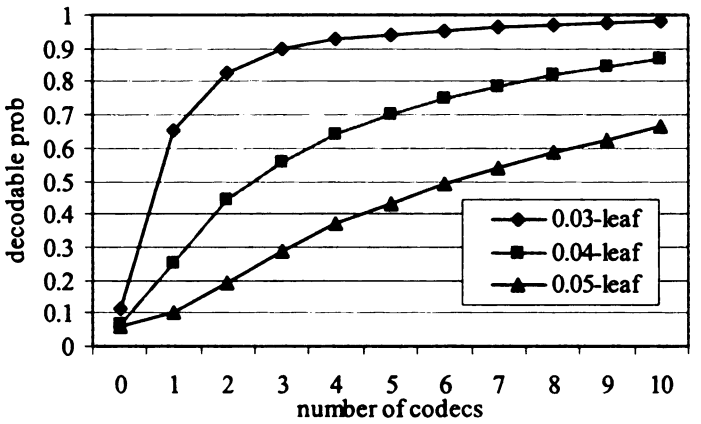



(b)

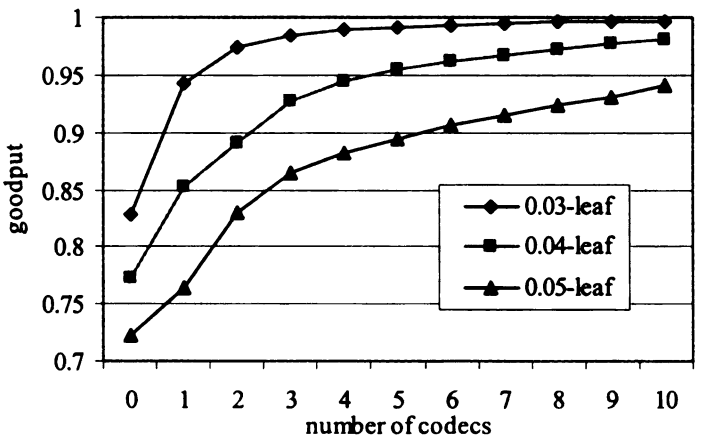
**Figure 21 decodable probability and goodput of p2p overlay multicast tree, $\rho = 0$.**

In proxy-based overlay networks where only leaf nodes are end users, the performance of intermediate codecs are much like those of the peer-to-peer overlay networks.

Figure 22 shows the decodable probability and goodput of leaf nodes when the packet correlation is set to 0.

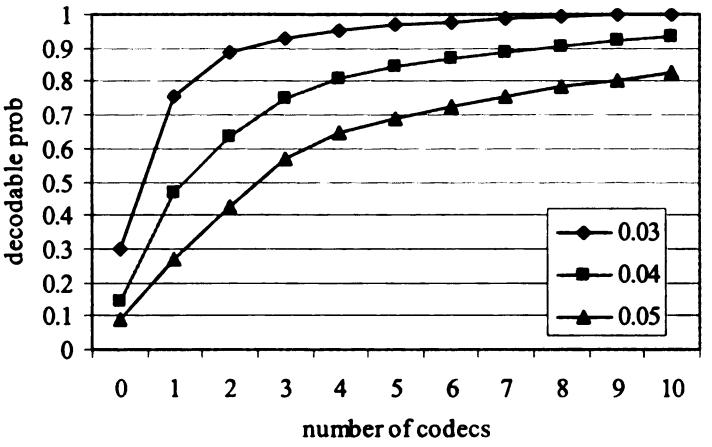



(a)

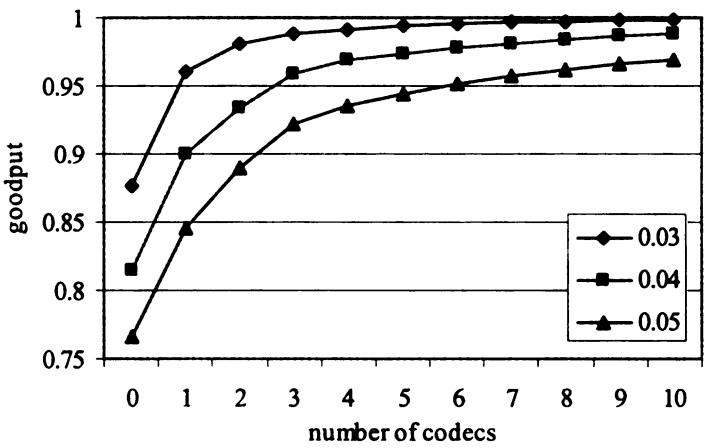



(b)

**Figure 22 decodable probability and goodput of proxy-based overlay multicast tree, $\rho = 0$.**




(a)




(b)

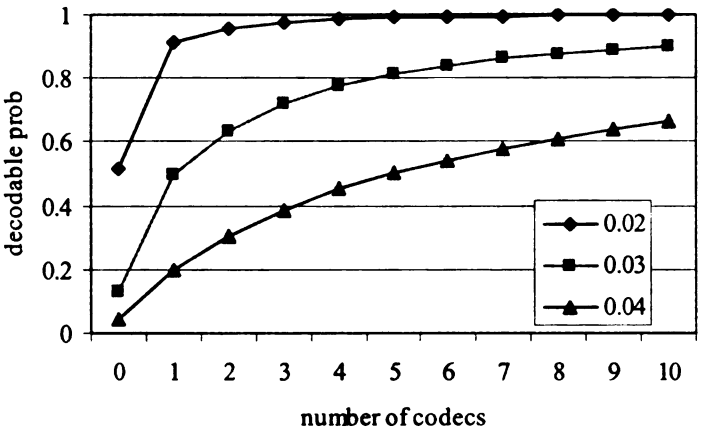
**Figure 23 decodable probability and goodput of p2p overlay multicast tree, $\rho = 0$, $p$ is normally distributed with mean of 3%, 4% and 5%, variance of 1.5%, 2%, 2.5% respectively.**

In the above analysis, we have assumed the average loss rates on each link of the multicast tree are the same. In reality, this is not the case. Different link may have different bandwidth and different background traffic, this will cause different network conditions and hence different loss rate. We do the analysis on the same set of trees we have described above. The positions of the codecs are not the same as that when all the links have the same loss rate, but the effectiveness of the codecs are not affected. Figure 23 shows the results when the average loss rate among the links of the multicast tree are

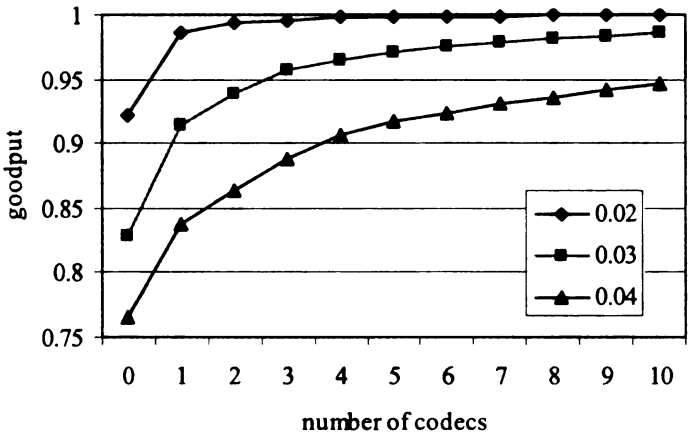
normally distributed, with mean of 3%, 4% and 5% and variance of 1.5%, 2% and 2.5% respectively.

### 4.3.2 Analysis Results for 600 Nodes Network

The same analysis is performed on a set of 600-hundred-node networks. Here we assume the binomial loss model, the average loss rates are set to be 2%, 3% and 4% respectively. We use the same transit-stub topology model as we have used in the 100-node network. Figure 24 shows the average decodable probability and goodput of the p2p overlay multicast trees when each link has the same loss rate. Figure 25 shows the results for the proxy-based multicast trees. Figure 26 shows the result when the loss rates are normally distributed among the links, with mean of 2%, 3% and 4%, variance of 1%, 1.5%, and 2% respectively.

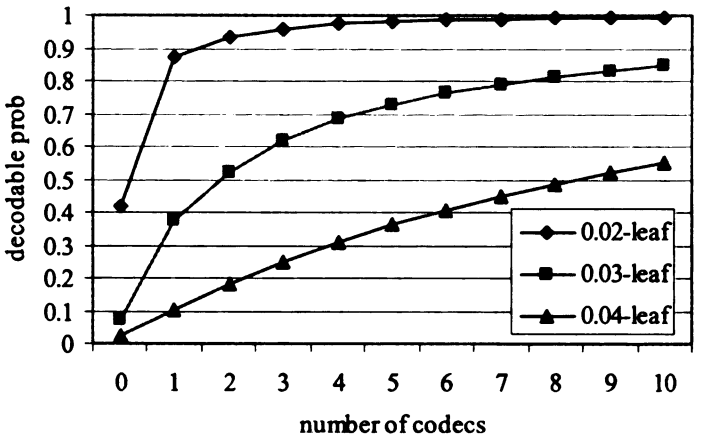



(a)

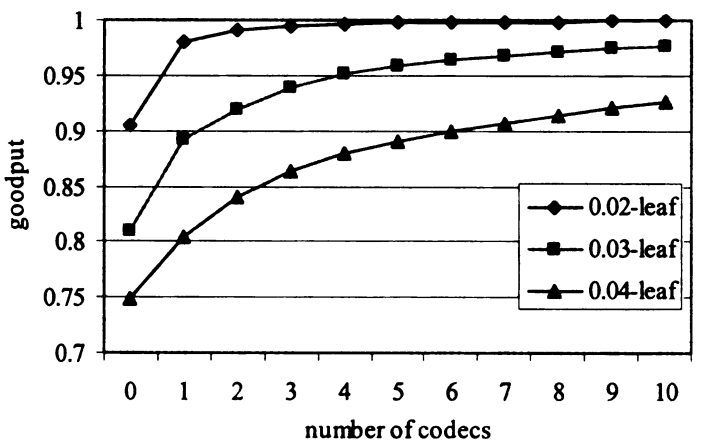



(b)

**Figure 24 decodable probability and goodput for p2p overlay multicast tree, 600-node. ρ = 0, p=2%, 3%, 4% respectively**

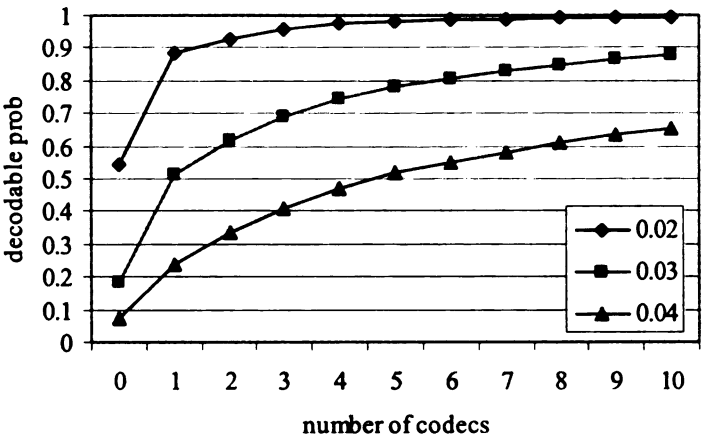



(a)

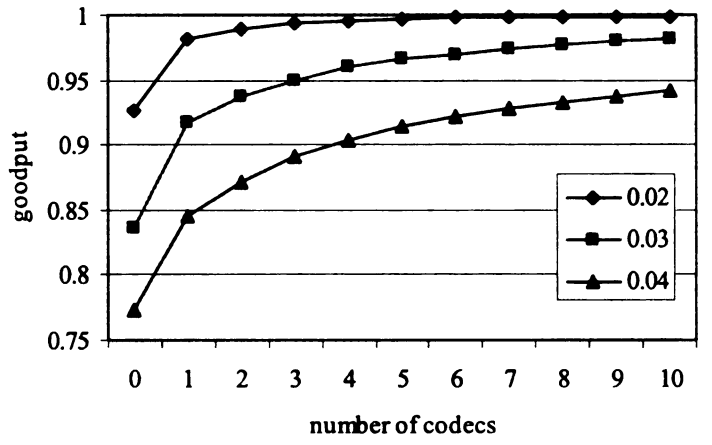



(b)

**Figure 25 decodable probability and goodput for proxy-based overlay multicast tree, 600-node. $\rho = 0$, $p$=2%, 3%, 4% respectively.**




(a)




(b)

**Figure 26 decodable probability and goodput of p2p overlay multicast tree,600-node, $\rho = 0$, $p$ is normally distributed with mean of 2%, 3% and 4%, variance of 1.0%, 1.5%, 2.0% respectively.**

### 4.3.3 Simulation Verification

We implemented the NEF framework using *network simulator2* (ns2) [82]. Here, FEC is used independently without ARQ, we will implement another version of NEF integrate FEC and ARQ to provide reliable multicast in our future work. We added a FEC packet header and traffic generator to the simulator to produce $RS(n,k)$ form of traffic. The FEC header has a FEC *block_ID* label to differentiate different blocks. It also has a flag

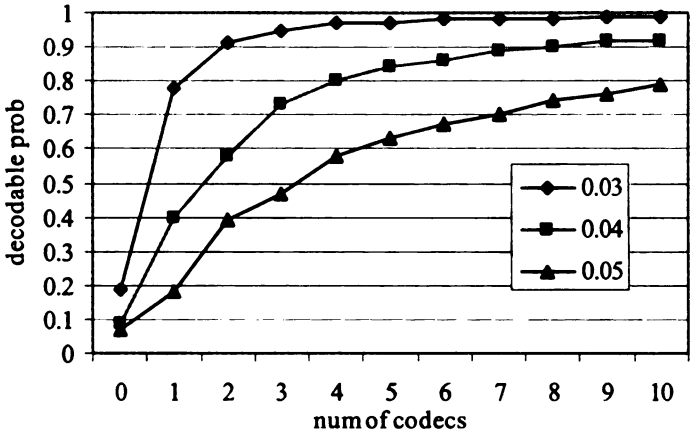
to indicate whether a packet is data, parity packet. The parameters $n$ and $k$ can be set by the application. The traffic generator transmits packets in FEC blocks, the first $k$ packets within a block are data packets and the remainder $n-k$ packets are parity packets. The transmission rate can be set by the application.

We have implemented primitives to build overlay networks upon the underlying topologies in ns2. As the algorithm to build the overlay network is not the main concern here, we use the Dijkstra's Shortest Path First (SPF) algorithm to build the overlay upon the underlying topology. The overlay we built forms a SPF multicast tree as we have used in our analysis. We have implemented a FEC UDP agent class and a FEC application class in the simulator. Each node in the graph will have a FEC UDP agent and a FEC application attached to it. The FEC application can detect how many data and parity packets that are lost in each FEC block. If the node is not a codec, the application merely replicates the packets and forwards them down streams. If the node is set as a codec, it will take action according to how many packets it received from the FEC block; if it receives less than $k$ packets; it just replicates and forwards; if it receives $k$ or more packets, it reproduces the lost packets and send these packets out on all the outgoing interface of the node for that multicast group. Notice that the reproduced packets are not multicasted to the entire group, but only to the sub-tree rooted at that NEF codec node. (We do not implement the actual Reed-Solomon encode and decode processes in the application; so the delay penalty incurred by the codec could not be measured here precisely.)

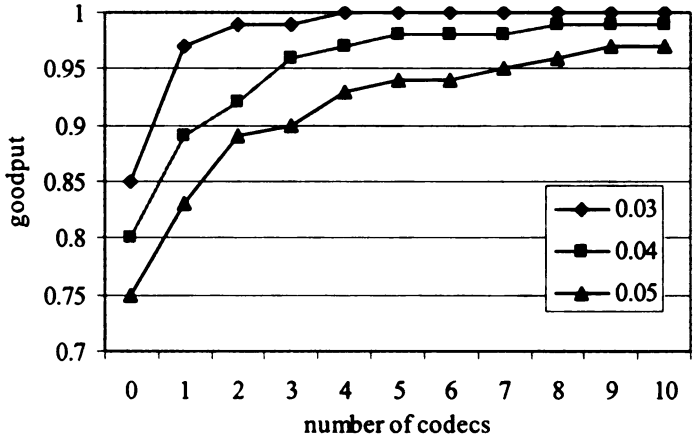
The network graphs are the 100-node graphs we used in analysis. The FEC code we use is $RS(255, 223)$, and the packet loss model we used is the BER model, the same as we

have used in our analysis. For each simulation, 30 blocks were sent. We first simulate the scenario when no intermediate nodes were set as codecs; then the number of codecs is increased from 1 to 10. The locations of the codecs were determined from our analysis. We run the simulations on each one of the ten simulation trees. The results are the average of all the simulations

Figure 27 (a) and (b) show the decodable probability and goodput average over all the nodes respectively. By comparing the simulations with the analysis results in Figure 21 (a) and (b), we can see that the differences between the simulation and analysis is within 1% for the decodable probability and 2% for goodput.. Figure 28(a) and (b) show the decodable probability and goodput average leaf nodes respectively. Again we can see the simulation results and the analysis results are very close. In comparison with Figure 22(a) and (b), we can see the biggest difference for the decodable probability over the leaf nodes is within 2%, and the biggest goodput difference is within 3%. Overall, the ns2 simulation results verify the analysis results: employing NEF codecs in the intermediate nodes improve the decodable probability and goodput significantly.

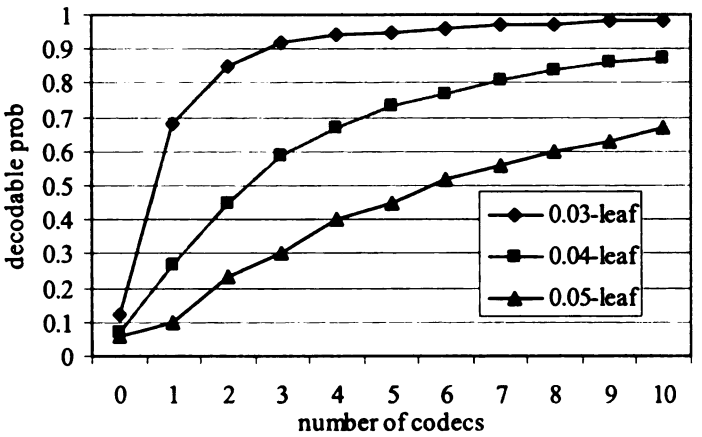



(a)

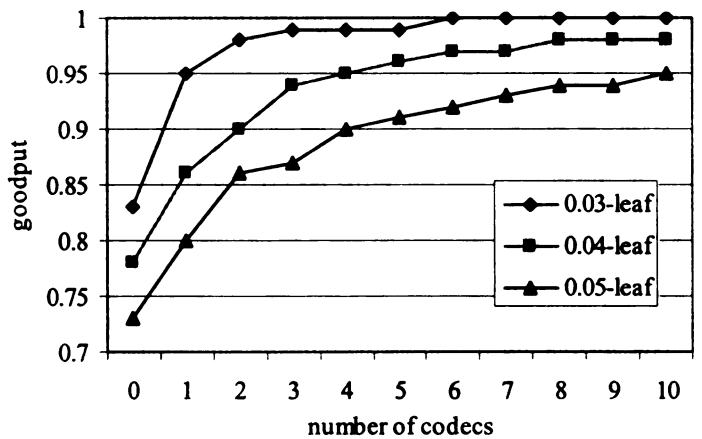



(b)

**Figure 27 simulation results: decodable probability and goodput for p2p overlay network, 100-node. $\rho = 0$, $p$ is set to 3%, 4% and 5% respectively.**




(a)




(b)

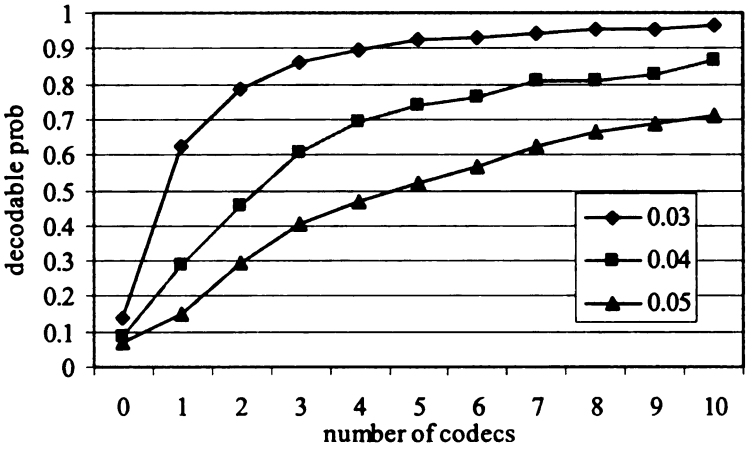
**Figure 28 simulation results: decodable probability and goodput for proxy-based overlay network, 100-node. $\rho = 0$, $p$ is set to 3%, 4% and 5% respectively.**

### 4.3.4 Analysis Results for Burst Loss

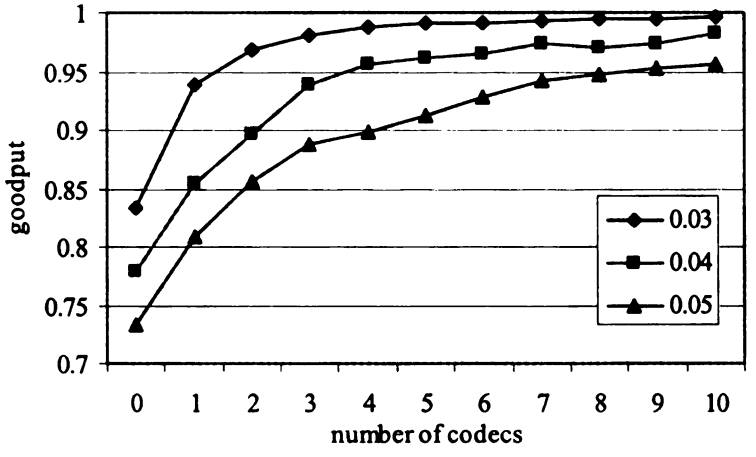
The above analysis has assumed that the packet loss follows the BER model, that is, there is no correlation between the lost packets. In reality, packet loss occurs in bursts. Previous work have showed that the loss burst length would impact the end-to-end FEC [41][76]. In this section, we analysis the impact of loss burst length on NEF [93]. The setup is the same as in the previous section, only that we will assume the Gilbert packet loss model here. For $(n,k)$ FEC codes, calculate one point of goodput need to calculate $n$ point of probability; calculate one point of decodable probability need to calculate $n-k$ point of probability. In section 3.3, we have seen that the calculation probability for Gilbert model is very complex and requires very high computation power, here we use the greedy algorithm to find the place of codecs that maximum the decodable probability but we will use the simulation to get the results for goodput improvement. This time in our simulation, a Gilbert loss model is inserted into each link of the multicast tree. For the decodable probability, the analysis results and simulation results are very close; the results we presented here are the simulation results, also we only present the results for 100-node network.

Figure 29, Figure 30 and Figure 31 shows the decodable probability and goodput the p2p multicast tree when packet correlation is set to 0.1, 0.5 and 0.9 respectively. Comparing the results, we can see that as packet correlation increases, the improvement in decodable probability and goodput becomes less dramatic. But even when the error packet correlation is very high, the improvement is impressive. In Figure 31, when the packet correlation is 0.9, for loss rate of 3%, we see that the first 3 codecs increase the decodable

probability from 35% to above 61%, and the goodput is increased from 85% to over 91%.

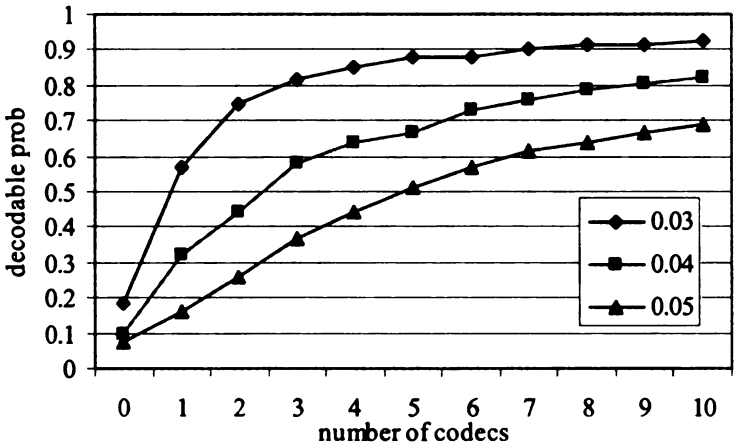



(a)

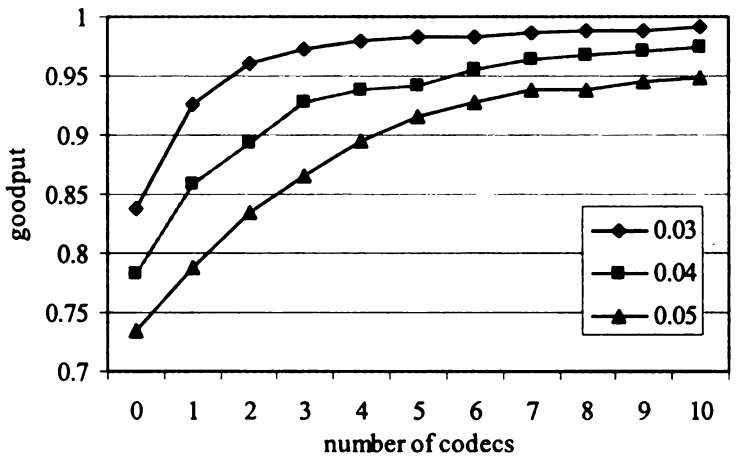



(b)

**Figure 29 decodable probability and goodput for p2p overlay multicast tree, 100-node. $\rho = 0.1$, $p$=2%, 3%, 4% respectively.**

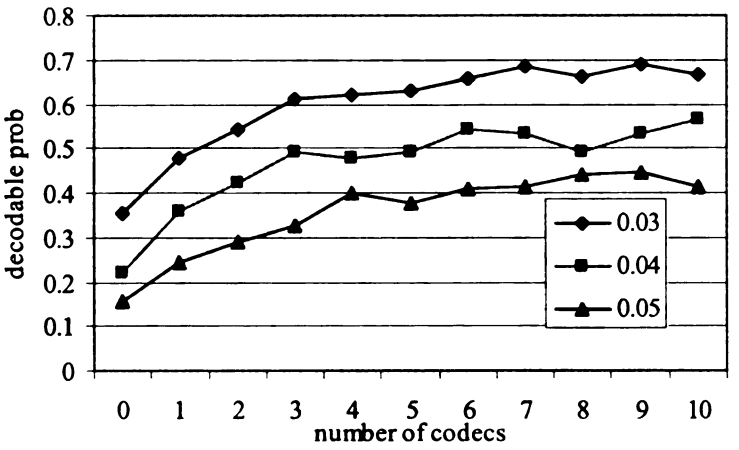



(a)

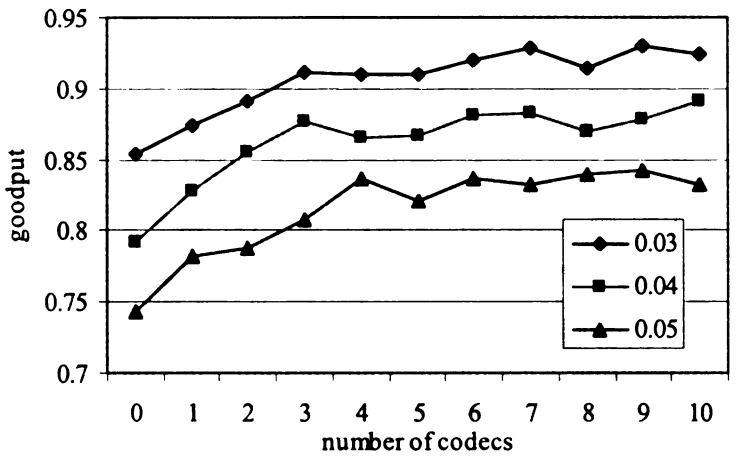



(b)

**Figure 30 decodable probability and goodput for p2p overlay multicast tree, 100-node. $\rho = 0.5$, $p$=3%, 4%, 5% respectively.**




(a)

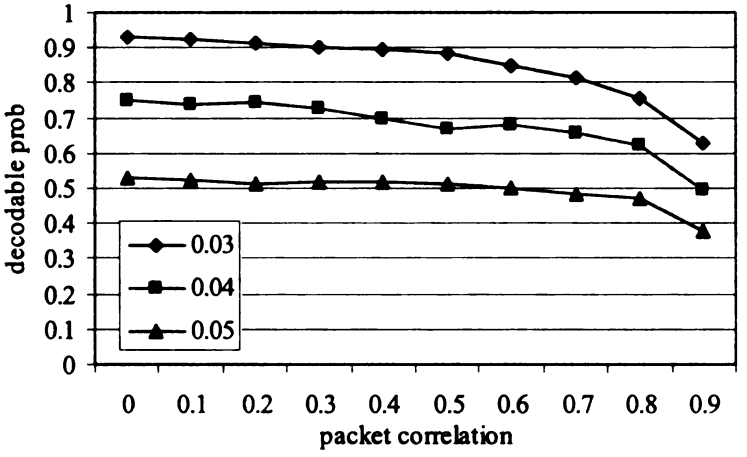



(b)

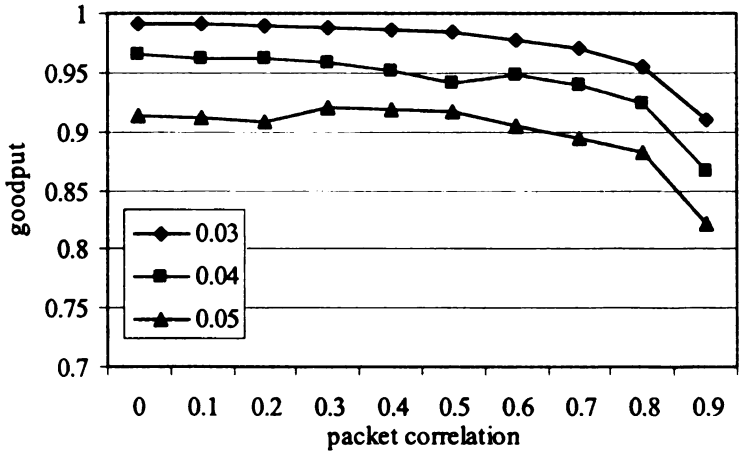
**Figure 31 decodable probability and goodput for p2p overlay multicast tree, 100-node. $\rho$ = 0.9, $p$=2%, 3%, 4% respectively.**

In order to get a better understanding of the impact of loss burst length on the effectiveness of NEF, we analyzed the decodable probability and goodput when the number of codec is set to 5 and the packet correlation is increased from 0 to 0.9. The result is plotted in Figure 32. Here, it can be observed that the decodable probability and goodput decrease as $\rho$ increases from 0 to 0.9. This implies that the erasure correlation

coefficient has a direct impact on the FEC codec performance. As the erasure correlation increases, the FEC codec performance is reduced due to an increase in the number of "long bursts" of lost packets in an FEC block. Nevertheless, the overall performance of the 5-NEF codecs stays very close to the optimum performance (at $\rho = 0$), and does not drop-off significantly only at high values of $\rho$ when it approaches 1. Even for $\rho = 0.9$, the overall goodput when $p = 3\%$ is around 90%.




(a)




(b)

**Figure 32 decodable probability and goodput when the number of codec is set to 5 and packet correlation is changed from 0 to 0.9**

For proxy-based overlay network, where only leaf nodes are considered as end users, we can see the similar pattern of decodable probability and message goodput improvement. The results are not presented here.

From the above analysis, we observe that for both binomial and Gilbert loss model, a relative few codecs can greatly increase the decodable probability and goodput. The reason for this is that in a transit-stub structure, a node in a stub domain often reaches to the majority of nodes in other stub domains through a few nodes in transit domain. A relative small number of codecs placed at some strategic points can affect most of the nodes in the whole network.

Another observation we can make is that the lower the per-link loss, the more effective the codec works, and the smaller number of codecs is needed to increase the decodable probability and goodput into an acceptable value. It can be seen that the slope of the increase is much bigger for the first few codecs when per-link loss is 3% than when it is 5%. Also the first few codecs contribute more improvements than the left, this is because the first few codecs are often placed at a relatively higher hierarchy of the multicast tree and serve more downstream children.

### 4.3.5 Overhead Analysis

In NEF, we have seen that intermediate codecs can improve the decodable probability and message goodput. In end-to-end FEC, this can be achieved by using lower rate FEC codes. In both cases, higher ratios of additional/overhead packets are sent into the network and these packets increase the overall overhead bandwidth. Here we measure the bandwidth overhead in terms of the cost for each message packet received [48]. The cost

of a message packet is the product of the average number of packets transmitted (including message packets and parity packets sent by the source and intermediate codecs) per message packet and the average number of links (hops) these packets traversed. For both NEF and end-to-end FEC, we do not consider control packets when we measure the cost of a message packet (for example, messages exchanged to build the overlay multicast tree).

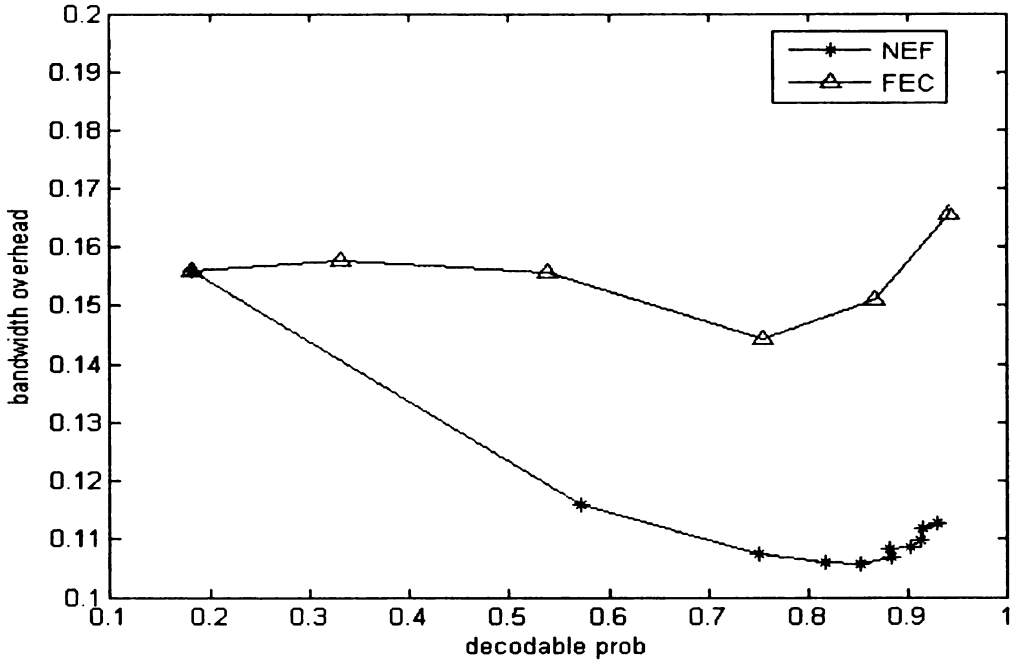
When there is no loss in the network, each client will receive exactly what the source has sent. There is no need for the use of FEC or retransmission. There is a bandwidth cost related to this ideal condition. We call this *ideal bandwidth cost*, and use *cost_ideal* to represents this. If there is loss in the network, we will use end-to-end FEC or NEF to add protection. The bandwidth cost of this two error protection schemes are *cost_FEC* and and *cost_NEF* respectively. We use the difference of the bandwidth costs correspond with these two schemes between the *ideal bandwidth cost* to represent the bandwidth overhead.

We use *ns*2 simulations to measure the average cost of a message packets. The packet correlation is set to 0.5. For NEF, we use *RS*(255,223) FEC code, the number of codecs is changed from 0 to 10. We measure the decodable probability and the average cost of a message packet each time a new codec is added to the multicast session. For end-to-end FEC, we start with *RS*(255,223) code, and we increase the FEC block size $n$ by 10 at each step of our simulation. For each different block size $n$, we measure the decodable probability and the average cost of a message packet.

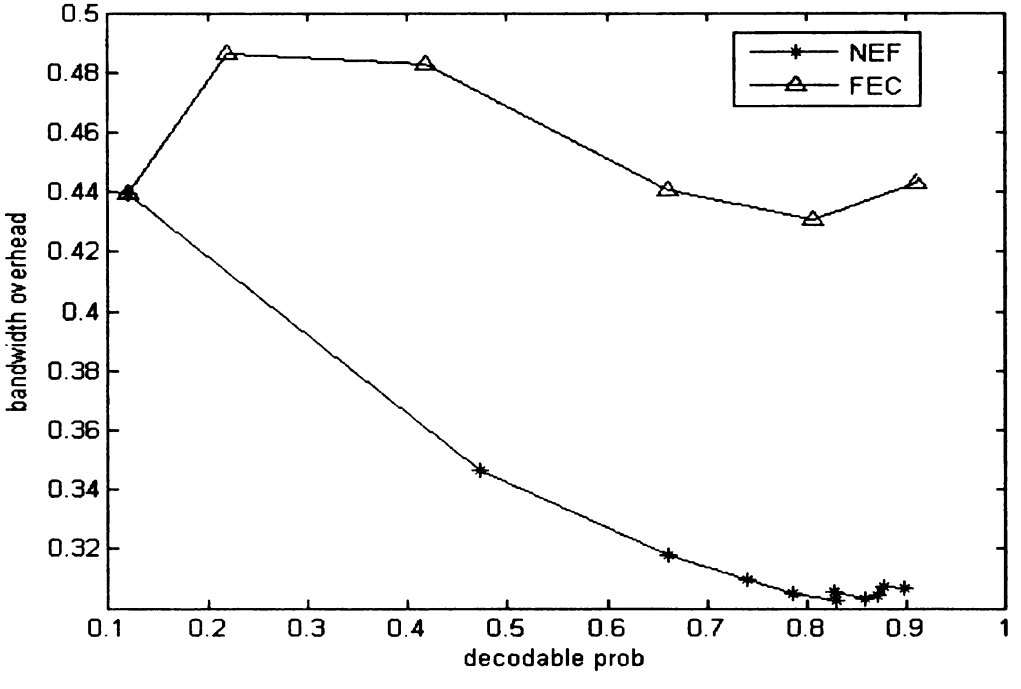
Figure 33(a) plots the average bandwidth overhead versus the decodable probability for NEF and end-to-end FEC for the set of p2p networks when the average packet loss rate

per link is set to 3% and the packet correlation is set to 0.5. For NEF, we can see that as the decodable probability increases (through adding codecs in the intermediate nodes), the bandwidth overhead decreases. For end-to-end FEC, the packet cost remains almost unchanged but increases when the decodable probability is above 0.9. Figure 33(b) shows the results for proxy-based overlay network. For both cases, we can see that the bandwidth overhead for end-to-end FEC is much higher than that of NEF when the decodable probability is the same.

Figure 34 and Figure 35 show the results when the packet losses per link are set to 4% and 5% respectively. For end-to-end FEC, we can see that as decodable probability increases (through decrease code rate), the bandwidth overhead increases and then decreases. For NEF, the packet cost decreases consistently but the slope of the decrease becomes smaller as the loss rate per link increases. Again, we can see the average bandwidth overhead for end-to-end FEC is much higher than that of NEF, and the difference between them becomes bigger as the loss rate per link increases.

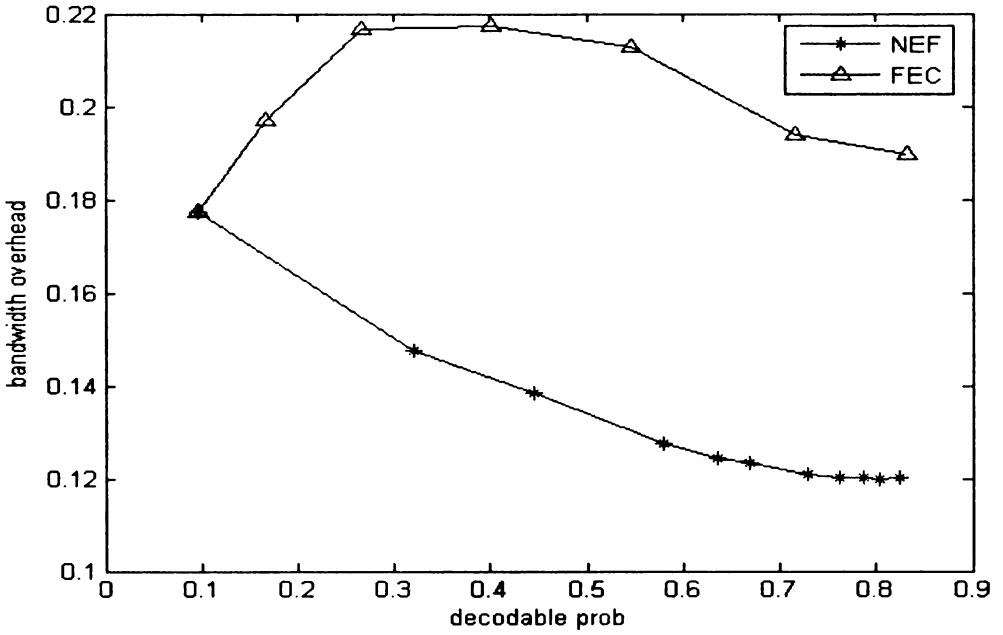



(a)

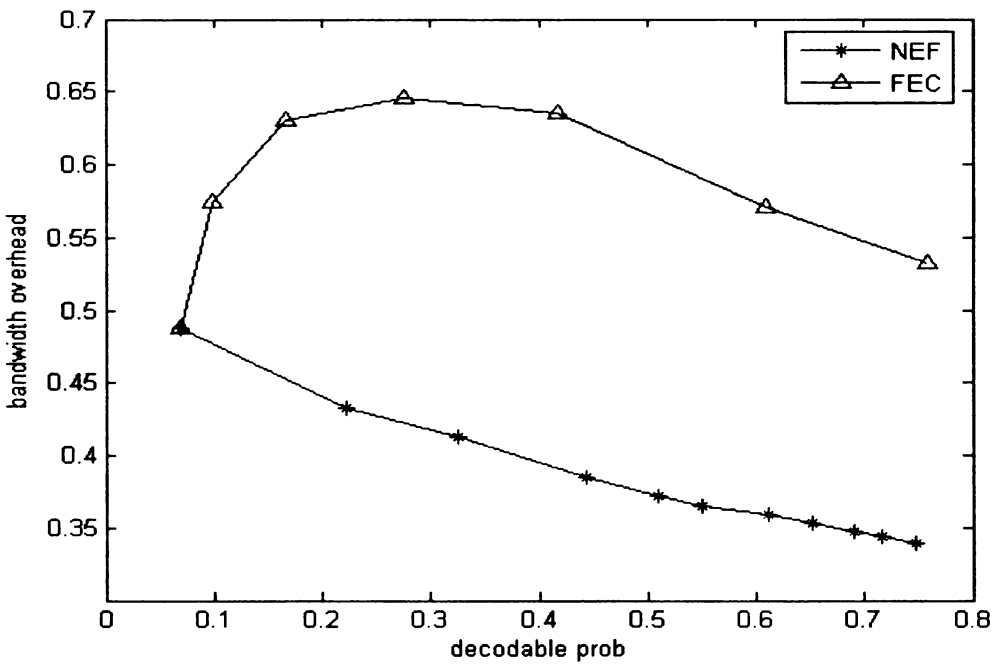



(b)

**Figure 33 Bandwidth overhead for NEF and end-to-end FEC, $\rho$ = 0.5, $p$=3% (a) P2P network; (b) proxy-based overlay network**

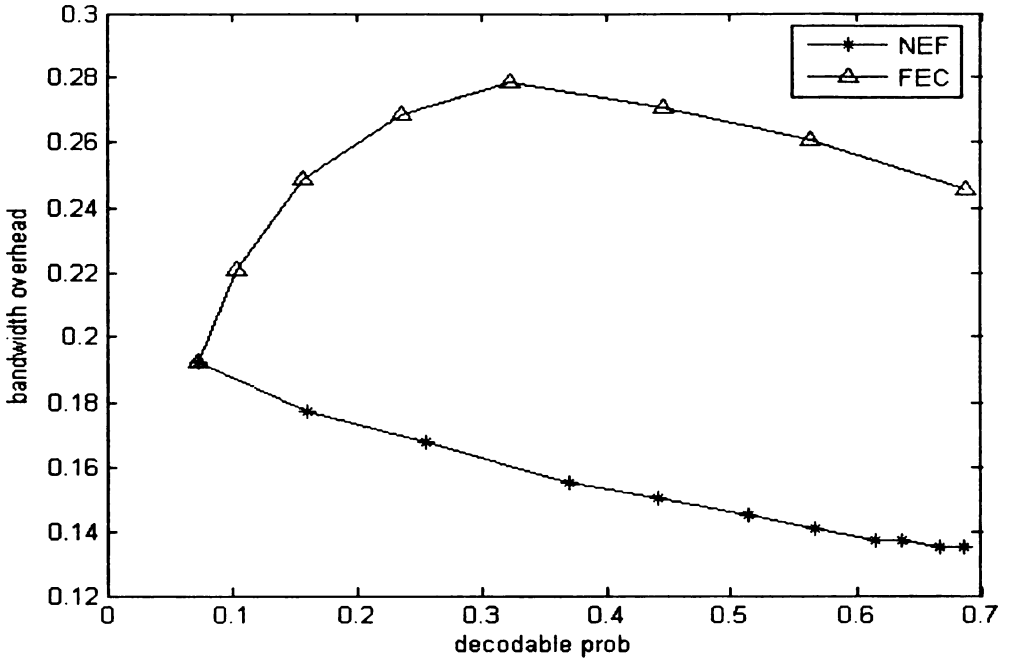



(a)

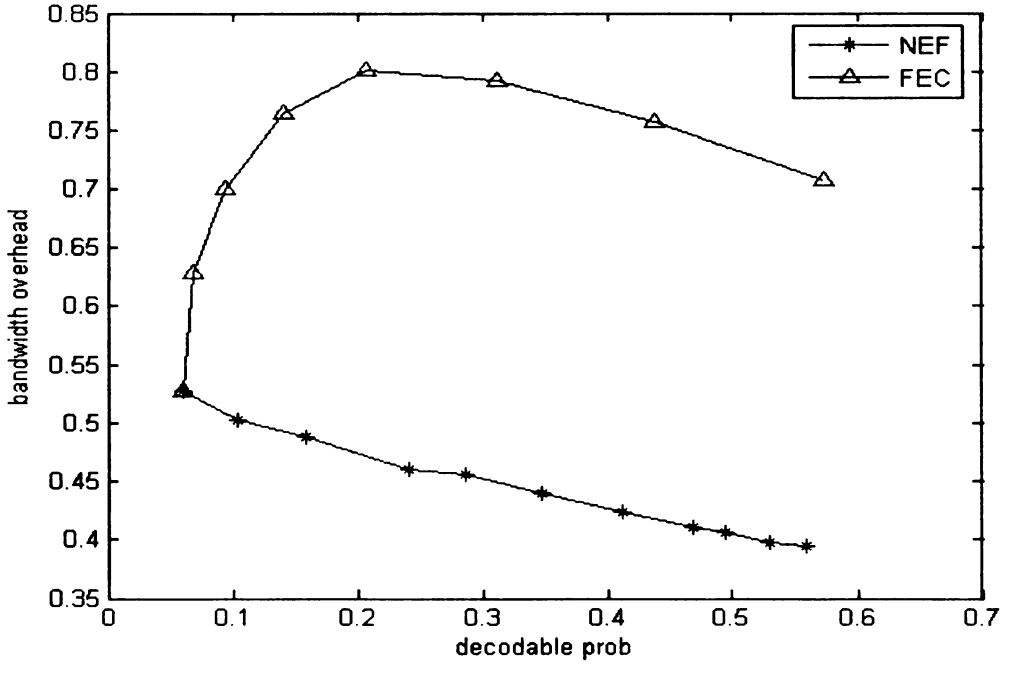



(b)

**Figure 34 Bandwidth overhead for NEF and end-to-end FEC, $\rho = 0.5$, $p$=4% (a) P2P network; (b) proxy-based overlay network**




(a)




(b)

**Figure 35 Bandwidth overhead for NEF and end-to-end FEC, $\rho = 0.5$, $p$=5% (a) P2P network; (b) proxy-based overlay network**

### 4.3.6 Performance of the greedy algorithm

The solution produced by the greedy algorithm may be sub-optimal. If there is only one codec, the greedy algorithm will give the best choice. In order to get an idea about how close we are to the optimum solution, we use the exhaustive search to find the first 2 and 3 codecs in the 100-node graph. As we have seen, the first three codecs give more improvements than the rest of the codecs; also doing exhaustive search for 4 or more codecs on a 100-node graph is too timing consuming. We run the exhaustive search on the set of trees we analyzed, with the per-link loss set to 3%, 4% and 5% respectively and the packet correlation is set to 0. In most case, the greedy algorithm finds the optimum solution.

Table 1 and Table 2 show the average decodable probability and goodput of the two algorithms when 2 and 3 codecs were placed in the tree. The difference of between the results of the two algorithms is small. Studying each tree separately, the worst case occurred on a tree when the codec number is 2 and link loss of 4%; in this case, the greedy algorithm gave a decodable probability and goodput of 45% and 88.4% respectively, while the optimum gave results of 59% and 93.3%, the difference is 14% and 5%. For the same case, when the codec number increases to 3, however, the difference decreases to 7% and 2% respectively. In general, as codec number increases, the difference between the results of optimal and greedy algorithm decreases. Over all, the greedy algorithm produces satisfying results.

**TABLE 1 AVERAGE DECODABLE PROBABILITY:**

**COPARISION BETWEEN OPTIMAL AND GREED ALGORITHM**

| Num of codecs | $p$=3% | | $p$=4% | | $p$=5% | |
|---|---|---|---|---|---|---|
| | opt | greedy | opt | greedy | opt | greedy |
| 2 | 0.9110 | 0.9044 | 0.6217 | 0.5794 | 0.3671 | 0.3671 |
| 3 | 0.9488 | 0.9488 | 0.7332 | 0.7146 | 0.4809 | 0.4809 |

**TABLE 2 AVERAGE GOODPUT:**

**COPARISION BETWEEN OPTIMAL AND GREED ALGORITHM**

| Num of codecs | $p$=3% | | $p$=4% | | $p$=5% | |
|---|---|---|---|---|---|---|
| | opt | greedy | opt | greedy | opt | greedy |
| 2 | 98.5% | 98.4% | 93.9% | 91.9% | 87.8% | 87.8% |
| 3 | 99.1% | 99.1% | 95.8% | 95.4% | 90.4% | 90.4% |

### 4.3.7 Necessity of the Greedy Algorithm

One reason that we need a greedy algorithm is that we want to limit the number of codecs in a multicast session even if every node is willing to act as codec. Encoding and decoding are computation expensive operations, too many codecs may cause longer delay penalty and may transmit too many unnecessary packets into the network. In the above section, we have observed that when we embedded ten codecs in a one-hundred node network, the maximum number of codecs per source-to-sink path is only two.

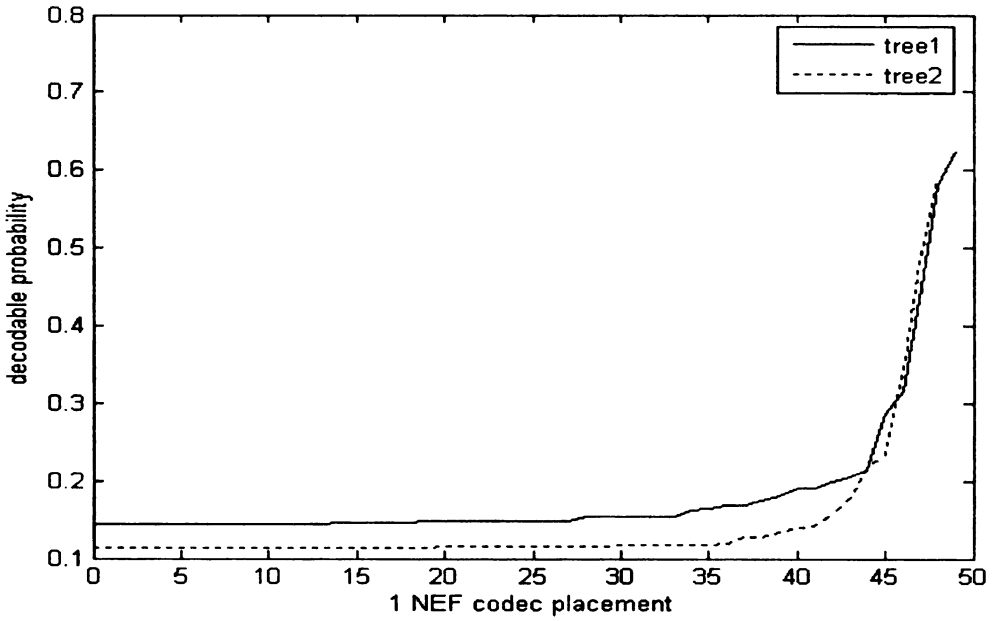
The other reason that we need the greedy algorithm is that not all possible codec arrangements necessarily lead to significant improvement in reliability and/or a near-optimal performance. We choose a proxy-based scenario to underline this argument. The results in this section are based on two arbitrarily chosen trees (for the sake of discussion we refer to these trees as "tree1" and "tree2") from the set of random trees considered in this paper.

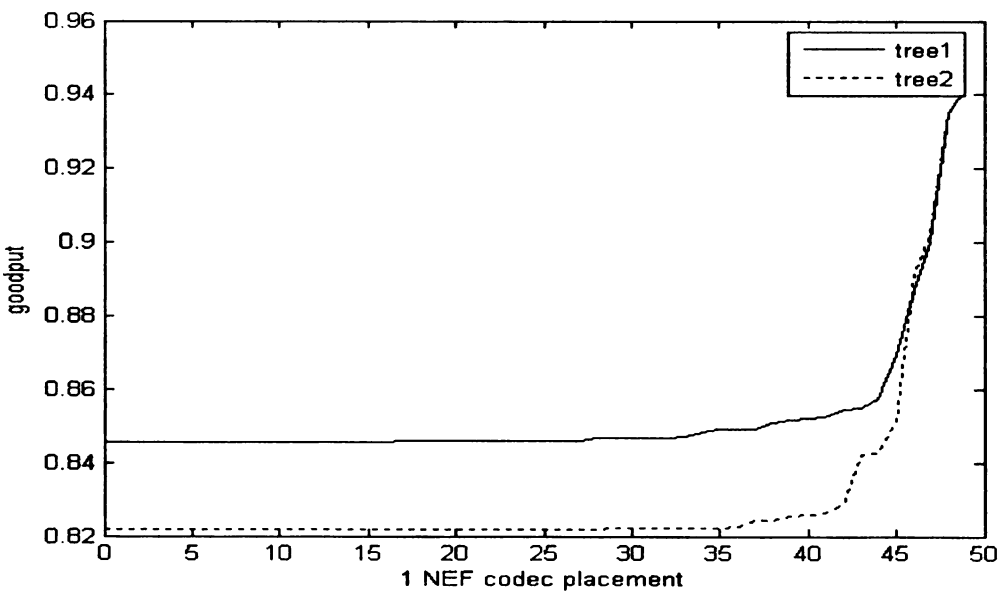
It can be observed in Figure 36 (a) - (b) that placing NEF codecs in an arbitrary manner does not necessarily lead to an increase in reliability. It should be appreciated that trivial placements (e.g. embedding a NEF codec on a leaf node) have been excluded in Figure 36 (a) - (b). The design of our greedy algorithm is such that single NEF codec embedding is always optimal. However, even for multiple codec embeddings the placement given by the greedy algorithm is almost equal (if not equal) to the optimal solution. For 2 NEF codec placement for tree2 the solution given by the greedy algorithm was equivalent to the optimal solution and for tree1 in the sorted list of placements the greedy solution was just 1 index below the optimal solution. Furthermore if all the considered solutions are assumed to be equally likely in a random NEF placement then:

- For a single NEF codec placement:
  - The decodable probability of a solution given by greedy algorithm is 62.57 % for tree1 and 58.92 % for tree2 as compared to the decodable probability of 18.54 % for tree1 and 14.54 % for tree2 obtained by averaging over all random placements.
  - The goodput of a solution given by greedy algorithm is 94.16 % for tree1 and 93.57 % for tree2 as compared to the goodput of 85.33 % for tree1 and 82.93 % for tree2 obtained by averaging over all random placements.
- For a placement of two NEF codec:
  - The decodable probability of a solution given by greedy algorithm is 79.82 % for tree1 and 83.39 % for tree2, the results for the optimal solution are 83.25% for tree1 and 83.39% for tree2 respectively, as compared to the decodable probability of 22.87 % for tree1 and 17.98 % for tree2 obtained by averaging over all random placements.

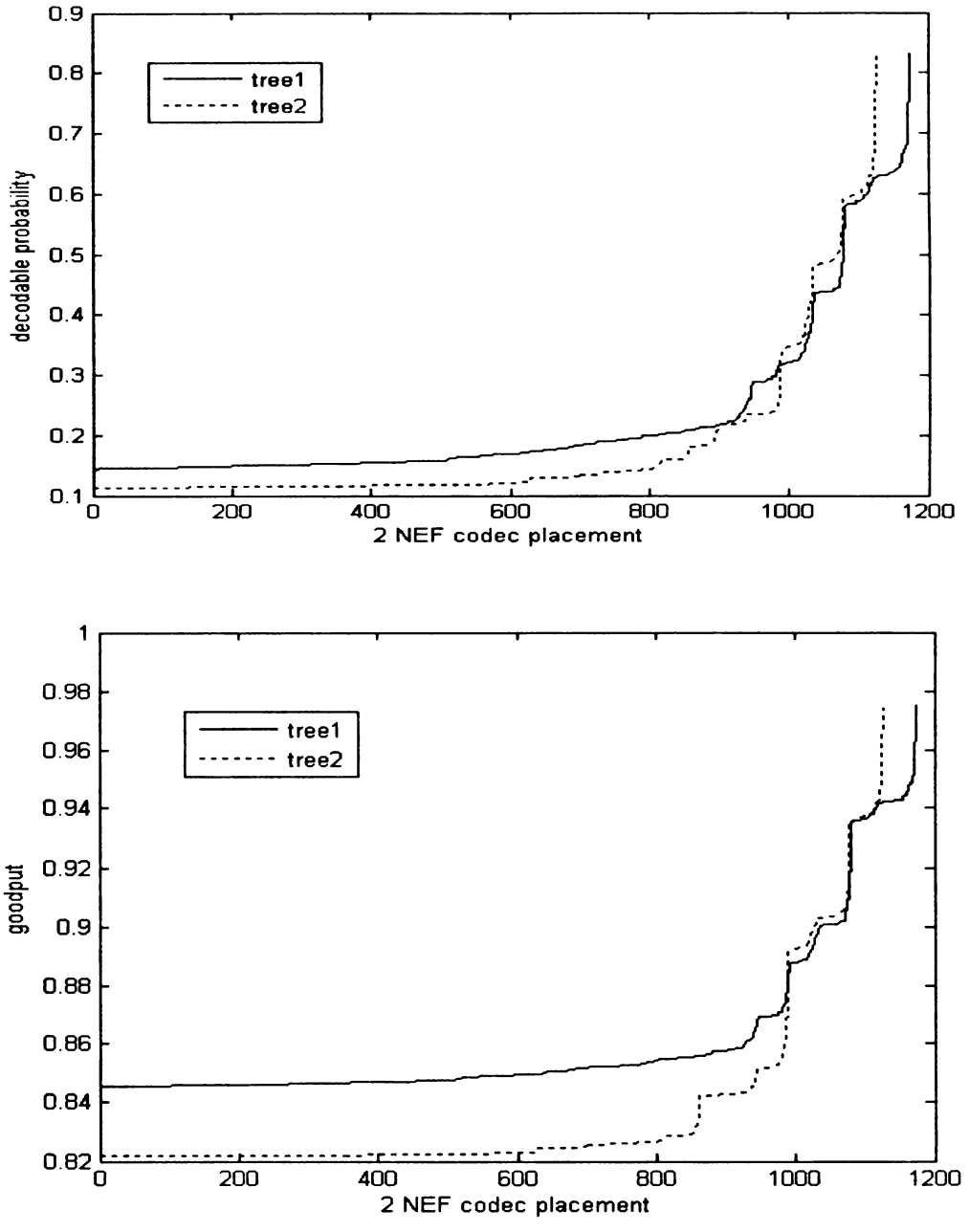
- The goodput of a solution given by greedy algorithm is 97.06% for tree1 and 97.46 % for tree2, the results for the optimal solution are 97.57% for tree1 and 97.46% for tree2 respectively, as compared to the goodput of 86.14 % for tree1 and 83.72 % for tree2 obtained by averaging over all random placements.

Thus from the above discussion it should be clearly evident that an efficient NEF codec placement algorithm is necessary and the greedy algorithm that we have proposed is indeed such an algorithm. The proposed algorithm not only is near optimal but also improves the decodable probability and goodput by a significant margin when compared with a random placement. It should be noted though that random algorithms where, all the solutions considered here are not equally likely, might provide better performance. However as the solution given by the greedy algorithm is almost always equal to the optimal solution, we believe that it would be difficult to design a simple enough random algorithm that can provide performance comparable to the greedy algorithm.


tree1
tree2
decodable probability
1 NEF codec placement
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
5
10
15
20
25
30
35
40
45
50


tree1
tree2
goodput
1 NEF codec placement
0.96
0.94
0.92
0.9
0.88
0.86
0.84
0.82
0
5
10
15
20
25
30
35
40
45
50

(a)




(b)

**Figure 36 The performance in terms of decodable probability and goodput for different NEF codec placements for two arbitrarily chosen topologies. The placements are ranked in a increasing order of efficiency. ( a ) Embedding a single NEF codec ( b ) Embedding two NEF codecs**

# Chapter 5 A Distributed Codec Placement Algorithm

The analysis in chapter 4 shows that a few codecs placed in the intermediate nodes of a network can greatly improve the decodable probability. In reality, it is impossible to know the loss rate of each branch before hand, especially the loss rates are always in dynamic change; further, the topology of a multicast session will change with time as users join and leave the session randomly; also it does not scale to run an centralize algorithm for a large network. We implement a simple distributed algorithm to place codecs in the intermediate nodes without the knowledge of full topology; the algorithm can also cope with the dynamics of the network. We use ns2 [82] to simulate the performance of the distributed algorithm.

## 5.1 Algorithm Design

In a distributed codec placement algorithm, nodes need to exchange information with each other. These information will be used by each node in its decision making process to decide if it should act as a codec. The exchange of information will cause management and traffic overhead. For management overhead, we mean that nodes need a mechanism to exchange and store the information. The traffic overhead has relationship with message size and how frequently these messages are exchanged. In a distributed algorithm, these overheads should be kept as low as possible.

In our algorithm, we assume that every node is willing to act as codec if it meets certain conditions; we also assume that nodes are willing to exchange information to each other for the codec placement purpose. Even if every node can act as codec, we want to limit the number of codecs in a multicast session. Encoding and decoding are computation

expensive operations, too many codecs may cause longer delay penalty and may transmit too many unnecessary packets into the network.

The distributed algorithm should deal with the dynamics of the network. In a multicast session, nodes join and leave randomly and the structure of the multicast distribution tree is changing constantly. This is especially true when an intermediate node leaves the multicast tree; all its children need to find their new parents and the structure of the multicast tree may change dramatically. The positions of the codecs should be changed to reflect the structure change of the network topology in a timely fashion. Another dimension of the network dynamics is that the conditions of the network, including the congestion conditions of nodes and available bandwidth resources, also change with time, this will cause variations in the loss rates of branches of the multicast distribution tree. A node may require a different number of parity packets to decode a FEC block at different times. A codec should adjust the number of parity packets it produces and transmits according to the requirements of its children at different times.

In order to limit the number of codecs in the network, we impose certain conditions for a node to be a codec. The first criterion for a node to be a codec is that on average, the node should be able to decode FEC blocks; only when a node can decode a FEC block can it be able to produce extra parity packets requested by its children. The second criterion for a node to be a codec is that it has children that need extra parity packets. If all the children of the node are able to receive enough packets to decode FEC blocks, there is no need for this node to send extra parity packets down stream.

Often there are multiple codec placement choices to satisfy the nodes' requirements in a multicast application. We use the same multicast tree in Figure 1 as an example, here we

assume this is a p2p multicast tree, each node in the tree is an end system. We redraw the tree in Figure 37. Assume node 0 is the source; node 1, 2, 3, 4 on average can decode FEC blocks; node 5, 6 and 7 on average can not receive enough packets to decode FEC blocks and requires 3, 5 and 8 extra parity packets respectively. One scheme is to have two codecs, node 3 and 4; where node 3 will send 3 extra parity packets to node 5, and node 4 will multicast 8 extra parity packets to node 6 and 7. The other scheme is to have just one codec, node 3; where it will multicast 8 parity packets to node 4, 5, 6 and 7. The later scheme may reduce the number of codecs, but it requires node 4 and node 5 to receive more extra packet.

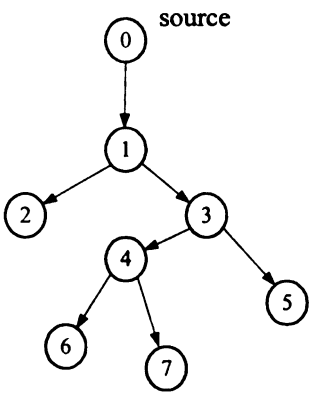



**Figure 37 a p2p overlay multicast tree**

We call those nodes that on average can decode FEC blocks as codec candidates. To select a limited number of codecs from these codec candidates, one approach is to let these candidates to exchange link state information with each other, decisions can then be made based on the partial knowledge of the topology of the network. This approach, while maybe feasible, is rather complicate, especially when the network is in a dynamic change. Also when codecs are added to the network, nodes down streams that could not receive enough packets to decode FEC blocks before can now receive extra parity

packets from the upstream codecs and maybe qualified as codec candidates, this makes the problem further complicate.

We designed a simple algorithm to solve this problem. In our approach, each node sends feedback messages to its parent periodically. In the feedback message, the node tells its parent the biggest number of parity packets its children have required and the number of children it has that on average can not decode FEC blocks. A codec candidate, on receiving the feedback message, if it finds that the number of children that can not decode FEC blocks passes a certain threshold, will make a decision to become a codec. Set a threshold on the number of children that can not decode FEC blocks can limit the number of codecs in a network. If the threshold is set to 1, then every node that on average can decode FEC blocks and have at least one child that needs extra parity packets will become a codec. If we set the threshold higher, the positions of codecs are pushed to the higher positions of the multicast tree and the number of codecs in the network will decrease.

For a node to acquire the information about the biggest number of parity packets its children have requested and the number of its children that can not decode FEC blocks, the feed back message should start from leaf the nodes of the network; an intermediate node should not send feedback until it receives feedbacks from all its immediate children. This, however, will cause a lot of time delay before the first codec is assigned. In a real time application, intermediate nodes first detect packet losses, and codecs should be put in places as soon as possible. In our algorithm, an intermediate node, once detects that it can not decode a FEC block, will send feedback to it parent immediately even if it does not get any feedback from its children. For example, in Figure 37, if node 3 detects

packet losses and can not decode a FEC block, it will send a feedback message to node 1 without waiting feedback messages from node 4 and 5. Node 1, on receiving feedback from node 3, if it decides to be a codec, will produce and transmit extra parity packets to node 3; the number of parity packets it sends, however, will be adjusted once the information about node 4, 5, 6 and 7 is obtained.

The time interval between consecutive feedback messages sent by a node should be selected carefully according to a specific application. While a shorter interval may cause more traffic overhead, a longer interval may not be able to reflect the dynamics of the network timely. In addition, some critical information may need to pass to the codec immediately. For example, In Figure 36, if node 1 is a codec and node 7 has asked the biggest number of parity packets among the children of node 1. If node 7 leaves the multicast session, the biggest number of parity packets required from node 1 should be updated immediately, otherwise, unnecessary redundant parity packets are sent downstream. On the other hand, given the same setting as in the previous example, if the link between node 4 and 6 becomes deteriorated and causes node 6 to loss a lots of packets, node 4 and 3 should pass this information immediately to node 1 so that node 1 can produce more parity packets as requested.

There are maybe multiple codecs along the path from the source to a receiver. The immediate codec of a node is the upstream codec that is nearest to the node. The packets a node receives consist of the original message and parity packets sent by the source and the parity packets produced by one or more codecs on the path between the source and the node.

Assuming we use $RS(n,k)$ FEC code. Each node estimates the average number of packets it receives per FEC block. We use *avg_recv* to represent this estimation. If $avg\_recv \geq k$, this node is a codec candidate. This estimation, however, cannot be used to calculate the number of extra parity packets it requires from its immediate codec in order to decode a FEC block.

Let $avg\_recv_i$ be the estimation of the average received packets per FEC block of node *i*. When there no codecs upstream, $k$ - $avg\_recv_i$ is the average number of extra parity packets needed for node *i*. If there are already one or more codecs upstream, then $k$ - $avg\_recv_i$ represents the incremental requirement based on the number of parity packets those codecs have already produced and sent.

Figure 38 shows an example. In Figure 38(a), there is no codec in the multicast tree, each number outside the circle is the number of parity packets that node requires; in this case, node 3 requires 2 packets and node 6 requires 10 packets etc. In Figure 38(b), node 1 receives feedback from node 2 and 3 and becomes a codec. Node 1 then produces and transmits 2 extra parity packets downstream, as represented by the number 2 by the side of the "codec" label. If these parity packets does not suffer to loss and the network condition does not change, then the average received packets for each node down stream node 1 will increase by 2, and $k$ - $avg\_recv_i$ will decrease by 2. In Figure 38(b), the number outside each circle is updated to reflect these changes. In this case, node 3 requires 0 parity packets and node 6 requires 8 parity packets, etc. Notice the number outside node 2 becomes -1, this means node 2 receives more parity packets than it needs. Figure 38(c) reflects the updated number when node 1 receives feedback message from node 6 and sends 10 extra parity packets downstream.

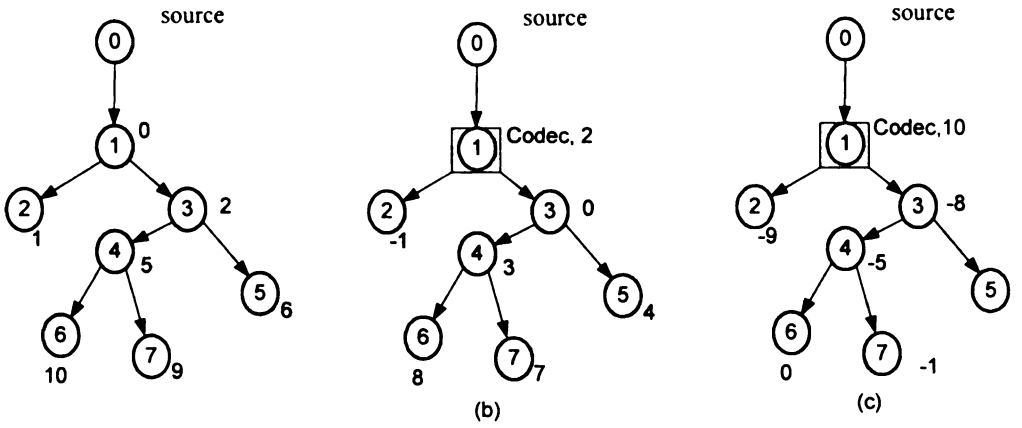



**Figure 38 the number of parity packets of a node requires changes as codec sends different number of packets**

Comparing Figure 38(a) and (b), using node 6 as an example. Node 6 uses *k - avg_recv* as the number of parity packets it requires and send this message to its parent node 4. When *k - avg_recv* is changed from 10 to 8, node 4 does not know whether this change is caused by a codec upstream sending extra parity packets or caused by the improved network condition on the branch that connect node 4 and 6. Comparing Figure 38(b) and (c), *k - avg_recv* of node 6 is changed from 8 to 0; again, node 4 does not know if the change is caused by a codec sending a different number of parity packets[5] or by the dynamic network condition. In either case, it is difficult for node 4 to update as exactly how many parity packets node 6 is requiring.

In order to solve this problem, we require each node to perform another estimation -- the average received packets per FEC block without counting the parity packets produced by its immediate codec upstream. If we use *avg_recv_imm* to represent this estimation, then *k - avg_recv_imm* represents the number of parity packet a node requires from its immediate codec upstream. In the above example, if the network conditions do not

[5] In a distributed environment, a node in general does not know how many parity packet a codec upstream produces and transmits.

change, then $k$ - *avg_recv_imm* of node 6 will not change no matter how many parity packets the codec (node 1) produces and transmits. Using the same example, in Figure 38(a), $k$ - *avg_recv_imm* of node 6 is 10. In Figure 38(b) and (c), if the network condition does not change, and node 1 produces and sends 2 and 10 parity packets respectively, $k$ - *avg_recv_imm* of node 6 will remain 10 unchanged. However, if the network conditions are changed, $k$ - *avg_recv_imm* of node 6 will be changed to reflect the differences.

A codec $i$ will also send feedbacks to its parent. The number of parity packets it requires is not the biggest number among its children, but the number it requires itself from its immediate codec upstream, again we use $k - avg\_recv\_imm_i$ to represents this number.

Initially, when a node $i$ decide to become a codec, if there are no codecs between node $i$ and the source, the node knows the largest number of parity packets required by each of its immediate children from itself. If there are codecs upstream, the node knows the largest number of parity packets required by each of its children from its immediate codec upstream, we use $req_j$ to represent this number for its child $j$; once this node becomes a codec, the biggest number of parity packets child $j$ requires from itself would then be $req_j - (k - avg\_recv\_imm_i)$.

In Figure 38(b), the immediate codec upstream of node 6 is node 1. Here, we can see that once node 1 becomes a codec, node 3 can now receive enough packets to decode FEC blocks and becomes a codec candidate. Node 3 has 4 children that cannot decode FEC blocks, if this number passes the threshold we have set, then node 3 becomes a codec. Now the immediate codec upstream of node 6 becomes node 3. After node 3 becomes a codec, node 3 will also send feedback to its parent, node 1; the number of parity packets it requires now is not the biggest number required among its children (in this case, 10),

but the number of parity packets itself requires (in this case, 2); the number of children that can not decode FEC blocks, however, is the same as that before node 3 became a codec (in this case, it is 5, including node 3 itself), this will remain node 1 to act as codec. Initially, node 3 knows the number of parity packets its children requires from its immediate codec upstream (in this case 10), as node 3 also requires 2 parity packets from its immediate codec upstream, it estimates that the biggest number of parity packets required from its children would be 10-2 = 8. After this initial stage, the number of parity packets in the feedback message would reflect the requirements from its children to itself. A node will keep a record for each codec upstream. When a node receives a parity packet produced by a codec, it will record its IP address and the number of hops between the codec and itself. A node can differentiate the packets it receives from different upstream codecs by their IP addresses. Further, a node can tell which codec is its immediate codec by the distance (the number of hops) between the codec and itself.

## 5.2 Algorithm Implementation

In the above section, we set out the principles that we are going to use in designing the distributed algorithm. In this section, we detail the implementation of the distributed algorithm in network simulator ns2.

In the proposed distributed algorithm, each node maintains a linked list of *child_info* data structure (except leaf nodes). Each *child_info* data structure stores information needed regarding each of its immediate child. The *child_info* has the following data members:

- *req_parity* indicates the parity packets requested by the child from its immediate codec upstream;

- *req_children* indicates the number of nodes in the sub-tree rooted at the child that, on average, can not decode FEC blocks if it does not receive parity packets produced by its immediate codec upstream .

Each node also keeps the following records about itself:

- *avg_recv,* an estimate for the average received packets per FEC block;
- *avg_recv_imm*, the average received packets per FEC block without counting the parity packets it receives from its immediate codec upstream.

In the following description of the distributed algorithm, we use the node number as index to reference the above parameters of a particular node. For example, $avg\_recv_i$ represents the average received packets per FEC block of node *i*.

Initially, there is no codec in the multicast session. Each node estimates its average received packets per block *avg_recv*. At this point, *avg_recv_imm and avg_recv* are equal. (Once there are codecs between a node and the source, these two parameters must be estimated separately). The number of parity packets each node requires from its immediate codec upstream is (*k*– *avg_recv_imm*), represented by *avg_req*. The parity packets sent by a codec, however, are suffered from losses. Assuming the loss rate from the nearest codec to node *i* is $r_i$, we adjust the number of parity packets node *i* requires to

$$avg\_req_i = (k - avg\_recv\_imm_i) * (1 + r_i)$$

Each parity packet sent by a codec have a sequential number and a group tag, this can help a node to estimate $r_i$. Also there is variation for the received packets per FEC block; this is especially true when loss occurs in bursts. Assuming the variation that node *i* receives $avg\_recv\_imm_i$ packets is $avg\_dev_i$ , then the number of parity packets node *i* requires is adjusted to

$$avg\_req_i = (k - avg\_recv\_imm_i) \times (1 + r_i) + avg\_dev_i$$

We use the following equations to calculate the estimations of the parameters listed above:

$$avg\_recv(K+1) = (1-g) \times avg\_recv\ (K) + g \times recv(K+1)$$

$$avg\_recv\_imm\ (K+1) = (1-g) \times avg\_recv\_imm\ (K) + g \times recv\_imm(K+1)$$

$$avg\_err(K+1) = |avg\_recv\_imm(K+1) - avg\_recv\_imm\ (K)|$$

$$avg\_dev(K+1) = (1-h) \times avg\_dev(K) + h \times avg\_err(K+1)$$

Here, *avg_recv* (*K*) represents the average received packets per FEC block up to the time when a node receives the *Kth* FEC block; *recv*(*K+1*) represents the packets received by a node for block *K+1*; In our estimation, *g* is set to 0.1 and *h* is set to 0.2.

Let $max\_req_i$ be the largest number of parity packets required by the node itself and its immediate children; $sum\_children_i$ be the total number of nodes in the subtree rooted at this node(including itself) that, on average, can not decode FEC blocks if they do not receive parity packets produced by its immediate codec upstream. If node *i* is a leaf node, then $max\_req_i = avg\_req_i$; if $avg\_recv\_imm_i < k$, then $sum\_children_i = 1$, otherwise, $sum\_children_i = 0$; if node *i* is an intermediate node, $max\_req_i$ is initialized to $avg\_req_i$; if $avg\_recv\_imm_i < k$, then $sum\_children_i$ is initialized to 1, otherwise, it is initialized to 0. For the intermediate nodes, *max_req* and *sum_children* will be updated when it receives feedback from its children.

Each node sends feedback to its parent periodically. In our algorithm, a node will send feedback each time a new FEC block is received. We still use *RS*(255,223) code as an example in our simulation, so a node will send one feedback (very small) packet per 255 (standard size) data packets.

In the feedback message, node $i$ reports to its parent $max\_req_i$ and $sum\_children_i$. As these information is passed from the leaf nodes toward the root of the multicast tree, eventually, each node will know the biggest number of parity packets required among the nodes that belong to the subtree rooted at this node; each node will also know the total number of nodes in this subtree that, on average, can not decode FEC blocks if they do not received extra parity packets from the immediate codec upstream this node.

Let $C_i$ represents the immediate children of node $i$, Figure 39 describes the action taken by node $i$ when it sends a feedback message. Here, a codec will also send feedback; however it will only send to its parent the number of parity packets it requires itself, not the biggest number required by its children.

Assume node $j$ is node $i$'s parent. When node $j$ receives a feedback from node $i$, node $j$ will first update node $i$'s information in $child\_info_i$, then it will make a decision to see if it can act as a codec. If $avg\_recv_j \geq k$ and the sum of nodes among its children that can not decode FEC blocks is bigger than a certain threshold, say *thresh_children* (we use the threshold to set a limit on the number of codecs), then the node can make a decision to become a codec. Figure 40 and Figure 41 summarize the action taken by node $j$ when it receives feedback from node $i$.

Procedure *Send_feedback*

Node $i$ send feedback its parent node $j$

IF node $i$ is a leaf node THEN

$max\_req_i = avg\_req_i$

IF $avg\_recv\_imm_i < k$ THEN

$sum\_children_i = 1$

ELSE IF $avg\_recv\_imm_i \geq k$ THEN

$sum\_children_i = 0$

END IF

END IF

IF node $i$ is a intermediate node THEN

$$max\_req_i = \max_{m \in C_i} (child\_info_m \rightarrow req\_parity, avg\_req_i)$$

IF $avg\_recv\_imm_i < k$ THEN

$$sum\_children_i = 1 + \sum_{m \in C_i} (child\_info_m \rightarrow req\_children)$$

ELSE IF $avg\_recv\_imm_i \geq k$ THEN

$$sum\_children_i = \sum_{m \in C_i} (child\_info_m \rightarrow req\_children)$$

END IF

END IF

IF node $i$ is codec THEN

Send $max\_req_i = avg\_req_i$

END IF

Send $max\_req_i$, $sum\_children_i$ to node $j$

END *Send_feedback*

**Figure 39 *send_feedback* procedure**

Procedure *Receive_Feedback*

Node *j* receive feedback from node *i*

$child\_info_i \rightarrow req\_children = sum\_children_i$

$child\_info_i \rightarrow req\_parity = max\_req_i$

Call Procedure *Decision_making*

End *Receive_Feedback*

**Figure 40 *receive_feedback* procedure**

Procedure *Decision_making*

Node *j* make a decision whether it should be a codec

$$sum\_children_j = \sum_{m \in C_j} (child\_info_m \rightarrow req\_children)$$

$$req\_max_j = \max_{m \in C_j} (child\_info_m \rightarrow req\_parity)$$

IF node *j* not a codec THEN

IF $avg\_recv_j \geq k$

and $sum\_children_j \geq thresh\_children$

and $req\_max_j > 0$ THEN

Set node *j* a codec

END IF

ELSE IF node *j* is a codec THEN

IF $avg\_recv_j < k$

or $sum\_children_j < thresh\_children$

or $req\_max_j \leq 0$ THEN

Set node *j* not a codec

END IF

END IF

End *Decision_making*

**Figure 41 *decision_making* procedure**

Once a node becomes a codec, if it can decode an FEC block, it can reconstruct the original FEC block; at the same time, it will produce the largest number of parity packets

required by its children (the maximum among *child_info→ req_parity*) and forward to each child the number of parity packets they have required (*child_info→ req_parity*). Each node along the forwarding path of the multicast tree also knows the largest number of parity packets required by each of its children (*child_info→ req_parity*), so the node does not necessarily forward all the parity packets it receives to all of its children, but only forward the number of parity packets each child has requested; this way, the transmission overhead is decreased.

The algorithm runs on top of an overlay multicast routing protocol. The random join and leave of nodes to a multicast session will not affect the performance of the algorithm. For example, when a node first joins the multicast session, its parent will allocate a *child_info* data structure for this node. The node will estimate its *avg_recv* and *avg_recv_imm* according to whether or not there are codecs upstream. When the node sends feedback to its parents, the parent will update the information stored in *child_info* for this node.

If a leaf node leaves a session, the parent of this node will detect this leave (by the routing protocol or other mechanism) and will delete the *child_info* data structure for this child. If this child has been the node that requested the biggest number of parity packets among all the children, the parent will then select the biggest number of parity packets required among all its other children and send a feedback to its parent.

If an intermediate node leaves a multicast session, the behavior of the parent of this node will be the same as that when a leaf node leaves a session. The children of this node will reset their estimation of *avg_recv* and *avg_recv_imm.* The overlay multicast routing protocol will find new parents for the children. The parents will allocate *child_info* data structures to these children as if they are newly joined nodes, and the children will restart

their estimation of *avg_recv* and *avg_recv_imm* once they receive packets from their new parents.

## 5.3 Simulation Results

We implemented the above distributed algorithms in ns2. Primitives to build overlay networks upon the underlying topologies are also implemented. The algorithm to build and maintain the overlay network is not a main focus of this work. Hence, we built an overlay network to form a Shortest Path First (SPF) multicast trees that we produced in chapter 4. On each link in the multicast tree, we insert a two state Markov error model with error packet correlation $\rho$ set to 0.5 and the average loss rates set to 3%, 4% and 5% respectively. We run the simulation on all the ten trees we produced in chapter 4, the results are the average of these simulations.

Table 2 shows the simulation results. Here, the source sends $RS(255,223)$ FEC blocks. The threshold *thresh_children* is set to 1, 5, and 10 respectively; the packet error correlation is set to 0.5 (as mentioned above). It can be observed that the *thresh_children* parameter does not have a significant impact on the overhead and goodput. As we have discussed, the *thresh_children* parameter is used to control the number of codecs in the distributed algorithm. In the distributed algorithm, codec use adaptive FEC; hence, when the number of codecs in the network is small, each codec may send more repair parity packets into the network, and this explains why the number of codecs (under the distributed algorithm) does not have the dramatic effect on the performance as in the centralized algorithm.

TABLE 3 SIMULATION RESULTS FOR DISTRIBUTED ALGORITHM

| | Thresh = 1 | | | Thresh = 5 | | | Thresh = 10 | | |
|---|---|---|---|---|---|---|---|---|---|
| Loss rate | 3% | 4% | 5% | 3% | 4% | 5% | 3% | 4% | 5% |
| Dec prob | 0.96 | 0.94 | 0.70 | 0.92 | 0.85 | 0.72 | 0.90 | 0.80 | 0.71 |
| goodput | 0.99 | 0.98 | 0.90 | 0.98 | 0.96 | 0.91 | 0.98 | 0.95 | 0.91 |
| overhead | 0.16 | 0.16 | 0.21 | 0.17 | 0.16 | 0.20 | 0.18 | 0.16 | 0.20 |

### 5.3.1 Performance analysis of the distributed algorithm

As we have discussed before, if there is no loss in the network, then each node will receive all the data packets reliably without any overhead. When there is loss in the network, we can use different error recovery schemes to increase the reliability of the receivers. Different error recovery schemes may achieve different levels of reliability and may incur different amounts of overhead.

For an error recovery scheme, we can adjust certain parameters to achieve the desired level of reliability, which, of course, will cause a certain amount of overhead. For example, a multicast application using end-to-end FEC for error recovery, if we decrease the FEC code rate, the application may become more reliable, but the bandwidth overhead may also increase. For end-to-end FEC, the only parameter that we can control is the code rate, though even for the same code rate, we still can use different FEC codes. In a centralized NEF codec placement algorithm, in addition to controlling the code rates, we can also control the number of codecs in the network. In the distributed NEF codec placement algorithm, we can control the code rates and the condition parameters that a node needs to satisfy in order for that node to be a codec.

In this section, we compare the performance of the distributed codec placement algorithm, the centralized codec placement algorithm and end-to-end FEC in terms of a "distortion measure" versus bandwidth overhead. Here, our distortion measure is the

average loss ratio of message packets experienced by the different schemes under consideration. We implement the three error recovery schemes in a multicast distribution tree, with the loss rate per-link of 3% and packet correlation of 0.5. The FEC codes we used is Reed Solomon codes with $k$ = 223 and $n$ is increased from 235 to 305 in steps of 10. For the centralized codec placement algorithm, for each FEC code we use, the number of codecs is changed from 1 to 10. In the distributed codec placement algorithm, for each code we use, the condition parameter *thresh_children* is set to 1, 5 and 10 respectively. In the centralized and distributed codec placement algorithms, if there are multiple implementations that achieve the same distortion, we select the implementation that has the minimum overhead.

Figure 42 shows the distortion versus bandwidth overhead of these three schemes. It can be observed that the two NEF schemes can achieve less distortion with less bandwidth overhead than that of end-to-end FEC. The distributed algorithm has a wider dynamic rang in overhead (from 0.05 to 0.18) than that of the centralized algorithm and end-to-end FEC. As we have explained before, we have more control on NEF than that of on end-to-end FEC. Comparing the two NEF schemes, the distributed algorithm is more flexible than the centralized algorithm. This is because: First, the number of codecs in the centralized placement algorithm is limited, while there is no limitation for the distributed algorithm. Second, the behavior of codecs in the centralized algorithm and the distributed algorithm is different. In the centralized codec placement algorithm, if a codec can decode a FEC block, it rebuilds the original FEC block; that is, for a codec that receives $k$ or more packets of a $RS(n,k)$ block, that same codec will send $n$ packets. In the distributed codec placement algorithm, a codec will send the number of parity packets a

child requested; in other words the number of parity packets a codec send may be larger or smaller than $(n - k)$.

When the overhead is between 0.14 and 0.18, the distributed algorithm causes a distortion (i.e., message packet loss rate) of about 1% to 2%. While the end-to-end FEC may cause a distortion of 4% to 25%. When the overhead is less than 10%, the distributed algorithm can obtain a distortion of less than 15%, this is better than the centralized algorithm. For overhead between 10% and 14%, the centralized algorithm outperforms the distributed algorithm, but the difference between these two algorithms is less than 5%.

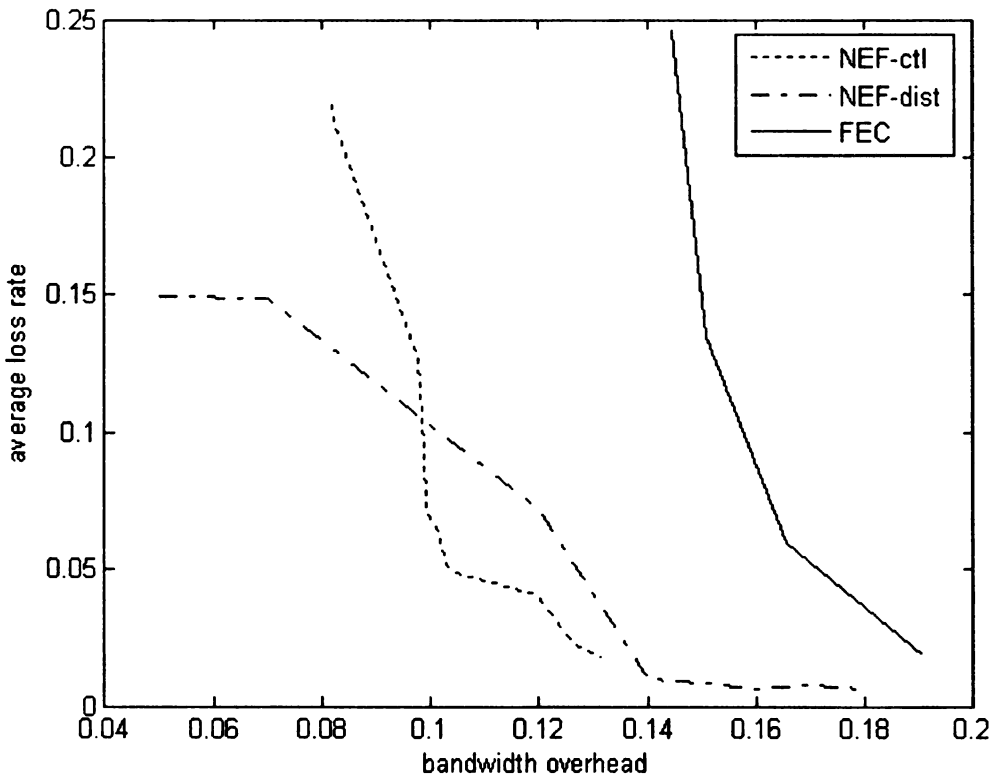



**Figure 42 Performance comparisons between distributed algorithm, centralized algorithm, and end-to-end FEC.**

# Chapter 6 A Hybrid ARQ-NEF (HANEF) Reliable Multicast Protocol

NEF is designed with realtime applications (audio/video) in mind. In some overlay network applications where reliable distributions of contents are desirable, NEF can be integrated with ARQ to achieve 100% reliability. Even for a realtime application, if the round trip time (rtt) delay between the receivers and retransmission servers is short, ARQ is still a viable option [38][83][84] to increase the reliability of an application. In this chapter, we describe the details of a Hybrid ARQ-NEF (HANEF) reliable multicast algorithm.

## 6.1 Related Work

### 6.1.1 The heterogeneous problem

In a multicast session, each receiver may require different number of parity packets for the same FEC block; also different receivers may have different round trip times to the recovery server. One of the main difficulties in designing reliable multicast protocol is to compromise between delay penalty and retransmission overhead. This can be shown in the following example. Figure 43 shows a 4-node network, where node 0 is the source, node 2 and 3 are two receivers. The numbers on the branches indicate the time delay between the nodes. The numbers below node 2 and 3 represent the number of parity packets these two nodes require in order to being able to decode a FEC block. The source first receives a feedback from node 3 requesting 6 packets; these parity packets are then produced and multicasted to all of the receivers. Node 2 then sends feedback before

it receives the retransmitted parity packets from the source requesting 10 packets; again these parity packets are produced and multicasted to all of the receivers. A total of 16 parity packets are retransmitted, which is much more than the requirement of the worst case node.

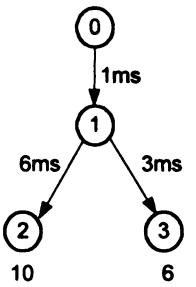



**Figure 43 synchronization among receivers is difficult for reliable multicast**

The above example just shows one round of feedback and retransmission; as retransmitted packets also suffer from losses, multiple rounds of feedbacks and retransmissions are required for each receiver to achieve reliability. As the number of receivers increases, the problem becomes worse.

One way to solve this problem is that, for each round, the source will wait until it receives feedbacks from all the receivers, as the round-based protocol proposed in [38]. In the above example, the source will not retransmit parity packets until it receives feedbacks from both node 2 and 3. This approach guarantees that the source will send no more parity packets than the requirement of the worst case node, but requires all the receivers adapt to the slowest receiver in the network. Also in the multicast session, as users join and leave the multicast session, it is difficult for the source to decide if it has received feedback from all the receivers.

An alternative way is that the source waits a random period of time before it sends out the retransmission packets. In this case, the source does not have to wait until it receives the feedback from the slowest node, but it will also send out more parity packets than the requirement of the worst case node. This approach is a compromise between delay penalty and bandwidth overhead. The time that the source should wait need to be designed carefully in order for this method to be efficient [48].

Another approach is that each node waits a random time period before it sends feedback; also the feedbacks are multicasted to all the receivers in the multicast session or to the receivers in local recovery groups [31][85]. In the above example, if node 3 holds back 7ms before it sends feedback and node 2 holds back 1ms, then the feedback message multicasted by node 2 will reach node 3 before it sends feedback itself. Once node 3 receives the feedback message, if the number of required parity packets is bigger than the number that it requires, it will suppress its feedback message. The performance of this scheme depends on the random time each node should wait before it sends out feedback. In the above example, if node 2 holds back longer than node 3, then there will be no improvement in the performance. The random time delay needs to be selected carefully and may need change according to different applications; also one or more feedback multicast groups need to be formed and may incur more management overhead.

### 6.1.2 Source based and distributed error recovery

There are lots of literatures on reliable multicast. Generally, they can be classified into two classes: source based error recovery and distributed error recovery [43]. In source based error recovery, only the source will process feedbacks and retransmissions. This can easily lead to the feedback explosion problem. Various techniques have been adopted

to suppress feedbacks. One scheme is to use a ring-based protocol, where only one node will send feedback to the source at one time [86]. Another scheme is to multicast feedbacks to all the receivers [87] [88], as we have described in the above subsection. Retransmissions can be multicasted to all the receivers or can be unicasted to a particular receiver. In [87] and [89], the authors have proposed a scheme where retransmission is unicasted or multicasted based on the number of receivers that sent the requests. All of the schemes we referenced above require the source to wait for a period of time before sending retransmission.

In a large multicast session, even when feedback suppression techniques are adopted, it is still very difficult for the source to process all of the feedbacks and retransmissions. In this case, receivers are organized into local groups where each group is served by a local error recovery server [31] [85]. Different feedback suppression techniques and retransmission methods can be adopted for local recovery.

We propose a reliable multicast algorithm that minimize the average delay penalty, at the same time, keep the redundant retransmitted packets no more than the requirements of the worst case node. The following subsection describes the algorithm in detail. Again we assume using a $RS(n,k)$ FEC code.

## 6.2 HANEF Algorithm Design

We add two more parameters in the repair request and retransmission packet header to synchronize the activities between the receivers and the repair servers (source or codecs): *round* and *repair_sent* (for feedback, these are set to *round_f* and *repair_sent_f* respectively; for retransmission, these are set to *round_t* and *repair_sent_t* respectively).

The meaning of these two parameters will be explained later. We first use a simple example to explain our approach.

In Figure 43, when node 3 first sends a repair request, it will require 6 parity packets. At the same time, it will set *round_f =1* and *repair_sent_f=0* in the feedback header. When the source receives the feedback message, it will retransmit 6 parity packets immediately, at the same time, the source will set *round_t=1* and *repair_sent_t=6* in the retransmission header. Also the source will record the retransmission status for this FEC block, in this case, one round of parity packets has been transmitted and the number of parity packets retransmitted is 6. Node 2 will send its first feedback message before it receives the retransmitted parity packets from the source. Node 2 will require 10 parity packets and it set *round_f=1* and *repair_sent_f=0* in the feedback header. When the source receives this feedback message, *round_f=1* and *repair_sent_f=0* in the feedback header indicate that node sent this feedback message before it receives the last retransmitted parity packets. The source will sent 10-6=4 parity packets. It will also set *round_t=2* and *repair_sent_t=10* in the retransmission header. Using this approach, we can see that a receiver sends feedback messages immediately once it detects that it can not decode a FEC block; and the source sends retransmission packets immediately after it receives a feedback. The number of parity packets sent by the source is no more than the requirements of the worst case node. In the following, we describe the receiver and sender side algorithms in detail separately.

### 6.2.1 Receiver Side Algorithm

Each FEC block has a *block_ID*. A receiver keeps the biggest *Block_ID* it has received in *current_ID*. It also keeps the following four parameters for each FEC block:

- *recv_pkts,* the number of packets the receiver has received for this FEC block.
- *recv_round,* the most recently retransmission round it has received, which is initialized to 0.
- *repair_sent_r,* the recorded total repair packets the source or codec has sent up to and including the *recv_round,* which is initialized to 0.
- *status,* this can be one of the four status: *RECV, WAIT, RECV_RET,* and *DONE.*

When a receiver receives the first packet of a block, the block enters into the *RECV* state. We assume that when the source sends the original data, packets from different FEC blocks are not interleaved. So when a receiver receives a block that has a *block_ID* bigger than *current_ID*, it knows that the transmission of the current FEC block has been terminated, and a new FEC block is commencing. The receiver will check the state of the current block. If it can decode the block, then this block enters into the *DONE* state, and the receiver does not have to keep any status information for this block. However, if the receiver receives less than $k$ packets of the current block, it will send the feedback message immediately and set the block into the *WAIT* state. The receiver will also setup a *wait_timer* for this block. This *wait_timer* expires after a *round-trip* time from the receiver to the source.

The header of a feedback message includes the following parameters:

- *block_ID*, the *block_ID* of the block that needs repair packets
- *req_num,* the number of repair packets this block needs, set to $k - recv\_pkts$.
- *round_f*, which is always set to *recv_round*+1.
- *repair_sent_f*, the total repair packets the source or codec has sent up to and including the *recv_round,* which is same as *repair_sent_r.*

When a receiver sends a feedback message for the first time, *round_f* is set to 1 and *repair_sent_f* is set to 0.

As we will describe later, in the head of the retransmission packet, the sender will include the *block_ID*, the *round_t*, which indicates the retransmission round of this block, and *repair_sent_t*, which represent the total retransmission packets up to and including this *round_t*. When a receiver receives this retransmission packet, *recv_round* is set to *round_t*, the receiver will also record *repair_sent_t* in *repair_sent_r*. The receiver then increments *recv_pkts* by one and checks if it can decode the FEC block. If it can, the receiver is *DONE* for this block; if it cannot, the receiver set the status of the FEC block to *RECV_RET*. Once a block is in a *RECV_RET* state, if the receiver continues to receive retransmission packets for that block but still cannot decode the block, the FEC block remains in the *RECV_RET* status; meanwhile *recv_pkts* is increased accordingly. However, if it does not receive any retransmission packets of the block within a certain *threshold*[6] time, the receiver will assume that this round of retransmission is over. If it still can not decode the FEC block, it will send another feedback and the status of the block is set to *WAIT*. This time the *round-trip* is set to be the round trip time from the node to the codec that sends the retransmission. Again the *wait_timer* will expire after the *round-trip* time.

[6] We assume that repair packets are sent consecutively at a certain *interval*, and we set some *threshold* time that is proportional to the *interval*: *threshold* = *a** *interval*, where *a* is some constant (e.g., *a*=10).

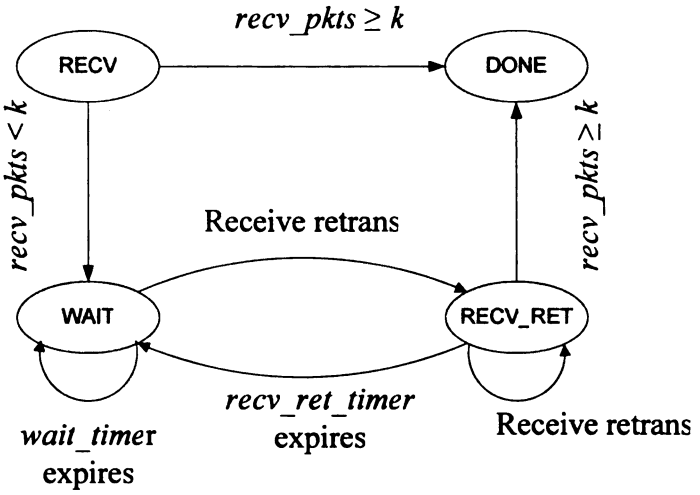



**Figure 44 state diagram of a FEC block at a receiver**

If a block is in *WAIT* state and the receiver does not receive any retransmission packets for this block within the *round-trip* time, the *wait_timer* expires and the receiver sends a feedback message for this block. This time the *recv_round* is increased by 1, the parameters in the feedback header are set in the same way as we have described before. The block stays in the *WAIT* status. Figure 44 shows the status transition diagram.

Figure 45 describes the receiver side algorithm in different state.

The send feedback and receive retransmission procedure of a receiver are described in Figure 46 and Figure 47 respectively.

```
IF status == RECV
   IF receive a packet for the current FEC block
      recv_pkts++
   END IF
   IF received the last packet of the current FEC block
      IF recv_pkts ≥ k
         Set status = DONE
      ELSE IF recv_pkts < k
         recv_round = 0
         repair_sent_r = 0
         Call send_feedback procedure
      END IF
   ENF IF
END IF

IF status == WAIT
   IF receive retransmission packets
      Call recv_retransmission procedure
   ELSE IF wait_timer expires
         Call send_feedback procedure
   END IF
END IF

IF status == RECV_RET
   IF receive retransmission packets
      Call recv_retransmission procedure
   ELSE IF recv_ret_timer expires
      Call send_feedback procedure
   END IF
END IF
```

**Figure 45 receiver side algorithm**

```
Procedure send_feedback
   req_num = k – recv_pkts
   round_f = recv_round + 1
   repair_sent_f = repair_sent_r
   send feedback message to its immediate codec upstream
   Set wait_timer
   Set status = WAIT
END Procedure
```

**Figure 46 *send_feedback* procedure**

```
Procedure recv_retransmission
   recv_pkts++
   recev_round = round_t
   repair_sent_r = repair_sent_t
   IF recv_pkts ≥ k
      Set status = DONE
   ELSE
      Set recv_ret_timer
      Set status = RECV_RET
      END IF
END Procedure
```

**Figure 47 *recv_retransmission* procedure**

## 6.2.2 Sender Side Algorithm

A sender (original source or codec) may choose to send the retransmission packets immediately or wait for sometime after it receives a feedback message. The later case only needs a small modification relative to the former one. We only focus on the case when the sender sends retransmission repair packets immediately.

Besides the *block_ID*, the sender also keeps the following parameters for each block:

- *current_round*, the most recently retransmitted round for this FEC block, initialized to 0.
- *current_sent*, the total retransmission packets that have been sent for this block up to and including *current_round*, initialized to 0.
- *last_time*, the time for the last request has been sent for this block. When certain interval have passed and the sender does not receive any feedback message for a FEC block, the sender can assume that all receivers have received this block reliably and can delete the record for this block.

When a repair server (the source or a codec) receives a feedback message, if the *round_f* in the feedback message is bigger than the *current_round*, this implies that the receiver

has received some or all of the most recently sent retransmission packets, and it is asking for more repair packets. In this case, *current_round* is set to *round_f* in the feedback message. And *current_sent* is increased by *req_num*. The repair server will multicast *req_num* number of repair packets down stream immediately. In the header of each retransmitted packet, the *round_t* is set to *current_round* and *repair_sent_t* is set to *current_sent*. Apparently, when the sender receives the first feedback message of a block, the *round_t* is set to 1 and *repair_sent_t* is set to the *req_num*, which is required by the node that has sent this feedback message.

On the other hand, if the *round_f* is equal to or smaller than the *current_round,* this implies that the node did not receive any of the most recently round of retransmission at the time it sent this feedback message. In this case, if *current_sent* is equal to or bigger than the sum of *req_num* and *repair_sent_f* in the feedback message, this feedback message is ignored; however, if *current_sent* is smaller than the sum of *req_num* and *repair_sent_f* in the feedback message, this means that even after the receiver receives all the most recently retransmitted packets, it still can not decode the block. Let *m* = *req_num* + *repair_sent_f* - *current_sent*. The *current_round* will be increased by 1 and the *current_sent* will be increased by *m*. after the repair server has updated these records, it will multicast *m* repair packets down stream. Again, in the header of each retransmitted packet, the *round_t* is set to *current_round* and *repair_sent_t* is set to *current_sent*.

Figure 48 and Figure 49 describe the *recv_feedback* and *send_retransmission* procedures at the sender side respectively.

```
Procedure recv_feedback
 IF current_round < round
    current_sent = current + req_num
    current_round = round
   m = req_num
 ELSE
    IF current_sent ≥ req_num + repair_sent_f
       m = 0
    END IF
    IF current_sent < req_num + repair_sent_f
       m = req_num + repair_sent_f - current_sent
       current_sent = current_sent + m
       current_round++
    END IF
 END IF
 While  m > 0
    Call Procedure send_retransmission
    m--;
END Procedure
```

**Figure 48 *recv_feedback* procedure**

```
Procedure send_retransmission
   round_t = current_round
   repair_sent_t = current_sent
   produce a parity packet and schedule to multicast it downstream
END Procedure
```

**Figure 49 send_retransmission procedure**

## 6.3 Algorithm Implementation in NS2

We implement the algorithm in network simulator ns2. In our implementation, the source will run the sender-side algorithm, leaf nodes will run the receiver-side algorithm and codecs will run both the sender-side and receiver-side algorithms (codecs also send feedbacks).

For *p2p* overlay networks where all intermediate and leaf nodes are receivers, if the leaf nodes receive all FEC blocks reliably, all intermediate nodes should receive all FEC

blocks reliably. To further reduce the average delay penalty in p2p networks, we can require all intermediate nodes to run the receiver-side algorithm. This option, however, is not implemented here.

A receiver that cannot decode a FEC block will send a feedback message along the reverse path of the multicast tree toward the source. When the feedback message reaches the first codec en-route, and if that codec can decode the block, it will use the required number of parity packets and multicast these packets downstream. Otherwise, if the codec could not decode the FEC block, then this codec itself is in need for some additional parity packets. Hence, in our HANEF reliable multicast model, a NEF codec could also generate a feedback message requesting parity packets from one of the upstream codec nodes or from the root node. Therefore, a codec that receives a feedback message and which cannot decode the FEC block will have to wait until it receives the retransmission packets from one of the upstream codecs or from the original source (root node). In HANEF, codecs will not forward the retransmitted repair packets transmitted by the source or other codecs opstream.

In the HANEF model, we do not employ any feedback suppression mechanism. However, the NEF framework naturally provides some form of suppression. A "natural suppression" is taking place since the NEF codec does not forward the feedback messages toward the root node (assuming that the codec can serve the feedback messages). Per each round of the HANEF model, and under the worst condition, the codec will send a single feedback message toward the root (when the codec cannot recover the FEC block). Nevertheless, it is feasible to support other elaborate forms of feedback suppression mechanisms in conjunction with the NEF framework. Since our

primary objective here is to analyze the impact of network-embedded FEC codecs, we opted to pursue a model that supports the most generic (and simplest) reliable multicast functions.

## 6.4 Simulation Results

### 6.4.1 Performance Comparison of HANEF and CER

The HANEF algorithm works well for source based error recovery scheme. In this case, the source will run the sender side algorithm and the leaf nodes run the receiver side algorithm we have designed above. We compared the performance of the source based HANEF with the Centralized Error Recovery (CER) scheme described in [43]. For simplicity, we did not implement any feedback suppression in CER, a receiver will send feedback to the source using unicast once it detects that it can not decode a FEC block. The source will wait for some time to accumulate feedbacks from receivers before it retransmits repair packets; we use the variable *Tw* to represent this wait time. As indicated by [48], the value of *Tw* will impact the performance of CER. Also in CER, the retransmissions are multicasted downstream. We implement HANEF and CER on the set of the trees that we have used in Chapter 4. The delay on each branch is set to 0.1 second; the loss rate per link is 3%. We compare the performance of HANEF and CER when *Tw* is changed from 0 to 300ms.

TABLE 4 and TABLE 5 show the average bandwidth cost (defined in chapter 4) and the average delay latency[7] for a node to receive enough packets to decode a FEC block for

[7] In [42] it has been shown that using software implementation on a 166MHz Pentium PC, the data rate of $RS(255,223)$ can reach 1.1M bps. For packet size of 512 bytes, encoding one packet only need 0.4ms. Using more advanced PC or dedicated hardware, the encoding rate can be much faster. Assuming a hop delay of 100ms, the encoding and decoding delays at codecs are negligible.

CER and HANEF respectively. For the CER approach, it can be seen that as *Tw* increases, the bandwidth cost decreases but delay latency increases. From TABLE 4, when *Tw* equals 300ms, the bandwidth cost is 1.38 and the delay latency is 1.28 seconds for CER; the bandwidth cost and delay latency are 1.11 and 1.21 seconds for HANEF. We can see that the bandwidth overhead of HANEF is much less than the CER approach, while the delay penalty of HANEF is close to that of CER.

**TABLE 4 COMPARISON OF BANDWIDTH COST AND DELAY FOR CER AND HANEF, AVERAGE OVER ALL NODES**

| | CER | | | | | | | HANEF |
|---|---|---|---|---|---|---|---|---|
| *Tw* (ms) | 0 | 50 | 100 | 150 | 200 | 250 | 300 | |
| Bwd_cost | 2.23 | 1.69 | 1.60 | 1.50 | 1.49 | 1.41 | 1.38 | 1.11 |
| Delay (s) | 1.11 | 1.06 | 1.13 | 1.12 | 1.18 | 1.19 | 1.28 | 1.21 |

**TABLE 5 COMPARISON OF BANDWIDTH COST AND DELAY FOR CER AND HANEF, AVERAGE OVER LEAF NODES.**

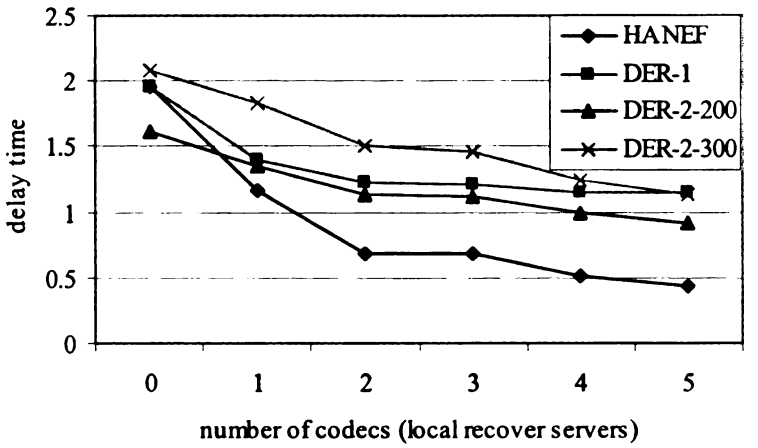
| | CER | | | | | | | HANEF |
|---|---|---|---|---|---|---|---|---|
| *Tw* (ms) | 0 | 50 | 100 | 150 | 200 | 250 | 300 | |
| Bwd_cost | 4.59 | 3.48 | 3.29 | 3.09 | 3.07 | 2.9 | 2.84 | 2.29 |
| Delay (s) | 1.20 | 1.19 | 1.27 | 1.27 | 1.34 | 1.35 | 1.45 | 1.41 |

## 6.4.2 Performance Comparison of HANEF and DER

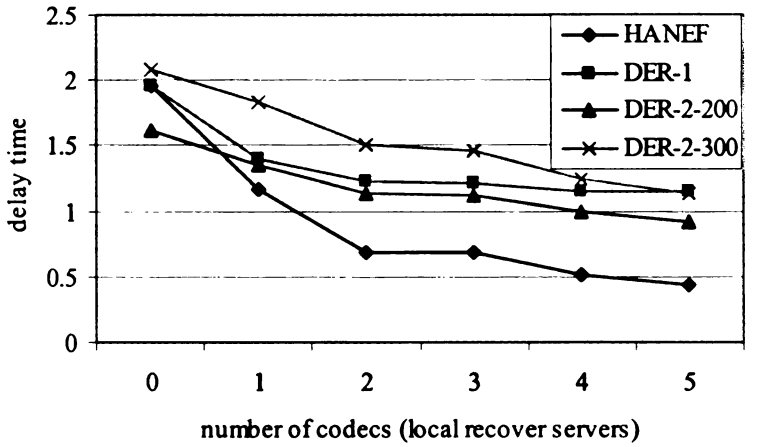
We compare the performance of HANEF with several Distributed Error Recovery (DER) schemes [43]. In the first DER scheme, we had the source and local recovery server run the sender side algorithm, and the leaf nodes run the receiver side algorithm.of our proposed HANEF approach that we have described above We call this approach HANEF/DER scheme, and for short we denote to this approach as DER-1. In the second DER scheme, we have all the local recovery servers run the centralized algorithm. That is, the local recovery server will wait for some time to accumulate the feedback messages

before they send out repair parity packets. We set the wait time to 200ms and 300ms, we label these scenarios as DER-2-200 and DER-2-300 respectively. Due to the time-complexity of the simulations, we only simulate these schemes on one of the trees we have used in chapter 4. .

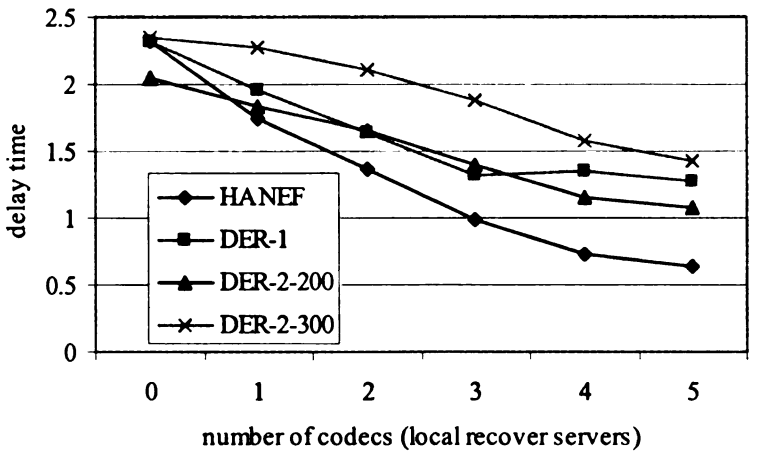
Figure 50 shows the average time delay for a node to receive enough packets to decode a FEC block for the four schemes (HANEF, HANEF/DER or DER-1, DER-2-200, and DER-2-300) when the average loss rate per link are set to 3%, 4% and 5% respectively. When there is no codecs (local recovery server) placed in the intermediate nodes of a distribution tree, this is equivalent to the centralized error recovery scheme. For this case, we can see that the time delay for HANEF and DER-1 is equal. When the loss rate equals 3%, the time delay associated with HANEF is larger than the DER-2-200 scheme and DER-2-300 scheme. When the loss rate equal 4% or 5%, the delay for HANEF is larger than DER-2-200 and smaller than DER-2-300 scheme. As the number of codecs (local recover server) increases, the time delay associated with HANEF decrease dramatically. With only 1 codec, the average delay for the HANEF scheme is better than DER-2-200 and DER-2-300 under all the loss rates. For example, when the loss rate equals 3%, with 1 codec (local recover server), the average delay for HANEF is 0.78s, while the delays for DER-1, DER-2-200 and DER-3-300 are 1.28s, 1.07s and 1.22s respectively. As the number of codecs (local recover server) increases, the improvement of HANEF over that of the other three schemes increases. When the number of codecs (local recover servers) equals to 5, with loss rate of 3%, the delay for HANEF is 0.23s, and is 0.95s, 0.77s and 0.81s for DER-1, DER-2-200 and DER-2-300 respectively.




(a)




(b)

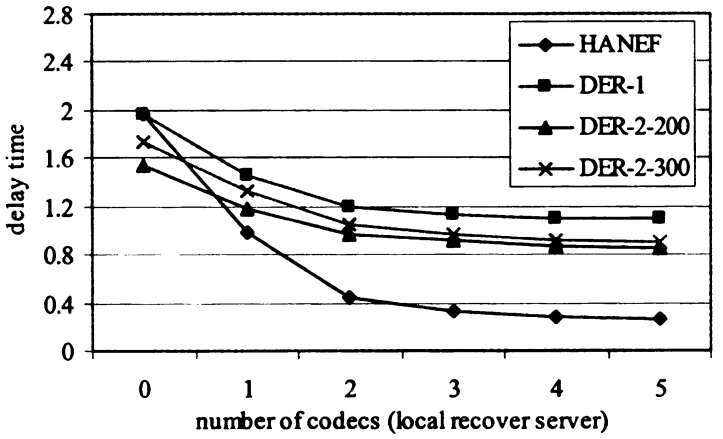



(c)

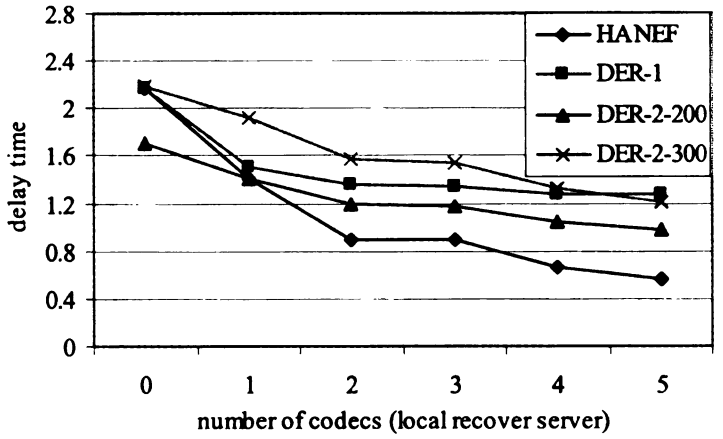
**Figure 50 time delay versus number of codecs ( local recover servers). Average over all the nodes. p=3%, (b) p=4%, (c) p=5%.**

Figure 51 shows the results for proxy-based overlay networks. In this case, only leaf nodes are considered as clients. Again, we can see that as the number of codecs (local recover server) increases, the time delay associated with HANEF is smaller than that of DER-1, DER-2-200 and DER-2-300.

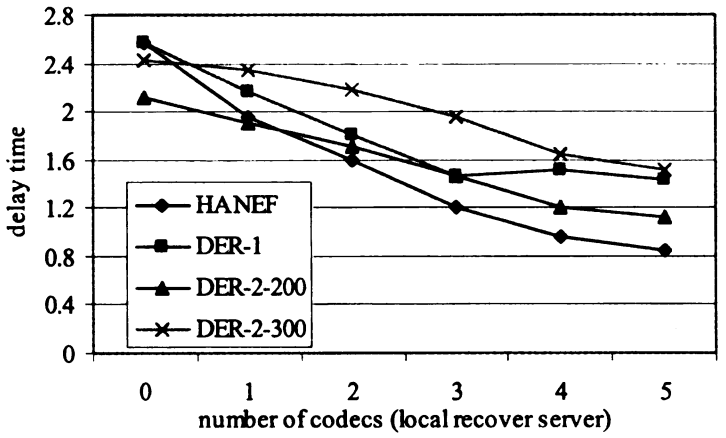
As we have mentioned at the beginning of this chapter. The heterogeneous nature of multicast clients means that to achieve reliability, the number of parity packets required for each client is different. To satisfy the exact number of parity packets requested by each node, the only way is to use unicast. The unicast scheme does not scale when the number of clients in a group becomes large; it is also not bandwidth efficient in a multicast environment. In all of the four reliable multicast schemes described above, the repair parity packets are transmitted using multicast by the source or local recover servers. This means that some nodes will receive more packets than they need. The unsolicited parity packets waste bandwidth. Figure 52 shows the average received packets per FEC block for each node to reliably decode a FEC block, the code used here is *RS*(255,223). In HANEF and DER-1, as the codecs (local recover servers) and the clients run the sender algorithm and receiver algorithm we have designed, the average received packets are almost equal for the two schemes and are both significantly lower than that of DER-2-200 and DER-2-300. For an example, with loss rate of 3%, when there is no codecs (local recover servers), the average number of received packets per FEC block is 250 for HANEF and DER-1, and is 319 for DER-2-200 and 299 for DER-2-300 respectively. When the number of codecs (local recover servers) increases to 5, the average received packets per FEC block is 239 for HANEF, 240 for DER-1, 279 for DER-2-200 and 267 for DER-2-300 respectively. We can also see that because DER-2-




(a)

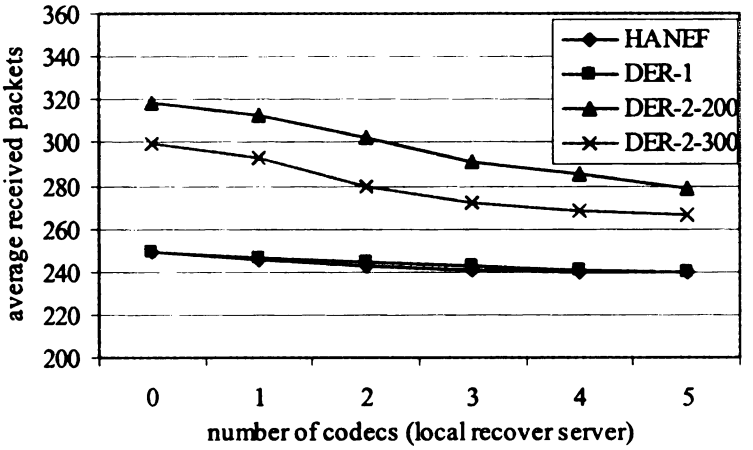



(b)

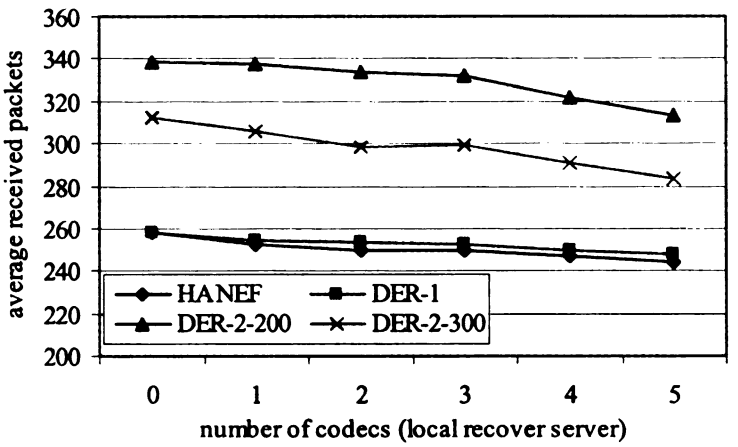



(c)

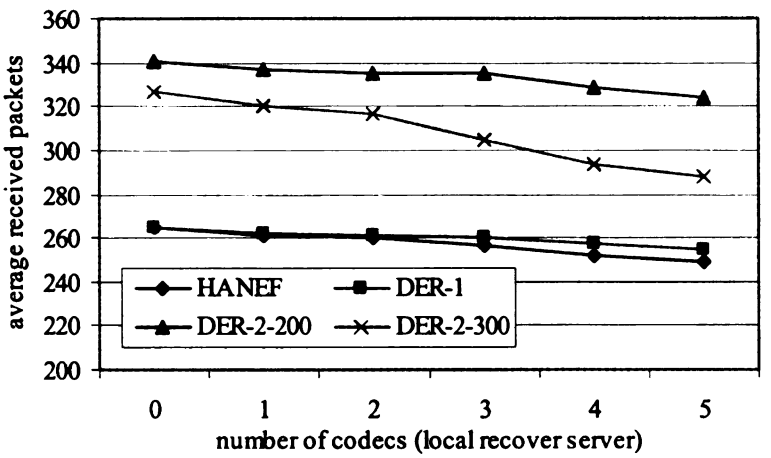
**Figure 51 time delay versus number of codecs (local recover servers), average over leaf nodes. p=3%, (b) p=4%, (c) p=5%.**




(a)




(b)

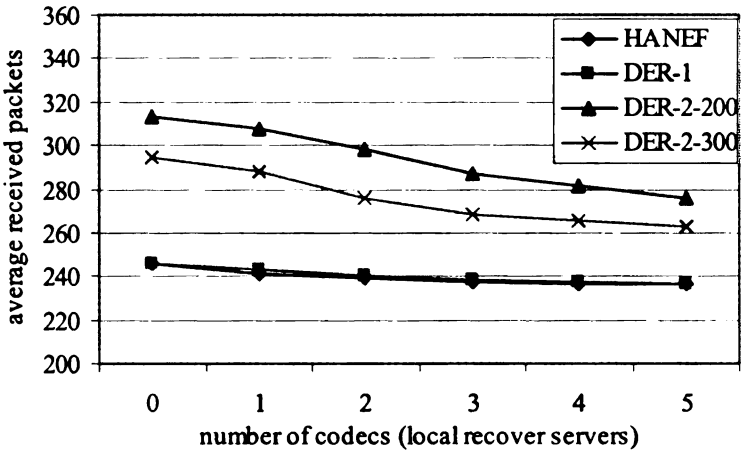



(c)

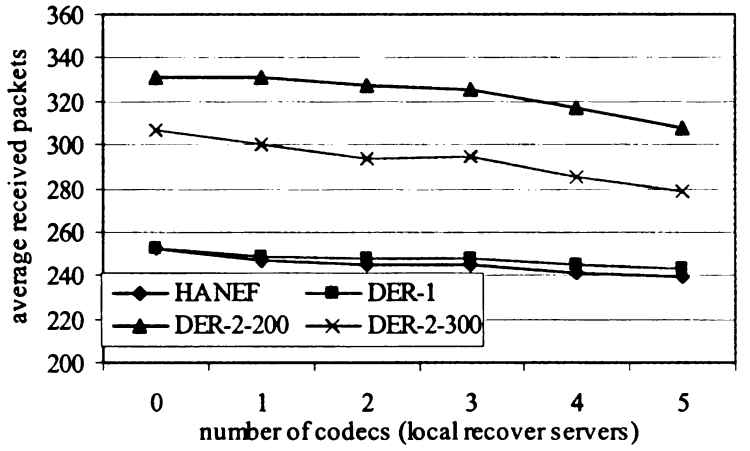
**Figure 52 Average received packets per FEC block, average over all nodes. (a) p=3%, (b) p=4%, (c) p=5%.**

300 waits longer than DER-2-200 to accumulate the feedback messages, the average received packets for DER-2-300 is smaller than that of DER-2-200. By comparing Figure 52 (a), (b) and (c), we can see that as the loss rate increases, the average number of received packets per FEC block increases.

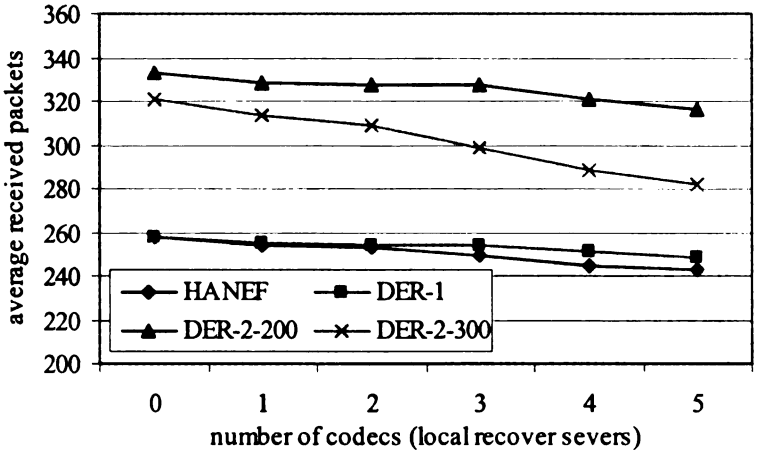
Figure 53 shows the average number of received packets when averaged over the leaf nodes.




(a)




(b)




(c)

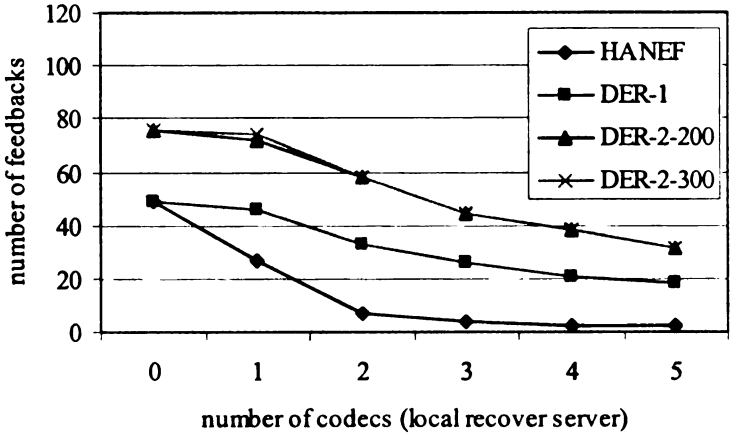
**Figure 53 average number of received packets per FEC block, average over leaf nodes. (a) p=3%, (b) p=4%, (c) p=5%.**

Processing feedback and retransmission consumes computation power. In HANEF and DER, codecs (local recover server) can share the burden of processing feedbacks and retransmissions for the source; also as feedback messages (or retransmissions) do not have to travel all the way back from each client to the source (or from source to each client), bandwidth overhead and delay penalty can be decreased, which has already been shown in our delay and bandwidth overhead analysis. Here we compare the feedback and retransmission overhead of HANEF and other DER schemes.

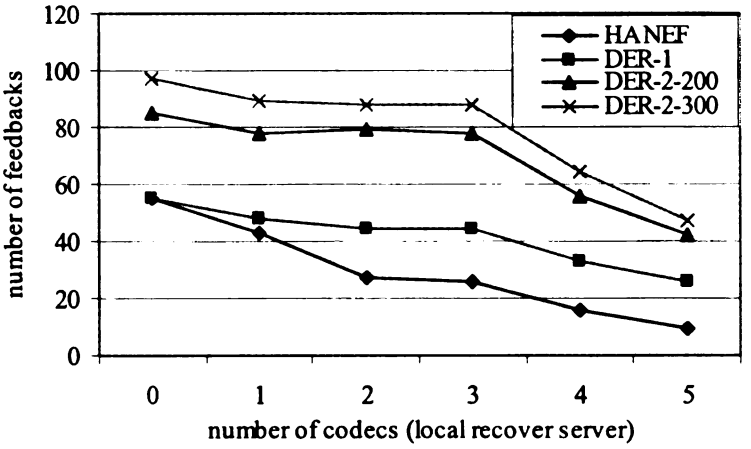
Figure 54 shows the maximum number of received feedback messages among the source and codecs (local recover servers) per FEC block for each node to receive enough packets to decode a FEC block. It can be seen that HANEF has the smallest number of received feedback messages among the four schemes. This is because in HANEF, as the number of codecs increases, the probability of receiving undecodable FEC blocks decreases dramatically. For example, when the number of codecs (local recover servers) is 2, the maximum number of received feedback message among the source and codecs (local recover servers) is 7, while this number is 33 for DER-1 and 58 for both DER-2-200 and DER-2-300. Even when there is no codecs (local recover servers), the number of received feedback message for HANEF and DER-1 is smaller than that of DER-2-200 and DER-2-300, this is because in HANEF and DER-1, retransmissions are sent immediately after a feedback message is received by a node, as retransmissions are multicasted to all the nodes, other nodes that have received the retransmissions and could decode the FEC block would suppress their feedback messages. The same reason can explain why the received feedback messages among the source and codecs (local recover servers) for DER-2-200 is smaller that that of DER-2-300; it is because DER-2-200 waits a shorter

time than DER-2-300 to send its retransmissions after it receives a feedback message. By comparing Figure 54 (a), (b) and (c), we can see that as the loss rate increases, the maximum number of received feedback messages among the source and codecs (local recover servers) increases, this is because retransmissions also suffer losses; hence, higher link loss rate means a node may need more (feedback and retransmission) rounds to receive a FEC block reliably.

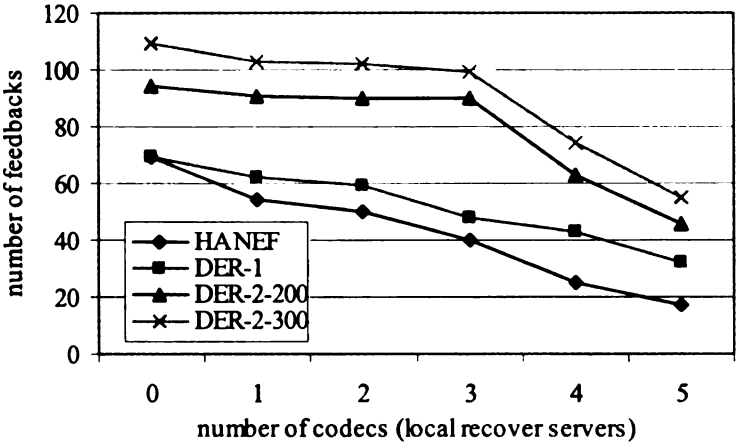
Figure 55 shows the maximum number of retransmissions among the source and codecs (local recover servers) per FEC block for a node to receive a FEC block reliably. Again, we can see that HANEF has the lowest number of retransmissions among the four schemes. For example, in Figure 55 (a), when the loss rate is 3% and there is no codecs (local recover servers), the maximum number of retransmissions for the source is 43 for HANEF and DER-1, 123 for DER-2-200, and 100 for DER-2-300, this is in accordance when we compare the performance of HANEF with centralized error recovery schemes. As the number of codecs increases, the number of retransmissions decreases dramatically for HANEF, while this number only decreases mildly for the other three schemes. For example, when the number of codecs (local recover servers) is 2, the maximum number of retransmission among the source and codecs (local recover servers) is 14 for HANEF, while this number is 41 for DER-1, 119 for DER-2-200, and 95 for DER-2-300, respectively. Comparing Figure 55 (a), (b) and (c), we can see that as the loss rate increases, the maximum number of retransmissions increases. Further, by comparing Figure 54 with Figure 55, we can see that the maximum number of received feedback messages among the source and local recover servers in DER-2-200 is smaller than that




(a)

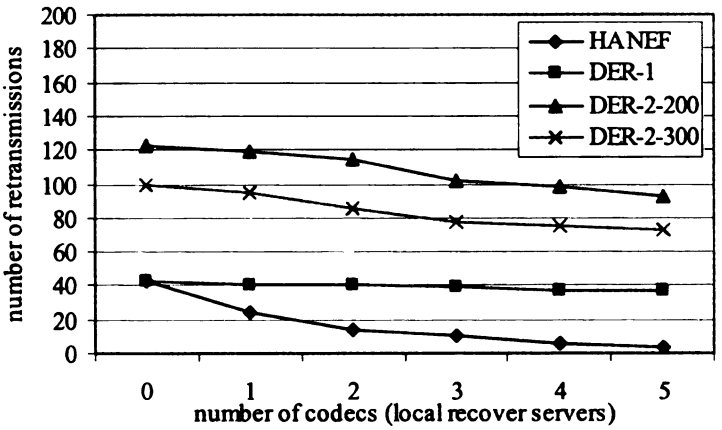



(b)

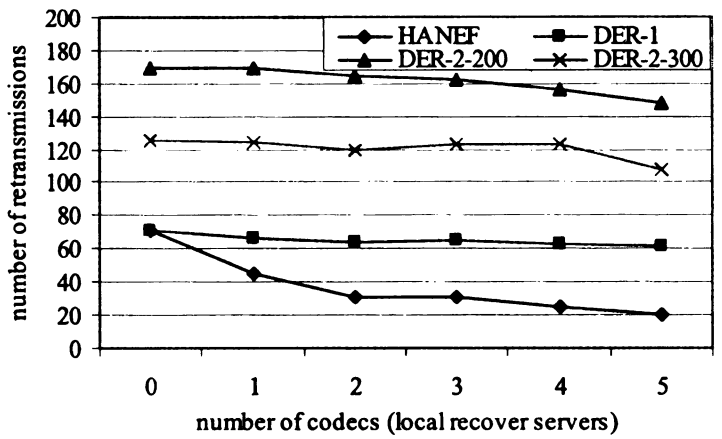



(c)

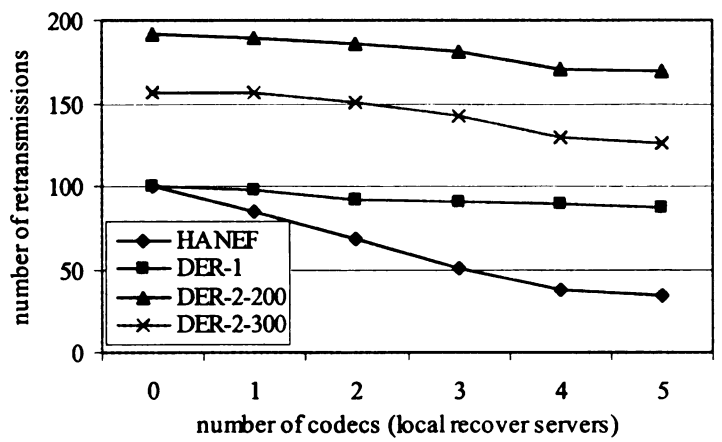
**Figure 54 maximum number of received feedbacks among source and codecs (local recover servers). (a) p=3%, (b) p=4%, (c) p=5%.**




(a)




(b)




(c)

**Figure 55 maximum number of retransmissions among source and codecs (local recover servers). (a) p=3%, (b) p=4%, (c) p=5%.**

of DER-2-300, while the maximum number of retransmission for DER-2-200 is bigger than that of DER-2-300.

## 6.5 Summary

In this chapter, we have designed and implemented a hybrid ARQ and NEF (HANEF) reliable multicast transport algorithm. We have compared the performance of HANEF with other centralized error recover schemes (CER) and distributed error recovery (DER) schemes. Through our simulations, we can see that HANEF outperforms CER and DER in terms of delay penalty and bandwidth overhead. In summary, our HANEF algorithm has the following advantages over other DER schemes: (1) our method does not require local clients to organize into different local multicast groups; also a client does not have to get the address of local repair server of that group. This is especially useful when multicast clients are dynamic. For example, if a local repair server leaves a group, then the clients in the group will have to organize into a new local multicast group or join other local multicast groups. This could produce significant management overhead. (2) feedback is sent immediately once losses are detected, and retransmission are sent immediately once a feedback is received, this makes our method more suitable for real time multicast applications. (3) The average time latency for a client to reliably receive a FEC block and bandwidth overhead is smaller in our method.

# Chapter 7 NEF for Multi-hop Wireless Networks

In multi-hop wireless networks, such as ad hoc networks, end users' nodes also work as routers. Hence, the NEF approach can be employed in such networks. In this chapter we focus on the use of NEF codecs for multicast applications in multi-hop wireless networks where nodes are static (i.e., not mobile). Codec placement in a mobile environment is a challenging problem, and it is left as part of our future work.

We first describe the wireless LAN protocols and the background of multicast applications in multi-hop wireless networks. We then show that NEF working with a modified MAC layer protocol can improve the decodable probability and goodput for multicast applications.

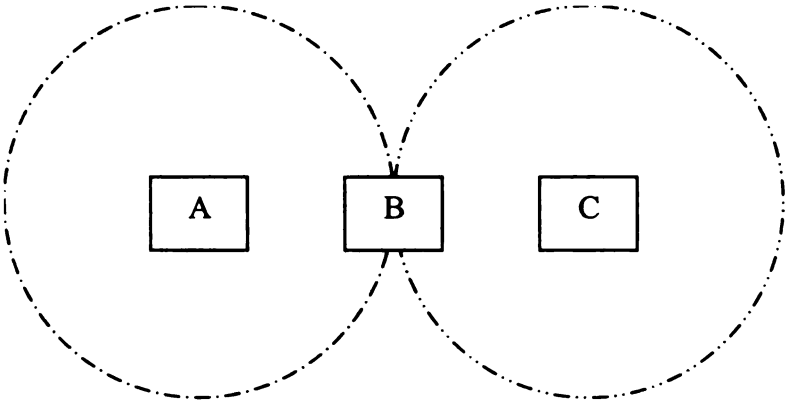
## 7.1 Introduction

### 7.1.1 Wireless LAN Protocols

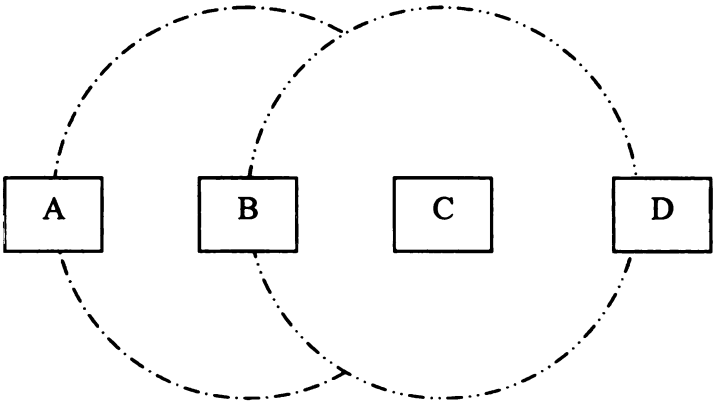
There two types of wireless local area networks, infrastructure based and ad hoc networks. In infrastructure based wireless networks, mobile terminals communicate with each other or end systems in the wired network through an access point (base station). In ad hoc networks, clients organize into networks spontaneously without infrastructure support; each end system is a user as well as a router. As a wireless channel has a broadcast characteristic and an end system or base station has certain transmission rang, using only carrier sense multiple access will cause the well known hidden terminal and exposed terminal problems. Figure 56 shows the phenomena of these two problems.

In Figure 56(a), terminal A and C want to send to terminal B. A starts transmitting, as C is out of rang of A, it can not sense that A is transmitting; C then starts transmitting, causing interference at B; this is the hidden terminal problem

In Figure 56(b), node C wants to send to node D and node B wants to send to node A. They can send simultaneously since A only receives signals from B and D only receives signals from C. B starts to send, C can not send because of carrier sensing, this is the exposed terminal problem




(a)




(b)

**Figure 56 (a) Hidden terminal problem. (b) Exposed terminal problem**

In order to solve the hidden and exposed terminal problems, current MAC layer schemes have added collision avoidance mechanism. For example, in 802.11, when a user needs to send data to a client, it first sends a Request to Send (RTS) message, if the client is ready to receive the message, it will send a Clear to Send (CTS) to the sender. The sender then transmits the data, the client, after receiving the data, will send an ACK message for acknowledgement.

As wireless channels are much less reliable than wired channels, 802.11 uses MAC layer retransmissions to add protection. Generally, for short packets (such as control message), the MAC layer will retry seven times; for long packets (data packets), the MAC layer will retry four times. This hop-by-hop retransmission mechanism greatly increases the reliability of a wireless channel.

### 7.1.2 Multicast in Multihop Wireless Networks

Multicast in wireless network has many applications, such as in sensor networks and military operations, etc. Generally, multicast routing protocols for wired network do not work for wireless networks due to limited resources (power, bandwidth), nodes movement, or channel characteristics (a link may be unidirectional). A lot of work has been done in designing multicast routing protocols for wireless networks; see for example [103]-[107]. Typically, these protocols can be classified as tree-based [104] or meshed based routing protocols [103]. In tree based protocols, each client has a single path to the source. The amount of bandwidth resources for building and maintaining a tree is often less then the mesh-based approach. However, a tree is more vulnerable to disruption due to nodes movement or clients leaving multicast sessions. A mesh based protocol is more

robust, yet it often incurs more bandwidth overhead; also a node needs to use cache to detect duplicate control and data packets due to routing loops.

Compared with other wireless multicast protocols, Multicast Ad hoc On-Demand Distance Vector routing (MAODV) incurs lower processing and memory overhead, as well as lower network utilization. Consequently, we use MAODV protocol in our simulations. In the following, we briefly introduce MAODV.

MAODV is an extension of AODV, and hence, its operation is similar to that of AODV. When a node needs to join a multicast group, a node broadcasts a *RREQ* message. When the *RREQ* message reaches a node already on the multicast tree, the node will unicast a *RREP* message back to the requesting node. Multiple nodes on the multicast tree may receive the *RREQ* message, so the node sending the request may receive multiple *RREP* replies, each indicating a route to the multicast tree. The node will select a route by sending a *MACT* message along that route. For nodes along a route that did not receive a *MACT* message, their states kept for the route will be expired. A node that sends a *RREQ* but does not receive any *RREP* within certain time will become the group leader itself.

Each multicast group has a group leader. The responsibility of the group leader is to initialize and maintain the group sequence number. The group sequence number is used to keep multicast routes update. For example, a node on a multicast tree receiving a *RREQ* request that has a bigger group sequence number than itself learn that its routes has expired and can not answer the request.

### 7.1.3 MAC Layer Multicast Protocols

In a wired network multicast application, a packet is replicated by the router (in IP multicast) or the peer (in overlay multicast) at a fork of the multicast distribution tree; a

copy of the packet is then transmitted on each branch of the distribution tree downstream. For wireless multicast applications, MAC layer multicast protocols may exploit the broadcast characteristic of the wireless channel to achieve bandwidth efficiency, as a packet may reach several receivers in a single transmission [94][95]. The 802.11 MAC layer multicast protocol works as described below:

When a node has a multicast packet ready for transmission, it first executes the contention phase. Once the node obtains access to the channel, it will proceed and multicast the data. There is no RTS/CTS handshake between the sender and receivers, and the receivers do not send ACKs. The source IP address of the multicast data is the address of the source of the group, the destination of the multicast packet is the group's multicast address, and the MAC layer address is set by the IP/MAC address mapping. Figure 57 shows an example of this. In Figure 57, node A needs to send a packet to node B and C; instead of transmitting the packet to B and C separately using unicast, A may choose to multicast (broadcast) the packet, B and C may receive the data from a single transmission. As there is no RTS/CTS handshaking, the probability of losses due to collision may increase. For example, in Figure 57, while D transmits to C, and A starts to multicast (broadcast) a packet to B and C, this may cause a collision at C.

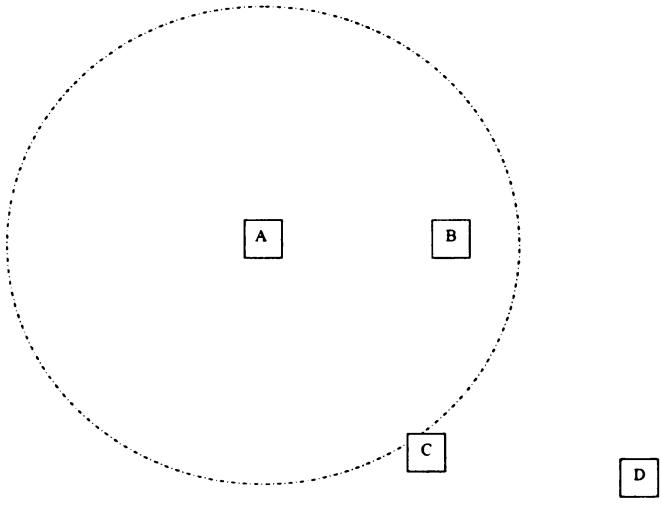



**Figure 57 mac layer broadcast**

A variety of MAC layer multicast protocols have been designed to decrease losses due to collision. In [96], the author tried to use RTS/CTS handshaking for 802.11 multicast/broadcast. When a node has a packet to broadcast, it first executes the contention phase. Once the node obtained the access to the channel, it broadcasts a RTS to its neighbors; if the node receives any CTS from its neighbors, it broadcasts the packet. Otherwise it backs off and executes another round of contention phase. The limitation for this approach is apparent; more than one neighbors sending CTS at the same time will cause collisions at the sender. Even if the sender receives a CTS, it does not mean that all of its neighbors are in the Clear to Send state. The work in [97] tried to solve the collision problem in [96] by adding a NAK phase; when a node sends a CTS and does not receive the data packet, the node will send a NAK message, this does not solve the problem completely because the NAK message may also suffer from collisions. In the *Broadcast Medium Window* (BMW) protocol proposed in [98], a node keeps a lists of its neighbors, it also has a send buffer and a receive buffer; when a node has a packet to broadcast, it first obtains access to the channel through a contention phase, then it sends out a RTS to

one of its neighbors with the sequence number of the packet in it. When the neighbor receives the RTS, it will check its receive buffer to see if it has received all the packets with the sequence number smaller than or equal to the sequence number of the upcoming packet; if all packets has been received, it will send a special CTS to notify the sender to suppress its data transmission; otherwise it will send a CTS with the sequence numbers of packets that has not been received in it; the sender, once receives the CTS, will send all the lost packets. This protocol requires lots of modification to the existing 802.11 protocol and requires at least a contention phase for each neighbor. The authors in [99] proposed a Batch Mode Multicast MAC (BMMM) protocol; the idea is that the sender uses its RTS message to instruct its receivers to send their CTS in order; it has also added a RAK (Request for ACK) message to coordinate the sequence of the receiver to send their ACK messages. The goal is to avoid CTS and ACK collision. The authors in [95] proposed a scheme for the sender to estimate the number of receivers in ready to receive state by measuring the power of a busy tone; this, however, needs accurate power regulation which is very difficult in practice.

In [100], a Leader-Driven Multicast protocol (LDM) was proposed. LDM was used in a wireless LAN to distribute video contents to a group of mobile hosts (MH). In LDM, a leader is selected among a group of MHs participated in a multicast session. The access point (AP) would send data stream to the leader using unicast; non-leader MHs would monitor the traffic from the AP to the leader, collect data and reconstruct the data stream. LDM needs a modified Networjk Interface Card (NIC) driver; and packet corruption correlation between the leader and non-leader MHs would impact the performance of LDM. The author also evaluated the performance of LDM when combined with FEC.

## 7.2 Using NEF for Multihop Wireless Multicast

As we have described in the above section, and as far as we know, there is no simple way to solve the reliable MAC layer multicast problem. For realtime multicast applications, where clients can tolerate limited packets loss and have strict requirements on time delay, we can use NEF to add more protection. In fact, NEF can work with a variety of MAC layer multicast protocols to increase the reliability of a multicast application. In this section we describe a scheme that combines NEF with the 802.11 MAC layer multicast/broadcast protocol and evaluate its performance.

We use MAODV as our multicast routing protocol. We downloaded a MAODV patch for ns2 from [106]. In the implementation of MAODV in [106], a multicast packet is forwarded to each neighbor using multiple unicast transmissions. In order for the MAODV [106] to work with the 802.11 MAC layer multicast protocol, we did some modifications to the MAODV [106] implementation. A node that receives a multicast packet will perform the following steps:

1. If the node is not on the multicast distribution tree, the packet is discarded.
2. If the node is on the multicast tree but the sender is not the next hop toward the source on the reversed distribution tree, the packet is discarded.
3. If the node is a group member, the packet is sent up the protocol stack to the correspondent application.
4. If the node has at least one next hop on the distribution tree, the packet is re-multicasted.

For the application layer Network-Embedded FEC (NEF) implementation, we use the distributed codec placement algorithm designed in Chapter 5. A node sends feedback

(using unicast) periodically to its parent. Any node that has at lease one child that on average can not decode a FEC block will become a codec.

The source will send $RS(n,k)$ FEC blocks to a group of users. Once a node on the multicast tree becomes a codec, if it receives $k$ or more packets it rebuilds the original FEC block and sends $n$ packets downstream; otherwise it just forwards what it has received.

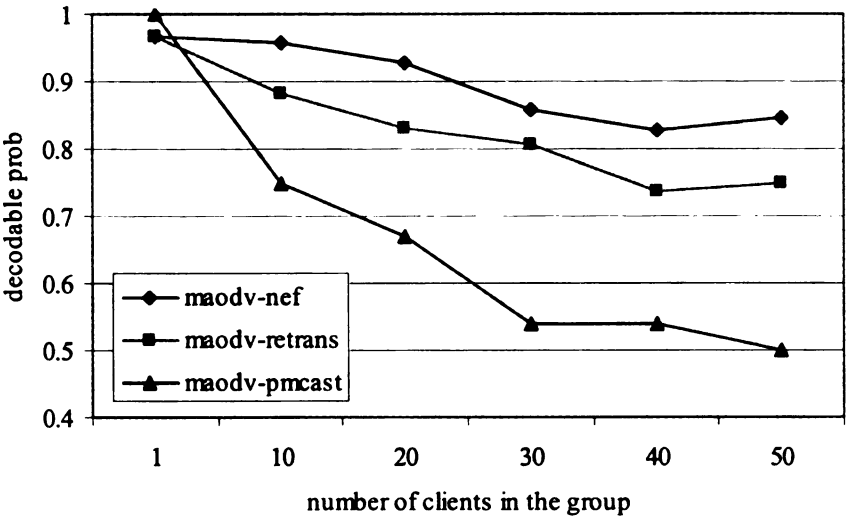
As we have mentioned above, in the implementation of MAODV in [106], a multicast packet is forwarded to each neighbor using multiple unicast transmissions. As unicast uses MAC layer retransmission to increase reliability, we call this scheme *maodv-retrans.* Another scheme is to use pure 802.11 multicast scheme at the MAC layer, we call this scheme *maodv-pmcast.* In our scheme, we implement NEF at the application layer and use pure 802.11 multicast at the MAC layer, we call this scheme *maodv-nef*. In the next section, we compare the performance of these three schemes using simulation.

## 7.3 Simulation Setup and Results

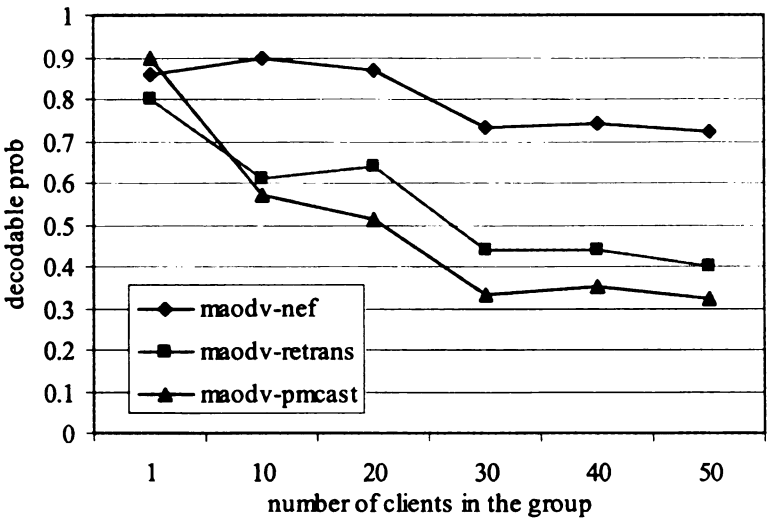
The topology we used in our simulation is 50 nodes placed randomly on a 500 x 500 square meter grid. We use the *Freespace* radio propagation model, the transmission range for each node is set to 100 meters. The data packet size is set to 512 bytes; the source rate is set to 32kbps, using $RS(25,20)$ code. The data rate of the channel is set to 5.5Mbps. The packet loss ratio is set to 3% and 5% per hop respectively and the packet correlation is set to 0.5. We run the simulation for 900 seconds.

Figure 58 shows the average decodable probability when the number of clients increases from 1 to 50. If there is only one client in the group (the unicast case), the decodable

probability for the three schemes are almost equal; As the number of clients in the group increases, *maodv-nef* outperforms the other two schemes; with *maodv-pmcast* has the lowest decodable probability. In Figure 58 (a), when all the nodes participate in the group, the decodable probability for *maodv-nef* is 0.85, and it is 0.75 and 0.5 for *maodv-retrans* and *maodv-pmcast* respectively. Figure 58 (b) shows the decodable probability when the loss rate per hop is 5%. By comparing Figure 58 (a) and (b), we can see that as the loss rate increases from 3% to 5%, the decodable probability improvement of *maodv-nef* over the other two schemes increases. In Figure 58 (b), when all the clients in the topology join the group, the average decodable probability for *maodv-nef* is 0.72, and is 0.54 and 0.32 for *maodv-retrans* and *maodv-pmcast* respectively. It can also be observed that generally, as the number of clients in the group increases, the decodable probabilities for all the three schemes decreases; the decrease is most significant for *maodv-pmcast*. For loss rate of 3%, when only one client in the group, the decodable probability of *maodv-pmcast* is 0.99, this number is decrease to 0.5 when all nodes in the topology join the group.



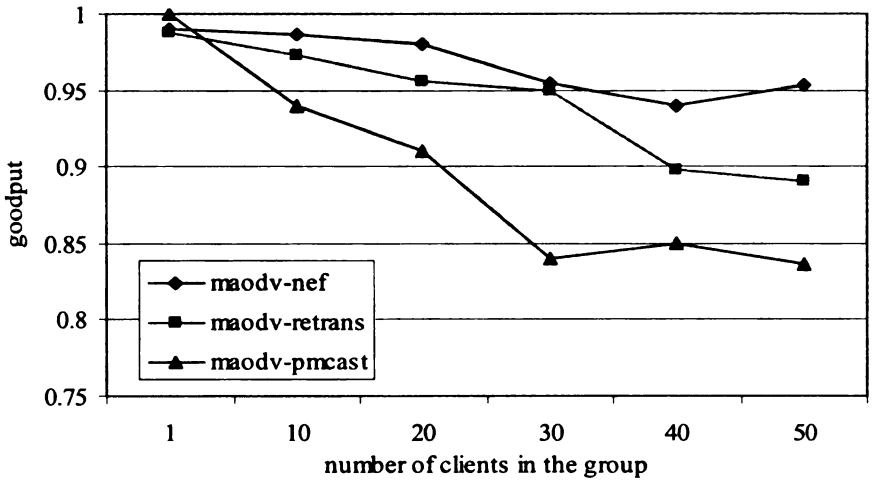

(a)



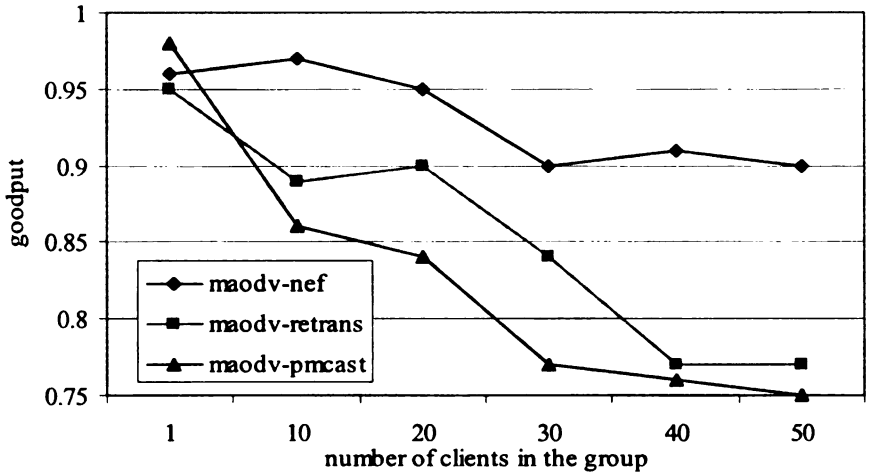

(b)

**Figure 58 Decodable probabilities versus number of clients in the group. (a) p=3% (b) p=5%**

We use the same definition of goodput as we have used in Chapter 3. For convenience, we repeat the definition here: the goodput for one node is the number of message packets the node receives divided by the number of message packets the source has sent. Average the individual goodput over all the clients in the group obtains the average goodput of the multicast session. Figure 59 shows the average goodput of a multicast session as the number of clients in the group increases from one to fifty. It can be seen that *maodv-pmcast* has the lowest goodput among the three schemes. The goodputs of *maodv-nef* and *maodv-retrans* are very close when the loss rate is 3% and the number of clients in the group is less than thirty(Figure 59 (a)); when the number of clients in the multicast group increase to more than forty, the goodput for *maodv-nef* outperforms that of *maodv-retrans* significantly. When the loss rate is 5%, the goodput of *maodv-nef* is better than that of *maodv-retrans* in most cases.




(a)




(b)

**Figure 59 Goodput versus the number of clients in the group. (a) p=3%, (b) p=5%.**

The throughput is the number of packets received by all the clients in the multicast session in a unit time. TABLE 6 and TABLE 7 show the throughputs when loss rate is set to 3% and 5% respectively. It can be observed that the throughput of *maodv-nef* is larger than that of *maodv-retrans* and *maodv-pmcast*, while the throughput of the later two schemes are very close.

TABLE 6 THROUGHPUTS FOR LOSS RATE OF 3%

| Num_clients | 1 | 10 | 20 | 30 | 40 | 50 |
|---|---|---|---|---|---|---|
| maodv-nef | 7 | 65 | 134 | 192 | 255 | 316 |
| maodv-retrans | 7 | 62 | 115 | 181 | 237 | 299 |
| maodv-pmcast | 7 | 60 | 123 | 176 | 239 | 290 |

TABLE 7 THROUGHPUTS FOR LOSS RATE OF 5%

| Num_clients | 1 | 10 | 20 | 30 | 40 | 50 |
|---|---|---|---|---|---|---|
| maodv-nef | 7 | 63 | 127 | 179 | 241 | 294 |
| maodv-retrans | 7 | 57 | 108 | 162 | 204 | 274 |
| maodv-pmcast | 7 | 55 | 114 | 162 | 217 | 264 |

## 7.4 summary

In this chapter, we studied the use of NEF for multicast applications in multi-hop wireless networks. We integrated NEF with 802.11 based MAC layer broadcast schemes, and have shown that it outperforms the MAC layer retransmission scheme and broadcast only scheme in terms of FEC decodable probability, message goodput and throughput. In the future, we will study combining NEF other MAC layer multicast/broadcast schemes to increase reliability of multicast application in ad hoc and multi-hop wireless networks.

# Chapter 8 Conclusions and Future Works

## 8.1 Summary

In this thesis, we present the using of network embedded FEC (NEF) for error corrections in p2p and proxy-based overlay multicast applications. Under NEF, FEC codecs are placed in the intermediate node of a network. These NEF codecs can detect and recover lost packets within FEC blocks at earlier stages before these blocks arrive at deeper intermediate nodes or at the final leaf nodes. This approach significantly reduces the probability of receiving undecodable FEC blocks among multicast end users. NEF can be designed to use independently in realtime multicast applications to obtain the desired level of playback quality; it can also be integrated with ARQ to obtain a one hundred percent reliability transmission.

In chapter 3, the probability analysis was performed on a Gilbert channel. In particular, a close function was obtained for a receiver to receive exactly $i$ packets when the sender send $n$ packets; this enable us to quantify the impact of packet loss correlation on the performance of FEC. Message goodput and decodable probability of NEF and end-to-end FEC on a cascaded Gilbert channel was then compared. It shows that over a wide rang of code rate, NEF outperforms end-to-end FEC. The last section of chapter 3 is the analysis of network-embedded FEC routes; this provides the foundation for the centralized code placement algorithm.

A centralized codec placement algorithm was designed and implemented in chapter 4. The algorithm assumes that the topology of the multicast tree and the loss rate for each link are known before hand and use a greedy approach to optimize the average decodable

probability. Analysis and simulation show that, under both random and burst packet losses, a relative few codecs placed in the intermediate nodes of the network can significantly improve the decodable probability and message goodput. In both p2p and proxy-based overlay network, NEF outperforms end-to-end FEC. It is also shown that in order to obtain the same level of reliability, NEF causes much less bandwidth overhead. The performance of the greedy algorithm was very close to that of the optimum.

In the centralized algorithm, we assumed that the topology of the multicast tree and the loss rate of each branch of the tree were known before hand. In a real world multicast session, nodes join and leave randomly and the structure of the multicast tree is changing constantly. Besides the structure change of the multicast tree, the available bandwidth for different users may also change over time; this may cause the loss rates on different branches of the tree change dynamically. Further, a centralized algorithm for placing and embedding FEC codecs may not scale well for a large network. In chapter 5, we present a distributed codec placement algorithm. The distributed algorithm use a simple feedback mechanism to collect information, a node would decide whether it should act as codec based on the average number of packets it received per FEC block and the number of children that can not decode FEC blocks. The distributed algorithm can cope with the dynamics of the network; its performance is close to the centralized codec placement algorithm.

NEF can be used independently for realtime applications which can tolerate certain level of losses; it can also be integrated with ARQ to achieve 100 percent reliable transmission. In Chapter 6, we designed and implemented a hybrid ARQ and NEF (HANEF) reliable multicast protocol. We compared the performance of HANEF with other CER and DER

reliable multicast protocols, simulation results shows that HANEF can decreased retransmission overhead dramatically and at the same time keep the delay penalty at the minimum.

In chapter 7, we studied the performance of NEF in multihop wireless multicast applications. We observed that NEF outperformed retransmission based and broadcast based multicast schemes in terms of goodput and throughput, especially when a large portion of nodes in the wireless network participated in the multicast session,

## 8.2 Future Works

Here, we highlight some of the key items that we plan to pursue under the NEF framework:

- When NEF is used in wireless networks, we have assumed that all nodes are static. In ad hoc networks, clients will move around at different speeds and directions. Codec placement in ad hoc *mobile* network is a challenging problem; one of our future studies is to explore the codec placement in ad hoc mobile networks.
- In chapter 7, we have only studied the performance of NEF when combined with a simple MAC layer multicast scheme. To fully understand the performance of NEF in wireless networks, combination of NEF with more efficient MAC layer multicast schemes need to be studied, especially with the LDM protocol proposed in [100].
- In NEF, intermediate codecs inject extra packets into the network; flow and congestion control, however, is not considered in this proposal. Another dimension of the future work is to integrate flow and congestion control into the

NEF framework. It has been shown that hop-by-hop congestion control is stable and responds more promptly than end-to-end congestion control scheme, both in wired and wireless networks [101][102]. Hop-by-hop congestion control requires participation of intermediate node. The integration of congestion control and NEF in a single scheme represents another future work effort.

- We have focused on Reed-Solomon (RS) codes for erasure (packet-loss) channels throughout this thesis. Extensions of the NEF framework under more recent channel coding schemes, such as Low-Density-Parity-Check (LDPC) codes, and other channels, such as error channels, represent another key aspects of future directions for this work.

# Bibliography

[1] S. Deering, "Multicast routing in a datagram internetwork." Ph.D Dissertation, 1991.

[2] H. Eriksson, "The Multicast backbone." *Communications of the ACM*, vol. 8, pp. 54-60, 1994.

[3] D. Waitzman, C. Partridge, and S. Deering, "Distance vector multicast routing protocol (DVMRP)." Internet Engineering Task Force (IETF) , RFC 1075, November 1988.

[4] J. Moy, "Multicast extension to OSPF." Internet Engineering Task Force (IETF), RFC 1584, March 1994.

[5] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, A. Helmy, D. Meyer, and L. Wei, "Protocol independent multicast version 2 dense mode specification." Internet Engineering Task Force (IETF), draft-ietf-pim-v2-dm-*.txt, November 1998.

[6] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei, "Protocol independent multicast sparse-mode (PIM-SM): Protocol specification." Internet Engineering Task Force (IETF), RFC 2362, June 1998.

[7] T. Ballardie, P. Francis, and J. Crowcroft, "Core based trees (CBT): An architecture for scalable multicast routing," in *ACM Sigcomm*, (San Francisco, California, USA), pp. 85-95, September 1995.

[8] Y. Rekhter and T. Li, "A border gateway protocol 4 (BGP-4)." Internet Engineering Task Force (IETF), RFC 1773, March 1995.

[9] T. Bates, R. Chandra, D. Katz, and Y. Rekhter, "Multiprotocol extensions for BGP-4." Internet Engineering Task Force (IETF), RFC 2283, February 1998.

[10] D. Farinacci, Y. Rekhter, P. Lothberg, H. Kilmer, and J. Hall, " Multicast source discovery protocol (MSDP)." Internet Engineering Task Force (IETF), draft-farinacci-msdp-*.txt, June 1998.

[11] S. Kumar, P Radoslavov, D. Thaler, C. Alaettinoglu, D. Estrin, and M. Handley, "The MASC/BGMP architecture for inter-domain multicast routing," in *ACM Sigcomm*,(Vancouver, CANADA), August 1998.

[12] M. Handley, D Thaler, and D. Estrin, "The internet multicast address allocation architecture." Internet Engineering Task Force (IETF), draft-ietf-malloc-arch-*.txt, December 1997.

[13] M. Handley, "Multicast address allocation protocol(AAP)." Internet Engineering Task Force (IETF), draft-ietf-malloc-aap-*.txt, August 1998.

[14] H. Holbrook and D. Cheriton,"IP multicast channels:EXPRESS support for large-scale single-source applications." In *ACM Sigcomm*,(Cambridge, Massachusetts, USA), August 1999.

[15] R. Perlman, C.-Y. Lee, J. Crowcroft, Z. Wang, C. Diot, J. Thoo, and M. Green,"Simple multicast: A design for simple, low-overhead multicast." Internet Engineering Task Force (IETF),draft-perlman-simple-multicast-*.txt, Feburuary 1999.

[16] S. McCanne, V. Jacobson, and M. Vetterli,"Receiver-Driven Layered Multicast," in *Proc. of ACM SIGCOMM '96*, August 1996.

[17] J. Liu, B. Li, and Y. Zhang, "A Hybrid Adaption Protocol for TCP-friendly Layered Multicast and Its Optimal Rate Allocation," in *Proc. of IEEE INFOCOM '02*, June 2002.

[18] Gu-In Kwon, John W. Byers, "Smooth Multirate Mulitcast Congestion Control", In *IEEE INFOCOM*, 2003.

[19] L. Rizzo,"pgmcc: A TCP-friendly single-rate multicast congestion control scheme," in *Proc. of ACM SIGCOMM '00*, 2000.

[20] A. Legout and E. Biersack, "PLM: Fast convergence for cumulative layered multicast transmission schemes," in *Proc. of ACM SIGMETRICS*, 2000.

[21] L. Vicicano, L. Rizzo, and J. Crowcroft, "TCP-like Congestion Control for Layered Multicast Data Transfer," in *Proc. of IEEE INFOCOM '98*, April 1998.

[22] J. Widmer and M. Handley, "Extending equation-based congestion control to multicast applications," in *Proc. of ACM SIGCOMM '01*, 2001.

[23] J. Byers, G. Horn, M. Luby, M. Mitzenmacher, and W. Shaver, "FLID-DL:Congestion Control for Layered Multicast," *IEEE J-SAC Special Issue on Network Support for Layered Multicast Communication*, vol. 20(8),pp.1558-1570, Oct. 2002.

[24] J. Byers, M. Luby, and M. Mitzenmacher, "Fine-Grained Layered Multicast," in *Proc. of IEEE INFOCOM '01*, April 2001.

[25] J. Byers and G. Kwon, "STAIR:Practical AIMD Multirate Multicast Congestion Control," *in Proc. of NGC '01*, 2001.

[26] M. Luby, V. Goyal, S. Skaria, and G. Horn, "Wave and Equation Based Rate Control Using Multicast Round Trip Time," in *Proc. of ACM SIGCOMM '02*, 2002.

[27] B. J. Vichers, C. Albuquerque, and T. Suda, "Source-adaptive multilayered multicast algorithms for real-time video distribution," In *IEEE/ACM Transactions on Networking*, vol. 8, no. 6, pp. 720-733, 2000.

[28] Qiang Liu, Jenq-Neng Hwang, "A New Congestion Control Algorithm for Layered Multicast in Heterogeneous Multimedia Distribution." In *ICME 2003*.

[29] J. P. Macker, "Reliable multicast transport and integrated erasure-based forward error correction", *Proceedings IEEE MILCOM*, November 1997, p973-7 vol.2.

[30] S. Floyd, V. Jacobson, C. Liu, S. McCanne, L. Zhang, "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing", *IEEE/ACM Transactions on Networking*, Dec. 1997 vol.5 no.6 p. 784-803

[31] J. Lin and S. Paul "RMTP: A Reliable Multicast Transport Protocol", In *Proc. of IEEE INFOCOM'96*. P. 14114-24 vol.3

[32] B. Levine, D. Lavo, and J.J Garcia-Luna-Aceves, "The Case for Concurrent Reliable Multicast Using Shared Ack Trees", In *Proc. ACM Multimedia'96 Conference*.

[33] J.J Metzner, "An improved broadcast retransmission protocol", *IEEE Transactions on Communications*, June 1984. vol.COM-32, no.6, p. 679-83

[34] S.Lin, D.J. Costello and M.J.Miller, "Automatic-repeat-request error-control schemes", *IEEE communication magazine*, 22(12):5-17, 1984

[35] Lui Rizzo and Lorenzo Vicisano, "A Reliable Multicast data Distribution Protocol based on software FEC techniques (RMDP)", *4th IEEE Workshop on High-Performance Communication Systems(HPCS'97)*(1997) p.134-148.

[36] J. Byers, M. Luby, M. Mitzenmacher and A. Rege, "A Digital Fountain Approach to Relaible Distribution of Bulk Data." *Proc. ACM SIGCOMM'98*, 28(4):56-67, Oct.1998.

[37] M. Yajnik, J. Kurose, D. Towsley, "Packet Loss Correlation in the Mbone Multicast Network," *IEEE Global Internet Conference*, London, November 1996.

[38] D. Rubenstein, J. Kurose, and D. Towsley, "Real-time Reliable Multicast Using Proactive Forward Error Correction," TR 98-19, CS Dept, Univ. of Massachusetts, Amherst, March 1998.

[39] P. Bhagwat, P. Mishra, and S. Tripathi, "Effect of Topology on Performance of Reliable Multicast Communication." *Proc. INFOCOM'94*, 2:602-609, June 1994.

[40] Marc Mosko, J. J. Garcia-Luna-Aceves, "An Analysis of Packet Loss Correlation in FEC-Enhanced Multicast Trees." *ICNP 2000*: 151- 161.

[41] J. Nonnenmacher, E. Biersack, and D. Towsley, "Parity-Based Loss Recovery for Reliable Multicast Transmission." *IEEE Trans. On Networking*, pages 349-361, Aug. 1998.

[42] http://www.4i2i.com/reed_solomon_codes.htm

[43] J. Nonnenmacher, L. Martin, J. Matthias, E. Biersack, G. Carle, "How bad is Reliable Multicast without Local Recovery?" *Proc. IEEE INFOCOM '98*, volume 3, pp. 972-9, San Francisco, CA, March 1998.

[44] Luigi Rizzo, "Effective Erasure Codes for Reliable Computer Communication Protocols", ACM Computer Communication Review, Vol.27,n.2, Apr.97 pp.24-36.

[45] Yatin Chawathe, Steven Mcanne, and Eric Brewer, "RMX: Reliable multicast for heterogenous networks." In *Proc. IEEE INFOCOM*, March 2000.

[46] V. Stankovic, R. Hamzaoui, and Z. Xiong," Packet Loss Protection of Embedded Data with Fast Local Search." In *Proc. Int'l Conf. Image Processing*. IEEE, September 2002.

[47] Yitzhak Birk, Diego Crupnicoff, "A Multicast Transmission Schedule for Scalable Multi-rate Distribution of Bulk Data Using Non-Scalable Erasure-Correction Codes." In *IEEE INFOCOM*, 2003.

[48] Sassan Pejhan, Mischa Schwartz and Dimitris Anastassiou, "Error control using retransmission schemes in multicast transport protocols for real-time media," *IEEE/ACM Transactions on Networking*, vol. 4(3), pp.413-427, June 1996.

[49] Y.-H. Chu, S.G. Rao, and H. Zhang, "A Case for End System Multicast." In *Proc. ACM SIGMETRICS*, June 2000.

[50] S. Banerjee, Bobby Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast." In *ACM SIGCOMM*, Pittsbrugh, PA, 2002.

[51] D. Andersen, H. Balakrishnan, M. Kaashoek and R. Morris, "Resilient overlay networks." In *ACM Symposium on Operating Systems Principles(SOSP)*, 2001.

[52] Dimitrios Pendarakis,Sherlia Shi,Dinesh Verma, and Marcel Waldvogel, "ALMI: An Application Level Multicast Infrastructure." *Proc. of the 3rd USNIX Symposium on Internet Technologies and Systems,* March 2001.

[53] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks." In *18th ACM SOSP*, Oct.2001.

[54] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Application-level Multicast using content-addressable networks." In *Proc. of NGC*, Nov. 2001.

[55] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "Scribe: A large scale and decentralized application-level multicast infrastructure." In *IEEE JSAC*, vol. 20, no. 8, October 2002

[56] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications." In *Proc. of ACM SIGCOMM*, San Diego, California, August 2001.

[57] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems." In *Proc. of Middleware*, Nov. 2001.

[58] B. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An Infrastruture for fault-resilient wide-area location and routing." U.C. Berkeley, Tech. Rep. UCB, CSD-01-1141, April 2001.

[59] M. Castro, M. B. Jones, A-M. Kermarrec, A. Rowstron, M. Theimer,H. Wang and A. Wolman, "An Evaluation of Scalable Application-level Multicast Built Using Peer-to-peer overlays", Infocom 2003, San Francisco, CA, April, 2003.

[60] http://www.gnutella.com/

[61] http://www.napster.com/

[62] http://www.akamai.com/

[63] Mark S. Daskin, "Network and Discrete Location: Models, Algorithms, and Application." 1995, John Wiley & Sons, Inc. New York.

[64] Gary, M. R. and D. S. Johnson, "Computers and Intractability: A Guide to the NP –Completeness." 1979, W. H. Freeman and Co. , New York.

[65] Dorit S. Hochbaum, editor, "Approximation Algorithms for NP-hard Problems." PWS publishing company, Boston, 1995.

[66] A.J.Goldman,"Optimal Center Location in Simple Networks", *Transportation Science, 5*(1971), 212-221

[67] Kariv, O. and S. L. Hakimi, "An Algorithm Approach to Network Location Problems. II: The *p*-Medians." *SIAM Journal on Applied Mathematics*, 37, 1979, p.539-60

[68] S. L. Hakimi,"Optimal locations of switching centers and medians of a graph." *Operations Research* 12 (1964) 450-459.

[69] J. E. Beasley, "A note on solving large p-median problems." *European J. Oper. Res.*, 21:270-273, 1985.

[70] Mark S. D askin, "A new approach to solving the vertex p-center problem to optimality: Algorithm and computational results." *Communications of the Operations Research Society of Japan*, 45:9:428-436, 2000.

[71] B. Li, M. J. Golin, G. F. italiano, X. Deng, and K. Sohraby, "On the optimal placement of web proxies in the internet." In *IEEE INFOCOM'99*, Mar. 1999, pp.1282-1290.

[72] A. Tamir, "An $O(pn^2)$ algorithm for the $p$-median and related problems on tree graghs." *Oper. Res. Lett.*, vol. 19, pp 59-64, 1996.

[73] V. Chvatal, "A greedy heuristic for the set-covering proble." *Mathematics of Operations Research*, 4(3):233-235, 1979.

[74] M. Fisher and P. Kedia, "Optimal solutions of set covering / partitioning problems using dual heuristics." *Management Science*, 36(6):674-688, 1990.

[75] F. Harche and G. L. Thompson, "The column subtraction algorithm: an exact method for solving weighted set covering, packing and partitioning problems." *Computers & Operations Research*, 21(6):689-705, 1994.

[76] W. Jian "QoS measurement and Management for Internet real-time multimedia services", Ph.D thesis, Department of Computer Science, Columbia University, 2003.

[77] H. Sanneck and G. Carle, "A framework model for packet loss metrics based on loss runlengths", *Proceedings of the SPIE/ACM SIGMM Multimedia Computing and Networking Conference 2000*. San Jose, CA, USA, pp.177-187.

[78] J. R. Yee and E. J. Weldon, "Evaluation of the performance of error-correcting codes on a Gilbert channel," *IEEE Trans. Commun.*, vol. 43, pp. 2316-2323, Aug. 1995.

[79] R. A. Howard, *Dynamic Probabilistic Systems*. New York: Wiley, 1971.

[80] M, Zorzi, R. Rao, "Lateness Probability of a Retransmission Scheme for Error Control on a Two-state Markov Channel", *IEEE Transactions on communications, Vol. 47, NO. 10*, pp.1537-1548.

[81] E. W. Zegura, "*GT-ITM: Georgia tech internetwork topology models (software),*" http://www.cc.gatech.edu /fac/Ellen.Zegura/gt-itm/gt-itm.tar.gz, 1996.

[82] http://www.isi.edu/nsnam/ns/.

[83] Cristos Papadopoulos, Gurudatta M. Parulkar, "Retransmission-Based Error Control for Continuous Media Applications," Proc. 6th Intl. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV).

[84] Sassan Pejhan, Mischa Schwartz, Dimitris Anastassiou, "Error Control Using Retransmission Schemes in Multicast Transport Protocols for Real-Time Media," IEEE/ACM *Transactions on Networking*, 4(3) : 413-427, 1996.

[85] R. Kermode, "Scoped Hybrid Automatic Repeat Request with Forward Error Correction (SHARQFEC)," Proceedings of ACM SIGCOMM'98, Vancouver, CA, September 1998.

[86] S. Armstrong, A. Freier, and K. Marzullo, "Multicast Transport Protocol," RFC 1301, Februray 1992.

[87] J. Crowcroft and K. Paliwoda, "A Multicast Transport Protocol," ACM Computer Communications Review, vol. 18, pp. 247-256, August 1988.

[88] T. Strayer, B. Dempsey, and A. Weaver, "Xpress Transfer Protocol Version 3.6," Technical report, XTP Forum, Santa Barbara, CA, 1992.

[89] R. Braudes and S. Zabele, "Requirements for Multicast Protocols," RFC 1458, TASC, Reading, MA, May 1993.

[90] H. Radha and M. Wu, "Overlay and Peer-to-Peer Multimedia Multicast with Network-Embedded FEC," IEEE International Conference on Image Processing (ICIP), September 2004.

[91] M. Wu, S. Karande, and H. Radha, "Network-Embedded Channel Coding for Optimum Throughput of Multicast Packet Video," Signal Processing: Image Communication, in press.

[92] M. Wu and H. Radha, "Network-Embedded FEC (NEF) Performance for Multi-Hop Wireless Channels with Memory," IEEE International Conference on Communications (ICC), May 2005.

[93] M. Wu and H. Radha, "Network Embedded FEC for Overlay and P2P Multicast over Channels with Memory," Conference on Information Sciences and Systems (CISS), March 2005.

[94] Prasanna Chaporkar, Saswati Sarkar, "Minimizing Delay in Loss-Tolerant MAC Layer Multicast," Third international Symposium on Modeling and Optimization in Mobile, ad hoc and wireless networks (WiOpt'05) April, 2005, Riva Del Garda, Trentino, Italy.

[95] Prasanna Chaporkar, Anita Bhat, Saswati Sarkar "An Adaptive Strategy for Maximizing Throughput in MAC layer Wireless Multicast," Mobihoc' 04, May 24-26, Roppongi, Japan.

[96] K. Tang and M. Gerla, "MAC layer Broadcast Support in 802.11 Wireless Networks," Proc. IEEE MILCOM 2000, pp.544-548, Oct. 2000.

[97] K. Tang and M. Gerla, "Random Access MAC for Efficient Broadcast Support," Proc. IEEE WCNC 2000, pp.454-459, Sep. 2000.

[98] K. Tang and M. Gerla, "MAC Reliable Broadcast in Ad Hoc Networks," Proc. IEEE MILCOM 2001, pp.1008-1113, Oct. 2001.

[99] Min-Te Sun, Lifei Huang, Anish Arora, Ten-Hwang Lai, "Reliable MAC Layer Multicast in IEEE 802.11 Wireless Networks," 2002 International Conference on Parallel Processing (ICPP'02).

[100] Peng Ge, "Interactive Video Multicast in Wireless LANs," Ph.D dissertation, Dep. of Computer Science and Engineering, Michigan State University, 2004.

[101] P. P. Mishra, H. Kanakia, "A hop-by-hop rate-based congestion control scheme", Proc. ACM SIGCOMM '92, Baltimore, MD, Aug. 1992, pp. 112-123.

[102] Yung Yi and Sanjay Shakkottai, "Hop-by-hop Congestion Control over a Wireless Multi-hop Network," In *IEEE INFOCOM*, 2004

[103] Sung-Ju Lee, William Su, Mario Gerla, "On-Demand Multicast Routing Protocol (ODMRP) for Ad Hoc Networks," Proceedings of IEEE ICNP'98

[104] Elizabeth M. Royer, Charles E. Perkins, "Multicast Ad hoc On-Demand Distance Vector routing (MAODV) ", Internet Draft, draft-ietf-manet-maodv-00.txt

[105] C.W. Wu and Y.C. Tay, "Ad hoc Multicast Routing protocol utilizing Increasing id-numbers (AMRIS)," In Proceedings of IEEE MILCOM'99, Atlantic City, NJ, Nov. 1999

[106] http://mobicom.cs.uni-dortmund.de/MobileP2P/software/ns-mcast.html

[107] Mingyan Liu, Rajesh R. Talpade, Anthony Mcauley, Ethendranath Bommaiah, "Ad hoc Multicast Routing Protocol (AMroute)," UMD Tech Report 99-8