8-2002

# Scalable and Reliable File Transfer for Clusters Using Multicast.

Hardik Dikpal Shukla
*East Tennessee State University*

# Scalable and Reliable File Transfer for Clusters Using Multicast

---

A thesis

presented to

the faculty of the Department of Computer Science

East Tennessee State University

---

In partial fulfillments

of the requirements for the degree

Masters of Science in Computer Science

---

by

Hardik D. Shukla

August 2002

---

Keywords: multicast, file transfer, scalable, reliable, cluster

# ABSTRACT

Scalable and Reliable File Transfer For Clusters Using

Multicast

by

Hardik D. Shukla

A *cluster* is a group of computing resources that are connected by a single computer network and are managed as a single system. Clusters potentially have three key advantages over workstations operated in isolation—*fault tolerance, load balancing and support for distributed computing.*

Information sharing among the cluster's resources affects all phases of cluster administration. The thesis describes a new tool for distributing files within clusters. This tool, the Scalable and Reliable File Transfer Tool (SRFTT), uses Forward Error Correction (FEC) and multiple multicast channels to achieve and efficient reliable file transfer, relative to heterogeneous clusters. SRFTT achieves scalability by avoiding feedback from the receivers. Tests show that**,** for large files, retransmitting recovery information on multiple multicast channels gives significant performance gains when compared to a single retransmission channel

## DEDICATION

To the most beautiful person I have ever known, my wife Naisargi, who constantly encouraged me to give my best effort,

My parents, whom I will always be indebted to

And

My mentor, Dr. Phillip Pfeiffer, with his reputed spirit for academic excellence in conjunction with his inexhaustible energy, who, despite many 3 AM sessions, provided an unwavering source of inspiration.

## ACKNOWLEDGEMENTS

The author wishes to express his sincere thanks to ETSU committee members; Dr. Phillip Pfeiffer, Dr. Martin Barrett and David Tarnoff.

The author sincerely appreciates the time and effort by Dr. Stephen Scott of Oak Ridge National Labs.

CONTENTS

LIST OF FIGURES

Number Page

# CHAPTER 1

# INTRODUCTION

## 1.1 Cluster Computing And Its Advantages

A *cluster* consists of a group of computing resources that are connected by a single computer network [Buyya99]. These resources can include servers, workstations, printers, and other networked systems. What distinguishes a cluster from a standard computer network is singleness of purpose. A cluster's resources are managed as a single system, using special-purpose software that coordinates the cluster's resources and creates an illusion of a single system.

Clusters, potentially, have three key advantages over workstations operated in isolation. The first is *fault tolerance*: the ability to recover from one or multiple system faults [Tanen95]. A computation on an isolated workstation can't finish if that workstation fails. If a cluster's resource fails during a computation, the rest of the cluster's resources assume the failed resource's workload. The cluster as a whole is available for more time to do useful work.

A second advantage is *load balancing*: the ability to distribute work evenly and fairly among the cluster's machines [Tanen95]. In a standard computer network one workstation can be bogged down while another sits idle. In a cluster with load balancing the cluster management software redistributes work from overloaded machines to other more lightly loaded resources. This decreases the computation time and increases the system's efficiency.

A third advantage is *support for distributed computation*: the ability to complete computations on two or more processors [Tanen95]. Distributed computing uses multiple, cooperating systems to solve specialized problems like matrix multiplication and climate monitoring by spanning computations across multiple PCs. MPI [MPIRef], PVM [Scott02], and C3 [Scott02] are examples of software that facilitate distributed computing on a cluster.

Some of the potential advantages of cluster computing are similar to those obtained from special-purpose systems designed for robustness or performance. Clusters, however, offers two additional potential advantages to fault-tolerant computing and supercomputing: cost and scalability.

Fault-tolerant systems, historically, have been custom devices in special configurations [Vyto93]. The same is true for supercomputers, which, historically, have cost far more than the commodity computers of their era [HenPa96]. Cluster computers, however, can achieve the benefits of supercomputing using off-the-shelf hardware [BelGrey02]. Because of significant improvements in the cost-performance of commodity computers and networks, cluster computing has a potential to offer the benefits of special-purpose systems at significantly low costs.

As for scalability, reconfiguring a special-purpose system to provide greater or lesser degrees of service is typically difficult to do. Clusters, on the other hand, are readily scalable. The size of a cluster—and, accordingly, its performance and reliability—can typically be increased or decreased by simply adding or subtracting off-the-shelf hardware.

## 1.2 Cluster Computing And Its Challenges

Cluster computing, however, is not simple to do well. Getting applications to run efficiently in distributed environments has proven to be a surprisingly difficult task [Tanen95]. The same is true for cluster administration [Scott02]. The applications development problem includes a variety of subtasks, such as program setup, distribution of tasks across a cluster, and managing communication among an application's constituent tasks. The cluster administration problem also includes multiple subtasks, like managing cluster membership, managing the access to a cluster, setting and monitoring the computing on an individual resource and configuring any cluster's resource automatically. Inefficient strategies for cluster operation make that cluster less attractive to use due to slower operation of programs and administrative hassles.

The focus of the work addressed in this thesis, information distribution, is an underlying concern in all phases of cluster administration. All of the above-mentioned cluster administration tasks require information sharing among the cluster's resources. Information distribution is also a potentially important problem of computation but, to a lesser degree, given need to maximize ratio of computation to communication, for reasons of performance.

Information can be distributed among a cluster's resources using mechanisms like common data repositories, configuration scripts, file distribution, or message passing APIs (MPI). The information may include operating system images, large datasets, executables, or files. In each case, key concerns related to the transfer of information include the speed with which information can be transferred; the reliability of the transfer mechanism in use; and the degree to which the mechanism will *scale*—i.e., support increases in the population of senders and receivers.

## 1.3 File Distribution In Clusters

### 1.3.1 Explanation of the problem.

The focus of this thesis is the creation of tools for distributing files in clusters. Any file distribution mechanism is based on one of four communication models:

- unicast, or communication between pairs of nodes
- broadcast, or one-to-all communication
- anycast, or one-to-one-of-many communication
- multicast, or one-to-many communication

Multicast has been a special focus of research due to its ability to conserve network bandwidth in situations where multiple receivers access transmissions from a single sender (cf. §2.2). The use of multicast, however, is complicated by the lack of support for reliability in IPv4 multicast (cf. §3.1).

The reliability problem has been a challenge to address. Mechanisms that promote scalability may do so by degrading reliability, and vice-versa. Standard schemes for making communications reliable use feedback from receiver to sender in the form of acknowledgements (ACK) or negative acknowledgements (NACK). Unfortunately, as the number of receivers increases, the number of ACKs or NACKs from the receiver to the sender increases. This may create a bottleneck around the sender, causing scalability problems in the form of feedback implosion [GeSch97]. Reducing ACKs or NACKs to improve scalability may hurt reliability.

A second issue in achieving reliability is the potential conflict between reliability and speed. Heterogeneous clusters complicate file distribution. If different systems in a cluster receive information at different rates of speed, then transmitting data too quickly or too slowly will result in data loss, excessive delays in transfer, or both. Ideally, a sender should transmit information in a way that is fair to all the receivers. Mechanisms to address the speed problem include receiver feedback, proxy servers, and transmitting the same information more than once.

### 1.3.2 Current Strategies For Reliable File Distribution.

Luigi has addressed the issue of implementing reliability in IPv4 multicast by transmitting forward error correction (FEC) packets along with the data packets [RiVic97]. In Luigi's approach the file to be transmitted is divided in fixed number of groups. Each group has a fixed number of data packets. When the file is sent, the sender constructs parity packets from the data packets and transmits the parity and data packets together. As the file is received, the receiver reconstructs lost data from parity packets. If enough packets reach the receiver successfully (cf. §4.2), the receiver can reconstruct any group even if all the data packets were lost. Reliability is thus achieved without any ACKs or NACKs, avoiding prohblems with feedback implosion. FCAST [GeSch97], a file distribution tool by Microsoft Inc., also uses this approach.

Sneha K. Kasera, Jim Kurose, and Don Towsley [KaKuTow00] introduced the use of multiple concurrent transmissions for improving the reliability of multicast. This approach uses multiple multicast groups to reduce receiver-processing overheads. A file to be transmitted is divided into a fixed number of groups with each group having a fixed number of data packets. In the first round of transmission data packets of all the groups are transmitted on a primary transmission channel. In the next and subsequent rounds of transmission instead of retransmitting the data packets of all groups on the same group, data packets of different groups are transmitted on different retransmission channels.

## 1.4 SRFTT: A Hybrid Strategy For File Distribution

This thesis's particular contribution is an efficient and robust file distribution tool—the Scalable and Reliable File Transfer Tool (SRFTT). To the best of author's knowledge, SRFTT is the first tool that combines Luigi's forward error correction technique and Kasera et al.'s multiple multicast channel technique. The result is a tool that enhances file transfer reliability while making allowances for heterogeneous receiver rates.

SRFTT transmits a file to a set of receivers using multiple rounds of communication. The first, setup round uses TCP to transmit session parameters to the receivers—information like file name, file size, file checksum, and session ID that govern the subsequent transfer. In the second round of communication, SRFTT transmits data packets on a primary transmission channel defined by a session parameter. Subsequent rounds of transmission use multiple communications channels to transmit FEC recovery packets. These rounds allow slow receivers to recover from overrun errors and all receivers to recover from damage to packets during transmission.

As a part of this work SRFTT's performance was evaluated under different loss rates for file transfers of 1 KB to 256 MB. Performance gains up to 33% are obtained by transmitting recovery packets on multiple channels instead of a single channel (cf. §6). The results of this evaluation show that for large files transmitting FEC packets on multiple multicast channels gives significant performance gains when compared to a single retransmission channel.

## 1.5 Roadmap For This Thesis

The rest of this thesis is divided into 7 sections. Section 2 is an overview of current multicast technology, including multicast mechanisms for IPv4 and commercial applications of multicast. Section 3 discusses the concept of reliability as it applies to multicasting. Section 4 reviews current reliable multicast tools, including FCAST, RMDP, SRM, and multiple multicast channels. Section 5 discusses SRFTT in detail. Section 6 discusses SRFTT performance under varied network conditions. Section 7 draws conclusions about the work and suggests possible improvements to SRFTT.

# CHAPTER 2

# OVERVIEW OF CURRENT MULTICAST TECHNOLOGY

Multicast is communication between a single sender and multiple receivers on a network [TechTarget01]. The current section discusses past work related to multicast-based file distribution. Section 2.1 defines multicast communication, comparing it to other communication mechanisms like unicast and broadcast. Section 2.2 describes key features of multicast transmission, including the Internet Group Management Protocol (IGMP), multicast send and receive, and algorithms that create a multicast spanning tree. Section 2.3 discusses commercial multicast applications.

The discussion below is along the lines of similar discussions in [GeSch97] and [Stardust01].

## 2.1 Multicast—A Definition And Comparison With Other Communication Models

Multicast is one of three communication models supported by IPv4 (Internet Protocol version 4). In multicast a sender uses a single send operation to send a packet of information to multiple receivers. Other communication models supported by IPv4 are unicast, where one 'send' operation sends to one receiver, and broadcast, where one 'send' sends to all receivers in a set of networks.

Multicast can offer substantial bandwidth savings when compared to unicast or broadcast. Consider, for example, the load created by sending the same packet to 5 receivers on a 25-host network by unicast, multicast, and broadcast. In unicast the initial relay device would receive 5 packets. In broadcast the initial relay device would receive 1 packet, but all 25 hosts would get the packet. In multicast only one packet is sent to the initial relay device. The relay devices at the network's junction replicate packets sent to multiple downstream devices—here, to the 5 target hosts.

## 2.2 Multicast Mechanism For IPv4

In IPv4 multicast communication is supported by mechanisms that handle different aspects of the multicast process. These mechanisms include IGMP, a protocol for managing multicast communication protocol; a multicast send mechanism; a multicast receive mechanism; and a set of multicast tree construction algorithms (relay algorithms).

### 2.2.1 Multicast Communication Protocol—IGMP.

An *mrouter* is a router that supports multicast [Stardust01]. All multicast hosts exchange group membership information with their local mrouters through Internet Group Management Protocol (IGMP). IGMP is described in RFC 1112 [Rfc1112]. IGMP's operation can be divided into the following subtasks: election of querier, Host Membership Query, and Host Membership Report.

The mrouter that periodically exchanges IGMP packets with the hosts on its subnet to check group membership states of its host is called the *querier*. A querier is *elected*—that is, chosen—when IGMP begins its operation sequence or the current querier fails. If there is more than one mrouter on the subnet, the mrouter with the lowest IP address becomes the *Querier*.

A querier, once elected, sends *Host Membership Query* messages on its subnet to discover the group membership states of its hosts. Host Membership Queries are addressed to all hosts on the subnet. The TTL of Host Membership Queries is one to prevent these queries from propagating outside the LAN.

The hosts on the querier's subnet reply to the Host Membership Query message by returning a *Host Membership Report* to the querier. By sending the Host Membership report, the hosts declare their interest in multicast sessions to the querier. The destination address of the Host membership report is all-host group and the TTL of the message is 1.

The host membership reporting algorithm uses random delay timers to protect the querier against message overload. If every host on the subnet were to send its host membership reports to the querier, the ensuing network congestion around the querier may bring the entire subnet network down. The following technique reduces the number of Host Membership Reports in a subnet. When a host receives a Host Membership Query message from the querier, the host doesn't reply immediately but instead starts a random delay timer based on a randomly chosen value. When the timer expires the host generates a Host Membership Report and sends the report on the multicast session address. On receiving the report message other group members suppress their Host membership reports. Thus only one host sends a report for one multicast session on a given subnet.

### 2.2.2 Sending Mechanism.

The IPv4 mechanism for supporting multicast, from the sender's point of view, resembles the IPv4 mechanism for supporting the sending of UDP datagrams [Comer00]. Each multicast session has a class D multicast address—an IP address between 224.0.0.0 and 239.255.255.255 inclusive—that uniquely identifies the multicast session. The sender sends the information on a class D address associated with the given multicast session using a datagram oriented sending mechanism. The sender need not be a part of the multicast session to which it sends data.

### 2.2.3 Receiving Mechanism.

The user's host application joins the class D multicast address associated with the multicast session. As a result of this join, a membership request is communicated to the mrouter of the host's subnet. The mrouter on the user's subnet forwards this membership request to the intermediate mrouters between the sender and the receiver, using relay algorithms discussed in the next section.

The receiving host's network interface card obtains a new MAC address for each multicast session the receiver joins. The receiving host's network interface card places the low-order 23-bits of the IP address into the low-order 23 bits of the Ethernet multicast address 01-00-5E-00-00-00 to obtain a new Ethernet Multicast address and listens for this new MAC address also. When the receiving host's mrouter receives a packet destined for a class D multicast address, it maps the multicast group address to the associated MAC address and sends the message on the subnet using this new MAC address. The receiver's NIC intercepts this message, matches it with its new MAC multicast address, and sends it to the protocol stack, which passes the message to the application.

It is possible for a receiver to receive messages from other multicast sessions it is not interested in. There are 28 significant bits in an IP host group address and the receiving host's. The mrouter maps only the lower order 23 bits to obtain a new Ethernet address. Hence, up to 32 host group addresses can map to the same Ethernet multicast

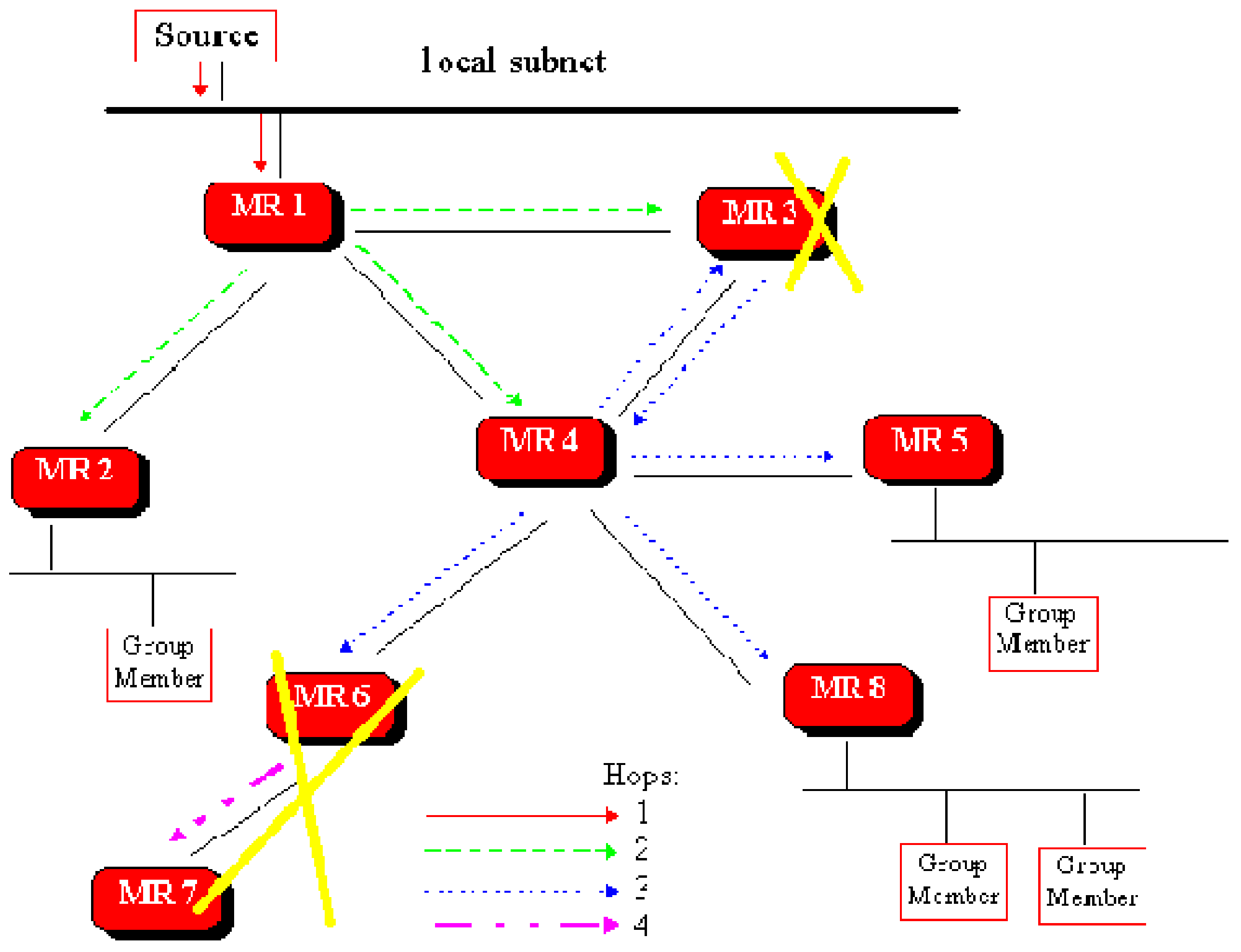

FIG. 1 DVMRP (adapted from [Stardust01])

address. The host's application layer discards messages from groups that aren't of interest.

## 2.2.4 Relay Algorithms.

IP Multicast packets for a given session are transmitted to the receivers via a spanning tree that connects a session's members. Protocols for constructing the spanning tree operate in one of the two modes depending on the relative proximity of the members of the group. These modes are referred to as Dense Mode and Sparse Mode.

### 2.2.4.1 Dense Mode Relay Algorithms.

All dense mode protocols are designed to work efficiently within a densely distributed network—one whose members are in close proximity. Dense mode protocols also assume that the underlying network has reasonably high bandwidth. Distance Vector Multicast Routing Protocol (DVMRP), Multicast Open Shortest Path First (MOSPF), and Protocol-Independent Multicast—Dense Mode (PIM-DM) are examples of dense-mode routing protocols.

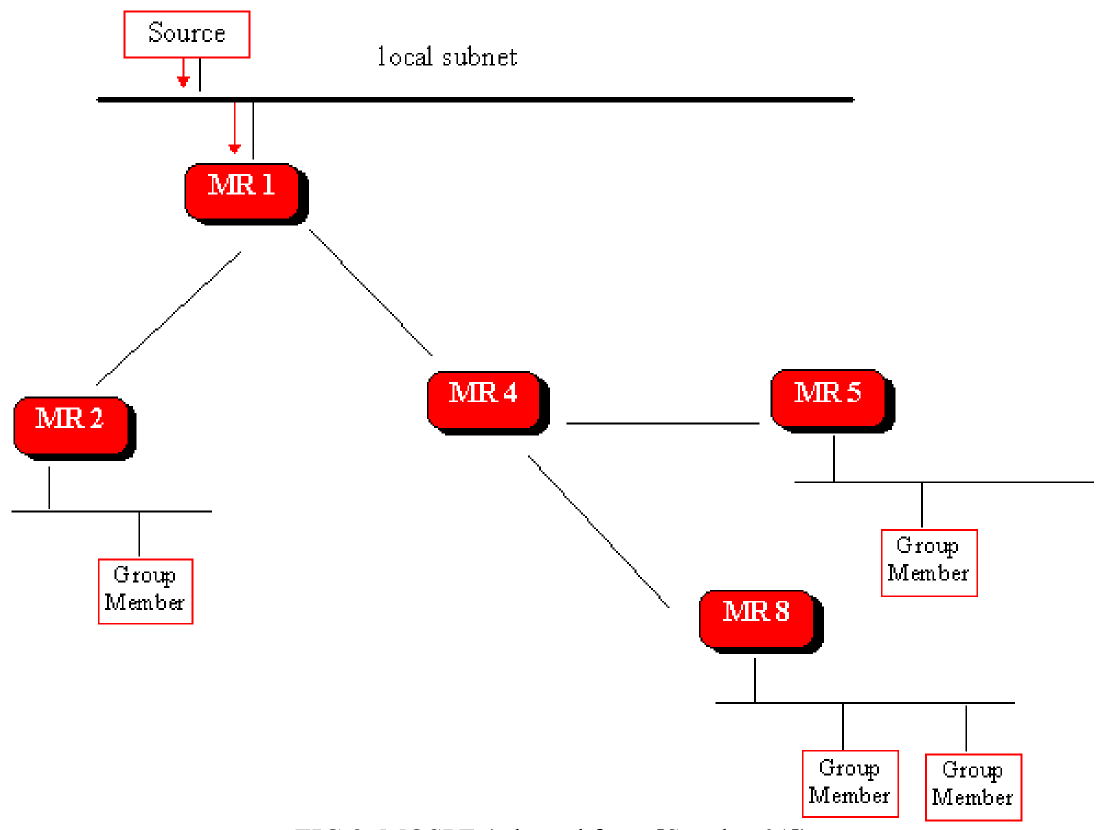

FIG 2. MOSPF (adapted from [Stardust01])

*DVMRP* (cf. Fig. 1) is described in RFC 1075 [Rfc1075]. DVMRP uses a special metric, the *DVMRP metric,* to construct a different spanning tree from each sender to all receivers. A DVMRP metric for a given source is constructed by taking into account the shortest number of hops from the source to all the receiver nodes. In the initial phase of tree construction the mrouter on the node's subnet multicasts a join message to all its adjacent mrouters. These adjacent mrouters multicast the join message to their adjacent mrouters. The forwarding of multicast messages continues till the join message reaches all the multicast members. Then the DVMRP metric is applied to the tree and the branches of the tree that don't provide the shortest path are pruned.

*Multicast Extensions to OSPF (MOSPF)* (cf. Fig 2) is defined in RFC 1584 [Rfc1584]. MOSP is a *link state protocol*. In a link state protocol parameters such as network traffic and QOS (quality of service) are considered along with the number of hops to calculate the shortest path from the sender to the receiver. All mrouters employing MOSPF periodically collect link state information from their neighboring mrouters. Hence, each router can compute a multicast tree from any source to all the members in the group. MOSPF employs *Dijkstra's algorithm* [Rfc1584] to compute the multicast tree. MOSPF is most suitable for small internetworks.

*PIM-DM* uses its underlying unicast protocol to check if the packet has arrived on the shortest path from the source. If the path is shortest, PIM-DM forwards the packet to its adjacent mrouters. Else, PIM-DM discards the packet. PIM-DM gives protocol independence at the cost of more overhead compared to DVMRP [Rfc1075] (cf. Fig 3).

#### 2.2.4.2 Sparse Mode Relay Algorithms.

All sparse mode protocols are designed to work efficiently within a sparsely distributed network—a network whose members are many hops away from each other. Sparse mode protocols are also intended for use in networks with limited bandwidth. *Core Based Trees (CBT)* and *Protocol-Independent Multicast—Sparse Mode (PIM-SM)* are examples of sparse-mode routing protocols.

In *CBT* a *core* router constructs a single spanning tree that is shared by all the group members throughout the entire multicast session. Mrouters send a join message to the core and join the spanning tree. Session members join the shared multicast tree through their subnet mrouters. When the core receives a join request, it returns an acknowledgment and accepts the new mrouter in the multicast session. Any mrouter between the core and the newly joined mrouter can intercept a join request and can acknowledge the join request. Some versions of CBT support the use of multiple cores to avoid bottlenecks around the core router.

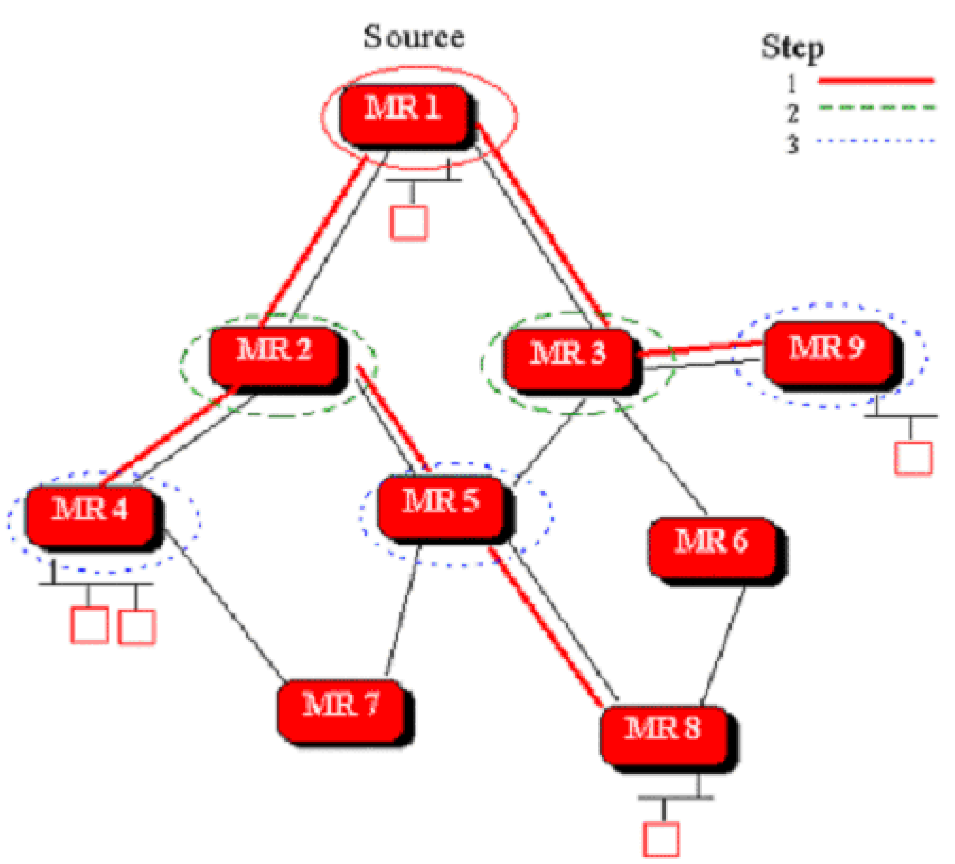

FIG 3. MOSPF Tree Construction (adapted from [Stardust01])

An mrouter called *a rendezvous point* (RP) constructs a multicast distribution tree. The rendezvous point is analogous to the core in the CBT protocol. During the initial phase of operation, the PIM-SM initially connects the senders and receivers to the rendezvous point and forms a group-shared tree (cf. Fig. 4). Later the router closest to the receiver can change its connection to a particular source by sending a PIM join message directly to the source on the shortest path and pruning the extra branches. Thus, because much of the network traffic is directed away from the RP, bottlenecks can be avoided around the RP.

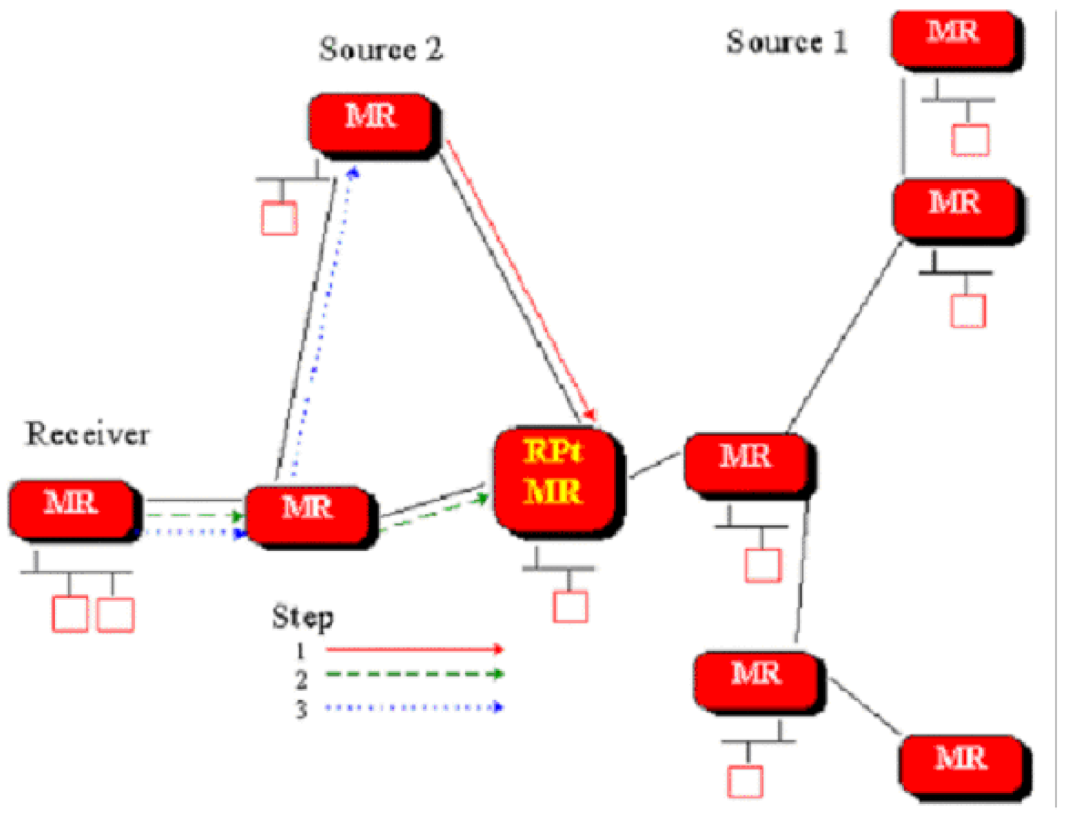

FIG 4. PIM-SIM (adapted from [Stardust01])

# CHAPTER 3

# RELIABLE MULTICAST

This section explores the limitations of the current IP multicast model with respect to scalability and reliability. Section 3.1 discusses the concept of reliability as it applies to multicasting. Section 3.2 discusses various concerns associated with implementing reliability in multicast communications, including the impact of application semantics on reliability implementation mechanisms, the need for reliability mechanisms in current IPv4 multicast model, mechanisms for adding reliability to the current IPv4 multicast data transmissions, and the use of forward and reverse flows to achieve reliability.

## 3.1 Reliability: A Definition

*Reliability*, in the context of network communications, can imply support for any combination of the following guarantees:

- A protocol maximizes the likelihood that receivers get data in timely way.
- A protocol guarantees the in-order arrival of data at all receivers.
- A protocol guarantees the correct arrival of data at all receivers.

Different flows require different kinds of guarantees. For example, applications that deliver voice or video in real time cannot tolerate out-of-order or untimely delivery, while file-transfer applications can typically tolerate delay and out-of-order delivery but not loss.

Different kinds of delivery algorithms provide different kinds of reliability. Each algorithm imposes its own form of overhead on a communication and cannot be used without imposing some kind of performance penalty.

## 3.2 Implementing Reliability In Multicast Communications

### 3.2.1 The Impact Of Application Semantics On Reliability.

An application's semantics dictate its mechanisms for implementing reliability. Multimedia applications consider transmissions with some losses as "reliable" as long as these losses don't degrade the quality of the transmission to the user to an excessive degree [Gonc02]. On the other hand, reliable data transfer applications such as reliable file transfer can tolerate retransmissions but can't tolerate the loss of even a single packet [Gonc02].

### 3.2.2 Need For reliability Mechanisms In Current IPv4 Multicast Model.

IPv4 multicast is unreliable [Comer00]. IPv4 multicast provides a best-effort delivery service that fails to guarantee the timely, ordered, or correct arrival of data. The resulting lack of overhead allows application developers to use IPv4 to build efficient multicast applications that provide application-specific guarantees. Unfortunately, this lack of reliability also forces developers who need reliability to implement guarantees on the top of the current IP multicast model.

### 3.2.3 Adding Reliability To IPv4 Multicast In Data Transmissions.

There are two basic approaches to implementing reliable communication in any multicast protocol. The first, *sender-oriented* approach, requires a transmission's point of origin to assume primary responsibility for detecting and correcting errors in transmission The second, *receiver-oriented* approach, makes reliability the responsibility of a message's receivers.

Schemes for ensuring reliable dataflow can also be classified according to the stream of information used to support reliability. *Forward error correction* (FEC) strategies use the flow of information from the sender to the receivers to *enhance* reliability (cf. §4). Here, reliability is achieved by transmitting extra parity information along with the original data. *Automatic repeat request (ARQ)* strategies use a second flow of information from the receivers to the sender to *enhance* or ensure reliability (cf. §4). Here, reliability is achieved by transmitting two kinds of control packets: packets that return the status of recent communications (ACK/NAK), and packets that direct the sender to slow or speed the flow of information to the receiver (flow control packets). *Hybrid* strategies employ a combination of FEC and ARQ to enhance or ensure reliability.

For total reliability, the receivers *must* transmit control packets to the sender.

The rest of this section describes complicating factors that affect the use of forward and reverse flows in reliable communications in more detail.

#### 3.2.3.1 Forward Flows.

Issues that complicate the use of forward flows in reliable communications include variable flow rates, asynchronous join/leave, network heterogeneity, and data ordering [Gonc02].

*Variable flow rates***.** Different receivers in the receiver set may have different data reception rates.  A sender needs to balance its flow between its fastest and the slowest receivers in the receiver set so as to minimize the idle time at fast receivers and transmission overruns at slow receivers.  The mechanism that guarantees a fair transmission rate is called a flow control mechanism.  Avoiding idle time and retransmissions, in general, requires the concurrent transmission of each stream of information at multiple rates of speed—either by the receiver itself or by fast, intermediary stations that accept, buffer, and relay data from the receiver at the rates required by the slower receivers.

Approaches for implementing efficient flow control mechanisms include the use of multiple multicast channels and proxy mechanisms.

*Multiple multicast channels.*  In the multiple multicast channels approach the sender transmits its data on different multicast channels in a way that enables any receiver that subscribes to all the channels to get the entire file.  Slower receivers subscribe to a few groups whose overall transmission rates match their reception speed. Receivers with high reception rates join all data channels and leave quickly.  The sender transmits the entire file on every channel to ensure reception by the slowest receiver.

In the proxy mechanism approach, a proxy is placed between the sender and receiver.  The proxy serves one or more LANs.  The proxy caches all the data transmitted by the sender and retransmits it according to the speed of its receivers on its LAN.

*Asynchronous join/leave.*  In a multicast session receivers can join and leave at any time.  How to retransmit information to receivers who have joined late is an important consideration for an efficient forward flow mechanism.  One solution to this problem uses a data reservoir that stores all the information transmitted by the sender. Receivers joining late can contact this data reservoir and obtain the information they have missed.

*Network heterogeneity.* Currently, some network relay devices don't support multicast. A mechanism called *IP tunneling* [Gonc02] is used to transmit information to receivers that have relay devices in their path from the sender that don't support multicast.

*Data Ordering.* Some application semantics require that information be assembled in correct order (like file transfer), while some application semantics (like video conferencing) don't require this.

#### 3.2.2.2 Reverse Flows.

Reverse flows consists of control packets (ACK or NACK) from the receivers to the sender. The primary issue that complicates the implementation of reverse flows is the excessive retransmission of control packets to the original sender. Too many control packets can result in *feedback implosion*—a situation where a bottleneck is created around the sender because of a large number of retransmission requests from the receivers [GeSch97]. Feedback implosion is addressed using *random delay timers:* the receivers wait for a random amount of time before transmitting ACK or NACK to the sender and the receivers suppress their requests for a packet to the sender if another group member requests the same packet.

# CHAPTER 4

# OVERVIEW OF MULTICAST MECHANISMS

This section describes research on reliable IP multicast. Section 4.1 discusses Software FEC techniques including two protocols that implement FEC: Reliable Multicast Distribution Protocol (RMDP) and Fcast. Section 4.2 discusses the Scalable Reliable Multicast (SRM) Framework. This discussion includes SRM mechanics, ADU considerations in an SRM framework, synchronization mechanisms in an SRM framework, and SRM framework operation. Section 4.3 discusses Kasera et al.'s multiple multicast channels framework including the framework's operation, implementation semantics, use of IP's multicast mechanism, local filtering mechanism, and operation with router support.

## 4.1 Software FEC Techniques [RiVic97]

FEC performs well in *lossy* network environments—networks where the overhead of parity information is offset by the time savings that the receivers obtain by using one parity block to recover from varied data packet losses.

FEC enables recovery from lost packets and bit level corruption. In IP multicast, however, the data link and physical layers rectify the bit level corruption errors. Thus, it suffices for a scalable and reliable multicast protocol to provide error recovery mechanisms for erasure loses only [RiVic97].

The FEC mechanism is divided into encoding and decoding phases. In the

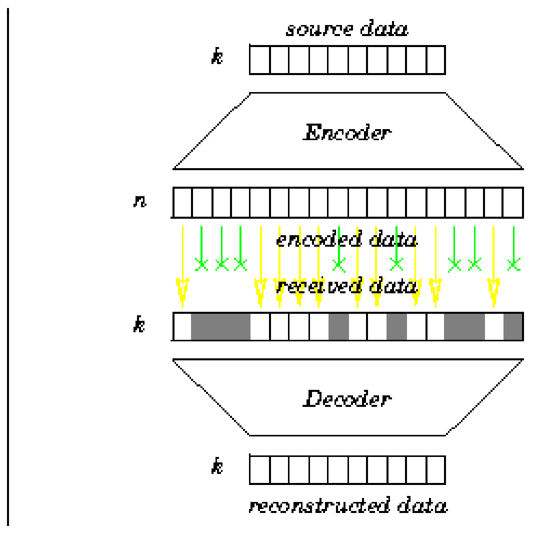

FIG 5 FEC Encoding/Decoding (adapted from [RiVic97])

*encoding phase* the sender constructs parity packets from the source packets [RiVic97]. In *decoding phase* the receiver reconstructs any lost packets by solving the differential equation on the parity packets and data packets.

Figure 5 illustrates the use of FEC in data transmission. FEC blocks are implemented using Rizzo and Vicisano's (n, k) encoding-decoding system. Here, n represents the total number of original data packets and k the total number of source packets. The number of parity packets transmitted along with the original data packets is n – k. The encoder (on the sender side) creates n – k parity packets from the k source packets by applying a differential equation on the source packets. The decoder (on the receiver side) can reconstruct the entire source packets if it receives any k out of the total n packets transmitted. The parity blocks are of the same size as the original data packets.

### 4.1.2 Protocols That Implement FEC.

#### 4.1.2.1 Reliable Multicast Distribution Protocol (RMDP).

RMDP is a reliable and scalable multicast protocol for file transfer. In RMDP the sender transmits a large number of parity packets relative to data packets. The authors use a round-based strategy to determine the existence of a multicast link between the client and the server.

In the first round of communication the client contacts the server by multicasting a join message to the server. If the server replies, the client can safely assume the presence of a multicast link with the server; otherwise, the client unicasts its request to the server. Session parameters are exchanged between the client and the server during this first round of communication.

In the next and subsequent rounds of communication the server transmits the data and parity (FEC) packets for a fixed period of time. If a client can reconstruct the original message from the packets it receives, it stops listening. But if the client cannot reconstruct the original message and the server has stopped transmitting, the client sends a continuation request to the server and asks for more packets.

The authors haven't incorporated congestion control mechanisms in their current design. RMPD claims to decrease the risk of feedback implosion from the clients to the server as the clients send feedback messages to the server only when the server has stopped transmitting and the client hasn't received all the packets.

4.1.2.2 Fcast.

Fcast [GeSch97] is a file transfer tool developed by Jim Gemmell, Eve Schooler, and Jim Gray of Microsoft. The discussion of Fcast that follows is organized along the lines of a similar discussion in [GeSch97].

Fcast uses multicast to send files on IP networks. Fcast receivers operate in one of two modes, according to the length of the time the receiver remains connected with the sender.

*Tune In, Download, and Drop Out (TIDDO) mode*. In this model the sender loops through a set of files and transmits the files continuously for a fixed period of time. Receivers subscribe to the appropriate transmission channels, receive the files, and drop subscriptions immediately after receiving their files. Receivers that want to receive the files again must resubscribe to the transmission channel. Thus, TIDDO mode is used where the time needed to establish a connection is low and bandwidth is limited [GeSch97].

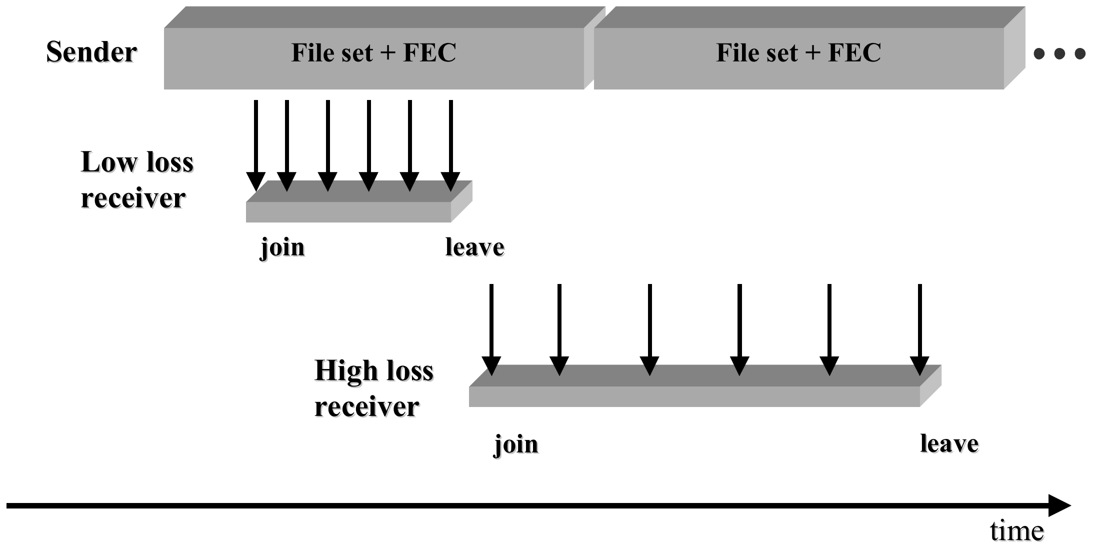

FIG 6. TIDDO mode (adapted from [GeSch97] )

*Satellite mode*. In satellite mode receivers remain subscribed to the transmission channel.

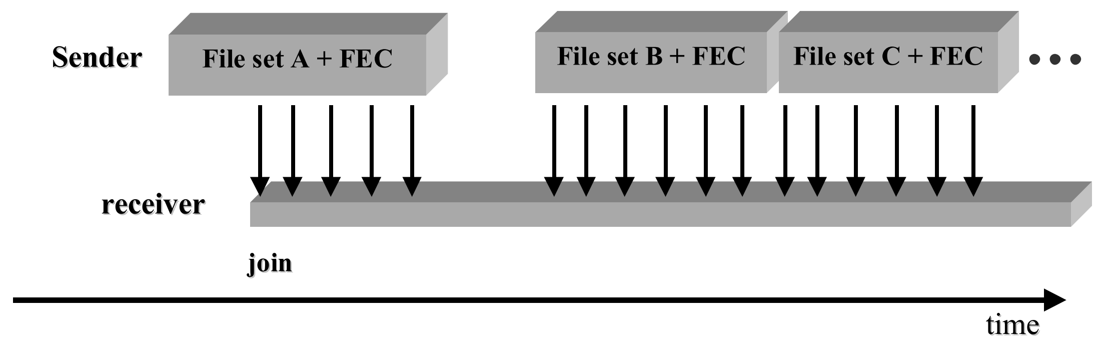

FIG 7. Satellite mode (adapted from [GeSch97] )

Here, the receiver may receive unwanted files, which wastes bandwidth. Hence, the satellite model is used when set up time is high and bandwidth is plentiful.

The Fcast sender and receiver components are implemented as ActiveX controls [GeSch97]. In Fcast a sender starts transmission without directly synchronizing with the receivers. The sender also transmits session parameters (multicast address, port number) through an outside mechanism as out-of-band data or email. The sender divides the file to transmit in fixed size groups, computing parity on a group-by-group basis, and ordering every group's parity information after that group's data. Each group is further divided into fixed size blocks (packets). The sender loops through the groups, transmitting one block from each group on each pass through the groups. Transmissions are ordered within groups: the $i^{th}$ block of each group is always transmitted before the $(i+1)^{st}$ block [GeSch97]. The sender also transmits each file's attributes, including that file's type (e.g., .zip or .ram), location, size, and name. Receivers subscribe to the appropriate transmission channel to receive files of interest and remain subscribed or drop out depending on their mode of reception.

The receiver caches received blocks in a temporary file until it receives all blocks necessary to reconstruct the entire file. After all blocks are received, the receiver reconstructs the file, sets its attributes and stores the file at the location specified by the sender.

Fcast's authors studied the effect of disk performance on file transfer time. The authors suggest that if the disk can't keep up with the speed at which packets are received from the network, many packets will be lost and require retransmission. They describe five techniques for optimizing fcast performance in environments where networks outperform disks.

*In-Memory*: The basic requirement for implementing In-Memory data transfers is that the receiver must have enough main memory to receive and decode the entire file in main memory. The In-Memory scheme is effective when files are small enough to fit in the main memory. One probable application area of the In-Memory technique is transmitting periodic updates.

*Receive-Order*: In Receive-Order mode, a receiver writes incoming blocks immediately to disk without sorting the blocks, thereby ensuring fast reception. The penalty of fast reception is paid during the decoding phase where blocks must be sorted before decoding.

*Group-Area:* Fcast divides each file into groups and every group into blocks. The area on disk that holds a group's blocks is called its Group Area. In Group-Area mode the blocks are written in their respective group-area on the disk without respect to ordering. Thus, when the reception is completed the blocks are partially sorted. The subsequent sorting and decoding of blocks takes less time than receive order scheme. The authors introduce the concept of a *bucket*: a logical compartment in main memory dedicated to holding related blocks. In the Group Area strategy, the main memory is divided into two buckets, one for receiving blocks and the other for writing the blocks.

*Crowd-Bucket:* Crowd–Bucket mode partitions a file's constituent groups into *crowds*. The receiver writes the blocks in their respective crowd-area. The operation of the Crowd-Bucket mechanism is similar to that of the Group-Area scheme.

*Crowd-Pool*: The Crowd-Pool mode of operation is similar to the Crowd-Bucket strategy, except that Crowd-Pool maintains a pool of buckets.

## 4.2 SRM Framework [FlJcLMcZh97]

The SRM framework is based on a multicast group delivery protocol model called Application Level Framing (ALF) by Clark and Tennenhouse. ALF explicitly includes an application's semantics in the design of that application's protocol [Ja95]. Birman et al.'s Lightweight Sessions (LWS) protocol extended ALF by adding receiver-based reliability [BiSchStep91]. SRM is built upon ALF and LWS. SRM maximizes information sharing among all group members. Wb, the web based distributed whiteboard tool, implements the SRM framework.

### 4.2.1 SRM Mechanics.

SRM is a receiver-oriented multicast mechanism. Each member of the group is individually responsible for detecting its loss and requesting retransmission.

#### 4.2.1.1 ADU.

The SRM framework divides the data to transmit into independent data units. Each such data unit is called an ADU (Application Data Unit). ADU’s have global, persistent, and unique identifiers that identify each ADU separately over time and space.

#### 4.2.1.2 Synchronization Mechanism.

Each member multicasts periodic session messages to the multicast group. Session messages establish a group’s current membership, inform all group members about the latest sequence number of ADU being transmitted, and support the calculation of round-trip times for message exchanges between pairs of group members. Session messages consume a very limited bandwidth compared to the data messages.

SRM members dynamically adjust the generation rate of session messages in proportion to the multicast group size.

#### 4.2.1.3 Operation.

Any member who detects a loss starts a random delay timer. The delay timer’s range depends on the distance between the source and the member that has detected the loss. The delay timer is chosen from the uniform distribution on $[C_I\, d_{SIA}, ( C_I + C_2 )\, d_{SIA}]$ seconds, where $d_{SIA}$ is host A's estimate of the one-way delay to the original source S of the missing data [FlJcLMcZh97]. Network parameters like network traffic, RTT, and expected delays govern the calculation of C1 and C2. When a timer expires, the member that has detected the loss multicasts its retransmission request on the multicast group.

Other members having loses identical with the previous one suppress their requests and start a new random delay timer randomly chosen from a uniform distribution on

$$2 \, i \, [C_I \, d_{SIA}, ( C_I + C_2 ) \, d_{SIA}]$$ [FlJcLMcZh97].

When any member receives a retransmission request from another member, it checks if it has the packets requested by the other group member. If so, it starts a random delay retransmission timer based on a value from the uniform distribution on $[D_I \, d_{SIA}, ( D_I + D_2 ) \, d_{SIA}]$ seconds, where $d_{SIA}$ is host B's estimate of the one-way delay to host A, and $D_1$ and $D_2$ are parameters of the repair algorithm dependent on the network parameters [FlJcLMcZh97]. If the member receives a retransmitted packet during the wait period and if it has set retransmission timer for the packet, it cancels its timer. Otherwise, after the timer expires it retransmits the packet on the multicast group. It is likely that a member nearer to the member that has the lost packet will receive the retransmission request first and retransmit the packet.

In order to prevent duplicate requests from triggering a responding set of duplicate repairs, host B ignores requests for data D for $3 * d_{SIB}$ seconds after sending or receiving a repair for those data, where host S is either the original source of data D or the source of the first request [FlJcLMcZh97].

## 4.3 Multiple Multicast Channels [KaKuTow00]

Sneha K. Kasera, Jim Kurose, and Don Towsley introduced use of multiple multicast channels for adding reliability to current multicast best effort delivery model. Their approach uses multiple multicast groups to reduce receiver-processing overheads.

In a typical ARQ-based reliable multicast scenario transmissions and retransmissions occur on the same multicast channel (group). Because all packet transmissions and retransmissions use one group, each receiver receives all retransmissions of a packet even after correctly receiving the packet. This imposes unnecessary receiver processing overhead because the network layer processes all the packets coming for the particular multicast group. The redundant packet travels its way up to the application layer where the application layer finally discards the packet. In a very lossy network receivers spend a lot of time processing unnecessary packets.

Multicast using multiple multicast channels attempts to solve the problem of redundant packets flowing in the network.

### 4.3.1 Operation.

Kasera et al.'s protocol operates as follows. Consider a system with a sender and R receivers such that data are to be transmitted from the sender to the receivers using G + 1 channels. One channel is used for the original transmission of packets from the sender. The remaining G channels are used for retransmissions and are mapped to ranges of sequence numbers such that a lost packet is retransmitted on the channel corresponding to the range to which it belongs. The receivers join the retransmission channels for the packets that they have lost and leave the group after they receive the packet.

The authors have demonstrated two scenarios, one with an infinite number of multicast groups and the other with a finite number of multicast groups. The authors have studied one to many and many to many communication methods in both the scenarios. Their data show that using a finite number of multicast groups can produce the effect of an infinite number of multicast groups.

### 4.3.2 Implementation Semantics.

#### 4.3.2.1 Using The IP Multicast Mechanism.

Each multicast group is implemented as a different IP multicast group. Joining and leaving the group means joining and leaving the IP multicast group.

When a receiver detects a packet loss, it determines the retransmission channel to join to recover the lost packet and starts a random timer. After the timer expires, the receiver multicasts the request on the multicast group. If another receiver receives a retransmission request for the same packet before the timer expires, it restarts its counter and again waits for a random amount to time.

#### 4.3.2.2 Local Filtering.

Joining and leaving a multicast group imposes extra overhead throughout the multicast tree. Each join and leave message results in processing overhead at all the intermediate routers. To avoid this, the authors have proposed a new concept of local filtering.

Initially a receiver joins the original transmission group and all retransmission multicast groups. The network transmits IGMP messages to the entire multicast tree. The local interface filters out the unwanted packets, based on the information provided by the local join and leave operations, to save the receiver from processing these packets.

#### 4.3.2.3 Adding Router Support.

Local filtering doesn't reduce network bandwidth. To reduce the use of network bandwidth the authors have proposed the use of retransmission-aware routers. All transmissions and retransmissions take place on only one channel. The router forwards retransmitted packets only to those receivers that have NACKed lost packets.

# CHAPTER 5

# SRFTT (SCALABLE AND RELIABLE FILE TRANSFER TOOL)

Scalable and Reliable File Transfer Tool (SRFTT), the subject of this thesis, is a tool developed to simplify information sharing among a cluster's nodes. SRFTT guarantees reliable data delivery for bulk data-transfer applications like file transfer, Web caches preloading software, and software upgrades distribution. SRFTT combines the following two strategies for error management to achieve reliability:

- Forward Error Correction (FEC) [RiVic97]: In this approach recovery packets are encoded and decoded using software FEC techniques. Data transfer tools like FCAST [GeSch97] and RMDP [RiVic97] (cf. §4.1.2.) use this approach.
- Multiple multicast Channels [KaKuTow00]: In this approach recovery packets are retransmitted on multiple retransmission channels to reduce reception time. (cf. §4.3)

The rest of this section discusses implementation issues related to SRFTT. Section 5.1 describes SRFTT's operation, and Section 5.2 describes SRFTT's component software.

## 5.1 SRFTT Operation

SRFTT makes four basic assumptions about its operating environment: the existence of a multicast link between the sender and the receivers, a static receiver set, a transfer unit of predetermined size and content, and a network of heterogeneous receivers.

SRFTT is session based. Each session has an identifier that uniquely identifies the session. Each session has a primary transmission channel to transmit the data packets and multiple retransmission channels to transmit the recovery packets. All the retransmission channels are implemented as distinct multicast channels.

SRFTT operation is divided into the following three steps:

- The sender transmits session information to all receivers. Session information includes session id, primary transmission channel address, file name, file size, and file checksum.
- The sender transmits the file on the session's primary transmission channel. The receivers join the primary transmission channel to receive the file.
- The sender transmits recovery packets on the session's retransmission channels. The receivers subscribe to the appropriate retransmission channel to receive the lost packets.

## 5.2 SRFTT Building Blocks

SRFTT is architected as a sender-receiver application where the sender acts as a server and the receivers as clients. The architecture of SRFTT senders and receivers is described in Sections 5.2.1 and 5.2.2 respectively. The SRFTT configuration file, which drives an SRFTT session, is described in Section 5.2.3.

### 5.2.1 SRFTT Sender.

SRFTT is sender-initiated: i.e. the sender initiates the file transfer. The sender divides each file to be transmitted into B groups, with each group having K data packets.

The parameter K is fixed and pre-configured. The value of B depends upon the file size and the value of K. A detailed description of B and K is given in Section 5.2.3.

In the first round of transmission, the sender transmits the session information to all the receivers using a reliable data delivery mechanism (TCP/IP). In the next round, the sender multicasts the data packets on the session's primary transmission channel. After sending the data packets, the sender multicasts multiple "stop" packets on the session's primary transmission channel. A "stop" packet is a special packet that signals the end of transmission on the session's primary channel. In subsequent rounds of transmission, the sender multicasts the recovery packets on all of the session's retransmission channels until a certain degree of redundancy has been achieved. The recovery packets are encoded using software FEC techniques [RiVic97].

The number of retransmission channels is fixed and pre-configured (cf. §5.2.3).

### 5.2.2 SRFTT Receiver.

On receiving the session information from the sender, the receiver joins the session's primary transmission channel to receive the file. The file is divided in B * K packets, where B = number of groups and K = number of packets per group (cf. §5.2.3). If a receiver receives all data packets correctly, the receiver leaves the primary transmission channel and reassembles the file from the received packets.

If a receiver fails to receive all the data packets correctly, the receiver joins the appropriate retransmission channels to recover the lost packets; receives the required number of recovery packets—i.e., the number of data packets lost for that group; leaves all retransmission channels; decodes the parity packets; and reassembles the file. To recover packets lost from a group i the receiver joins the retransmission channel i % T, where T = total number of retransmission channels.

Both sender and receiver are multithreaded. The primary thread handles primary transmission channel, and the secondary threads handle retransmission channels. SRFTT uses the PISCES [Pfei02] and PFC [Pfei02] libraries to handle socket and thread management respectively.

There is no feedback from the receivers to the sender. This addresses the problem of feedback implosion and makes SRFTT scalable.

### 5.2.3 SRFTT Configuration File.

SRFTT uses a configuration file to configure parameters like packet size, group size (number of packets per group), number of retransmission channels, and network loss. For optimal SRFTT performance, these parameters must be tuned to the given application or environment.  This sub-section is a detailed description of the SRFTT configuration parameters.

*Packet size:* IP multicast is built on the top of IP protocol and IP multicast packets are transmitted as IP datagrams.  The network infrastructure de-fragments a packet in multiple IP datagrams if the packet size exceeds a certain limit (normally 512 or 1024 bytes).  Increasing the packet size helps to transmit the file in fewer packets, thus reducing overhead while increasing the cost of fragmentation and reassembly.

*Group size*: Group size is defined as number of packets in a group.  Increasing the number of packets in a group increases the robustness of the transmission (in terms of erasure recovery capability) but also increases encoding and decoding costs [RiVic97].

*Number of retransmission channels:* If the number of SRFTT retransmission channels is less than the number of groups in a file, some retransmission channels must carry recovery packets for two or more groups. In this scenario, multicast groups that support multiple retransmission channels multiplex among their assigned channels, delaying retransmission, but decreasing the overall load on the network.

*Loss rate:* A network's loss rate is defined as the number of packets lost by the network per 100 packets transmitted. SRFTT initially transmits FEC packets depending upon a pre-decided loss rate of the network. Different networks have different loss rates. If the initial loss rate is configured below the network loss rate, the network load increases because of the retransmission requests by multiple receivers (cf. §5.2.2). On the other hand, if the initial loss rate is configured above the network loss rate the network load increases because of the sender transmitting unnecessary FEC recovery packets.

# CHAPTER 6

# PERFORMANCE

This section reviews tests of SRFTT performance. SRFTT's performance was tested relative to different file sizes under varying loss rates and retransmission channel counts. The results obtained show that for large files retransmitting recovery information on multiple multicast channels gives significant performance gains when compared to a single retransmission channel.

The balance of this chapter is divided into three sections. Section 6.1 describes the methodology for testing SRFTT. The description reviews test parameters in detail. Section 6.2 describes the SRFTT test procedures. Section 6.3 analyses the results obtained from the tests.

## 6.1 SRFTT Test Methodology

File transfer time and scalability are the defining attributes for a reliable file transfer mechanism. Faster transfer times and more graceful degradation under heavier loads equates to better performance.

Two important test parameters for any file transfer tool are file size and loss rate.

- *File size*. Modern day communications demand transmission of several GB of information. A file transfer tool should be able to transfer files up to several GB with minimal degradation in performance. SRFTT was tested for file sizes of up to 256 MB. Due to resource limitations, SRFTT was not tested beyond 256 MB.

- *Loss rate*. Loss rate, the number of packets dropped or corrupted beyond repair during transmission, is typically measured as a percentage of all packets transmitted. Normally, loss rates increase and decrease with increases and decreases in network load. Network loss rates normally average between 10% and 20% [Comer00]. In extreme situations, the network loss-rate can jump to 50%. A reliable file transfer tool should incorporate recovery mechanisms for extreme conditions. SRFTT was tested under stimulated network loss rates of up to 50%.

A third SRFTT test parameter was the number of retransmission channels. SRFTT transmits recovery packets on multiple retransmission channels. As the number of retransmission channels increases, the error recovery time decreases due to reduction in receiver-wait time and, hence, performance increases [KaKuTow00]. Also, increasing the number of retransmission channels beyond a certain point doesn't improve performance gains for a given network state [KaKuTow00].

Scalability is a fourth important concern for a file transfer tool. A file transfer tool should be able to service hundreds of clients without considerable degradation in service. Currently SRFTT has been tested with 10 clients due to resource limitations.

## 6.2 SRFTT Test Procedures

SRFTT was tested on Dell Intel Pentium III 933 MHz workstations with 3Com's 10/100 Mb/s NIC cards running Windows 2000. The network capacity is 10Mb/s.

SRFTT's performance was evaluated using files with fixed and incrementally increasing sizes. The tests were designed to assess the effect of file size, transmission loss rate, and retransmission channel count on transfer time.

SRFTT divides the file to be transmitted in G groups. Each group can be represented as a set of (k, n) packets where

k = total number of data packets in a group,

n = total number of packets in a group, and

n – k = number of FEC recovery packets in a group.

The value of n was 255 for all the tests because of FEC library limitations [RiVic97]. The network infrastructure fragments a packet into multiple IP datagrams if the packet size exceeds a network-specific limit (normally 1476 bytes for Ethernet) [Comer00]. The size of the data packet as well as the recovery packet is 1 KB to avoid fragmentation by the network infrastructure.

### 6.2.1 Testing Procedure: Fixed File Size.

In the fixed file size tests**,** a 2MB file was repeatedly sent over the network to determine how SRFTT's performance was affected by varying the number of retransmission channels, network loss-rates, and values of k. This testing procedure is similar to that of Luigi [RiVic97] with one addition—a new dimension, the number of retransmission channels, was added to the tests. The value of k was set to 1, 8, 16 and finally 32 packets per group. Increasing k beyond 32 had no effect on file transfer.

For each k, a predetermined number of data packets were deliberately dropped to simulate varying network loss-rates. On successive test runs, 0%, 1%, 2%, 5%, 10%, 15%, 20%, 25%, 30%, 35%, 40%, 45% and 50% of all packets were dropped. A network loss rate of 0% yielded a loss of around 60 packets (3%), suggesting one of two things: the network either lost packets, or the network speed outpaced disk speed. Gemmell and

Schooler's research suggests that the second explanation is more probable [GeSch97]. Accordingly, the loss-rates mentioned earlier are an approximation and variations in the range of ±3% are possible.

For every k and network loss-rate the number of retransmission channels was set to 1, then 2, then 3. Increasing the number of retransmission channels beyond 3 did not change file transfer time.

## 6.2.2 Testing Procedure: Incremental File Size.

In the incremental tests, a succession of test files was sent over the network to

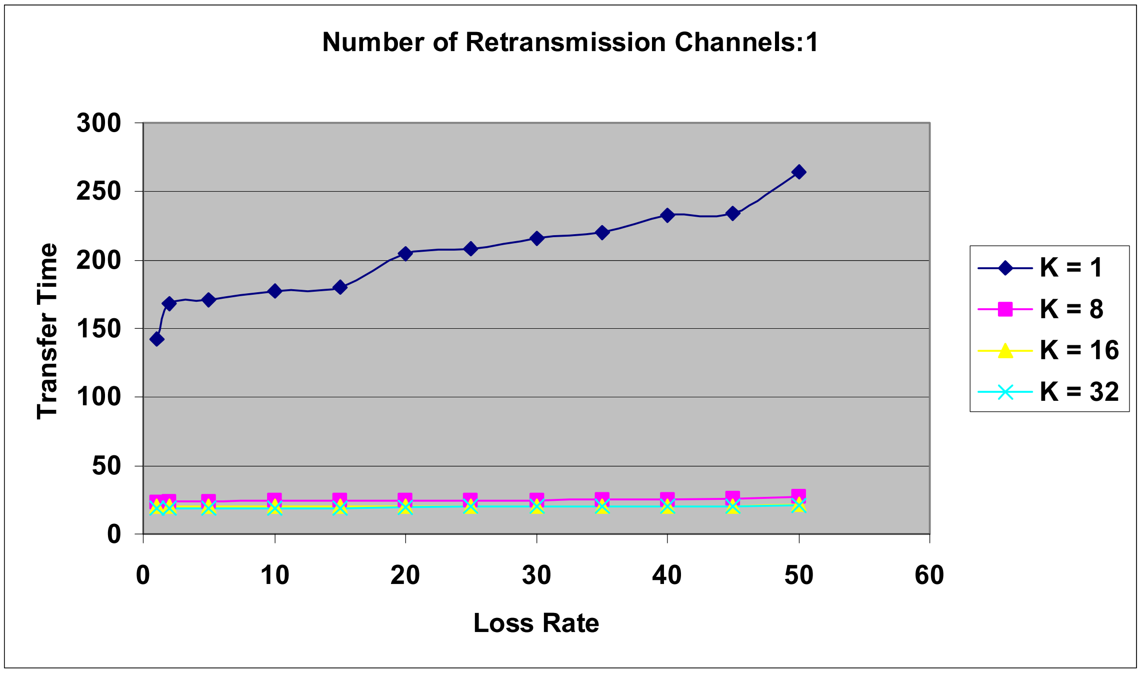

FIG 8 Loss rate vs Transfer time with different values of k and single retransmission channels

determine SRFTT's performance relative to file size and retransmission channel count.

File sizes were chosen to be 1 KB, 1 MB, 2 MB, 4 MB, 8 MB, 16 MB, 32 MB, 64 MB, 128 MB, and 256 MB. For consistency with fixed file size tests, each file was tested using 1, 2, and 3 retransmission channels. Loss rate and k were held at 25% and 16 respectively.

## 6.3 Data And Analysis

As shown in Figures 6.1, 6.2, and 6.3, SRFTT's performance improves as the

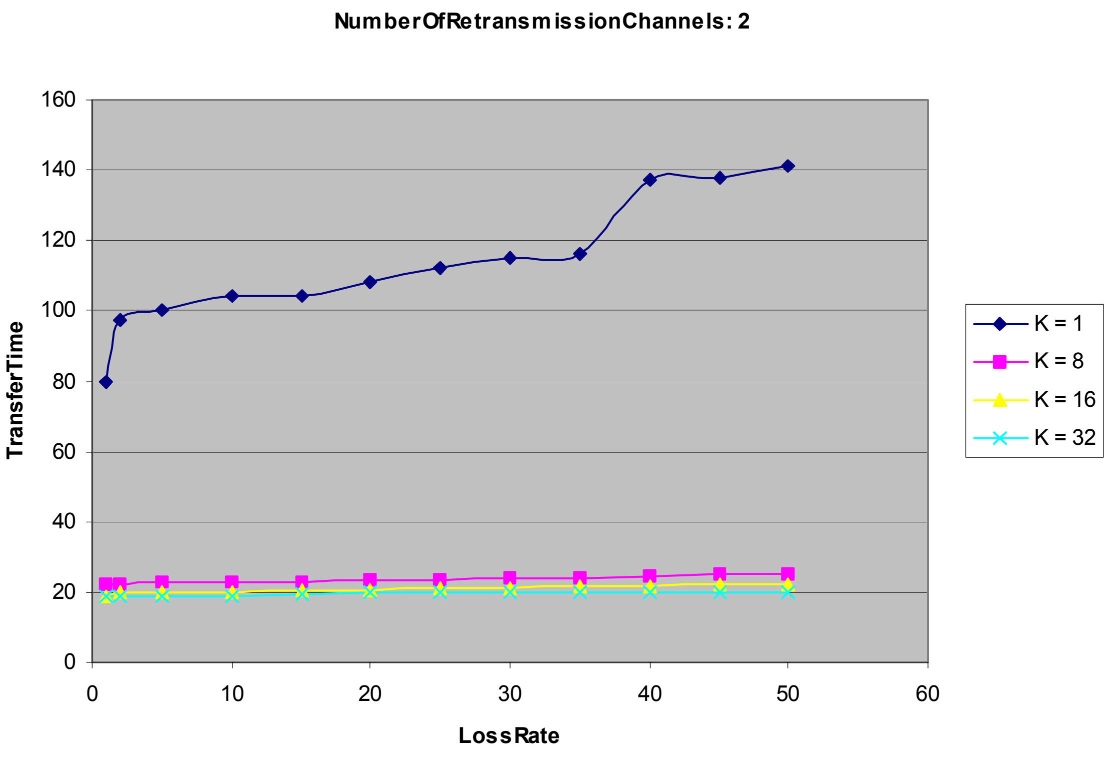

FIG 9 Loss rate vs Transfer time with different values of k and two retransmission channels for a 2MB file

number of packets per group (k) increases when other parameters are held constant. These data are consistent with Luigi's results [RiVic97]. As k increases the number of data packets in a group increases.

The FEC encoder generates n – k recovery packets from all the data packets of a group. Because all data packets are involved in generating recovery packets, increasing k increases the number of data packets involved in generating recovery packets. The FEC decoder can recover any data packet of a group if the number of distinct recovery packets received equals the number of packets lost for that group. Thus, as the number of data packets in a group increases, a single recovery packet can help recover more data packets. Consider, for example, the scenarios where k = 1 and k = 8. When k = 1, one recovery packet can help recover only one packet. When k = 8, one recovery packet can help recover 8 packets. Hence, as k increases, the efficiency of each recovery packet and the robustness of each transmission increases—but at the cost of increases in encoding and decoding overhead [RiVic97].

For a 2 MB file, SRFTT's performance doesn't improve drastically on increasing the number of retransmission channels. However, adding retransmission channels has a noticeable effect on the transmission of 128 MB and 256 MB files. Becuase packets per group (n) is held constant, increasing file size increases the number of groups (G). As noted in Chapter 5, retransmission of recovery packets occurs on multiple multicast channels. To recover a group's packets, the receiver must join that group's multicast channel. For small files having fewer groups, the cost of joining multiple retransmission channels offsets the benefits of using multiple channels [KaKuTow00].

SRFTT's performance increases with the increase in number of retransmission channels. This observation is consistent with Kasera et al.'s results [KaKuTow00]. As noted in Chapter 5, SRFTT transmits recovery packets of a group on a particular retransmission channel according to the following equation:

*Retransmission channel = Group mod (Number of retransmission channels*).

The number of retransmission channels is usually less than the number of groups. Hence, one retransmission channel may service multiple groups. In this scenario recovery packets of multiple groups are interleaved on retransmission channels. As the number of retransmission channel increases, the time for the recovery packets of a group to be transmitted again on the group's retransmission channel decreases. This reduces the wait time of a receiver waiting for a particular group's packets. Hence, the efficiency

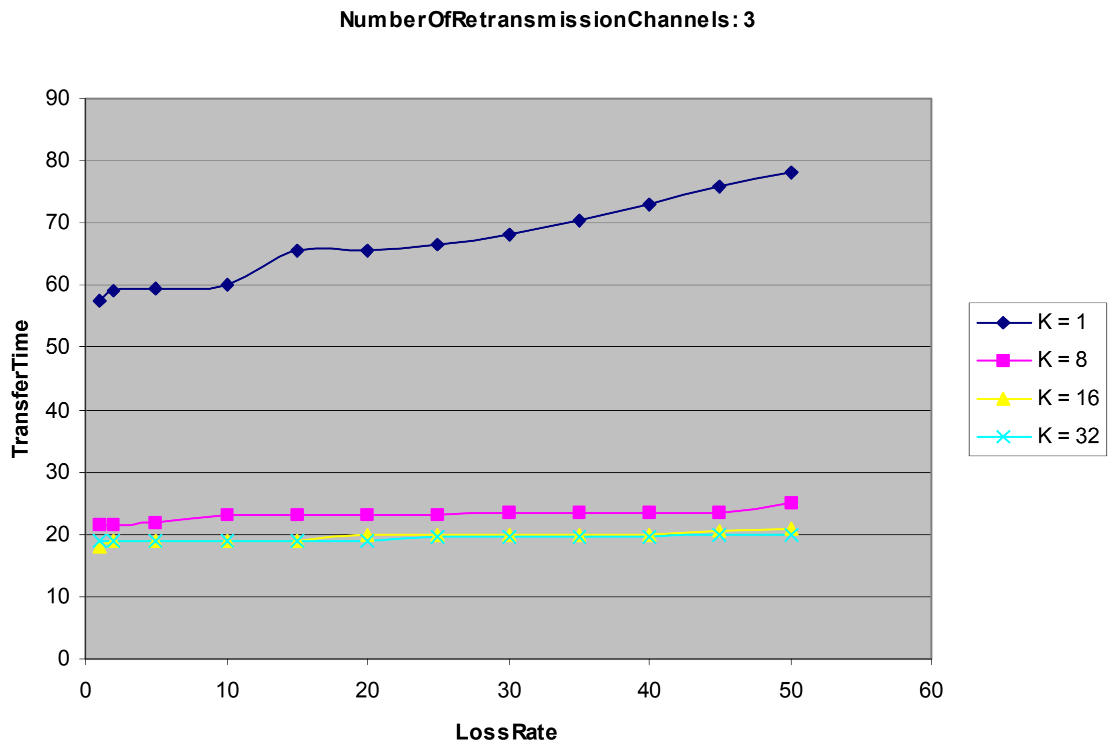

FIG. 10 Loss rate vs Transfer time with different values of k and three retransmission channels for a 2MB file

increases with the increase in the number of retransmission channels.

As the file size increases, the effect of increasing the number of retransmission channels becomes pronounced (cf. Fig. 6.4). Increasing the number of retransmission channels from 1 to 3 yields a gain of nearly 33 for file sizes larger than 32 MB. Further increases in the number of retransmission channels yield further improvements. For example, increasing the number of retransmission channels from 3 to 7 decreases the reception time for a 64MB file with k = 16 and loss-rate = 25% from 2361 sec. to 2185 sec.

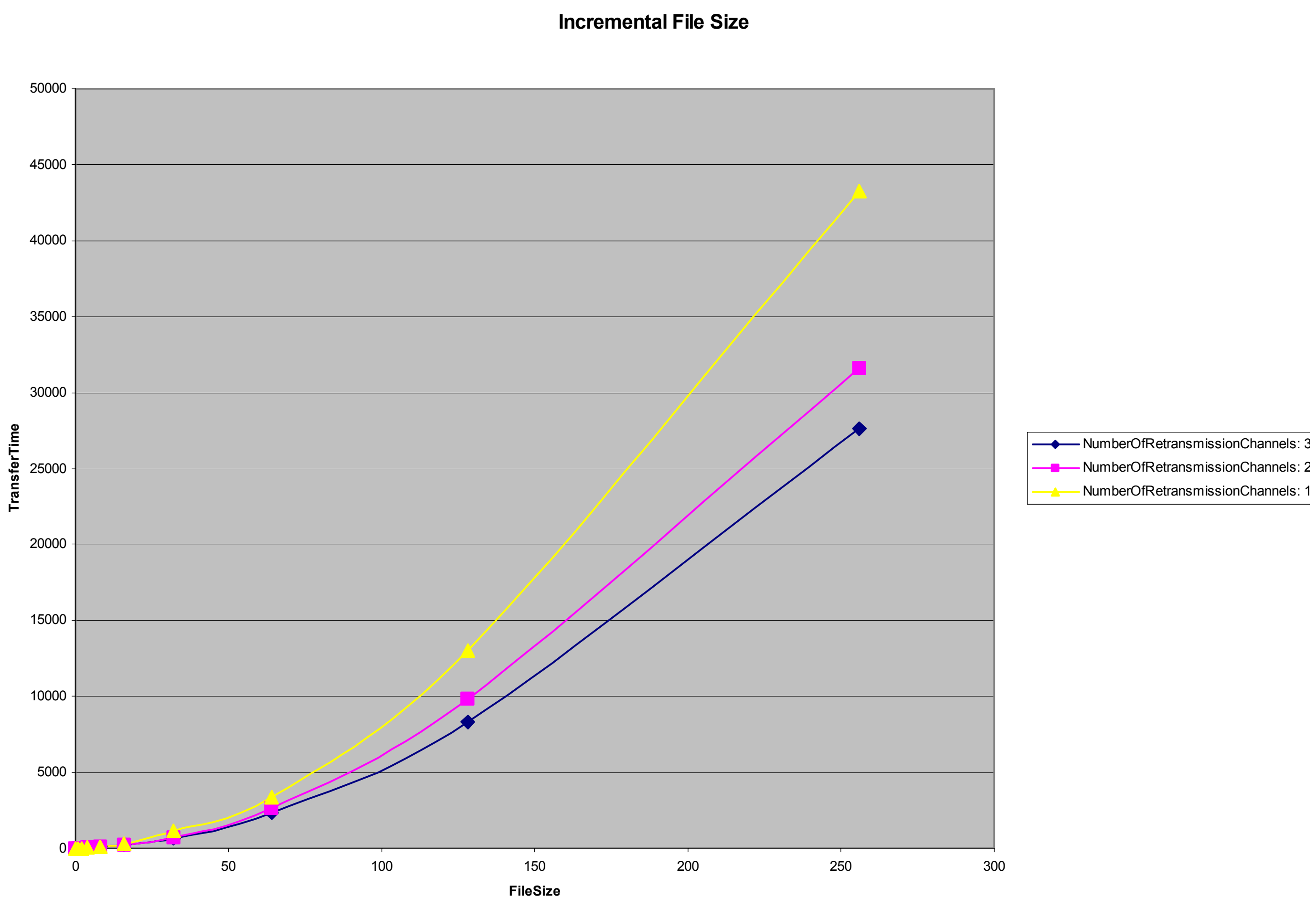

FIG 11 Incremental File Size

# CHAPTER 7

# CONCLUSIONS AND FUTURE WORK

This thesis has described the implementation and operation of a new tool for file transfer, specifically designed for use in heterogeneous clusters. This tool, SRFTT, combines two techniques to achieve reliability in multicast communications: software FEC and multiple multicast channel approaches. Even though both approaches have overheads, the cost of encoding/decoding FEC packets is offset by the ability to recover from multiple packet losses using a single FEC packet and the cost of joining and leaving multiple retransmission channels is offset by the reduced receiver reception time. SRFTT performs better in heterogeneous environments. For large files increasing the number of retransmission channels improves SRFTT's performance significantly.

SRFTT's implementation will be available soon as freeware. Currently SRFTT is tested on the Windows platform. With minor modifications, SRFTT can be implemented on UNIX and Linux platforms. SRFTT has been tested for file sizes up to 256 MB and 3 retransmission channels. More testing, however, should be performed on SRFTT to determine the tool's ability to transfer massive files relative to larger and less uniform networks of PCs. These tests should include transfers of larger, multi-gigabyte files, using more retransmission channels and more target hosts. SRFTT should also be enhanced to support flow control mechanisms in order to improve the tool's performance.

# APPENDIX

## OBSERVATION TABLE

**Fixed File Size**

**Number of Retransmission channels = 1**

| **K** | **Loss Rate** | **Time** | **K** | **Loss Rate** | **Time** | **K** | **Loss Rate** | **Time** | **K** | **Loss Rate** | **Time** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 142.5 | 8 | 1 | 23 | 16 | 1 | 20 | 32 | 1 | 19 |
| | 2 | 168.5 | | 2 | 24 | | 2 | 20 | | 2 | 19 |
| | 5 | 171 | | 5 | 24 | | 5 | 20 | | 5 | 19 |
| | 10 | 177 | | 10 | 24.4 | | 10 | 20 | | 10 | 19 |
| | 15 | 180 | | 15 | 24.5 | | 15 | 20 | | 15 | 19 |
| | 20 | 204.5 | | 20 | 24.5 | | 20 | 20 | | 20 | 19.5 |
| | 25 | 208 | | 25 | 24.5 | | 25 | 20.5 | | 25 | 20 |
| | 30 | 216 | | 30 | 24.5 | | 30 | 20.5 | | 30 | 20 |
| | 35 | 220 | | 35 | 25.5 | | 35 | 20.5 | | 35 | 20 |
| | 40 | 233 | | 40 | 25.5 | | 40 | 20.5 | | 40 | 20 |
| | 45 | 234 | | 45 | 26 | | 45 | 20.5 | | 45 | 20.5 |
| | 50 | 264 | | 50 | 27.5 | | 50 | 21.5 | | 50 | 21 |

**Number of Retransmission channels = 2**

| K | Loss Rate | Time | K | Loss Rate | Time | K | Loss Rate | Time | K | Loss Rate | Time |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 80 | 8 | 1 | 22 | 16 | 1 | 19 | 32 | 1 | 19 |
|  | 2 | 97.5 |  | 2 | 22 |  | 2 | 20 |  | 2 | 19 |
|  | 5 | 100 |  | 5 | 22.5 |  | 5 | 20 |  | 5 | 19 |
|  | 10 | 104 |  | 10 | 22.5 |  | 10 | 20 |  | 10 | 19 |
|  | 15 | 104 |  | 15 | 23 |  | 15 | 20.5 |  | 15 | 19.5 |
|  | 20 | 108 |  | 20 | 23.5 |  | 20 | 20.5 |  | 20 | 20 |
|  | 25 | 112 |  | 25 | 23.5 |  | 25 | 21 |  | 25 | 20 |
|  | 30 | 115 |  | 30 | 24 |  | 30 | 21 |  | 30 | 20 |
|  | 35 | 116 |  | 35 | 24 |  | 35 | 21.5 |  | 35 | 20 |
|  | 40 | 137 |  | 40 | 24.5 |  | 40 | 21.5 |  | 40 | 20 |
|  | 45 | 138 |  | 45 | 25 |  | 45 | 22 |  | 45 | 20 |
|  | 50 | 141 |  | 50 | 25 |  | 50 | 22 |  | 50 | 20 |

**Number of Retransmission Channels = 3**

| K | Loss Rate | Time | K | Loss Rate | Time | | Loss Rate | Time | K | Loss Rate | Time |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 57.5 | 8 | 1 | 21.5 | | 1 | 18 | 32 | 1 | 19 |
| | 2 | 59 | | 2 | 21.5 | | 2 | 19 | | 2 | 19 |
| | 5 | 59.5 | | 5 | 22 | | 5 | 19 | | 5 | 19 |
| | 10 | 60 | | 10 | 23 | | 10 | 19 | | 10 | 19 |
| | 15 | 65.5 | | 15 | 23 | | 15 | 19 | | 15 | 19 |
| | 20 | 65.5 | | 20 | 23 | | 20 | 20 | | 20 | 19 |
| | 25 | 66.5 | | 25 | 23 | | 25 | 20 | | 25 | 19.5 |
| | 30 | 68 | | 30 | 23.5 | | 30 | 20 | | 30 | 19.5 |
| | 35 | 70.5 | | 35 | 23.5 | | 35 | 20 | | 35 | 19.5 |
| | 40 | 73 | | 40 | 23.5 | | 40 | 20 | | 40 | 19.5 |
| | 45 | 76 | | 45 | 23.5 | | 45 | 20.5 | | 45 | 20 |
| | 50 | 78 | | 50 | 25 | | 50 | 21 | | 50 | 20 |

***Incremental File Size***

Loss Rate = 25 = 25, k = 16

| ***3 Rexmit Ch*** | | ***2 Rexmit Ch*** | | ***1 Rexmit Ch*** | |
|---|---|---|---|---|---|
| ***FileSize*** | ***Time*** | ***FileSize*** | ***Time*** | ***FileSize*** | ***Time*** |
| 0.001 | 0 | 0.001 | 0 | 0.001 | 0 |
| 1 | 10 | 1 | 10 | 1 | 10 |
| 2 | 20 | 2 | 20 | 2 | 20 |
| 4 | 51.5 | 4 | 52 | 4 | 54.5 |
| 8 | 113 | 8 | 118 | 8 | 129 |
| 16 | 259.5 | 16 | 263 | 16 | 314 |
| 32 | 640 | 32 | 739 | 32 | 1172 |
| 64 | 2361 | 64 | 2658 | 64 | 3380 |
| 128 | 8306 | 128 | 9814 | 128 | 13024 |
| 256 | 27634 | 256 | 31583 | 256 | 43241 |

Random Samples

No change by increasing the number of retransmission channels to 4 or 5 on a 2Mb file

K = 16, NRT = 7, Loss rate = 25 and file size = 64 Mb-- Transfer time = 2185 sec

# BIBLIOGRAPHY

*Unless specified otherwise, all web pages referenced below were on-line as of 3 June 2002. All RFCs may obtained at www.ietf.org*

[BelGray02] Bell, G., and Gray, J., "What's next in high performance computing?" Communications of the ACM, Vol 45, No 2, Feb 2002

[BiSchStep91] Birman, K., Schiper, A., and Stephenson, P. ``Lightweight Casual and Atomic Group Multicast", ACM Transactions on Computer Systems, Vol.9, No. 3, pp. 272314, Aug. 1991

[Buyya99] Buyya, R., High Performance Cluster Computing, Vol 1, Ch. 1, Prentice-Hall, 1999

[Comer00] Comer, D., Internetworking with TCP/IP Vol.1: Principles, Protocols, and Architecture, 4/e, Chapter 12-13, Prentice Hall, c.2000

[FlJcLMcZh97] Floyd, S., Jacobson, V., Liu, C., McCanne, S., and Zhang, L., "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing", IEEE/ACM Transactions on Networking, Volume 5, Number 6, December 1997

[GeSch97] Gemmell; J. and Schooler, E., "Using Multicast FEC to Solve the Midnight Madness Problem", citeseer.nj.nec.com/schooler97using.html, Sept. 1997

[Gonc02] Goncalves, M., IP Multicasting, McGraw-Hill, c. 2002

| | |
|---|---|
| [HenPa96] | Hennessey, J. and Patterson, D., Computer Architecture–A Quantitative Approach, Chapter 7, §13, Morgan Kaufman Publishers, c.1996 |
| [Ja95] | Jacobson, V., ``Lightweight Sessions—A new architecture for real time applications and protocols'', Third Annual Principal Investigators Meeting, ARPA, Santa Rosa, CA, Sept. 1995 |
| [KaKuTow00] | Kasera, S., Kurose, J., and Towsley, D., "Scalable Reliable Multicast Using Multiple Multicast Channels", IEEE/ACM Transactions on Networking, June 2000 |
| [MPIRef] | http://www-unix.mcs.anl.gov/mpi/ |
| [Pfei02] | Pfeiffer, P., csciwww.etsu.edu/phil/freeware.htm, 2002 |
| [Rfc1075] | Waitzman, C., Partridge, D., and Deering, S., "Distance Vector Multicast Routing Protocol" Stanford University-- November 1988 |
| [Rfc1112] | RFC 1112: Deering, S., "Host Extensions for IP Multicasting", Aug. 1989 |
| [Rfc1584] | RFC 1584: Moy, J., "Multicast Extensions to OSPF", March 1994 |
| [RiVic97] | Rizzo, L. and Vicisano, L, "A Reliable Multicast Data Distribution Protocol Based on Software FEC Techniques (RMDP)", Proceedings of the Fourth IEEE HPCS Workshop, Chalkidiki, Greece, 1997 |
| [Scott02] | Scott, S., "Projects Page", www.csm.ornl.gov/~sscott/Projectspage.htm, 2000 |

[Stardust01] Stardust Technologies, "Introduction to Multicast Routing", www.stardust.com/community/whitepaper/introrouting.html [last referenced in July 2001; since removed from website]

[Tanen95] Tanenbaum, A. S., Distributed Operating Systems , Prentice Hall, c.1995

[TechTarget01] TechTarget Inc, searchnetworking.techtarget.com/sDefinition/0,sid7_gci212610, 00.html

[Vyto93] Vytopil, J, Formal Techniques in Real Time and Fault Tolerant Systems, Kluwer Academic Publishers, c.1993

## VITA

HARDIK DIKPAL SHUKLA

| | |
|---|---|
| Personal Data: | Date of Birth: September 9, 1977<br>Place of Birth: Gujarat, India<br>Marital Status: Married |
| Education: | L.D. College Of Engineering, Ahmedabad, India;<br>Instrumentation and Control, Bachelor of Engineering, 1999<br>East Tennessee State University, Johnson City, TN, USA;<br>Computer Science, Master Of Science, 2002 |
| Professional Experience: | Intern, Oakridge National Labs<br>Oakridge, TN, *(Summer 2001)*<br><br>Intern, Birla AT&T Communications<br>Gandhinagar, India, 1997-1998<br><br>Research Assistant, Department of Computer Science<br>East Tennessee State University, TN, 1999-2000 |