FTP-M: An FTP-like Multicast File Transfer Application

Manamohan Mysore, George Varghese

Abstract— **Research in Reliable Multicast has focused on providing scalable, semi-reliable delivery of data. Such solutions based on schemes like SRM have proved adequate for multicast delivery of multimedia data or whiteboard applications, but are inadequate for the strict delivery semantics of file transfer. This paper describes a simple application called FTP-M that provides an efficient one-to-many multicast data transfer by simply extending the FTP protocol and user interface. FTP-M relies on a *strictly-reliable* Reliable Multicast protocol layer beneath to send data reliably to multiple receivers. In this paper, we describe the design and implementation of FTP-M. We suggest that FTP-M has the potential to be adopted as a standard way to achieve multicast file transfer.**

Keywords—**Reliable Multicast, File Transfer, Application**

I. Introduction

One of the significant advances in computer networks in the last decade has been the invention of IP multicast. This invention has opened up the possibility of a variety of new applications in today's Internet. Among the applications that have been proposed to use IP multicast are streaming multimedia applications that disseminate video/audio data simultaneously to several destinations, and conferencing applications. However, a very important application of IP multicast, one-to-many file transfer has not received attention commensurate with its importance. While the research in this field has mainly focused on providing a reliable or semi-reliable transport layer over IP multicast, the application layer aspect of achieving multicast file transfer has not received as much attention.

This paper presents a simple application, FTP-M that extends FTP protocol [9] and interface to provide a natural and convenient interface to one-to-many multicast file transfer. FTP-M relies on a *strictly-reliable* [1] Reliable Multicast protocol layer beneath to send data reliably to multiple receivers. The FTP-M mechanism itself is thus decoupled from the problem of *strictly-reliable* multicast. In this paper, we describe the design and implementation of FTP-M. We show how to modify FTP to make is useful in a multicast environment. We also describe FTP User Interface extensions to provide a natural and convenient interface to multicast file transfer. We then go on to describe our actual implementation experience. Using an illustrative reliable multicast protocol called TCP-M, that provides strict reliability by extending TCP, we show how FTP-M can be implemented with ease. We argue that FTP-M has the potential to be adopted as a standard way to achieve multicast file transfer.

M. Mysore is a research scientist at Nomadix Inc., Westlake Village. Email: mmysore@ieee.org

G. Varghese is a professor at Dept. of Comp. Sci. & Eng., UC – San Diego. Email: varghese@cs.ucsd.edu.

[1] Refer to Section II

The rest of the paper is organized as follows. Section II introduces the context of our work by briefly surveying existing solutions to achieve multicast file transfer. In Section III, we present a brief overview of TCP-M, the *strictly-reliable* multicast protocol that we have used in this paper to illustrate FTP-M. After setting the stage for our core contribution, we describe the design and implementation of FTP-M in Section IV. In Section VI, we conclude the paper and outline directions for future research.

II. Related Work

An important offshoot of IP multicast is the field of Reliable Multicast. This topic aims at improving reliability of IP multicast data transfers. Traditionally, Reliable Multicast has focused mainly on *improving* reliability of multicast data transfers. These protocols do not guarantee delivery of data and hence leave much of the reliability issues to be resolved by applications. Of late, there have been a few proposals to enhance such protocols to include the notion of strict reliability. While these new protocols do simplify the problem of conveniently achieving multicast file transfer, an easily understandable and quickly deployable "killer-app" is needed. In the following sub-section, we give an overview of the existing research in reliable multicast in order to set the ground for our discussion of FTP-M, the multicast file transfer application that we propose.

A. Reliable Multicast

One can classify existing reliable multicast methods into two broad categories: *semi-reliable* and *strictly-reliable*. Among the first class of *semi-reliable* multicast protocols and congestion control techniques, MTP[1],SRM [2], RMTP [6], MTCP [10], PGM [14], and PGMCC [12] are quite well-known. These solutions are meant for applications such as multimedia data transfers, conferencing, and whiteboard applications and do not guarantee transfer of data as unicast transport protocols such as TCP do. For example, as they scale by having the source be unaware of the receivers, the source can never be sure that every receiver has received every message. Using these protocols for file transfer would need additional effort to build another layer on top of these protocols to guarantee delivery of data. Such a layer would then be required to keep track of the intended receivers and which among them received data correctly.

On the other hand, the second class of strictly-reliable reliable multicast protocols includes protocols such as IRMA [5], SCE [16], MFTP [13] and TCP-M[4]. These protocols provide an implicit or explicit way of informing the application which receivers have finally received the data that was

sent via multicast. This presupposes that the multicast source knows the intended receivers. These are ideally suited for use in multicast file transfer applications such as FTP-M.

B. Existing Multicast File Transfer Applications

While there are other multicast file transfer applications that have been proposed (FCAST [3], RMDDP [11], MFTP [13], XFTP [16], MDP [7]), these build a new file transfer protocol and user interface from scratch. However, FTP-M reuses and very simply extends a well known file transfer protocol (FTP) and interface (`ftp`) to achieve multicast file transfer. This makes FTP-M more suitable for easy and large scale deployment. Moreover, FTP-M does not depend on the use of a particular strictly-reliable reliable multicast protocol to function and can be used with any such protocol.

We now briefly describe TCP-M, an illustrative strictly-reliable multicast protocol that is used in our implementation of FTP-M.

III. TCP-M

To illustrate the working of FTP-M, we use a TCP-like reliable multicast protocol called TCP-M, that was initially proposed by Ghosh, et. al [4]. By reusing much of the TCP code and socket interface, TCP-M minimized the changes to FTP we need to make. FTP-M uses TCP-M as an example multicast reliability layer to ensure strictly-reliable delivery of data to recipients. In the following subsections, we briefly describe the design and implementation of TCP-M.

A. Introduction to TCP-M

TCP-M, as described in [4], is a TCP-like reliable multicast transport protocol that provides the following features:

- **Strict Reliability**: since unicast TCP is reused as the transport layer.
- **TCP-friendliness**: due to the unicast TCP layer at the source.
- **Scalability**: by enlisting router support to do Ack-fusion (Refer to [4])
- **Convenient API**: by using the standard BSD Socket API [15] enhanced with `getsockopts` and `setsockopt`.

B. The working of TCP-M

The essential idea behind TCP-M is to reuse a well known unicast transport protocol to serve as the transport layer at the source as in [5], [16][2]. This idea makes sense in a multicast scenario with unidirectional data transfer, if one does a many-to-one mapping somewhere. In TCP-M, this is achieved at a new adaptation layer called the Group Module (GM) introduced between the TCP and IP layers at the source. Figure 1 shows the positioning of the GM layer. Effectively, the source GM makes it appear to its TCP that it has a single peer, while in reality, a connection is being established to multiple receivers via multicast. The GM layers at the receivers capture any modifications that might be needed in the receiver's protocol stack.[3]

In normal operation, packets originating from the source TCP are sent to the multicast group by the source GM. Receiver GMs, on receiving multicast packets, make appropriate modifications to fool the receiver TCP that they are genuinely from its peer. Receiver TCPs send Acks to their peers and the GM layers direct the those Acks to the source TCP. The GM layer at the source does the many-to-one mapping, i.e., fuses all incoming Acks and passes up a summarized Ack to the TCP above. The details of connection establishment and tear-down may be found in [4].

As an enhancement, TCP-M enlists router support to fuse Acks as they proceed towards the source. However, the absence of Ack-fusing routers does not pose a problem to the working of TCP-M.

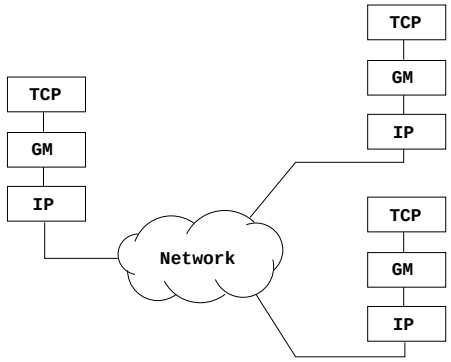

Fig. 1. TCP-M Architecture

C. Implementation of TCP-M

We implemented TCP-M in NetBSD kernel 1.3.2. Our implementation included the source and receiver GMs. Router Ack-fusion was not implemented. For modularity and easy maintenance, we chose to implement the GM as a separate protocol layer in NetBSD. The API was implemented as `getsockopt` and `setsockopt` extensions to the BSD Socket Interface [15]. `setsockopts` were provided to enable TCP-M semantics for a socket, to set the list of receivers at the source, to set the multicast group for the transfer and to boot out slow receivers. `getsockopts` were provided to return the list of currently active receivers, to query receiver RTTs, and to return certain statistics. The semantics of `socket`, `bind`, `connect`, and `accept` were not modified. For details, the reader is referred to [8].

Having described TCP-M and its implementation, we now go on to present the nuts and bolts of FTP-M.

IV. FTP-M

FTP-M is an application-layer protocol-interface combination that ensures fast and convenient multicast file transfer. Although the FTP-M idea works with any strictly-reliable

[2]However, the detailed mechanisms of TCP-M are different from [5], [16]

[3]In the case of NetBSD 1.3.2 TCP, receiver side modifications are very small but unavoidable

multicast transport protocol, we chose TCP-M as its transport service since we already had a usable prototype of TCP-M. In addition, TCP-M provided a comprehensive list of features – TCP-like congestion control, scalability, containment of Ack-implosion, easy API, and resilience to failures (refer Section II).

A. FTP-M features

Before going on to describe the intricacies of FTP-M, it is useful to summarize its features:

- FTP-M extends the standard FTP interface to achieve one-to-many file transfer.
- It uses a strictly-reliable Reliable Multicast protocol (such as TCP-M) to push data to multiple destinations.
- The actual modifications made to FTP are minimal. Most modifications are on the client side. The FTP server needs minimal extensions to function as an FTP-M server.
- The FTP-M interface has two modes of operation: one, the *normal mode*; two, *one-to-many multicast mode*. While functioning in multicast mode, FTP-M allows only *push* operations (such as `put` and `mput`); all *pull* commands (such as `get` and `mget`) are disallowed in this mode.

Before proceeding, a brief overview of the FTP protocol and interface is appropriate.

B. FTP

The FTP model is illustrated in Figure 2. FTP (unicast) uses two connections: a control connection, and a data connection. The control connection is active all through a file transfer session while the data connection is established and closed on demand. As can be seen in Figure 2, the control connection between the peer *protocol interpreters* that implement the File Transfer Protocol uses FTP command-reply sequences (such as USER, PASS, STOR, RETR, etc.). The data connection between the *data transfer processes* follows no specific protocol and involves the transfer of a sequence of bytes. The user interface provides a simpler interface to the FTP commands. For example, the user interface `put` command uses the STOR FTP command provided by the protocol interpreter. FTP replies are number-coded to facilitate easy interpretation.

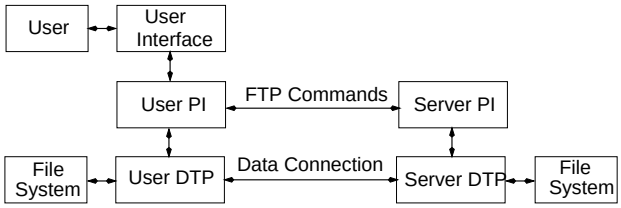

Fig. 2. The FTP Model

FTP provides two ways transferring data. In the first way, called *active mode* transfer, the client side creates a socket that waits[4] on a certain port p, and using the PORT command informs the server of port p. The server then initiates a connection to port p on the client machine to start the data transfer. In the second way, called *passive mode* transfer, the client side tells the server using the PASV command that it instead wants to `connect` to a waiting server socket. In response, the server waits on a new socket and informs the client of its port q. The client then initiates a connection to port q on the server machine to accomplish data transfer.

C. FTP-M model

Having described the working of FTP in brief let us now go onto examine the design of FTP-M. Figure 3 illustrates the FTP-M model. Let us examine Figure 3 in greater detail. In the FTP-M model, the client maintains control connections to several servers[5]; these control connections use enhanced FTP commands and replies. However, whenever there is a need to push data, a single reliable multicast (here, TCP-M) connection is initiated to all FTP-M servers. At any point of time, the user can work or negotiate with only one control connection; hence, the client protocol interpreter *points* to only one control connection (termed the *active* control connection) at any point of time. An FTP-M data transfer functions in two phases. In the first phase called the *negotiation phase*, the user establishes control connections to all the servers and if necessary, prepares each server (by say, moving/deleting/renaming files or changing working directory) for the planned multicast data transfer. In the second phase termed the *data transfer phase*, the user issues a suitable push interface command (`put`, `mput` or `send`) to reliably multicast the file. These two phases could repeat several times during an FTP-M session.

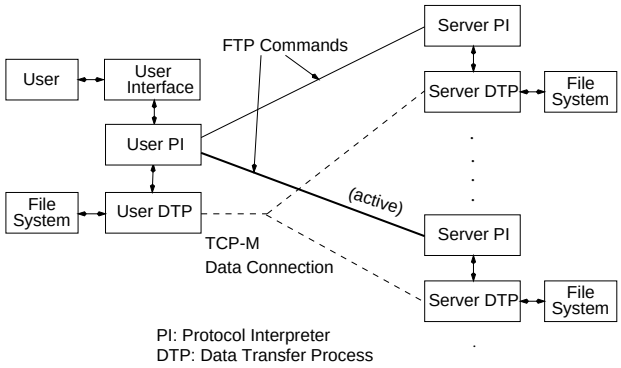

Fig. 3. The FTP-M Model

D. FTP-M User Interface

Figure 4 shows an example FTP-M session. In the example below, the user first switches the FTP-M user interface to the multicast mode by using the `multi` command[6]. He

[4] using the `accept` system call

[5] As mentioned in Section II, the multicast source needs to know the intended receivers

[6] FTP-M interface defaults to standard FTP interface if the `multi` command is not issued

then opens a control connection to pc1 as pat and readies the stage for the planned file transfer. He then opens another control connection to pc2 as al and performs similar actions. Now, the active control connection points to pc2. Later, he realizes that he needs to check the directory contents in pc1 and hence opens pc1 again, which switches the active control connection now to pc1. Finally, he does a put to attempt to transfer the file using TCP-M to both pc1 and pc2. However, pc1 crashes in the middle of the transfer, but the file is transfered successfully to pc2. The failure of pc1 is explicitly reported by FTP-M. Note that this interface reuses most of the FTP user interface.[7]

ftp-m> multi	230 User al logged in.
ftp-m> open pc1	ftp-m> ls
Connected to pc1	-rw-r- - - - - 1 al ... unwanted
220 pc1 FTP-M server ready	ftp-m> del unwanted
Name : pat	ftp-m> mkdir Doc
331 Passwd required for pat.	ftp-m> cd Doc
Passwd:	ftp-m> open pc1
230 User pat logged in.	Switching to pc1
ftp-m> ls	ftp-m> ls
-rw-r- - - - - 1 pat ... .zshrc	-rw-r- - - - - 1 pat ... .zshrc
ftp-m> mkdir Temp	ftp-m> lsopen
ftp-m> cd Temp	pat@pc1 : Temp (active)
ftp-m> open pc2	al@pc2: Doc
Connected to pc2	ftp-m> put README
220 pc2 FTP-M server ready	pc1 failed during transfer.
Name : al	Transfer partially successful.
331 Passwd required for al.	ftp-m> bye
Passwd:	221 Goodbye.

Fig. 4. An example FTP-M interaction

Below, we enumerate the various modifications FTP-M makes to the existing FTP interface. The usage of the following commands changes in multi mode.

1. open: multiple opens are now allowed.

2. put and mput: in multi mode, these commands use a TCP-M connection to push the file.

3. close: if specified with a hostname argument, it closes the control connection to hostname. If no argument is used, it closes the active control connection and switches the active control connection to the first in the list of connected servers.

4. get and mget: these are disabled in multi mode, since only push commands are available in this mode.

In addition, a few new commands are introduced for added convenience.

1. lsopen: This command is used to list all open connections. This command can be subsumed under the more general status command, but is incorporated to retain specificity.

2. run: This command, if used with a file argument loads and executes a configuration file that stores the mapping between the servers in consideration and the files that need to be transferred. This too can be avoided by storing FTP-M commands in a file and redirecting their contents[8] to an FTP-M execution.

[7] It should be noted that while the Figure 4 uses the prompt ftp−m> for clarity, a real implementation would instead use the normal ftp> prompt.

E. Behind the Scenes

We now describe how the above interface is made possible in FTP-M. As was mentioned earlier, there are two modes (active and passive) in FTP. Since FTP-M uses TCP-M where the source initiates connection to the receivers, the data transfer mechanism has to use the passive mode, where the client initiates the connection. In usual FTP, the passive mode causes the server side to wait on a usual TCP socket. In FTP-M, since we want the server side to wait on a TCP-M socket, a new protocol command (PASM) is introduced. PASM causes the remote end to wait on a TCP-M socket and return the address and port number to the client side. With this knowledge, let us examine the FTP-M protocol command reply sequences that correspond to the example FTP-M transfer shown in Figure 4.

$\rightarrow_1$ USER pat	$\leftarrow_2$ 226 Transfer complete
$\leftarrow_1$ 331 Passwd needed for pat.	.
$\rightarrow_1$ PASS ____	.
$\leftarrow_1$ 230 User pat logged in.	$\rightarrow_1$ PASM
$\rightarrow_1$ PORT 24,94,3,173,4,196	$\leftarrow_1$ 227 Entering passive
$\leftarrow_1$ 200 PORT cmd successful	mode(132,239,17...)
$\rightarrow_1$ LIST	$\rightarrow_2$ PASM
$\leftarrow_1$ 150 Opening ASCII mode	$\leftarrow_2$ 227 Entering passive
$\leftarrow_1$ 226 Transfer complete.	mode(132,239,17...)
$\rightarrow_1$ MKD Temp	$\rightarrow_1$ STOR README
$\leftarrow_1$ 257 “Temp” dir created.	$\leftarrow_1$ 150 Opening BIN mode
.	$\rightarrow_2$ STOR README
.	$\leftarrow_2$ 150 Opening BIN mode
$\rightarrow_2$ USER al	Data transfer completes...
$\leftarrow_2$ 331 Passwd needed for al.	pc1's death reported...
$\rightarrow_2$ PASS ____	$\leftarrow_2$ 226 Transfer complete
$\leftarrow_2$ 230 User al logged in.	$\leftarrow_1$ (dead)
$\rightarrow_2$ PORT 24,94,3,173,4,197	$\rightarrow_1$ QUIT
$\leftarrow_2$ 200 PORT cmd successful	$\leftarrow_1$ (dead)
$\rightarrow_2$ LIST	$\rightarrow_2$ QUIT
$\leftarrow_2$ 150 Opening ASCII mode	$\leftarrow_2$ 221 Goodbye.

Fig. 5. FTP-M cmd sequence corresponding to Figure 4

Figure 5 shows the FTP-M protocol exchange. The commands directed to pc1 are preceded with $\rightarrow_1$ and those directed to pc2 are preceded with $\rightarrow_2$. The ‘$\leftarrow$’s have a similar meaning. Notice the use of the PASM command. Notice also that pc1's failure during the TCP-M transfer is reported by TCP-M to the application and is handled elegantly.

F. Implementation

We implemented FTP-M on the NetBSD version of ftp and ftpd. The extension of ftp to include the multicast extensions took just around 400 lines of well-placed code. ftpd modifications took around 100 lines of code. We hope

[8] using a standard shell redirection mechanism

this shows the simplicity of our design. For lack of space, we do not mention performance figures except to state that using 733 Mhz Pentium III PCs, file transfer time for a 80 MB file on a 100 Mbps Ethernet increased by around 30% (due to the overhead of TCP-M versus standard TCP) when using two receivers instead of a single receiver; thereafter, file transfer time remained nearly constant up to 12 receivers. By measuring CPU utilization with 12 receivers and extrapolating, we expect that the file transfer time would remain at this value till around 216 receivers at which point the CPU would saturate (the ack fusion overhead of TCP-M grows with the number of receivers.)

As mentioned earlier, the API exported by TCP-M to the application layer was a simple `getsockopt`/`setsockopt` extension to the BSD Socket Interface. This was much not of a deviation from the API used by unicast `ftp` or `ftpd` to interact with TCP. This was another factor that aided the transition from FTP to FTP-M. While several reliable multicast protocols have been implemented with modified BSD Socket APIs ([16] and [5] to mention a few), this API advantage may not be available in the case of an implementation of FTP-M over a general strictly-reliable reliable multicast protocol. We argue that inserting code to use a non-standard API would not add a significant amount of development effort.

program	added	modified
`ftp`	`multi` `lsopen` `run`	`open` `close` `put` `send` `get` `recv`
`ftpd`	PASM	-

TABLE I
Table detailing `ftp` and `ftpd` modifications

Table I summarizes the modifications made to the `ftp` and `ftpd` programs.

V. Discussion

In the preceding subsections, we have seen how FTP-M provides an efficient and elegant way of achieving multicast file transfer by simply extending FTP. Since FTP-M does not directly depend on any particular reliable multicast protocol, issues concerning scalability, response to congestion, deployability are based almost entirely upon the reliable multicast protocol used. It is true that if one wanted to transfer a file to several thousands of receivers, the FTP-M interface may not be appropriate. We intend FTP-M for small to medium scale multicast scenarios. In typical usage scenarios such as fast network backups, web-server mirroring, and software update distribution, we argue that FTP-M would provide a familiar and convenient way of transferring files via IP multicast for hundreds of receivers.

VI. Conclusions and Future Work

In this paper, we introduced FTP-M, which extends the existing FTP protocol and user interface to provide a simple and convenient way of achieving multicast file transfer. FTP-M is general enough to be implemented to use any strictly-reliable multicast protocol. In this paper, we described the implementation of FTP-M using TCP-M, an example strictly-reliable multicast protocol. The basic implementation required less just a few hundred lines of code modifications to FTP client and server, proving the elegance of the FTP-M idea. We hope that that the generality, convenience of use/implementation and familiarity of FTP-M makes it suitable for adoption as a standard application in today's Internet.

One could extend this work by defining a general and convenient API between the multicast file transfer application and the strictly-reliable protocol. One could also implement FTP-M with a variety of strictly-reliable multicast protocols to get a deeper understanding of the application-protocol interdependence.

VII. Acknowledgments

We thank Girish Chandranmenon, Lili Qiu, Geoff Voelker and Joe Pasquale for their insightful comments.

References

[1] S. Armstrong, A. Freier, and K. Marzullo. Multicast Transport Protocol. *Network Working Group, RFC 1301*, February 1992.

[2] S. Floyd, V. Jacobson, S. McCanne, C. Liu, and L. Zhang. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. In *Proceedings of ACM SIGCOMM '95*, pages 342–356, August 1995.

[3] Jim Gemmell, Eve Schooler, and Jim Gray. Fcast Multicast File Distribution. *IEEE Network, Vol. 14, No.1*, pages 55–68, Jan 2000.

[4] Rajib Ghosh and George Varghese. Congestion Control in Multicast Transport Protocols. Technical Report WUCS-98-19, Washington University in St. Louis, June 1998.

[5] Kang-Won Lee, Sungwon Ha, and Vaduvur Bharghavan. IRMA: A Reliable Multicast Architecture in the Internet. In *Proeedings of IEEE Infocom '99, New York, NY*, March 1999.

[6] John C. Lin and Sanjoy Paul. RMTP: A Reliable Multicast Transport Protocol. In *Proceedings of IEEE INFOCOM '96*, pages 1414–1442, March 1996.

[7] J P Macker and R B Adamson. The Multicast Dissemination Protocol Toolkit. *Proceedings of IEEE MILCOM 99*, Nov 1999.

[8] Manamohan Mysore. TCP-M: A TCP-friendly Transport Protocol for Multicast File Transfer Applications. *Thesis, University of California – San Diego*, June 2000. Available at http://mysore.ucsd.edu/~ mmysore/ms-thesis.ps.

[9] J. B. Postel and J. K. Reynolds. File Transfer Protocol. *RFC 959*, October 1985.

[10] I. Rhee, N. Balaguru, and G. N. Rouskas. MTCP: Scalable TCP-like Congestion Control for Reliable Multicast. Technical Report TR-98-01, Department of Computer Science, NCSU, January 1998.

[11] L Rizzo and L Vicisano. Reliable Multicast Data Distribution protocol based on software FEC techniques. *Proceedings of the Fourth IEEES Workshop on the Architecture and Implementation of High Performance Communications Systems, HPCS '97, Chalkidiki, Greece*, June 1997.

[12] Luigi Rizzo. pgmcc: a TCP-friendly single-rate multicast. *SIGCOMM Conference 2000, Stockholm*, pages 17–28, August 2000.

[13] K. Robertson, K. Miller, M. White, and A. Tweedly. Starburst Multicast File Transfer Protocol (MFTP) Specification. *Internet Draft, Internet Engineering Task Force*, April 1998.

[14] Tony Speakman, Dino Farinacci, Steven Lin, and Alex Tweedly. PGM Reliable Transport Protocol. *Internet Draft: draft-speakman-pgm-spec-01.txt*, January 1998. Expires July 1998.

[15] W. Richard Stevens. *UNIX Network Programming, Volume 1, Second Edition: Networking APIs: Sockets and XTI*. Prentice Hall, 1998.

[16] R. Talpade and M. H. Ammar. Single Connection Emulation: An Architecture for Providing a Reliable Multicast Transport Service. In *Proceedings of the 15th IEEE Intl Conf on Distributed Computing Systems, Vancouver*, June 1995.