

IMPLEMENTATION AND PERFORMANCE EVALUATION OF A TCP-BASED  
MULTICASTING PROTOCOL IN A REAL-WORLD ENVIRONMENT

by

JOHN RUSSELL

Advisor  
DR. HALA ELARAAG

A senior research paper submitted in partial fulfillment of the requirements  
for the degree of Bachelor of Science  
in the Department of Mathematics and Computer Science  
in the College of Arts and Science  
at Stetson University  
DeLand, Florida

Spring Term  
2004

## **ACKNOWLEDGMENTS**

Funding for this project and presentation at NCUR 2004 in Indianapolis, Indiana provided by the Dean's Fund at Stetson University.

## TABLE OF CONTENTS

|                         |    |
|-------------------------|----|
| ACKNOWLEDGMENTS .....   | 1  |
| TABLE OF CONTENTS.....  | 2  |
| 1. INTRODUCTION .....   | 4  |
| 2. Related Work .....   | 6  |
| 3. Implementation ..... | 14 |
| 4. Results.....         | 17 |
| 5. Conclusion .....     | 17 |
| REFERENCES .....        | 18 |

## ABSTRACT

Traditionally, multicast data transmission has been accomplished over the UDP protocol. However, there are many advantages to using TCP, including congestion control and guaranteed in-order delivery. There are many current research projects striving to implement multicast over TCP. One example is TCP-SMO, currently being developed at Stanford University. Many TCP based multicast protocols target very large scale applications, conceivably having millions of receivers. This introduces complexity to their design, which TCP-SMO avoids by targeting mid-to-large scale applications having a few hundred or thousands of receivers. As a result, the implementation of TCP-SMO is relatively simple, requiring less than one thousand lines of code to be changed in the TCP implementation. However, much of the research into TCP-SMO has been done in a laboratory environment. In this paper, we propose to study TCP-SMO's performance in a real-world environment. We propose to implement TCP-SMO on a Linux operating system serviced by a broadband cable ISP. Our Linux machine will multicast packets to a number of receivers located across the internet. We can then alter the implementation, i.e., alter queuing algorithms, ACK processing algorithms, and congestion control and avoidance mechanisms to further study and improve TCP-SMO's performance.

## 1. INTRODUCTION

The growth and popularity of the internet is due in large part to the Transmission Control Protocol (TCP). TCP has become fast and responsive to network conditions through such mechanisms as slow start, fast retransmit, and congestion control and avoidance. However, TCP is by its very nature designed to provide communication between a single source and a single receiver. It is not intended to provide data transfer between a source and multiple receivers.

There are several reasons for this limitation. TCP is a connection-oriented protocol. No data can be transferred until the source and a single receiver establish a connection through a three-way handshake. Information unique to that connection, like initial sequence number and maximum segment size is exchanged. Similarly, no data can be sent after a connection is terminated by a two-way handshake. To provide multicast functionality, a source must be capable of handling asynchronous joins and leaves. Also, TCP favors reliability over timely delivery, while many multicast applications, like streaming audio or video, place a high value on timely delivery.

Currently, multicast over TCP can be accomplished by creating a separate TCP connection from the server to each receiver. This technique causes unnecessarily high demand on the server, and causes data duplication as identical information is sent out multiple times. Several methods are currently being developed to address this problem.

The focus of this paper is called TCP-SMO, for Single-source Multicast Optimization. TCP-SMO seeks to provide good performance for multicast data transmission while retaining the benefits of TCP like congestion control, fast retransmit, and in-order data delivery.

The rest of this paper is organized as follows. Section 3 discusses related work in the field of TCP-based multicast. Section 4 presents further detail on TCP-SMO. Section 5 discusses how we intend to implement TCP-SMO and how we intend to change it in order to study its performance. Finally, section 6 presents our conclusions.

## 2. Related Work

In addition to TCP-SMO, many other protocols have been proposed that seek to implement multicast over TCP [LEE99, LIA03]. A good survey and taxonomy of multicasting protocols can be found in [OBR98]. Obraczka [OBR98] proposes a classification scheme that includes features like:

- Data propagation - describes the methods used to propagate data among all the participating sites.
- Reliability mechanism - or how the protocol recovers from lost data.
- Feedback Control - or how the protocol handles the amount of control information generated by the receivers.
- Flow and congestion control - or the mechanism used to prevent the server from overloading the receivers and causing network congestion.
- Group management - or how the protocol handles and organizes membership in the multicast group.

The Reliable Multicast Transport Protocol (RMTP) [OBR98] is one such method. RMTP addresses feedback control by organizing group members into a control tree. This tree then controls feedback propagation and processing. Intermediate nodes in the tree process ACKs from child nodes and buffer data that has already been sent. They can then retransmit data to their child nodes as needed, relieving some of the load from the server.

RMTP allows dynamic joins and leaves at any time during a multicast session, and on the receiver end, knowledge of group membership is not necessary. Congestion control is provided by reducing the transmission window to only one packet if the number of retransmission requests reaches a certain threshold.

The Illinois Reliable Multicast Architecture (IRMA) is a multicast architecture that supports TCP as its transport protocol without modifications to the end host. IRMA guarantees reliable, in-order delivery, ACK-based reliability, and support for end-to-end flow and congestion control. IP multicast is presently accomplished via a virtual network of multicast capable routers called the MBONE. Any multicast routing protocol must generate a directed tree in the MBONE. Data packets are sent from sender to the receivers via this tree. ACKs from the receivers are sent back to the sender along the same tree, but in the opposite direction. With IRMA, ACKs from multiple receivers are aggregated in such a way that the standard TCP concept of cumulative acknowledgement is preserved. That is, each node in the tree forwards an ACK with the minimum of the sequence numbers and an estimated advertised window from the ACKs it receives from downstream. IRMA also preserves the standard TCP algorithms for sequencing, reliability, and flow and congestion control.

Dynamic joins are handled by the intermediate routers. IRMA uses the currently unassigned multicast address 224.0.0.3 to signal a reliable multicast join. When a receiver wants to join a multicast group, it broadcasts a request to join this group. An intermediate router in the multicast tree uses the information in this packet to initiate the joining process. This way, the newly joining receiver does not interact with the source

when joining a multicast group, and it does not receive copies of packets from before it connected, only from the point where it joined the group.

TCP-RTM (Real-Time Mode) [LIA03] seeks to allow applications such as streaming live video or audio to be transmitted over TCP. TCP-RTM is implemented by making changes to the TCP packet reception algorithm. However, TCP-RTM is still intended for one-to-one communication, although it can be used in conjunction with other protocols like TCP-SMO to provide multicast.

## **2.1. TCP-SMO**

As stated, the purpose of TCP-SMO is to allow multicast applications to benefit from the many advantages of TCP. TCP-SMO uses the SSM channel model [BHA01]. A channel is defined by an (S,G) pair, where S is the server address and G is a multicast group address. The server maintains a separate TCP connection to each receiver, as well as a multicast channel that is associated with these connections. Each packet is sent to the multicast address G, and it behaves as if it is sent out on each separate connection. To the server, this appears as n TCP connections (n being the number of receivers), as well as an additional multicast connection.

There are several issues that need to be addressed when going from unicast to multicast. For example, how the server processes ACKs from multiple receivers, how to manage the relationship between the multicast connection and the multiple TCP connections, how to handle group membership, and others. We will now describe in

detail how TCP-SMO handles these issues.

## **2.2. connection management**

For each multicast channel (S,G), a TCP-SMO server creates a master-socket that manages the multicast connection. Additionally, the server creates a child-socket for every individual TCP connection to each receiver. The TCP connections are needed so that a common sequence number can be used for each subscriber.

The master-socket contains a TCP Control Block (TCB) [POS81] that is responsible for managing TCP state variables like sequence numbers, RTT, etc. This is called the master-TCB for the channel. Additionally, each child-socket contains a child-TCB. The master-TCB is responsible for organizing and keeping track of the child-TCBs. When the master-TCB sends a packet to the multicast channel, it informs each child-TCB so they can update their control information. Additionally, each child-TCB receives and processes ACKs, then send it to the master-TCB for further processing.

## **2.3. send window advancement**

For a regular TCP connection, a pointer called SND.UNA (send unacknowledged) [POS81] keeps track of the next sequence number the sender expects will be acknowledged by the receiver. SND.UNA can be advanced when a receiver ACKs that new data has been received. With multicast, the issue of when to advance SND.UNA is

more complex. TCP-SMO provides three options to this problem.

Wait for the slowest receiver. In this case, the master-TCB will advance its send window only when a packet is acknowledged by all receivers in a reasonable amount of time. If a receiver does not acknowledge a packet in this time frame, it is assumed to be too congested and can be removed from the channel.

Specify an ACK-count threshold. For example, the server can specify that if 90% of the receivers have acknowledged a packet, then the master-TCB's send window can be advanced.

Buffer-limited. In this case, when the master-TCB's send-buffer is full, the oldest packet is removed and the send window advanced. The optimal size of the send buffer has not yet been discovered.

The type of data being transferred can determine which scheme to use. For example, reliable data transfer like file distribution calls for a conservative approach to ensure each packet is delivered. These applications are best served by option a from above.

Conversely, for real-time audio or video streams it is more important to keep pace with the real-time rate, so slower receivers should not mandate the send window advancement. Options b or c from above are better suited for real-time applications.

## **2.4. ACK processing**

As a multicast audience grows large, the number of potential ACK packets grows accordingly. This is called an ACK implosion problem, as the server runs the risk of

being bogged down by ACK packets. Fortunately, ACK packets are small (typically 40 bytes), and on high bandwidth networks, even a lot of ACKs will account for only a small percentage of bandwidth. In addition, an Adaptive Acknowledgement scheme can further reduce ACK implosion. A TCP receiver usually generates one ACK packet for every two data packets received. However, after the slow-start phase and if no packet loss occurs, the receiver can generate an ACK once for every four to eight packets received, reducing the ACK burden on the server.

## **2.5. RTT estimation**

By default, TCP-SMO uses the maximum RTT to any one of its receivers as the RTT for the multicast channel. If an RTT to a particular receiver is disproportionately larger than others, the receiver may be dropped from the channel or its RTT ignored. To retransmit lost packets, TCP-SMO has a choice whether to do unicast or multicast transmission. This decision is based on a threshold number that indicates the point where a unicast transmission would use more bandwidth than multicast. The optimal value for this threshold has not yet been determined.

## **2.6. congestion and flow control**

Each child-TCB is responsible for its own congestion window size (cwin) using standard TCP algorithms. The master-TCB derives its cwin from the values of the child-

TCB congestion windows according to configurable options. The default value for the master cwin is the minimum cwin of its children. As with the send window advancement, if a cwin is disproportionately small, that receiver may be removed from the channel.

## **2.7. implementation and socket API**

One of the design goals of TCP-SMO is to make as little change as possible to the TCP API. This makes implementation and deployment of TCP-SMO much easier. A TCP-SMO server is constructed in much the same way as a standard TCP server. A new socket option must be used to specify the multicast channel parameters, and a receiver must subscribe to the multicast channel before connecting to the server. Specifically, the server must complete the following steps in order to begin multicasting as a TCP-SMO server:

1. Create a TCP socket called the listening-socket using the standard system call `socket()`.
2. Call `bind()` to bind the socket to (source address, source port).
3. Impose a new socket option, `TCP_SMO` on the listening-socket to create a new socket called the master-socket that contains a master-TCB that is responsible for the multicast TCP parameters. This socket option also provides the master-socket with the appropriate multicast address and port as its destination.
4. Call the standard `listen()` and `accept()` on the listening-socket to wait for incoming requests.

5. When the server receives a TCP SYN packet from a receiver, it creates a TCB and replies with a SYN/ACK. This TCB contains a pointer to its parent, the master-TCB. The child-TCB is added to a table in the master-TCB, which helps the master-TCB manage the various child-TCBs.

6. When the server receives the ACKs from its receivers, the 3-way handshake is complete, and the connections to each receiver are in place. The server can then send data to the master-socket, which is in turn multicast to the receivers.

7. ACKs from the receivers are sent to their respective TCBs, and are then forwarded on to the master-TCB for further processing.

8. Connection tear-down is similar to standard TCP connection tear-down. The only extra step is to remove the leaving connection's TCB from the master-TCB's child list.

The following steps must be completed by a receiver wishing to join a TCP-SMO session.

1. Create a standard TCP socket.
2. Subscribe to channel (S,G) using standard socket options.
3. Send SYN with 4-tuple (server address, server port, receiver address, receiver port).
4. Receive SYN/ACK, reply with another ACK, set up TCB.
5. The receiver can now begin receiving packets from the server. After data transmission is finished, the receiver can close the socket and unsubscribe from the channel.

### **3. Implementation**

Because implementing TCP-SMO involves making extensive changes to the TCP networking code, all code is implemented in the Linux kernel. The server was written on a system running Mandrake 9.2, kernel version 2.4.22, with an 800 mhz processor. Clients were implemented on various Linux versions including Redhat and Mandrake using kernel versions 2.4.25.

According to the specifications in LIA02, a new socket option is defined at the TCP level. This option, TCP\_SMO, is used as a flag to indicate to the kernel that special handling is needed for any traffic associated with the socket. Alternatively, a data structure can be defined and used to pass various options to the kernel to further improve the robustness and flexibility of TCP-SMO. These options might be used to indicate how the kernel should handle disassociating slower receivers, or a minimum number of clients needed before transmission will begin.

When the TCP\_SMO socket option is called, a new socket is created. In kernel space, this is done with similar function calls as those used when creating a socket in user space. The purpose of this socket is to control the state variables associated with the individual connections. Each client that participates in the multicast will have its own TCB, but will share the read buffer and state information from the master-TCB. To accomplish this, we created structures representing the master-TCB and child-TCB in a new header file called

tcpsmo.h located in /usr/src/linux/include/net/. Additionally, we define functions to get and set the sequence number, snd\_una, and other TCP state variables. These functions are implemented in a file called tcpsmo.c located in /usr/src/linux/net/ipv4/.

Traditionally, a TCP server application will send packets to each new client that connects to it. With TCP-SMO, the server will only send one packet addressed to the multicast group. To accomplish this, several fields must be manipulated in the TCP and IP header. In the TCP header, the source port must be changed to the port associated with the multicast group. Currently we implement this with a module that is insertable into the kernel. The IP header contains the destination IP address. This must be changed so that the packet is addressed to the multicast group. This is similarly implemented with a module.

On the client side, the TCP multicast packets were being received by the hardware device, and processed by the kernel at the IP layer, yet were not being delivered to the client application. Originally we thought this was due to the fact that the packets had a class D multicast address in the IP header. We thought that this would introduce a problem with routing, as we were not sure that the kernel would know how to properly handle a TCP packet addressed to a multicast address. However, the functionality for mapping multicast addresses to listening applications is handled at the IP level, so the proper routing functions were still called.

Although the TCP and IP header checksums were being recalculated by the server prior to transmission, the problem on the client side still seems to involve a checksum. We were eventually able to trace the packets to the function `tcp_v4_check`, located in

/usr/src/linux/include/net/tcp.h. this function returned a value which caused tcp\_v4\_rcv (located in /usr/src/linux/net/ipv4/tcp\_ipv4.c) to drop the packet. This issue has not been resolved at the time of this writing.

Additionally, a problem was encountered when implementing a client on a Windows computer. An error occurred when setting the socket option IP\_ADD\_MEMBERSHIP on a socket of type SOCK\_STREAM. The socket option is defined at the IP level, yet on two different Windows platforms, the error occurred, leading us to believe that the Winsock API will not natively support TCP-SMO.

## **4. Results**

Due to the problems with implementing this project, there are no results to publish at this time. Before results can be gathered in a real-world environment, the issues involving the Windows socket implementation must be resolved. Because our implementation differs from the one outlined in [LIA02], we feel that further work into the area of TCP based multicast at this university can yield a unique implementation that can be included in future version of the Linux kernel.

## **5. Conclusion**

Much future work still remains to adequately implement and test TCP-SMO in a real-world environment. In the Linux kernel, different queueing algorithms can be included at compile time, and these algorithms may improve performance. Scalability is an important issue, and has been addressed in [LIA02] in a lab environment, but still remains to be tested in the real world.

## REFERENCES

- [BHA03] Bhattacharyya, Supartik, et al. "Internet Draft: An overview of Source-Specific Multicast (SSM) Deployment", 2001
- [LEE99] Lee, Kang-Won and Ha, Sungwon and Bharghavan, Vaduvur "IRMA: A Reliable Multicast Architecture for the Internet", IEEE 1999
- [LIA02] Liang, Sam and Cheriton, David "TCP-SMO: Extending TCP to Support Medium-Scale Multicast Applications", IEEE 2002
- [LIA03] Liang, Sam and Cheriton, David "TCP-RTM: Using TCP for Real Time Multimedia Applications"
- [OBR98] Obraczka, Katia "Multicast Transport Protocols: A Survey and Taxonomy", IEEE Communications Magazine, January 1998
- [POS81] Postel, J "RFC 793: Transmission Control Protocol", Sep 1981