Efficient reliable IP multicasting

Michael Henderson, *University of Missouri – Columbia***, Adam Ryan,** *Truman State University***, Dr. Haibin Lu,** *University of Missouri – Columbia***, Dr. Wenjun Zeng,** *University of Missouri - Columbia*

Abstract

Unicast connections are a single connection between two computer hosts. While they may work well for one-to-one communication or non-real-time applications, these connections may not be as efficient for one-to-many or many-to-many communications. One alternative to this is IP Multicast, which allows efficient one-to-many and many-to-many communications. The focus of our research was IP Multicast, which is mainly used for one-to-many connections. One of IP multicast's weaknesses is its lack of reliability due to its use of the User Datagram Protocol (UDP) for data transmission. Our project focused on IP multicast in a wireless (802.11 b/g) local area network (WLAN) environment. We hoped to find more about reliably multicasting so to specifically address the problems with IP Multicast and possibly find the solution to such problems to ensure more efficient information exchange between computers using such programs as Classroom Presenter and ConferenceXP. Using Microsoft's .NET platform and the C# programming language, we wrote two programs; a server (sender) and a client (receiver), which uses IP Multicast to communicate simple strings of text with a pseudo-NACK implementation. Our test bed consisted of eight desktop computers which had the Microsoft Windows XP Professional operating system and Microsoft's .NET Framework version 2 installed. We ran simple performance tests on both wired (LAN) and wireless (802.11b/g) connections. While our final results show that IP Multicast is an efficient method of transmitting data, it is still unreliable over wireless networks even in small scale settings. The problem with IP Multicast is the lack of feedback mechanism in the overall procedure. This creates reliability and scalability issues making it hard to ensure that data is properly transmitted from both ends. The use of extra protocols/algorithms is recommended when reliability is in need in an IP Multicast situation.

Table of Contents

I. Introduction

Most network connections today are *Unicast* (one-to-one) connections. They are basic and can be used for reliable data transmissions back and forth between the two connected nodes. However, these connections are not appropriate for communications from one sender to multiple receivers (one-to-many) or for many senders to many receivers (many-to-many). *Multicast* connections, particularly IP Multicast connections, may be more appropriate. IP Multicast connections may be particularly useful for one-to-many communications, as the sender need only transmit data packets once, and multicast-enabled routers will copy the packets and send them to joined receivers. As the standard is implemented, there is no need for a sender to know who the receivers are, nor vice-versa. Rather, senders and receivers need only know the multicast IP address (from the D block of IP addresses) and the port that will be used.

Barring loss or corruption of the data, multicasting can more efficiently use network resources in many situations. These could be such things as discovering resources and routers, multimedia conferencing, serving video or other multimedia to many receivers, distributing the same files or data to many receivers at once, and many other situations where data is needed by many receivers in a short period of time.

With the rise in popularity of wireless networks, including both IEEE 802.11 (WiFi) and mobile (cellular), efficient use of network and host resources is important in order to both enable maximum battery life and the most use of network bandwidth/other resources available. Multicast has its uses in wireless networks as well, although as we will see later on, it also presents its own set of challenges.

In this paper, we focus specifically one the use of IP Multicast for one-to-many communications in local area networks (LAN), both wired (Ethernet) and wireless (IEEE 802.11). We examine IP Multicast in light of making transmissions both efficient and reliable.

Topics that are beyond the scope of this paper include multicast-related security, multicasting in ad hoc networks, many-to-many multicast applications, and interdomain multicasting. While the authors believe that these are engaging topics, to discuss them would detract from this paper.

II. Troubles with IP Multicasting

Though IP Multicasting can be an efficient way to quickly distribute data to many receiving hosts, it has one great weakness: it is unreliable.

IP Multicast is unreliable due to its use of the User Datagram Protocol (UDP) [1], which unlike the Transmission Control Protocol (TCP) [2] used on the World Wide Web, lacks any sort of data reliability methods built in. UDP is a best-effort protocol. Native features of TCP like sequence numbering and ordering of packets, receiver acknowledgements of successful packet reception, and retransmission of lost packets are not included for UDP. This reduces the required overhead for each packet, and also allows for IP Multicast's feature of receivers and senders not needing to know one another, but may become an Achilles' heel when packets are lost, delivered out of order, or corrupted, as the transport layer provides no assistance.

Wireless networking adds the potential for even more problems. Due to the use of radio waves for transmissions, these networks are susceptible to background noise and interference, which can cause packet corruption or loss. Furthermore, with a more limited channel capacity than wired networks, efficiency is important so as to maximize the utility of the network for all of its users. Therefore, multicast is less robust in a wireless than a wired network, thus needing additional considerations.

It is therefore up to programmers of multicasting applications to decide on their needs for reliability. For those implementing applications where delays are not tolerable but some packet loss is, such as conferencing and other real-time uses, such as live streaming, some of the methods we will describe for adding reliability to multicast may not be useful, as retransmission cannot be an option; rather, said application would need to move on and wait for the next packet, ignoring the lost packet. For others who can tolerate some delay, and for those whom reliable data delivery is more important, there are several different options and methods to implement to attempt to increase the reliability of IP Multicast. Besides various proposed and implemented protocols (discussed later in the Related Works section of this paper), particularly for routing, there are application-layer solutions. These include adding sequence numbers to packets, adding error checking and correction capacities, positive and/or negative feedback schemes, and retransmission schemes. Such solutions come at a cost, however. These costs may take the form of any or all of the following: increased packet sizes due to additional data sent per packet for increased reliability; more application overhead at the sender and/or receiver in order to process the extra information that may be sent with packets; additional delays due to feedback and retransmissions; and, most counterproductively of all, less scalability to large numbers of users.

III. A Brief Discussion of Different Reliability Schemes

We would like to take a little closer look at some of the previously-mentioned

application-layer reliability-enhancement schemes. We will view them in terms of their costs versus their potential benefits.

The first method a programmer might use to improve the reliability is some form of error detection/correction scheme. Whether using hash code checks or Forward Error Correction (FEC), these methods can be very useful in helping a receiver detect, and, in the case of FEC, correct, errors due to transmission (such as bit flips). They might be appropriate for uses where receivers must receive data with no errors. If the program only uses error detection, then it must be paired with a packet sequence numbering method as well as a retransmission scheme. Either way, though, these error-recovery schemes create high overheads for both the processing to create, decode, and compare the checking data as well as the additional packet overhead. If small errors are permissible in the application (such as when sending simple text messages, or when such an error might distort a frame of video), we recommend that programmers pass in order to keep a much higher efficiency.

Adding sequence numbering to packets is a simple, and often necessary, technique that the programmer can utilize. By merely appending the data with a short number at the sender, and then parsing it out on the receiver's end, clients will be able to have a reference for other schemes, such as feedback and retransmission requests. Although extra considerations based upon the size of the number of packets to be sent may be needed, we recommend programmers add packet sequence numbers to their multicast applications if they need to improve reliability, though sequence numbers alone will do nothing to improve reliability.

Feedback schemes, whether positive acknowledgements (ACKs) or negative acknowledgements (NACKs), can be useful for the sender to know whether packets are successfully being received.

With an ACK-based setup, sequence numbering is theoretically not necessary, but still recommended. While an immediate response to each packet received can be done without knowing a packet number, there is no way for the sender to know which packet a receiving is acknowledging, unless the sender waits after transmitting each packet. Such a scheme is somewhat pointless, however, if the sender is not keeping track of all the different receivers, a difficult task not necessary to implement IP Multicast. The sender needs to know which receivers have not received the packet so it can resend it to them. However, there is always the problem of knowing who is in a receiving group, both before beginning as well as during the whole session, as receivers can drop membership from a multicast group without the sender's knowledge. Many different approaches to allow such discovery have been proposed [3, 4, 5, 6]. On the flipside of this problem is another: how do receivers know where to send their acknowledgements? If they use the multicast address, there is the risk of flooding the group with useless ACKs and blocking data from the sender. If receivers are to respond to the IP address of the sender, they must know it beforehand, something not required for normal IP Multicasting. If Source-Specific Multicast (SSM) is being used, however, receivers will already know this and therefore the application can easily put it to use. (For more on SSM, see [7, 8]). However, there is another problem in having potentially a large group of receivers trying to acknowledge receipt: they will overwhelm the sender with their responses. Thus, the scheme does not scale well.

Although there are various proposals on ways to reduce the number of ACKs needing to be sent ***(see more in the "Related Works" section)***, we feel that due to both the need for the sender to know all receivers as well as the scaling issue, that ACK-based feedback schemes are not useful beyond small, controlled groups of receivers, such as in a closed LAN or a small business, where senders can easily check and maintain a record of all receivers.

An alternative to the positive feedback mechanism is the negative acknowledgement (NACK) method. NACK implementations generally require sequence numbering of packets, in order to allow receivers to keep track of the packets received. In a most basic form, a receiver would keep track of the sequences numbers against an internal counter of its own, making sure it receives all packets. If a packet is missed, the receiver sends a NACK to the sender, and the sender will retransmit. Programmers might wish to have the receivers include the sequence number of the expected packet in the NACK transmission in to ensure that the sender knows which packet needs to be resent. More robust applications would also have receivers watching for packets that are correctly delivered but are merely out of order (as is possible when using UDP). As with the use of ACKs, a NACK scheme has the problem of how receivers will communicate back to the sender. Again, sending to the multicast address could flood the group and interfere with the sender, causing further mayhem. Likewise, sending directly to the sender requires knowledge of the IP address, just as with the ACK scenario. With a few adaptations, a NACK scheme is more viable feedback mechanism to use for multicasting and may be quite helpful for reliability, worth its costs of implementation. As discussed later in this paper, in our experiment we created what we dubbed a "pseudo-NACK", a rudimentary, application-layer NACK scheme, without any retransmission methods built into the sender.

Finally, there is the use of retransmission mechanisms by the send to help receivers recover lost or corrupted packets. Without such a mechanism, the only of the other groups of schemes for reliability enhancement that would have any purpose is the error correction methods. If a sender is not planning on doing retransmissions, there is little purpose for implementing a feedback mechanism. Assuming that some method has been used to inform the sender what packet(s) to resend, a programmer needs to only create a cache or store of previously transmitted packets, from which it can retrieve the needed packet. The specific issue for the programmer is to consider how *long* to store a packet or alternatively, how *many* packets to store after sending. At one end, the sender might only store the last packet sent. This is not likely to be very robust if there are several router "hops" between the sender and receiver(s). The other end is of course to save everything transmitted during the session, which can put a tax on system memory (RAM, hard drives, or both depending on how the programmer decides to keep the store). The optimal length/size of this store depends on the specific situation, so we recommend careful testing and adaptation as needed in order to find the best balance between reliability and efficiency.

In light of these issues and trade-offs, we performed a small experiment in creating our own IP Multicasting program, dubbed "MCast".

IV. Assumptions

We kept the following list of assumptions in mind while developing our test program for this study:

1. *There is no outside interference on the network.*
2. *All servers and clients that are running the program will be on the same Local Area Network (LAN) or Wireless Local Area Network (WLAN).*
3. *All servers and clients will never be on a mixture of LAN or WLAN connections.*
4. *Network Address Translation (NAT) is not affecting the IP addresses of the servers or clients.*

V. The Test Bed

The test bed that was used in this study consisted of eight computers with the Microsoft Windows XP Professional operating system, Service Pack 2; the Microsoft .NET Framework 2.0; and the most recent version of our test program.

VI. Our Attempts: "MCast"

The original goals of this study were to research multicasting and implement it into a program called "Classroom Presenter". However, the second version of Classroom Presenter had already done so by the time we were in the design phase of our project. It was decided that in order for us to monitor a simple multicast connection, we had to write our own program and test it on a wireless network as well as a wired network. A sample program in one of our reference books, "Multicast Sockets: Practical Guide for Programmers" by David Makofske and Kevin Almeroth, is the base of MCast which we then built upon it in order to meet our needs.

Since the emphasis of this research project was the efficiency and reliability of IP Multicasting, we had to design a program to measure both aspects in an experimental setting. We developed MCast using the C# language in Microsoft's Visual Studio 2005. After researching numerous methods of multicasting over both wired and wireless networks, we found the simplest method to be "IP Multicasting". IP Multicasting consists of one or more servers sending information to a multicast address (an IP address within the range of 224.0.0.0 to 239.255.255.255) [1, 2] and clients who are listening for data to be transmitted to them. Such connections are known as one-to-many or many-to-many connections depending on the number of active servers and clients interacting with the multicast address.

The sender half of MCast (MCSend) was designed so that it would be able to send valid strings of text to the multicast address as well as the multicast port. The port number on the sender and receiving end was initialized to the same number to ensure the fastest delivery of each packet. Although this was a good idea at first, one problem did arise in which senders had a difficult time sending data to and listening on the same port. Sockets were used for the multicast connection due to its compatibility with IP multicast. Each string of data that is sent over the multicast address is appended with a sequence number (a simple counter we called the "pCount" initialized at zero) at the end which the receiver uses to compare with its own sequence. Whenever a message was sent, pCount would increment by one until the program was terminated by the user. If a receiver failed to properly receive a packet, then it would send a NACK message to the sender containing the IP address(es) of the receiver(s) which didn't receive the packet.

Instead of having the receivers send an acknowledgement (ACK) message every time it successfully received a packet from the sender, it is much more efficient for the receiver to tell the sender when it doesn't receive the packet as this is a less likely event to happen in most multicast connections. More efficient in this case would mean that less bandwidth is used since fewer receivers are communicating information back to the sender at one time. A timer was also implemented to measure the time it took MCSend to send a packet to the multicast address. We planned on measuring the time it took for MCSend to send a message and the receiving end to print it out, but was eventually thrown out because it didn't fit in with our experiment. All of the information that is displayed on the console (messages sent, timestamps, etc) as well as the sequence number at the end of each message were saved to a text file named "mc_send_log.txt" which was later used for compiling and interpreting results.

MCast's receiving end (MCReceive) connected to the multicast address and listened for any data sent to it. Upon detecting any packets being transmitted to the multicast address via the sender program, it retrieves the data and prints it out to the screen. However, if a receiver does not properly receive a packet, then it will fall out of synch with the sender and begin sending NACK messages to the sender via a Unicast connection which is opened when the program starts until the user exits MCReceive. A socket was opened and used to create the connection just like MCSend did. When a receiver first joins the multicast group, its counter is automatically synchronized to the same number that the sending end is on. It does this by stripping off the sequence at the end of the message and stores it in the pCount variable while printing out the message as usual. The

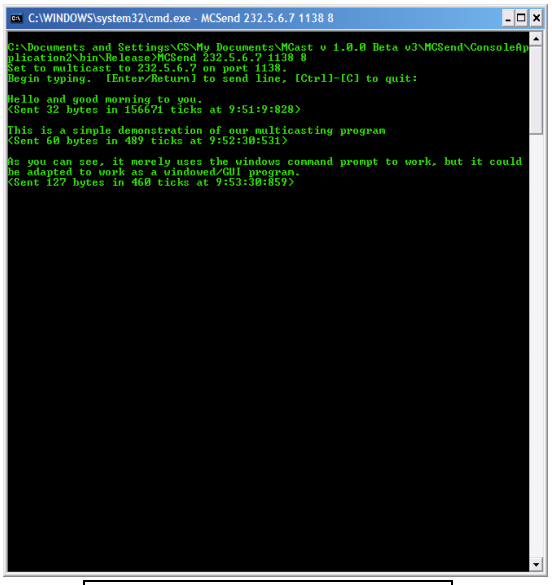

Fig. 1 - MCSend in operation

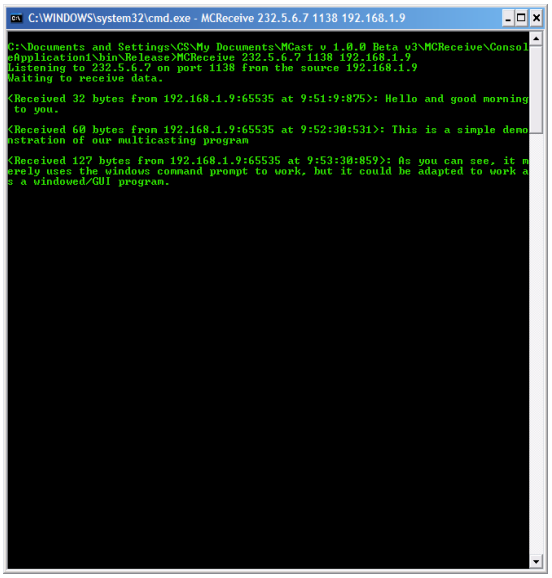

Fig. 2 - MCReceive in operation

program contains one method to do both at the same time. This counter then increments independently with each packet received from the multicast group address, and compared to the sequence number attached to each additional packet received (which is stripped off each message and stored in a separate variable for the comparison). Since all of this is happening in a "while loop" *(a*

block of code that repeats itself as long as the given conditions are true and breaks at the end of the cycle that it is on if said conditions become false). The following pseudo code describes the method behind the counter:

```
While the user isn't done
{
        //Some Code Here

        pCount = update pCount method;

        if(sender sequence != pCount)
                Send NACK message;
        Else
                Print message;

        Increment pCount;

        Raise a flag which make the update pCount
        method return pCount;
}

Update pCount method
{
       If(flag is down)
            pCount = number appended to the  most
            recent packet received;
       Else
           pCount = pCount;
}
```

The only problems that arose from this implementation is if a receiver joins the multicast group and fails to receive the first packet that was sent to it, then no NACK message is sent and the program behaves normally as if nothing happened. Also, for reasons still unknown, MCReceive printed out the wrong address that it was supposed to. Instead of printing out the multicast address and port as intended, it prints out the Unicast connection IP address and hard coded port which were used to send the NACK messages to the sending side. As MCSend saves all of its information within a text file, MCReceive does the same with the only difference being the name of the file which it saves to ("mc_receive_log.txt").

In order for the program to work, the appropriate command line arguments need to be inserted after the program name (MCSend, MCReceive) or else an error will print out instead of the startup message instructing the user the proper input. These are the command line arguments which each end uses:

For MCSend: A multicast address, multicast port and time to live (the amount of hops a packet travels through the network before it is discarded and ignored) need to be present.

Ex. MCSend 232.5.6.7 1138 2

For MCReceive: The same multicast address and multicast port need to be present as well as the IP address of the sending end which is used to transfer the NACK messages from the receiver to the sender.

Ex. MCReceive 232.5.6.7 1138 192.168.1.9

Due to time constraints and the focus of the project, we have decided not to implement the following list of features in our program (although it is very much capable of handling them):

1. *A way for MCSend to retransmit any packet that was not received any of the clients.*
2. *A more precise measure of time between the packet being sent and it being received. Instead of the measure being in ticks, fractions of a second could be used instead.*
3. *A Graphical User Interface (GUI) – This would make the program much more appealing to customers if we were developing this for commercial use.*
4. *Better efficiency – the code used in the program does what it was intended to do. However, some parts can be redone to ensure maximum efficiency.*
5. *Fix current bugs – though the bugs present in the current version of the program do not*

interfere with the performance, it is something that will need to be addressed in future versions of the program.

VII. The Known Bugs of MCast

There are a few known problems within the program which we have addressed but haven't fixed at this time. Perhaps in the near future, should this program be implemented in future project, these bugs can be dealt with as well as any unknown problems in the program. The following list describes the known bugs in MCast:

a. There is no way for MCSend to retransmit data if a MCReceive client fails to properly receive a packet.

b. MCSend prints out the wrong port which it receives the NACK message on. Instead of it print out "65535" like it should, it prints out the multicast port that the messages are being sent over.

c. MCSend and MCReceive cannot be run on the same computer if there is more than one client connected to the multicast group. This causes the receiver (the one that is playing both roles) to send a NACK message within a few messages.

VIII. Testing on the Network

The testing phase for the project only had one test with multiple runs for a variety in data rather than multiple tests with multiple runs. We felt that a real life test over a real network would be the best way to monitor our program's performance. Simple stress tests were performed over the network to ensure the program was working correctly and there were no problems with it which would create inaccuracies with the results. One of the test bed computers was designated as the sender (server) while the other seven were assigned the role of receiver (client).

The original plan was to have the sender also be a receiver, but the idea was later discarded due to difficulties in which the sender was having trouble relaying the messages to itself via the same port it was communicating on. During the initial testing phase, a fraction of the total number of the computers was used for each run (starting with two receivers then adding two more for a total of four then adding in the remaining clients). It was then decided to just use all of the computers in each run instead of starting out with a few and add more in because there was no difference in the time it took for the transfer of packets from the sender to the receiver.

The goal of our test was to see how long it took for all receivers to send NACK messages to the sender at once. If a receiver became out of synch with the sender then it would send the NACK message to the sender, and the packet number of that packet was recorded to tell how many receivers were not receiving messages at that time. When all seven of the receivers sent a NACK message to the sender then the test was terminated and reset for the next run. One of the reasons why we implemented such a test was because we lacked the necessary resources for a large ($N > 30$) scale study. While we could have used additional computers outside the test bed, we also lacked the necessary amount of participation to do so. A more detailed procedure is given later in this section should anyone wish to do this experiment themselves.

Both wired (LAN) and wireless (WLAN) networks were used during testing so we could compare the difference in performance between the two. Our main concern, however, was the results of the wireless network since we were seeking to create an environment similar to that of a classroom setting using Classroom Presenter. An arbitrary number of runs of the test were first performed on the wired network followed by the wireless network. The only data which we were interested in was that from the MCSend program since it contains how many NACK messages are being sent to it and the packet number it was on at the time. The results from each run were compiled into a single spreadsheet and sorted by the pCount then the amount of NACK messages which were present at the time. Once all of the testing was complete then all of the results were compiled into a master spreadsheet which contains information from the wired network runs as well as the wireless network runs.

The following is a more detailed version of the procedure we used for the test:

Before any of the testing began, we made sure that each computer had the latest version of the MCast program (previous implementations aren't compatible with the most recent version). Then all of the computers were either switched to the wired or wireless network in the room depending on which type was being tested at the time. The Command Console was opened on all terminals and awaited for its respected side of the program (MCSend or MCReceive) to be started with all the right command line arguments (IP addresses, ports, time to live, etc).

First, the sending end opened up the multicast connection and was ready to send messages to the multicast address regardless whether or not any receivers were listening in on the connection. Then all seven receivers connected to the multicast address and waited until the sender sent a message to them via the multicast connection. A couple of packets were sent over the network in order to tell whether or not the program was working correctly. Then messages of different sizes (measured in bytes) ranging from the smallest size available (a blank message) to the largest size (255 characters) allowed across the connection. If no receivers were giving into the stress then a constant stream of messages were then sent over the network until all seven receivers were sending NACK messages at the same time. The sending end recorded all of the messages it sent (with the sequence number still attached to it) as well as all of the NACK messages it received from the receiving members of the multicast group via the thread the sender is listening to. The results from each additional run were appended at the end of the file with a time stamp to separate one run from the other. The receiving end results were only used to ensure continuity between runs.

IX. Results and Observations

After a sufficient amount of data was collected and compiled, it was put into a scatter plot in order to show the differences between the program's performance on the wireless network versus the performance we observed over the wired network. The main reason why the wired network was included in this study was because we needed a reliable medium to compare the wireless network to (which is inherently unreliable and unpredictable). The final results were put into a scatter plot because there was no way to average the amount of receivers sending NACK messages to the sender at the time (measured by pCount). A range of results was thrown out because it was either constant or redundant data. The amount of messages sent during the wired connection runs greatly outnumbered those sent during the wireless connection runs. Once we reached a certain point during the wired connection test in which it was inevitable that no receiver would be sending a NACK

anytime soon, the test was terminated and recorded it as a perfect run. In order for the comparison to be even, we also took out all of the data points from *X = 101* and above from both sides.

According to the scatter plot (Figure X), the runs done on the wired network show a perfect performance every single time (i.e. none of the receivers were dropped during the entire run). This shows that the wired connection was reliable for the number of users it was handling during the testing phase. While the wired connection results show a stable trend, the wireless connection results show something completely different. Looking back at the scatter plot, there are data points in a much greater variety of places which show that the connection itself was unpredictable in the sense that it was not known when a receiver would send a NACK message to the sender.

How can this be applied to IP multicasting? The advantage of IP multicasting over a wireless network is that it takes up less bandwidth of the connection. Initially this type of connection will take up more bandwidth than a Unicast connection, but each additional member of the multicast group will take up less bandwidth than another Unicast connection. Since IP multicasting currently has scalability issues in which the reliability of the transferred data diminishes as the number of members in multicast group grows. Implementing this type of multicast connection over a wireless network only compounds the reliability issues the connection already has from the beginning. What makes this method so appealing is its use of UDP packets where each packet can travel independently of each other unlike TCP packets which require a more direct approach with the transmission of the information. One of the reasons why IP multicasting is so unreliable is because it

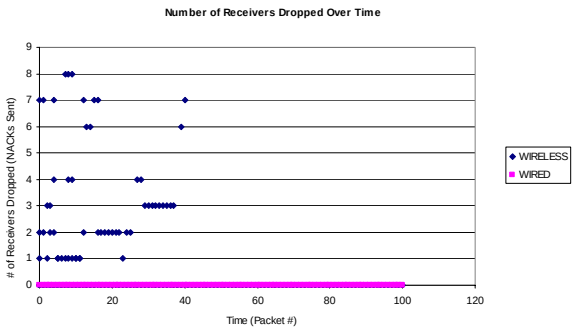

Fig. 3 – Final test results

uses UDP packets which are also unreliable in nature. [1, 2] While there are more methods of multicasting that exist, this one is the easiest to set up and monitor.

Currently the second version of Classroom Presenter makes use of IP multicasting. Even though this program has more effective and efficient algorithms than MCast, the scalability issues still pose a threat to the program's performance due to the wireless multicast connection it mainly utilizes.

X. Related Work

Besides the "simpler" methods for creating reliable multicasts discussed earlier, various other methods have been proposed and suggested. We will briefly examine a few of them.

Of the many approaches we came along in our research, there was a common theme in many of them: some form of additional grouping. The first of these approaches is the grouping with leaders approach. In it, receivers are divided into different groups, and assign them leaders. Members within these subgroups communicate with the leader, and in turn if retransmission is required, the leader communicates for the whole group to the sender [9]. A slight alteration on this

approach is to have the leader maintain a cache and to have it do the resending instead of the original sender [3]. While a leader-based approach was found to be more effective for preventing problems within a controlled wireless environment [10], such solutions require extra work in software in order to assign leaders and have them maintain information on their peers inside their subgroup.

Another similar scheme is the tree-based ACK approach, where again receivers are grouped, except there is a more hierarchical structure, and nodes only report up to the next level of the hierarchy [11]. As was pointed out, this tree method faces problems if the members within it change—it is not very dynamic. Furthermore, special protocols are required to manage these trees for different multicast sessions, as was pointed out by the authors.

Grouping is also popular for methods of data rate and congestion control. For rate control, receivers are grouped according to their maximum possible throughput rates, in order to allow the sender to cater more specifically to their network capabilities without overwhelming any buffers [12, 5, 13, and 14]. While this might be effective, to implement such a procedure requires a great deal of planning and testing to ensure that it works properly, or the programmer may only aggravate matters for the worse.

There is one final use for grouping receivers together: to keep from overwhelming the sender [6]. Such a set up might employ multiple multicast addresses for the different groups, and further hierarchies underneath. Our comments, again, are to plan and test carefully.

We also came across an interesting approach that did not involve grouping receivers together in any way. Rather, it was a suggestion specifically meant for wireless multicasting situations [15]. The authors proposed using two channels for wireless multicast—one for messages, and one for sending busy tones to notify that the message channel is not clear for sending. There are two flaws to this approach: first, it requires tying up a channel that could be used for transmitting data instead of sending a busy signal; and second, this approach, as the authors indicated themselves, is not really compatible with the IEEE 802.11 standards. We feel that programmers might be better advised to pursue reliability via other methods.

XI. Conclusion

While IP multicast remains to be the most popular method of multicasting on the Internet today, much improvement is needed to ensure maximum efficiency and reliability for data transfer, security, and other such aspects of networking. Through testing and results, we have concluded that IP multicast can be reliable over a LAN connection for a small scale ($N < 30$) commercial setting. However, it still remains unreliable and unpredictable over a WLAN connection for the same amount of clients in a multicast group. However, it is safe to say that it is more efficient than Unicast connections in the same environment. This concept has been reiterated throughout this paper and still holds true to today's standards. Different methods still need to be researched and developed before multicasting technology is perfected to a point where minimal data loss is present and the proper error correcting algorithms are in place.

XII. Acknowledgements

This project would not have been at all possible if it was not for Dr. Wenjun Zeng, Dr. Haibin Lu, the University of Missouri – Columbia, and the National Science Foundation for heading up the REU Site for Home Networking Technologies program within the Computer Science department. The academic advisors of the students who participated in this project are also acknowledged because it was them who informed the students about the research opportunity. We would also like to thank the ACM, IEEE, and IETF for all the resources that were used in this paper. Finally, a thank you to the Microsoft Corporation for the non-restricted, nonprofit use of their programs.

XIII. References

[1] "IP Multicast," *Wikipedia*, Wikimedia Foundation, Inc., 2007. [Online]. Available: http://en.wikipedia.org/wiki/IP_Multicast [Accessed July 16, 2007].

[2] "Classful Network," *Wikipedia*, Wikimedia Foundation, Inc., 2007. [Online]. Available: http://en.wikipedia.org/wiki/Classful_network [Accessed July 16, 2007].

[3] Bartoli, Alberto. "Group-based multicast and dynamic membership in wireless networks with incomplete spatial coverage," *Mobile Networks and Applications*, vol. 3, no. 12, 1998. [Online] Available: The ACM Digital Library, http://portal.acm.org/ [Accessed July 16, 2007].

[4] Auerbach, Joshua, et al. "Multicast Group Membership Management," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, February 2003. [Online] Available: The ACM Digital Library, http://portal.acm.org/ [Accessed July 16, 2007]

[5] Deb, Supratim, and R. Srikant. "Congestion Control for Fair Resource Allocation in Networks with Multicast Flows," *IEEE/ACM Transactions on Networking*, vol. 12, no. 2, April 2004. [Online] Available: The ACM Digital Library, http://portal.acm.org/ [Accessed July 16, 2007]

[6] Ammar, Mostafa H. "Probabilistic Multicast: Generalizing the Multicast Paradigm to Improve Scalability," *INFOCOM '94. Networking for Global Communications. 13th Proceedings IEEE*, June 1994. [Online] Available: IEEE Xplore, http://ieeeexplore.ieee.org/ [Accessed July 16, 2007].

[7] Holbrook, H., and B. Cain. "RFC 4607: Source-Specific Multicast for IP," *RFC Index*, The Internet Engineering Task Force, August 2006. [Online] Available: The RFC Database, http://www.rfc-editor.org/ [Accessed July 16, 2007].

[8] Bhattacharyya, S. "RFC 3569: An Overview of Source-Specific Multicast (SSM)," *RFC Index*, The Internet Engineering Task Force, July 2003. [Online] Available: The RFC Database, http://www.rfc-editor.org/ [Accessed July 16, 2007].

[9] Kuri, Joy, and Sneha Kumar Kasera. "Reliable Multicast in Multi-Access Wireless LANs," *Wireless Networks*, vol. 7, no. 4, August 2001. [Online] Available: The ACM Digital Library,

http://portal.acm.org/ [Accessed July 16, 2007].

[10] Dujovne, Diego, and Theirry Turletti. "Multicast in 802.11 WLANs: An Experimental Study," *Proceedings of the 9th ACM international symposium on Modeling analysis and simulation of wireless and mobile systems*, 2006. [Online] Available: The ACM Digital Library, http://portal.acm.org/ [Accessed July 16, 2007].

[11] Levine, Brian Neil, David B. Lavo, and J. J. Garcia-Luna-Aceves. "The case for reliable concurrent multicasting using shared ACK trees," *Proceedings of the fourth ACM international conference on Multimedia*, 1997. [Online] Available: The ACM Digital Library, http://portal.acm.org/ [Accessed July 16, 2007].

[12] Bhattacharyya, Supratik, et al. "Efficient Rate-Controlled Bulk Data Transfer Using Multiple Multicast Groups," *IEEE/ACM Transactions on Networking*, vol. 11, no. 6, December 2003. [Online] Available: The ACM Digital Library, http://portal.acm.org/ [Accessed July 16, 2007].

[13] Floyd, Sally, et al. "A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, December 1997. [Online] Available: IEEE Xplore, http://ieeeexplore.ieee.org/ [Accessed July 16, 2007].

[14] Kasera, Sneha Kumar, et al. "Scalable Reliable Multicast Using Multiple Multicast Channels," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, June 2000. [Online] Available: IEEE Xplore, http://ieeeexplore.ieee.org/ [Accessed July 16, 2007].

[15] Chaporkar, Prasanna, Anita Bhat, and Saswati Sarkar. "An Adaptive Strategy for Maximizing Throughput in MAC layer Wireless Multicast," *Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, 2004. [Online] Available: The ACM Digital Library, http://portal.acm.org/ [Accessed July 16, 2007].

[16] Makofske, David, and Kevin Almeroth. *Multicast Socket: Practical Guide for Programmers*. Boston: Morgan Kaufmann Publishers, 2003.

For Further Reference and Reading

[17] Almeroth, Kevin C. "The Evolution of Multicast: From the MBone to Interdomain Multicast to Internet2 Deployment," *IEEE Network*, vol. 14, no. 1, Jan-Feb. 2000. [Online] Available: IEEE Xplore, http://ieeeexplore.ieee.org/ [Accessed July 16, 2007].

[18] Atwood, J. William. "A Classification of Reliable Multicast Protocols," *IEEE Network*, vol. 18, no. 3, May-June 2004. [Online] Available: IEEE Xplore, http://ieeeexplore.ieee.org/ [Accessed July 16, 2007].

[19] Braudes, R. and S. Zabele. "RFC 1458: Requirements for Multicast Protocols," *RFC Index*, The Internet Engineering Task Force, May 1993. [Online] Available: The RFC Database, http://www.rfc-editor.org/ [Accessed July 16, 2007].

[20] Calderon, Maria, et al. “Active Network Support for Multicast Applications,” *IEEE Network*, vol. 12, no. 3, May-June 1998. [Online] Available: IEEE Xplore, http://ieeeexplore.ieee.org/ [Accessed July 16, 2007].

[21] Deering, Stephen E., and David R. Cheriton. “Multicast Routing in Datagram Internetworks and Extended LANs,” *ACM Transactions on Computer Systems*, vol. 8, no. 2, May 1990. [Online] Available: The ACM Digital Library, http://portal.acm.org/ [Accessed July 16, 2007].

[22] Deering, Stephen, et al. “The PIM Architecture for Wide-Area Multicast Routing,” *IEEE/ACM Transactions on Networking*, vol. 4, no. 2, April 1996. [Online] Available: IEEE Xplore, http://ieeeexplore.ieee.org/ [Accessed July 16, 2007].

[23] Diot, Chris, et al. “Deployment Issues for the IP Multicast Service and Architecture,” *IEEE Network*, vol. 14, no. 1, Jan-Feb. 2000. [Online] Available: IEEE Xplore, http://ieeeexplore.ieee.org/ [Accessed July 16, 2007].

[24] Dubray, K. “RFC 2432: Terminology for IP Multicast Benchmarking,” *RFC Index*, The Internet Engineering Task Force, October 1998. [Online] Available: The RFC Database, http://www.rfc-editor.org/ [Accessed July 16, 2007].

[25] Mockapetris, Paul V. “Analysis of Reliable Multicast Algorithms in Local Networks,” *Proceedings of the eighth symposium on Data communications*, 1983. [Online] Available: The ACM Digital Library, http://portal.acm.org/ [Accessed July 16, 2007].

[26] Ratnasamy, Sylvia, Andrey Ermolinskiy, and Scott Shenker. “Revisiting IP Multicast,” *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, 2006. [Online] Available: The ACM Digital Library, http://portal.acm.org/ [Accessed July 16, 2007].

[27] Stopp, D. and B. Hickman. “RFC 3918: Methodology for IP Multicasting Benchmarking,” *RFC Index*, The Internet Engineering Task Force, October 2004. [Online] Available: The RFC Database, http://www.rfc-editor.org/ [Accessed July 16, 2007].

[28] Thaler, D., B. Fenner, and B. Quinn. “RFC 3678: Socket Interface Extensions for Multicast Source Filters,” *RFC Index*, The Internet Engineering Task Force, January 2004. [Online] Available: The RFC Database, http://www.rfc-editor.org/ [Accessed July 16, 2007].

[29] Van Meighem, Piet, Gerard Hooghiemstra, and Remco van der Hofstad. “On The Efficiency of Multicast,” *IEEE/ACM Transactions on Networking*, vol. 9, no. 6, December 2001. [Online] Available: IEEE Xplore, http://ieeeexplore.ieee.org/ [Accessed July 16, 2007].

[30] Varshney, Upkar. “Multicast Over Wireless Networks,” *Communications of the ACM*, vol. 45, no. 12, December 2002. [Online] Available: The ACM

Digital Library, http://portal.acm.org/ [Accessed July 16, 2007].

[31] Whetton, B., et al. “RFC 3048: Reliable Multicast Transport Building Blocks for One-to-Many Bulk-Data Transfer,” *RFC Index*, The Internet Engineering Task Force, January 2001. [Online] Available: The RFC Database, http://www.rfc-editor.org/ [Accessed July 16, 2007].