SLAC-PUB-9387

Exploring the use of a reliable IP Multicast to distribute BaBar's Online Event processing and Filter software to a large number of farm machines

Tomoko Ishihara

Office of Science, Energy Research Undergraduate Laboratory Fellowship

Reed College

Stanford Linear Accelerator Center

Menlo Park, California

August 12, 2002

Prepared in partial fulfillment of the requirements of the Office of Science, Department of Energy Research Undergraduate Laboratory Fellowship under the direction of S.Luitz in SLAC Computing Services (SCS) at Stanford Linear Accelerator.

Work supported in part by the Department of Energy contract DE-AC03-76SF00515.

Contents

1 Introduction 3

2 Materials and Methods 5

3 Results 6

4 Discussion and Conclusions 7

A Acknowledgments 9

B References 9

entries go here...

Exploring the use of a reliable IP Multicast protocol to distribute BaBar's Online Event processing and Filter software to a large number of farm machines. Tomoko Ishihara (Reed College, Portland, OR, 97202) Steffen Luitz(SLAC, Menlo Park, **CA,** 94025)

Abstract

Currently, the problem at hand is in distributing identical copies of OEP and filter software to a large number of farm nodes. One of the common methods used to transfer these softwares is through unicast. *Unicast* protocol faces the problem of repetitiously sending the same data over the network. Since the sending rate is limited, this process poses to be a bottleneck. Therefore, one possible solution to this problem lies in creating a reliable multicast protocol. A specific type of multicast protocol is the Bulk Multicast Protocol (Morris, 1997). This system consists of one sender distributing data to many receivers. The sender delivers data at a given rate of data packets. In response to that, the receiver replies to the sender with a status packet which contains information about the packet loss in terms of Negative Acknowledgment. The probability of the status packet sent back to the sender is $\frac{1}{N}$, where N is the number of receivers. The protocol is designed to have approximately 1 status packet for each data packet sent. In this project, we were able to show that the time taken for the complete transfer of a file to multiple receivers was about 12 times faster with multicast **than** by the use of unicast. The implementation of this experimental protocol shows remarkable improvement in mass data transfer to a large number of farm machines.

1 Introduction

The goal of the BaBar experiment is to study CP violation by observing the decay of B-mesons (Nuclear Instruments & Methods in Physics Research). The subject of CP violation currently attracts many researchers to become actively involved in the BaBar collaboration. The BaBar detector is used to measure decays of bottom and charm mesons as well as other interesting processes.

The BaBar Data Acquisition (DAQ) system currently uses a farm of 60 computers in which Online Event Processing and filtering take place. Currently, unicast is used to distribute the executables and configuration files to the 60 farm nodes. Unicast is not ideal for data distribution, since the sender must distribute the same data in repetitious cycles to the receiver computers. The condition that unicast protocol operates under a limited sending rate can result in a bottleneck.

As a possible solution to this problem, a multicast protocol could be used. A specific type of multicast protocol that relatively simple to implement as an experimental protocol is called the Bulk Multicast Protocol “BMTP” (Robert Morris, 1997). In applying this protocol to the data distribution problem, the problem of the sender delivering the same data repetitiously could be eliminated. After the sender distributes the packets to the farm machines using multicast, they each respond to the sender by returning a status packet. The status packet aids the sender in determining the packets that were lost in the transmission, so that the sender can multicast the missing parts to the receivers again. The receivers respond to the sender with status packets in terms of Negative Acknowledgment (NAK) that contain information about the receiver’s request for the lost data. Each receiver only sends a status packet with a probability of $P_s = \frac{1}{N}$, where N is the number of receivers. This makes the total rate of status packets independent of the number of receivers.

After exploring the process of using a multicast protocol, the functions of the sender and the receiver need to be further explained. In the multicast setting, the sender first needs to read the file from a disk. The data contained within the file is divided into sections we call segments. For each segment, a network packet is created after adding a header. Therefore, the size of a segment has to be small enough to fit into a network packet (ca. 1500 bytes). After multicasting a network packet, the sender checks for the arrival of status packets from the receivers. The status packet information is used to recalculate the number of receivers, adjust the sending rate and to resend lost packets identified by receiver NAKs.

Now turning to the receiver’s perspective, they collectively begin with an empty set of data arrays and prepare for a packet to arrive from the sender. When a receiver obtain a packet, it splits it into a segment and a header. This process occurs, so that the receivers are able to copy the segment in the correct position of the array. The receivers then need to calculate their receive rate so that they can notify the sender about the maximum rate they can receive and they need to identify lost packets and re-request them from the sender.

Even if the receivers do not obtain any new packets from the sender, it is important for the receivers to remain actively involved in the process of completing the array. During the waiting period, the receivers will check to observe if any packets are missing. When they recognize that a NAK needs to be sent, they calculate the waiting interval and the probability at which it should be delivered to the sender. While the receivers prepare the NAK, the receivers can check if they have received the entire array of packets. When the array has been completely received, the receiver can write the array to a file.

In this project, multicast protocol will be implemented to ensure efficient data transmission between the sender and the receivers. Through observing the data transmission rate of the sender, we hope to analyze the receiver's request for retransmission by studying the number of missing data reported along with NAK. The goal of this project is to compare the time taken for unicast and multicast to complete the data transfer of a large file, so that the improvement in data transfer between the two protocols can be analyzed.

2 Materials and Methods

Two programs were written in C (Kelley and Pohl 1998)in order to distribute a file from the sender to multiple receivers. The sender needs to read a file from a disk then load the file to a large array. The array is then divided into a sequence of 1400 byte (SEGSIZE) sized segments. Then, a protocol header is attached to each segment to turn it into a network packet.

The header contains information regarding the segment number and an estimate of the number of receivers. The segment carries the actual data that will be transmitted. In order to transfer the entire array, the sender calculates the number of segments and the remainder of segments. The SEGSIZE used to calculate these values is a global constant shared between the sender and the receiver. This information is crucial in placing the received data into the receivers data array.

After the sender starts transmitting data with an initial rate (in pkts/s), the number of clients, file size and the start and length of each segment are calculated. In order to create a network packet (Stevens 1994) the header needs to be attached to the segment containing specific values for the session id, number of clients, number of bytes in the packet, segment number, packet number, retransmission, and total number of bytes in the file. After the network packet has been completed, the data packet needs to be sent. After sending the packet, the sender checks if any status packets have been received. If there is a status packet waiting, the sender recalculates the new sending rate by adjusting it to the minimum rate reported by the receivers. The sender adjusts the retransmission rate by a constant K, where $K > 1$. The value of K used in our sender program was 1.05. Using K in calculating the retransmission rate slightly overloads the slowest receiver, but allows the rate to increase at the cost of packet loss. Therefore, the total rate limit is determined by the retransmits requested by NAK and the normal data packet transfer. After the sender's rate measurement interval is finished, the sender shows the receive rate reported by the IP address of the receiver, the new send rate and the time interval measured since the last rate was reported by the client.

Now turning to the receiver's program, the receiver begins with an empty array identical to that of the sender. When the client receives a packet from the sender, it calculates the number of segments in a file and partitions the packet into a segment and a header. The client copies the received data by placing each of the segments into the entire array. Since the client copies a received segment into the array starting at (Segment number $*SEGSIZE)+$ each byte of SEGSIZE, this ensures that the packets are transmitted into the correct place and that they are of the correct size inside the array. For example, by using the formula to calculate the transfer of a packet with segment number 0, the formula becomes, $(0 * 1400) + 1400 = 1400$ bytes. This means that the packet with segment number 0 contains 1400 bytes to be transferred to the empty array of the receiver. The client is able to set the receive time counter in order to store the time (in ms) at which the packet was received. The counter is set to 0 before the packet is transmitted by the sender. If the counter reports a positive number, this signifies that the packet has been received. However, if a negative number is reported, this means that the packet was lost in the network. If there is a missing packet, the segments that lie between the previous segment number and the current segment number are marked as missing. Then the receive time counter proceeds forward until it finds an array element marked by 0 or a negative time interval and represents that as the first missing packet. After that, the program checks if the complete file has been received.

In calculating the probability of sending a status packet, the ratio of missing packets to sent packets represented by $\frac{m}{s}$ was defined to be in the range of $0 \leq ratio < 1$ and the following equation, $P_s = \frac{1}{N} min(\frac{m}{s}, a)$ was used. *a* is a constant that increases the probability of sending a status packet in the case of having a flawless receiver (a receiver that does not miss any data packets from the sender). Our value of *a* was defined to be 0.1. The condition was imposed that if the ratio was not 0, then the missing rate took the value of the ratio. However, if the ratio was 0 in an instance of a perfect receiver, there was a 10% chance that the client will report a status packet to the sender. In an alternative implementation, $P_s = \frac{1}{N} + \frac{b(\frac{m}{s})}{N}$ was defined so that there was an increased probability due to the multiple of the constant b that the client would reply to the sender. Finally, the possbile candidates for NAK were decided and sent after the clients waited for a given time interval and found a receive time (marked by negative time) that had been in the array for a long period of time.

3 Results

In this project, the data transfer between the sender and the clients could be analyzed quite successfully. First, Figure 1 depicts an array diagram explaining the content of a single data packet that is transmitted from the sender. The data packet comprises two divisions, namely the header and the segment. The header contains seven different integer elements, each having a size of 4 bytes. The segment contains the data needed to be transferred in 1400 byte sized elements.

Figure 2 reveals the process of data transfer between the sender and the clients when the programs are executed. When the sender distributes a packet, the packet is placed in the correct segment number of the receiver's array. When the packet arrives to the receiver, the receiver stores the time taken for the packet to be delivered in the Receive Time Counter Array. The counter marks the packet as missing when it records a 0 or a negative time interval.

Figure 3 shows a sample of a sender and a receiver after the program has been executed. The sender displays the total number of bytes in the file, the number of segments and the remaining bytes of the segment before showing the receive rate reported by the client. After the sender obtains the receive rate, it reports the IP address of the receiver along with the new send rate and the time elapsed since the last receive rate was reported. On the side of the receivers, they estimate the number of clients, report the packet receive rate, the number of missing packets, the ratio of $\frac{m}{s}$ and the NAK.

Figure 4 and 5 show how the receive rate reported by the client varies with the time interval measured by the sender. The data points were reported by the sender every 500ms. Specifically, Figure 4 shows data taken from one receiver and Figure 5 reveals data retrieved from 37 identical sun machines.

Lastly, Figure 6 shows the graph of Receive Time counter vs Segment number for the clients after all data has been received. The data points show the time at which each of the segment numbers were received.

4 Discussion and Conclusions

In Figure 3 and 4, a sample of the sender and the receiver outputs are displayed. Since the client program was executed before the sender, the number of NAK transmitted to the sender is almost always 0. However, when there are status reports occasionally delivered, the number of NAK is consistent with the number of packets that are missing. This means that the client is requesting data to be retransmitted for the packets that it had failed to receive. The client calculates the missing packets and the sent packets by analyzing the packet sequence number. This number is a unique number assigned to each packet that increases by 1 every time a data packet is transmitted from the sender.

In calculating the probability of sending a status packet, the method of $P_s = \frac{1}{N} + \frac{b(\frac{m}{s})}{N}$ was used instead of $P_s = \frac{1}{N} min(\frac{m}{s}, a)$. By using the first probability formula, the estimate of the number of receivers approximated by the sender was calculated to be a value of 1, since the probability of sending a status packet was $P_s = \frac{1}{N} + \frac{b(\frac{m}{s})}{N} \geq \frac{1}{N}$. As a result, the probability of sending a NAK was $\frac{1}{N}$ when the first probability equation was applied. When the second probability equation was used, the following problem was encountered. In a situation with a perfect receiver, where $m = 0$:

$$P_s = \frac{a}{N}$$
$$\sum_N P_s = a$$
$$N * a = 1$$

$$N = \frac{1}{a} \tag{1}$$

Since the estimate for the number of receivers became $N = \frac{1}{a}$ the probability for sending a status packet was $P_s = \frac{1}{N} min(\frac{m}{s}, a) \geq \frac{a}{N}$. This caused the sender's estimate of the receivers to become too small and greatly decreased the probability of sending the NAK.

In Figure 5, the receive rate of the client (in pkts/s) grows exponentially with respect to the time interval. This result shows that the sender is retransmitting the data according to the new sending rate defined by $newsendrate = K * min$, where $K = 1.05$. Since the sender defines the newsendrate with a 5% increase from the minimum receive rate of the client, this means that the NAK feedback is functioning correctly. In determing the frequency of the transfer of NAK, a random number r that generates numbers in the range of $0 \leq r \leq 1$ was used. If $r \leq \frac{m}{N}$ a status packet would be sent. While preserving the probability of sending a status packet as $\frac{1}{N}$, the client with the slowest receive rate is able to respond to the sender much more often compared with the client that missed only a few data packets. The effect of applying the random number method to the probability of sending a status packet became apparent when testing on multiple Sun machines. As shown in Figure 6, the rate of packet transfer still maintains the exponential growth as seen in Figure 5. Since the receiver is sending a status report if $r < P$, where $P = \frac{1}{N} = 1$, the receiver prevents overcrowding the sender with numerous status packets. As a result of this, the sender is able to choose the receiver that transmits very large number of NAK and increase the newsendrate at an exponential rate.

Figure 7 shows the time taken for the segment number to be completely transferred to the receiver's array. As seen in Figure 8, when the receiver's program was executed before the sender's, the retransmitted data packets by the sender appear as a small line during the time interval of about 45000 ms to 45130ms. Since the client receives these data packets exactly after 1000ms it has been declared as missing, the receive time counter accurately estimates the time at which NAK must be sent.

Unlike the previous figures, Figures 9 and 10 both reveal the time elapsed for all of the segment numbers in the file to be transferred when the sender was started before the receiver. At the beginning of Figure 9, there are two lines that join before 10000ms. The bottom line is due to the sender's retransmission of data and the top line is a result of the receiver getting caught up to the sender's rate. In the close up picture, there is a stream of retransmitted data between 47300ms and 47450ms. Unlike Figure 8, there are a large number of retransmits, because the receiver takes time to reach the rate of transfer of packets by the sender. Again, the client's request for missing data occurs 1000ms after it has been missing.

Lastly, the time taken for the complete transfer of the file was measured for one client and multiple clients. In both of the runs, the sender transferred a file containing $n = 100,000,000$

bytes at the initial rate of 2000 pkts/s. For one receiver, the time taken for the transfer was 28.6 s. This amounts to about a rate of 3MBytes/s. Comparing this rate to a rate of unicast transfer which is typically 5MBytes/s, the multicast protocol shows improvement in the efficiency of mass data distribution. When the protocol was tested on 37 identical clients, the time taken for the completion of transfer was 1min 4s (64s). Compared to the unicast transmission, unicast would take approximately about 12 min for transferring the data to all of the clients. Therefore, by using multicast, a factor of 12 was gained, which shows remarkable improvement in the data transfer.

For further improvements to the protocol, the probability of sending a status packet could be adjusted, since the rate of transfer drops quite rapidly due to the slowness of the clients. Other possible solutions that could eliminate such a sudden drop in the rate of sending data packets need to be explored. Although multicast works, the final goal of the project was to distribute Babar's Online Event Processing and filter software and save the data to a file. If there was more time available, this task could have been accomplished.

A Acknowledgments

I would like to thank the DOE for selecting me to participate in this summer ERULF intern program at Stanford Linear Accelerator. This program has been an invaluable experience for me as I have learned a lot from the project as well as from the lectures and tours that we were able to participate in.

I would also like to thank my mentor Steffen Luitz for assigning me a project where I was able to explore an unrelated field to my major. As a physics major, I feel that having programming skills are very helpful in working at a research environment. I also would like to thank Steffen for all of the helpful insights that he gave to me in order to assist me with the completion of my project.

Lastly, I would like to thank Helen Quinn and Sekazi Mtingwa for organizing this program that I was able to take part in. I feel that the ERULF program has offered me the opportunity to work closely with my mentor and gain an overall knowledge about the kinds of experiments that are currently operating at SLAC.

B References

Aubert, B., et al. (2001). The BaBar Detector. Yale University, New Haven, CT.pp.11. Kelley, A.,Pohl,I. (1998). A Book on C Programming in C. Addison-Wesley.

Stevens, R.W. (1994). TCP/IP Illustrated, Volume 1 The Protocols. Addison-Wesley Publishing Company.

Morris, R. (1997). “Bulk Multicast Transport Protocol,”Proceedings of INFOCOM’97. Cambridge,

MA.pp.1-9.

C Figures

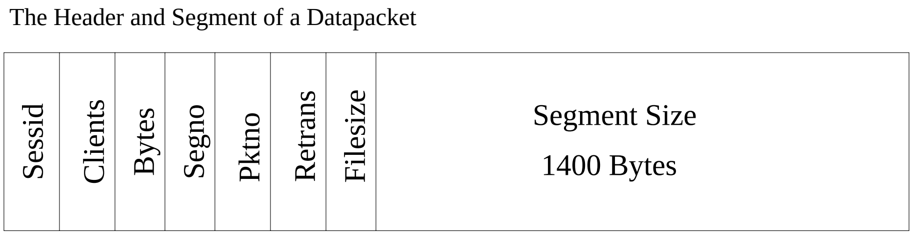

Figure 1: Diagram of a data packet

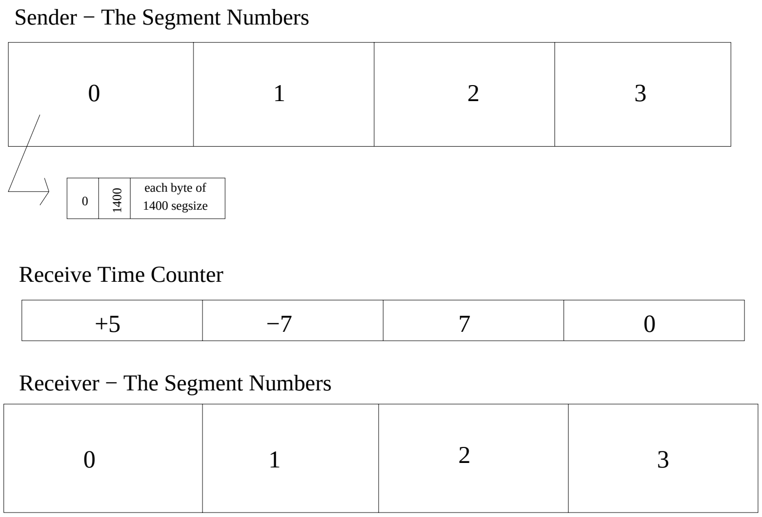

Figure 2: The transfer of the packet from the sender to the receiver

```
this is n, the total number of bytes in the entire file:100000000

this is the number of segments in the packet:71428

this is the remaining bytes of the segment:800

setup_network: 0

rec_rate: 200.995026 by 134.79.156.151, new_send_rate: 211.044769 time: 1016
rec_rate: 200.995026 by 134.79.156.151, new_send_rate: 211.044769 time: 1519
rec_rate: 210.789215 by 134.79.156.151, new_send_rate: 221.328674 time: 2022
rec_rate: 210.789215 by 134.79.156.151, new_send_rate: 221.328674 time: 2524
rec_rate: 220.558884 by 134.79.156.151, new_send_rate: 231.586823 time: 3028
rec_rate: 220.558884 by 134.79.156.151, new_send_rate: 231.586823 time: 3529
rec_rate: 231.075699 by 134.79.156.151, new_send_rate: 242.629486 time: 4031
rec_rate: 231.075699 by 134.79.156.151, new_send_rate: 242.629486 time: 4532
rec_rate: 240.759247 by 134.79.156.151, new_send_rate: 252.797211 time: 5036
rec_rate: 240.759247 by 134.79.156.151, new_send_rate: 252.797211 time: 5539
rec_rate: 252.495010 by 134.79.156.151, new_send_rate: 265.119751 time: 6041
```

Figure 3: Sample of a sender output

```
clients: 1, rate: 210.789200, missing: 0, ratio: 1.000000, naks: 0
clients: 1, rate: 220.558884, missing: 0, ratio: 1.000000, naks: 0
clients: 1, rate: 231.075699, missing: 0, ratio: 1.000000, naks: 0
clients: 1, rate: 240.759232, missing: 0, ratio: 1.000000, naks: 0
clients: 1, rate: 252.495026, missing: 0, ratio: 1.000000, naks: 0
clients: 1, rate: 264.735260, missing: 0, ratio: 1.000000, naks: 0
clients: 1, rate: 276.171478, missing: 0, ratio: 1.000000, naks: 0
[...]
clients: 1, rate: 510.489532, missing: 0, ratio: 1.000000, naks: 0
clients: 1, rate: 534.465576, missing: 0, ratio: 1.000000, naks: 0
clients: 1, rate: 533.932129, missing: 23, ratio: 1.206093, naks: 0
clients: 1, rate: 558.882263, missing: 0, ratio: 1.000000, naks: 23
clients: 1, rate: 585.414612, missing: 0, ratio: 1.000000, naks: 0
clients: 1, rate: 611.388611, missing: 0, ratio: 1.000000, naks: 0
[...]
clients: 1, rate: 1420.579468, missing: 0, ratio: 1.000000, naks: 0
clients: 1, rate: 1452.547485, missing: 0, ratio: 1.000000, naks: 0
clients: 1, rate: 1402.597412, missing: 99, ratio: 1.329341, naks: 0
clients: 1, rate: 1424.575439, missing: 0, ratio: 1.000000, naks: 99
clients: 1, rate: 1473.526489, missing: 0, ratio: 1.000000, naks: 0
clients: 1, rate: 1528.471436, missing: 0, ratio: 1.000000, naks: 0
```

Figure 4: Sample of a receiver output

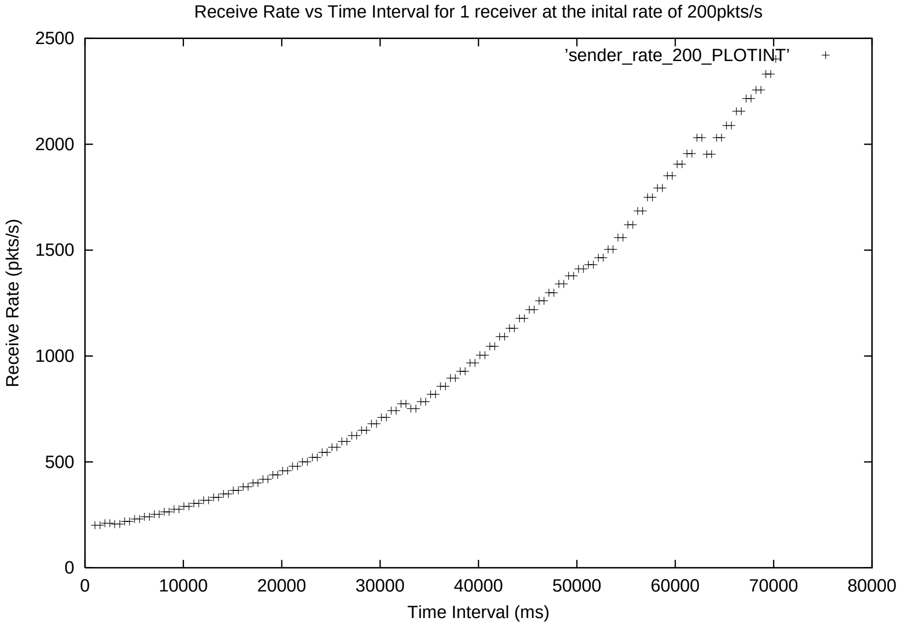

Figure 5: Graph of Pkt Rate vs Time Interval as reported by the sender for 1 receiver

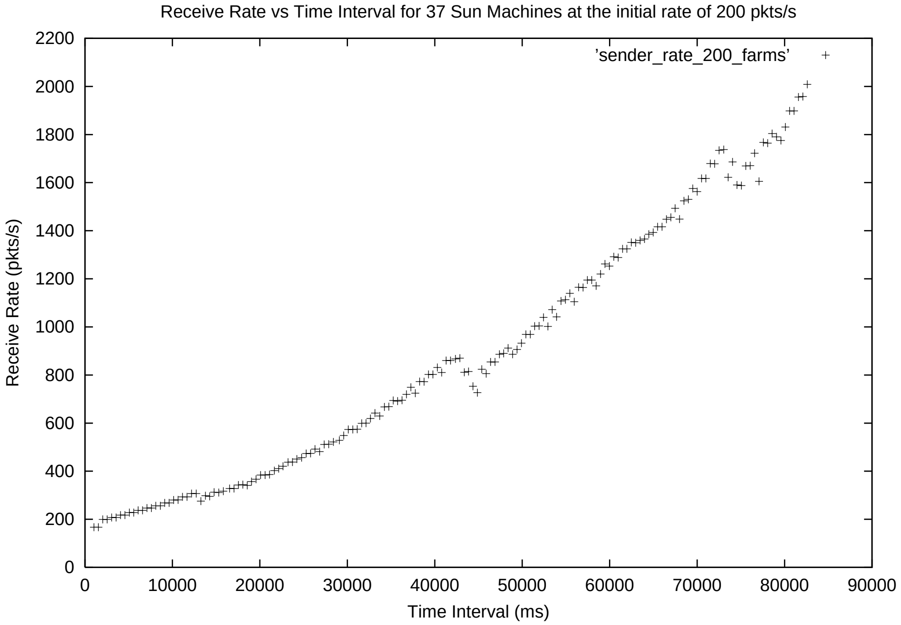

Figure 6: Graph of Pkt Rate vs Time Interval as reported by the sender for 37 Sun Machines

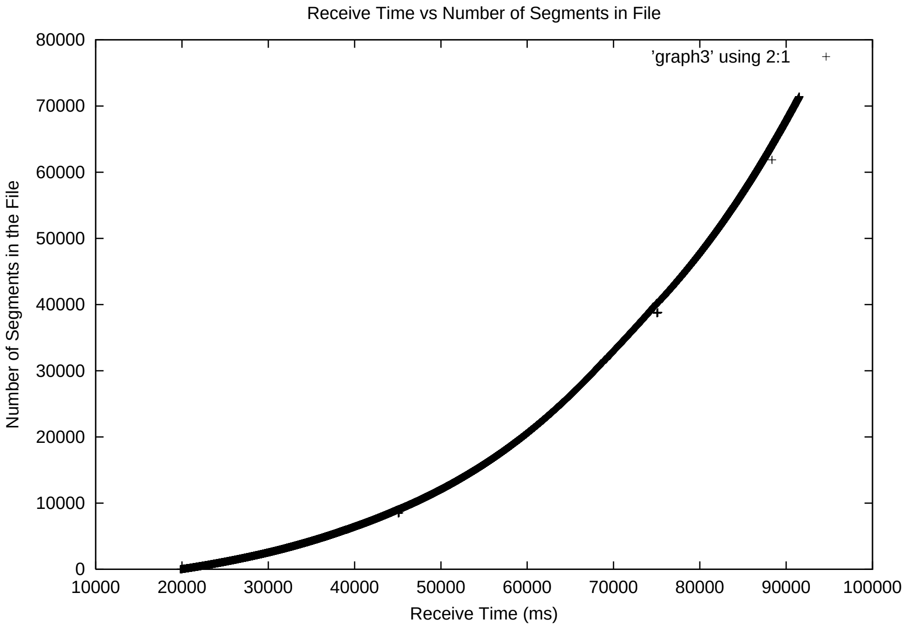

Figure 7: Graph of Segment Number vs Time when the the receiver was started before the sender

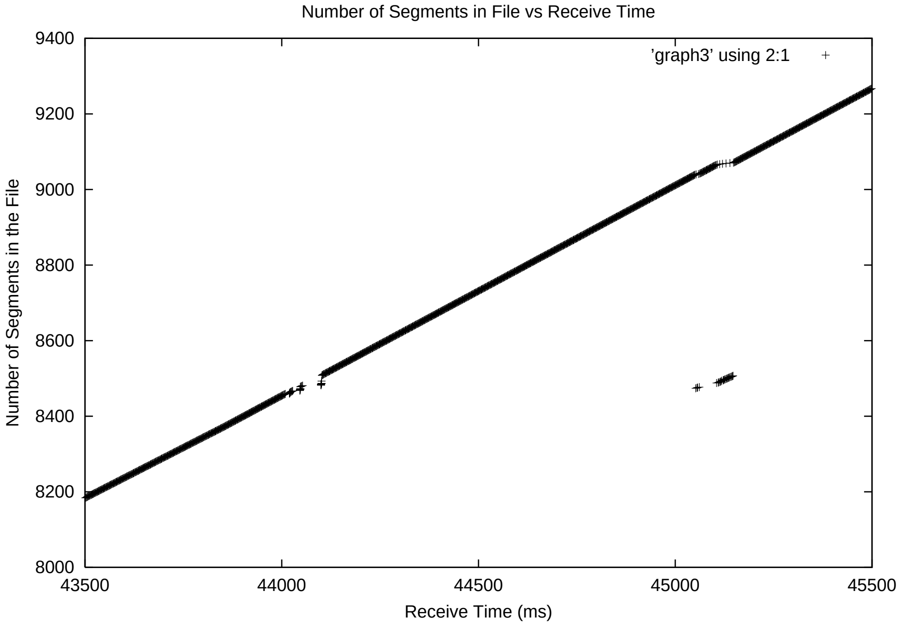

Figure 8: A close up view of the Graph of Segment Number vs Time when the receiver was started before the sender

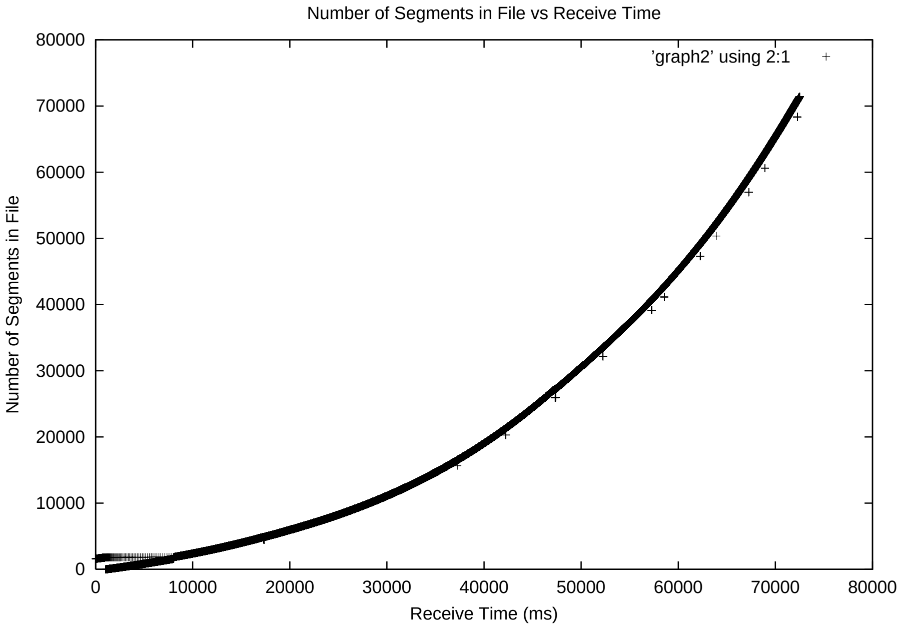

Figure 9: Graph of Segment Number vs Time when the sender was started before the receiver

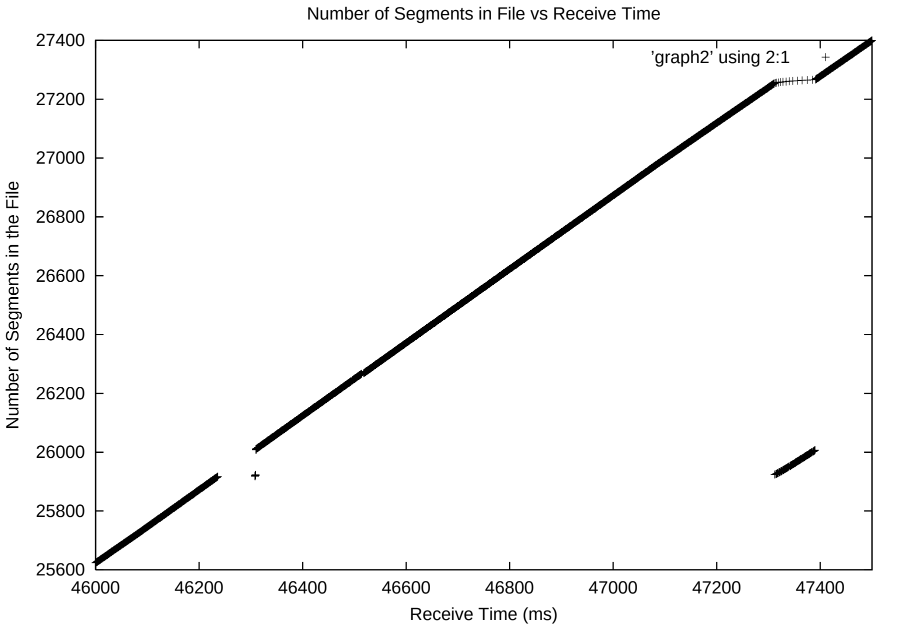

Figure 10: A close up view of the Graph of Segment Number vs Time when the sender was started before the receiver