Reliable Source Specific Multicast

- Implementation and Analysis of Results

By Ramakrishnan Venkitaraman
(rxv024000@utdallas.edu)
April 2003

1 Introduction

This document discusses the results that were observed as a part of a sample implementation of the protocol suggested in the previous deliverable. The aim of the implementation is to increase our understanding of the proposed protocol and to understand the scalability issues that arise with an end to end Implementation.

This document is organized as follows

1 Introduction 1
2 Analysis of implementation results 1
2.1 First Run 1
2.2 Second Run 3
2.3 NACK Loss 4
2.4 Busy Waiting 5
3 Future Implementations 7
4 Conclusion 7

2 Analysis of implementation results

2.1 First Run

The server sends packet freely until the window size is reached. Say if the window size is denoted by winSize then the server will send the packets starting from the first packet to packet number winSize without waiting in between (As of now I am not using slow start). Once winSize is reached the sender will have to wait until it is sure that it will not receive a retransmission request packet that its going to do away with in the sending window. It is done using time outs (max RTT) as discussed in the second deliverable.

In the implementation I use the following values,

Time out to forward the sending window pointer = the max RTT of the receiver set

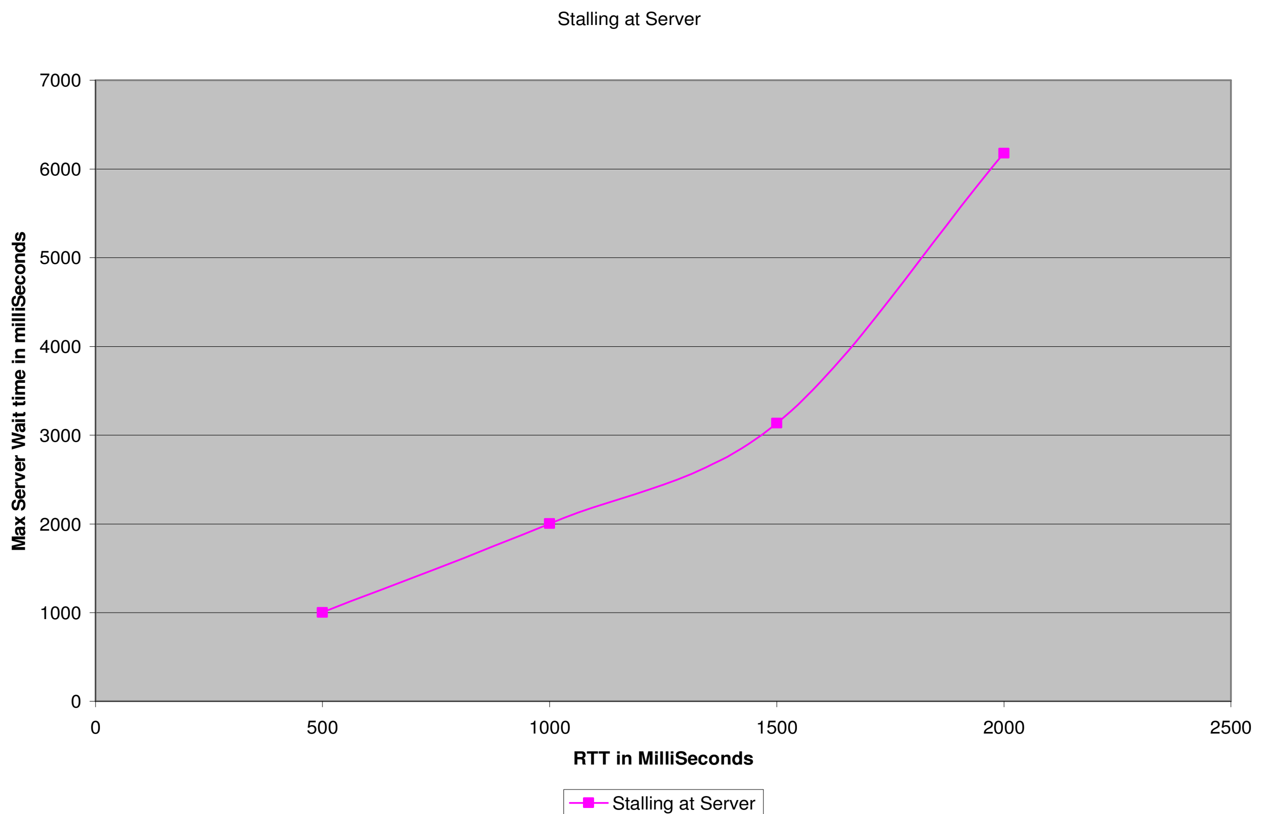
Say the sender is ready for transmission of packets at time t1 if it sees that the sending window is full it waits until the sending window pointer advances. During this period of time the server is said to be in the “**Busy Waiting**” state because its idle and the computing power is consumed just doing noting and checking if the sending window pointer has advanced or not. During the data collection phase of the implementation we call this as the idle cycles spent at the server.

Idle cycles **=** Number of cycles that were spent in the busy waiting state where a cycle refers to one check sending window for space.

The graphs were generated by using data transfers of about 315 packets (File Transfer) as an SSM session with the maximum RTT ranging from 500 milliseconds to 2500 milliseconds.

The size of the receiver set does not drastically affect our current analysis which tries to find the maximum amount of time that the sender will be sitting idle before sending a packet waiting for the sending window to have more space. This problem arises because the sender cannot advance the window until it is sure that it will not receive future NACKs. So if we have even one receiver with a very large RTT then the sender will wait for that time interval before it advances the pointer in the sending window and the sender will be waiting idle without sending packets until this advancement happens.

The following graph shows the plot of the values that we got for one of the runs.



In the graph the X axis corresponds to maximum Round Trip Time in milliseconds and the Y axis corresponds to the maximum amount of time that the server was waiting (idle) between successive packet transmissions for a given Maximum RTT.

As can be seen with increase in the estimated RTT the sender will stall for increasingly larger intervals of time.

From the graph we find that for an RTT of about 500 milliseconds the server waits for as much as one second between successive packet transmissions. So, though the sender is capable of sending at a far higher rate, he is forced to restrict his inter packet transmission rate to one second.

The situation becomes worse as the RTT increases, for an RTT of 1000 milliseconds the packet transmission rate at the sender is restricted to one for every 2 seconds and with the RTT of 1500 milliseconds the packet transmission rate at the sender is restricted to one for every 3 seconds.

The situation becomes quite unacceptable as with an RTT value of 2000 milliseconds in which case the sender waits for more than 6 seconds between packet transmissions. This in practical scenarios will be quite unacceptable and with this mechanism we will not be able to transmit big files as it could take days to transmit them.

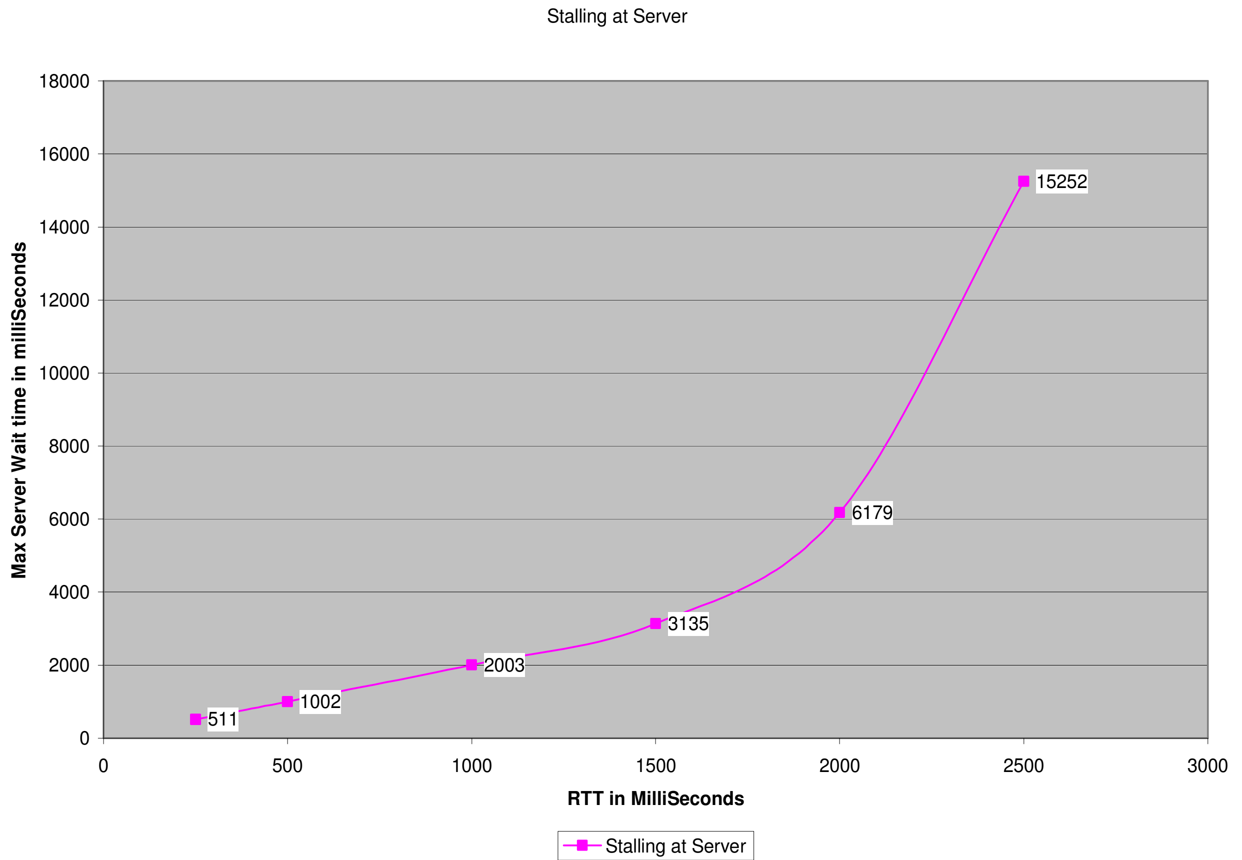
2.2 Second Run

The following graph shows the plot that was got from another run of the Implementation.

It also includes two additional runs in which we have a run with a maximum RTT value of 250 and also a run with the maximum RTT value of 2500. We observe the following results.

The following graph corresponds to the dataset

RTT (milliseconds)	Server Stalled for milliseconds
250	511
500	1002
1000	2003
1500	3135
2000	6179
2500	15252



In the graph the X axis corresponds to maximum Round Trip Time in milliseconds and the Y axis corresponds to the maximum amount of time that the server was waiting (idle) between successive packet transmissions for a given Maximum RTT.

For a RTT value of 250 milliseconds the stall time is around 500 milliseconds. The stall time for the runs with 500 through 2000 milliseconds give similar results but when an RTT value of 2500 is used the stall time increases to as much as 15 seconds plus between packets.

This graph suggests some sort of exponential rise in the stall time at the sever side even for small increases in the RTT value.

2.3 NACK Loss

Through out this implementation I have assumed the NACK that the receiver sends to the sender requesting retransmission will safely arrive at the sender without loss or packet errors and this assumption may be unrealistic.

A pure end to end solution is very tough for this because the sender will get to know of the loss only if the receiver is to send a NACK for that and if that NACK is lost there the sender has no idea of this.

The receiver will still receive a retransmission if the sender received a NACK from another receiver for the packet (that was lost). Note that we also send the retransmissions as a Multicast (SSM). But incase this NACK which was lost was the only NACK for that packet then we need some other mechanism to handle this.

The simple approach that I can think of now is for the receiver to wait for some time after sending the NACK and if it does not get the retransmission send the NACK again asking for retransmission. As can be clearly seen this approach has the following disadvantages

1. A design using this will make the sender to wait for an even longer amount of time before being able to move the back pointer of his sending window and this will in turn increase the packet sending rate at the sender (which already is in an unacceptable state).

2. Say if the retransmitted NACK packet is also lost in this case the sender again will not be retransmitting and the sender would have by this time moved the back pointer of the sending window implicitly assuming that all the receivers have received it. So even if the receiver sends the NACK request again for the third time, and if the sender receives it, the sender will NOT have the requested packet in the sending window and will be in no way be able to retransmit it.

So, some more approaches to deal with this problem are

1. Get network layer support and/or Local retransmissions so that we have a greater amount of luxury in deciding the amount of time that the sender can wait for NACK's.

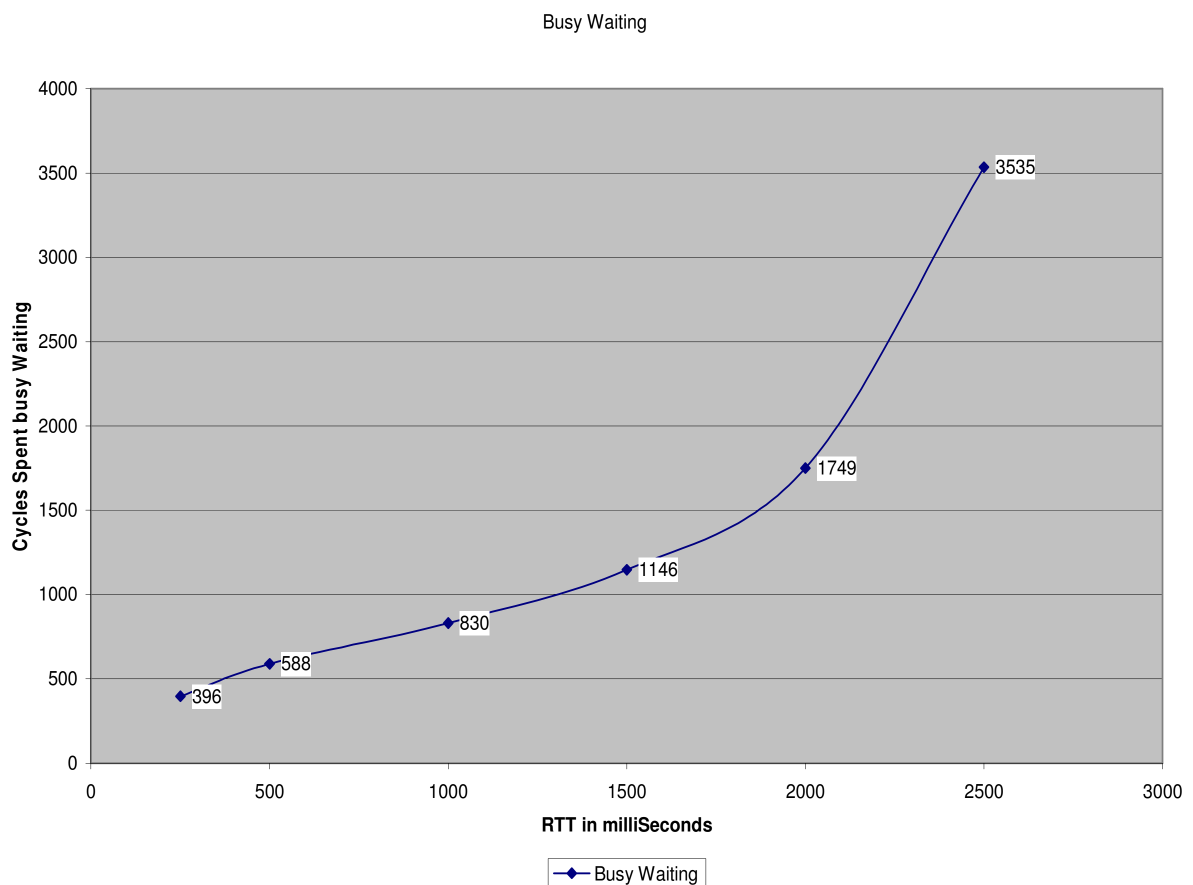
2. Else, if the type of data transfer is like a file transfer then it is very likely that the sender will have the requested retransmission of file contents say in his disk though not always at the current sending window. In such cases the sender will remember the set of retransmissions that it was unable to fulfill and once the end of file is reached it will retransmit/satisfy those requests again by getting it from the disk and send it to the receiver.

2.4 Busy Waiting

The following graph shows the amount of time that the sever spends in busy waiting cycles with the increase in the RTT. A busy waiting state in the will indicate the underutilization of the computing power at the sender side.

The graph corresponds to the dataset

RTT (milliseconds)	Busy Waiting Cycles
250	396
500	588
1000	830
1500	1146
2000	1749
2500	3535



In the graph the X axis corresponds to maximum Round Trip Time in milliseconds and the Y axis corresponds to that the server spent (idle) polling the sending window for presence of space between successive packet transmissions for a given Maximum RTT.

Similar to the previous case we find a steady rise in the degree to which the server is underutilized which can be seen from the increasing number of idle cycles.

For an RTT of 2500 milliseconds more than 3500 states as wasted as busy cycles. This will be unacceptable for practical considerations.

3 Future Implementations

My current implementation focuses only on the sender side sliding window effects for the variations in RTT and does not implement all the features that were discussed as a part of the second deliverable of the paper.

So future implementations will be done to analyze the effects of Correlated Loss which deals with how a loss of a packet can affect the whole system. For a system with N nodes a loss of a packet may mean that

- All the N nodes lost the packet
- A larger set among the N nodes lost the packet
- A small subset of the N nodes lost the packet
- Only one node lost the packet

So we can assign probabilistic values to each of these events and try to simulate the same and analyze the effect for various probability values of the same. More over we can also model the issues related to redundant NACKS and NACK suppression as discussed in the second deliverable. This will further increase our understanding of the current system and help us in better modeling future systems that use Reliable SSM.

4 Conclusion

A pure End to End Implementation though always welcome is to be carefully designed if it to support a wide set of receivers who are spread across the globe with different values of RTT. By careful design I mean that the design should some how share the heavy load (idling waiting for receiver's response) at the server side by say having more representatives of the sender as "Helpful Intermediate node" as discussed in the second deliverable.

The most efficient implantation of Reliable SSM will be the one that will get help at the network layer in the process of Retransmission and NACK Suppression. So if such network layer support is acceptable then we must go for such an approach as an pure End to End approach will be less efficient than the one that network layer support.