The Effectiveness of Selective Acknowledgments in a Lossy Network

Stephen Harding

November 4, 2013

Abstract

In a network where packet loss is uncommon, the naive strategy of only acknowledging packets when all previous packets in the sequence have been received and acknowledged works fairly well. However, in a network where packet loss is ubiquitous, this strategy can lead to excessive and gratuitous retransmission of packets. i.e., the recipient is missing a packet and has received several packets that are after the missing packet but never sends an acknowledgement to the sender for these packets. The sender ends up retransmitting all of the unacknowledged packets because it has no way of knowing that the recipient was actually only missing a single packet. The standard way of addressing this problem is selective acknowledgements [1]. This allows the recipient to acknowledge ranges of packets which has the effect of notifying the sender of the missing packets so they can retransmit only the necessary packets.

In this paper, we examine the effectiveness of selective acknowledgements in a network environment where there is a high rate of packet loss.

1. Introduction

To study the effectiveness of selective acknowledgements (`sack`) we perform a full factorial experiment in which we hold constant a rate of packet loss and vary the network latency. We measure retransmission rates with selective acknowledgments on and off for each latency setting.

The rest of this document is laid out as follow. We describe the experimental setup in detail in section 2. In section 3 we present the results of the experiments. In section 4 we discuss the results presented in section 3 before concluding in section 5.

2. Experimental Setup

We perform a full-factorial experiment with two factors: network latency and selective acknowledgement. The selective acknowledgement factor has two levels, either on or off. For the network latency factor, we consider several different latencies. The experiment is conducted on a network of three virtual machines where there are two endpoints on different subnets and a router that connects the two. The network delay and latency is simulated with `tc` rules on the router. A client program is run on one endpoint (a FreeBSD machine) and a server is run on the other (an Ubuntu machine). The client sends a random string of 1,000,000 characters to the server and the server sends back an MD5 checksum of the transmitted string. Since only the server is receiving large amounts of data, we only vary the state of the selective acknowledgement `TCP` option on the server.

The latency factor has the following levels: 0ms, 5ms, 5ms ± 2ms, 50ms, 50ms ± 10ms, 100ms ± 40ms, and 500ms. Levels with a ± are varied by the amount after the ± and are normally distributed. Levels without a ± have no variation in the latency other than the variation inherent in the actual network.

`TCP` dumps are recorded on the client machine in order to measure the rate of retransmitted packets. The network loss is implemented by a `tc` rule: `tc qdisc change dev eth1 root netem loss 25% 25%`

2.1 Bias and Distortion

We take two measures to minimize bias and distortion. First, treatments are not blocked. The treatments are performed at random by building a list of experiments to be performed and randomizing that list prior to beginning the experimentation. Second, no other extraneous processes or network connections are allowed to run during any of the experiments. All network communication to change the testbed is done in-between treatments.

3. Results

Since there are 196 entries in the full factorial experiment we must restrict ourselves to a few representative results. [1] To analyze the results we inspect the `TCP` dumps and generate text files which contain information about packet retransmission which we can then analyze with the `R` statistical tools.

To begin, consider the 0ms delay. We find that difference in the mean rate of retransmission when `sack` is varied is statistically significant with a t-test [2], with the sample means of 107.8 and 125.4 for `sack` on and off respectively. We see a similar reduction in retransmission in the 5ms case: 130.2, and 145.0. The benefit is slightly reduced in the 5ms $\pm$ 2ms case: 128.1, and 139.4. Interestingly, the 100ms $\pm$ 40ms case resulted in no statistical significance but the 500ms with `sack` on performed better than 100ms $\pm$ 40ms with 139.3 and 150.4 respectively. Even more interesting is the following. The treatment with 500ms with no variation with `sack` on resulted in no statistical significance for a difference of means with each of the following treatments: 100ms $\pm$ 40ms `sack` on, 50ms `sack` on and off, 50ms $\pm$ 10ms `sack` on and off, 5ms $\pm$ 2s `sack` off, and 5ms `sack` off. The most dramatic reduction in retransmission is the 500ms case: 139.3, and 212.6 with `sack` on and off respectively.

4. Discussion

We can see from the results that selective acknowledgements can have a dramatic impact on the number of retransmissions that are sent in a lossy network. Furthermore, it is clear that selective acknowledgements enjoy the greatest benefit in a network environment where the network latency is constant. In fact, as we saw in the previous section, with selective acknowledgements enabled, even a very high delay is much better than a lower delay with high variation.

5. Conclusion

Clearly selective acknowledgements are a beneficial `TCP` option in a lossy network. While we did find some cases where enabling selective acknowledgements did not make an obvious beneficial difference, we found no cases where enabling selective acknowledgements performed worse than disabling them.

Future work would benefit from adding a factor to the experiment in which the levels would vary congestion control parameters.

References

[1] S Floyd, J Mahdavi, M Mathis, and A Romanow. Tcp selective acknowledgment options. 1996.

[1] The reader is invited to view the results in more detail by examining the raw data found at github.com/stharding/netslab.git

[2] We justify the use of a t-test here by making use of the standard normality tests available in `R` (e.g. shapiro.test, qqnorm, qqline, etc.) The reader is welcome to reproduce these results by obtaining the repository listed in footnote 1