Tackling bufferbloat in 3G networks - DRWA implementation

Pulkit Yadav
2010CS10234

Shivam Singh
2010CS10255

ABSTRACT

Cellular network providers have started to keep extensively large buffers in their base stations to avoid the bursty traffic causing any packet drops. These large buffers lead to large delays in the network as packets get collected in these buffers. This phenomenon, termed as Bufferbloat, has caught the attention of the internet research community lately. Through this project we implemented the Dynamic Receiver Window Adjustment(DRWA) algorithm in the kernel of an android device and performed experiments to gauge the effectiveness of the algorithm in tackling the bufferbloat problem. Our results show XXX improvement by using DRWA.

1. MOTIVATION

The problem of Bufferbloat can lead to serious performance degradations, especially for certain application like VoIP, online gaming, etc. Thus there is a need to identify the magnitude of the bufferbloat problem and then implement solutions to mitigate it. As proposed by Haiqing Jiang and others in their paper, "Tackling Bufferbloat in 3G/4G networks", their algorithm DRWA can lead to approximately 50 % decrease in delays leading to the same percentage increase in throughput. The results claimed by the authors are groundbreaking and can help in mitigating the bufferbloat problem. Thus we implemented DRWA to check for the claims made by the authors.

2. INTRODUCTION

Bufferbloat results in extremely long delays as congestion window of TCP flows usually exceed far above the bandwidth-delay product(BDP) of the network. The excess packets do not get dropped but are stored in the large buffers in the network. This causes large queueing delays and is detrimental to the performance of a lot of applications. Dynamic Receiver Window Adjustment (DRWA) algorithm has been designed to take RTT information from the network in adjusting the receiver advertised window. This way the receiver is able to perform better flow control and limit the congestion window of the sender when delay starts increasing due to bufferbloat. The DRWA algorithm has been discussed in the next section.

3. DRWA ALGORITHM

Before elaborating on the algorithm we first discuss the method by which RTT information is gathered. The estimation of RTT is done in one of the following two ways :-

1. Using Timestamp Option, if available, is the best way to estimate RTT. Timestamp option can be used by a TCP connection if both sender and receiver want to use it. It is requested at the start of the connection in the SYN packet.

2. RTT is estimated as the the time between when a byte is first acknowledged and the receipt of data that is at least one window beyond the sequence number that was acknowledged. It is used if timestamp option is not available.

The execution of the algorithm is triggered by the event when some data is copied to the user space. If the elapsed time is less than the estimated RTT then nothing is done as receiver window adjustment is performed after atleast RTT amount of time. If the elapsed time is more, then a smoothed congestion window estimation is done using the data received in last RTT as can be seen below :-

$$cwnd_{est} \leftarrow \alpha * cwnd_{est} + (1 - \alpha) * data\_rcvd$$

Using the above congestion window estimation and the RTT information, the receiver window to be advertised is set as follows -

$$rwnd \leftarrow \lambda * \frac{RTT_{min}}{RTT_{est}} * cwnd_{est}$$

where RTTmin is the minimum of all the RTT samples that has been estimated and lambda is the factor which determies the tradeoff between the delay and throughput of the flow by defining how aggressive the flow will be.

4. RELATED WORK

The issue of bufferbloat has been known for a while now. With the increase in size of buffers at routers, bufferbloat is now affecting real-time applications like VoIP, online gaming, etc. more than ever. A few solutions have been proposed to tackle this pressing issue. Some solutions have been proposed like reducing the buffer size on the OS and network devices [2], but this is not feasible in todayâĂŹs broadband or wireless systems.

Some research as done by [3] have proven that bufferbloat does not pose much problem in real networks. We would also aim to verify the existence of bufferbloat and also the magnitude to which it affects real networks.

To tackle bufferbloat in cellular networks, an algorithm to resize receiving buffer dynamically called DRWA was proposed by Haiqing Jiang [1] et al which was shown to be effective in 3G/4G cellular networks. We aim to test its validity and effectiveness by implementing and testing it on an Android based mobile device.

5. PROJECT PHASES

5.1 Understanding existing implementation

We used Samsung Galaxy Tab 8.9 as our testing device. Source code of android version 4.0.4 with kernel version 3.0.8 was read to understand the existing implementation of receiver window adjustment. The modular structure of TCP kernel can be understood from figure 1. The receiver module is a part of a larger TCP module which is a part of the Networking module.We found out that the implementation in the kernel was Dynamic Right Sizing (DRS). DRS is a uni-directional adjustment algorithm in the sense that it increases the receiver window size if it thinks that it is limiting the senderâĂŹs congestion window but does not decrease it. Specifically, in each RTT, the kernel makes an estimation of the congestion window that the sender is using and then advertises a receiver window that is twice the estimated congestion window of sender, if it has that much of buffer space available.

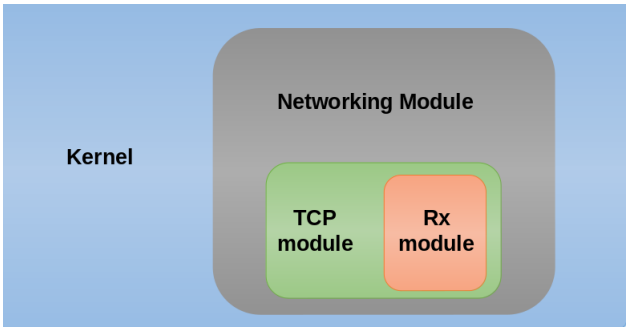

Figure 1: Modular Structure of Kernel

5.2 Code Modifications

After understanding the source code of the kernel we implemented DRWA into the kernel tcp receiver module. This entails including two new variables into the tcp socket structure - rtt_min and cwnd_est. rtt_min is used to store the minimum rtt that is observed from the start of the connection. cwnd_est is used to store the smoothed estimated of congestion window that is performed by the algorithm. It is based on the amount of data that is received each estimated RTT. Once these variables are added to the tcp socket structure, the existing DRS implementation is modified to DRWA.

5.3 Kernel Compilation

We downloaded Android 4.04 kernel source code for our device from Samsung website (opensource.samsung.com). Then, changes were made in the networking module to replace the existing DRS (Dynamic Right Sizing) algorithm by DRWA (Dynamic Receive Window Adjustment). After making the required modifications, the kernel was compiled using Android NDK toolchain (The exact procedure and instructions can be found in the appendix). This generated the kernel image zImage which was then installed on the device.

5.4 Kernel Installation

For being able to install custom kernel images on the device, it is necessary to first install custom recovery image on the device. It facilitates changing kernel images and backing up existing data. For this purpose, we installed ClockWorkMod (CWM) recovery on the device. Once this was done, we needed to pack the generated kernel image zImage into an installabe format.

For this, we compressed the kernel image alongwith some essential kernel modules and installation scripts into a .zip file and placed it into the memory of the device. Then, we install the new patched Android OS by rebooting the device and installing this .zip file from the CWM recovery menu.

6. EXPERIMENT SETUP

Two sets of experiments were carried out - the first experiment was carried out to assess the adaptive nature of DRWA and the second experiment was carried out to assess the efficacy of DRWA in tackling the bufferbloat problem.

For the first experiment a video was downloaded from youtube(size : 20MB) and packet dump was collected using an android application called Shark. Internet was accessed using an Airtel 3G HSPA network.To assess the adaptive nature of DRWA, the experiment included moving the device from a strong signal area into a weak signal area and then back to a strong signal area. This experiment was done twice - once with DRWA implemented in the kernel and once with default implementation(DRS) in the kernel. The packet dumps were then analyzed using tcptrace.

For the second experiment another video was downloaded from youtube(size : 40MB) and packet dumps were collected. While the video was being downloaded an android application called Ping was used to ping the server from where the application was being downloaded. The ping appilcation was configured to use 10 pings before reporting the average RTT. The server was pinged at two points in time during the download - first when 20% of the video had been downloaded and second when 80% of the video had been downloaded. This was done to measure the extent to which bufferbloat increases the average RTT for a TCP connection. This experiment was performed three times on a single day, morning, evening and late night to aaccomodate for changes in traffic patterns. The average of the three iterations was then calculated and has been reported in the results section.

7. EXPERIMENT RESULTS

For the first experiment, the throughput results(figure 2 and figure 3) show a clear behaviour that when DRWA is used, the connection is able to respond much more smoothly to the drop in signal strength. The red line corresponds to the instantaneous throughput while the white line corresponds to the average throughput.This shows that DRWA is much more responsive to changes in network. This is primarily due to the RTT information that is incorporated in DRWA algorithm to estimate the queueing delays in the network.

For the second experiment the average RTT for 20% mark

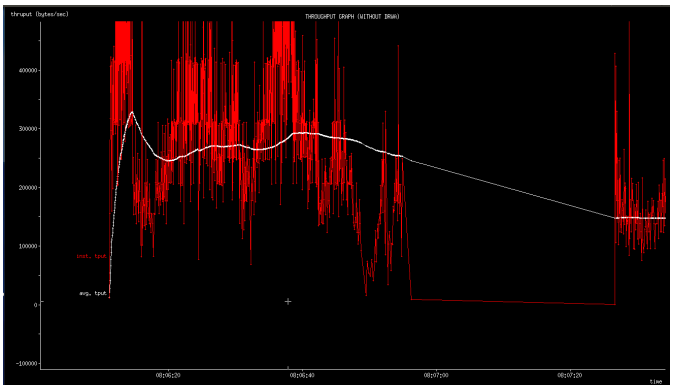

Figure 2: Throughput v/s time plot(Without DRWA)

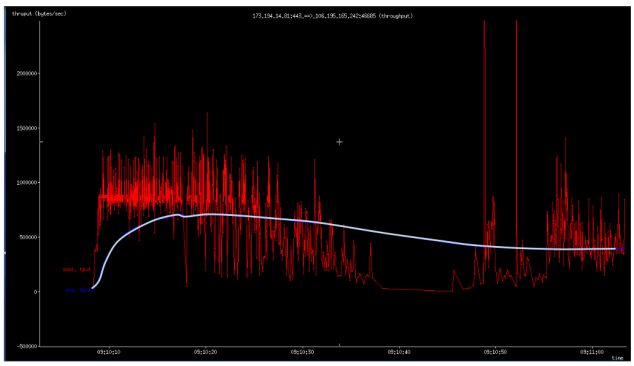

Figure 3: Throughput v/s time plot(With DRWA)

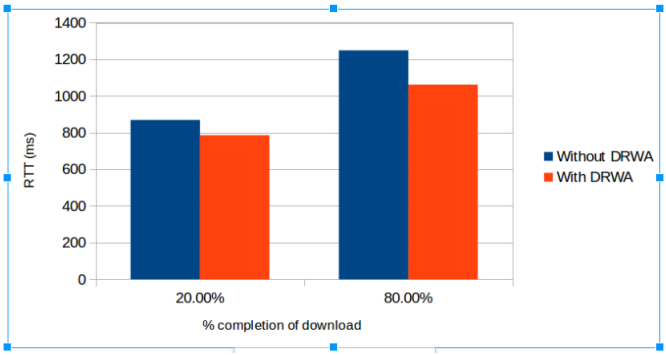

Figure 4: Average RTT while download

and 80% mark, as can be seen from figure 4 show a clear improvement in terms of RTT when using DRWA. The large differences in RTT between the 20% mark and the 80% mark verify the existence of bufferbloat. RTT for the kernel implementation without DRWA shows approximately 50% increase in RTT between the 20% mark and 80% mark which verifies existence of large buffers in Airtel 3G HSPA network. Also, at the 80% mark, the RTT with DRWA kernel implementation shows approximately 17% improvement over the kernel without DRWA. This is a significant improvement owing to the ease of application of the solution.

8. CONCLUSION

Through this paper we implemented Dynamic Receiver Window Adjustment algorithm which is an easy-to-apply solution as it just requires patching the kernel on the receiver side. We carried out experiments to be able to conclude that the problem of bufferbloar does exist in cellular networks today. Also, by using DRWA we could see improvements in average RTT of a TCP connection by upto 17% which is significant given the ease with which the solution can be deployed in real networks. Thus, the problem of bufferbloat in cellular networks is one which needs to be addressed as it can lead to extremely bad performance for certain applications. DRWA provides a good starting point in tackling the bufferbloat problem in 3G networks.

9. REFERENCES

[1] H. Jiang, Y. Wang, K. Lee, and I. Rhee. Tackling Bufferbloat in 3G/4G Networks. In ACM Internet Measurement Conference, Nov. 2012.

[2] Gettys, Jim; Nichols, Kathleen (January 2012),Bufferbloat: Dark Buffers in the Internet, Communications of the ACM 55 (1), ACM, pp. 57 - 65

[3] M. Allman. Comments on bufferbloat. ACM SIGCOMM Computer Communication Review , 43(1):30-37, 2012

Appendix

Installing CWM recovery

1. Download CWM recovery for your device. For Samsung Galaxy 8.9, it can be downloaded from `https://github.com/Koderok/drwa-android-ics/blob/master/cwm_recovery.zip`
2. Place it into the root folder of sdcard of your device.
3. Reboot into download mode. The method varies from device to device. For Samsung Galaxy 8.9, it can be achieved by pressing power and Volume-down button simultaneously for a few seconds.
4. Go to the custom recovery menu.
5. Select "install update from zip" option.
6. Browse to CWM recovery zip and confirm.
7. The CWM recovery will now be installed and your device will be rebooted.

Building the kernel

The original Android 4.0 kernel source code in which we made modifications can be downloaded from the official Samsung website - file GT-P7300_ICS_Opensource_Update1.zip from `http://opensource.samsung.com/reception/receptionSub.do?method=sub&sub=F&searchValue=p7300`

In this code, we made changes to implement DRWA algorithm as described in section <>. The source code with our DRWA implementation can be downloaded from `https://github.com/Koderok/drwa-android-ics`.

The following instructions inform how to compile and install custom patched Android 4.0 kernel on an Android device (tested on Samsung Tab 8.9 P7300). This assumes that you have downloaded the patched kernel source code.

1. Download Android NDK toolchain from its official website - `https://developer.android.com/tools/sdk/ndk/index.html`
2. Make sure NDK binaries are in your environment PATH variable. You might have to do something like this: PATH=$PATH:/<directory_where_ndk_was_downloaded>/android-ndk-r9d/toolchains/arm-linux-androideabi-4.6/prebuilt/linux-x86/bin
3. Then, to generate .config file which will later be of use in kernel compilation, run the following command: make ARCH=arm CROSS_COMPILE=arm-linux-androideabi-samsung_p5_defconfig
4. Then, issue the following command to finally compile the kernel: (Remember to replace 4 by the number of cores on your machine) make -j4 ARCH=arm CROSS_COMPILE=arm-linux-androideabi- A binary file zImage would be generated in the directory arch/arm/boot.
5. Download the kernel package .zip file from `https://github.com/Koderok/drwa-android-ics/blob/master/drwa_kernel.zip` and place the generated file zImage into its kernel/ directory.
6. Place this .zip file into the root of sdcard of device.
7. Reboot into download menu.
8. Clear system cache and data.
9. Select âĂIJinstall zip from sdcardâĂİ option from CWM recovery menu.
10. Browse to the kernel zip file. Confirm when prompted.
11. The patched kernel will now be installed on your device. Reboot upon completion.