Rodrigo Sanchez and Sam Groenjes

Professor Youn

TCP Friendly Congestion Control Algorithms

December 12, 2016

TCP Friendly Congestion Control Algorithms

Introduction

This report is about Transmission Control Protocol (TCP) friendly congestion control algorithms. With the increase in streaming and multicast applications, non-TCP (UDP) traffic has increased disproportionately. While this is great for companies and consumers of streaming applications, these UDP using applications usually forgo the congestion control resulting in these applications using an unfair share of bandwidth. The reason for this greed of streaming applications is that users of the applications would rather have an unthrottled connection with the loss of a few packets provided to them by UDP as opposed to the safer, but throttled connection from TCP. However, the problem with these streaming applications receiving more bandwidth than they should is that it creates the possibility for a congestion collapse and a severe decrease in

TCP traffic. This possibility is caused by the fact that UDP does not have congestion control like TCP allowing UDP the opportunity to monopolize the bandwidth. Some important TCP based applications are web browsers and email clients so the importance of TCP is apparent. TCP friendly congestion control algorithms are algorithms designed to remedy this problem.

The main question we will be addressing in this report is which algorithms perform best when the network is receiving a constant amount of data that causes congestion or receiving bursts of data that cause congestion.

Related Work

Drop Tail

The simplest algorithm we will test is Drop Tail. With this, when the router queue has reached its maximum capacity, any new arriving packets are dropped until the queue has enough room to accept incoming packets.

RED

Random Early Detection or RED is another one of the algorithms we will be using. Floyd and Jacobson write that, " The gateway detects incipient congestion by computing the average queue size. The gateway could notify connections of congestion either by dropping packets arriving at the gateway or by setting a bit in packet headers. When the average queue size exceeds a preset threshold, the gateway drops or marks each arriving packet with a certain probability, where the exact probability is a function of the average queue size" (1) . Gateways in a RED system set the average queue size

at a low threshold while occasionally allowing bursts of packets. The probability of the gateway notifying a connection to lower its window size is proportional the connection's share of bandwidth through the gateway. Gateways in a RED system are designed to go with a transport layer congestion control protocol like TCP. These gateways have no bias towards traffic that is burst and tries to avoid situation where various connections simultaneously decrease the size of their window. RED gateways simplifies the job of congestion control required by the transport protocol and it should be able to be applied to other transport layer congestion control mechanisms that are not the current version TCP, which includes rate-based protocols as opposed to window-based flow control. However, there are a few specific things that are designed solely for TCP/IP networks such as being able to simply mark or drop a packet and have that be enough to notify the transport layer protocol that there is congestion. As opposed to a DECbit congestion control scheme, where the fraction of packets arriving with the congestion indication bit is computed by the transport layer protocol. Adding on to the situation where simultaneous connections decrease their window size, or Global Synchronization, this is especially relevant to TCP/Tahoe because each connection has to go through slow-start, which reduces the window size to one, in response to a dropped packet. RED can also be useful in other scheduling or dropping algorithms. In a situation where packets were marked essential or optional, RED congestion control mechanisms could be applied so that the packets marked optional are the first packets to be dropped.

A key advantage to RED is that it can be applied over a wide range of gateways. Because it works with current protocols, and it does not require all gateways in

existence to adhere to the same congestion control method, RED can applied slowly. RED, in its simplicity, can be implemented with no change to current transport protocols in the TCP/IP networks.

Fair Queueing

The idea of basic fair queueing (FQ) scheduling is fairly simple, but more fair than the even simpler First In First Out. Basically, what fair queueing does is it separates each data flow into its own queue and then proceeds to service each queue in round robin (cyclic) fashion. That is to say, queues 1,2,3 and 4 would each be serviced once per cycle.(Zhao and Lindahl).

The motivation for the creation of FQ was to limit ill-behaved connections. Because in FIFO, or droptail, a user could just send high amounts of data at high speed to control an arbitrarily high fraction of the bandwidth. Some requirements were thought of prior to the creation of FQ such as independence of packet arrival time, disallow abuse by the mentioned ill-behaved connections, and ability to control the delay allocation independently of the bandwidth and buffer allocation.

Immediately obvious, a huge flaw with FQ is that if a connection sends a packet with a size that is significantly bigger than the other connections packets, then there is unfairness among the network. The next algorithm attempts to alleviate this problem.

Deficit Round Robin

Another algorithm is Deficit Round Robin (DRR), which is a variation on fair queueing. DRR takes into consideration the length of packets to be more fair. It does this by setting a maximum packet size for each queue and then sending the packet

back if the packet length is less than the max. The difference between the max and the packet length is stored for future use. If the packet length is greater than the max that queue is skipped over and the value stored is equal to the max. On the next round, each queue is now compared to the max value and the value stored in the previous round.

An example, there are *x* connections that send packets of various sizes to the gateway. Each connection is allowed to send *k* amounts of data in each round robin cycle. Each connection's packets are sent into *Q* queues and then starts the round robin. During the first cycle of round robin, a packet in *Q* can be sent if its length is not greater than the max *k.* If the packet is sent, the difference between the packet length and *k* is added to *credit.* For empty queues *credit* is reset to zero. If the length of the packet is greater than *k* the packet is not sent and *credit* is increased by *k* as the difference between *k* and zero is *k.* This happens for the next cycle, but the packet length is now compared to *k* + *credit.* If max was 2 then the next comparison will be made against 2 + (Max - Previous Packet Length). In the case that the packet was not allowed through the Previous Packet Length is zero.

Stochastic Fair Queueing

Stochastic Fair Queueing (SFQ) is a variation of fair queueing that is suitable for high-speed software or firmware implementation due to its mapping from flow to gateway queue. SFQ is a class of algorithms that are probabilistic variations of FQ (Jacobson and Floyd 1). One of the main algorithms is one in which a hash function is utilized. A hash function is defined to map flows to a fixed set of queues. Two or more

flows may collide; to avoid this the hash function is periodically altered so the flows are unlikely to collide after another pass of the queues.

Motivation for the creation of Stochastic Fair Queueing comes FQ's problems with high speed implementation. FQ has a time complexity of O(log(n)). It is possible for the average computational cost of FQ to exceed the worst case scenario of SFQ.

Class-based Queueing

Class-based queueing is another algorithm that distributes bandwidth proportionally after grouping traffic into classes. The grouping into classes is based on IP addresses, protocols and application types. Each class is a level in a hierarchy which tells the router which packets to drop or delay when congestion occurs. The router queue is typically divided into eight sections and thus eight classes in the hierarchy. Packets in a higher class that guarantee a certain bandwidth do not get dropped while packets in lower classes are dropped according to the algorithm.

Work Completed

In this section, we outline the experiments and analysis for TCP and UDP traffic using the different active queue management algorithms defined in the previous section. The simulation will be done using Network Simulator 2. The network graph will consist of 6 sources that communicate with 6 different destinations. Sources 1 through 4 will use a TCP Reno agent and 5 and 6 will use a UDP agent. These flows will bottleneck as shown in Figure 1 before reaching their destination. We will look at six scenarios. In the first five scenarios the gateway bandwidth capacity will be set to different target

values lower than the aggregate flow traffic so the link experiences different amounts of congestion. Each TCP and UDP agent is set to initially send 0.8 Mbps with packets of 500 bytes with a total bandwidth of 4.8 Mbps. The bottleneck capacities for the five situations are: 4Mbps, 3.428Mbps, 3Mbps, 2.667Mbps, and 2.4Mbps. These values relate to 120%, 140%, 160%, 180% and 200% bottleneck link utilization (LU). Lastly, to simulate a more realistic scenario, the flows will send traffic in bursts so the bottleneck does experience congestion. In each scenario these different active queue management algorithms will be used: DropTail, Fair Queueing, Stochastic Fair

Fig. 1. Simulation network topology

Queueing, Deficit Round Robin, Random Early Detection, and Class-based Queueing. NS will send data about the enqueue, dequeue, received, and dropped packets on each link in the network graph to a trace file. Grep, a command-line utility for searching text data sets, is used to count the number of packets received from each node in the graph. Raj Jain's fairness index and each node's average throughput can then be computed for each algorithm-LU pair. Network animator (nam) will help visualize the network flows for the burst traffic scenario.

Results

First of all, Raj Jain's fairness index equation,

rates the fairness of a set of values where there are *n* users and x_i is the throughput for the *i*th connection. The index has bounds $1/n$ to 1. The index is k/n when *k* connections share the bandwidth equally and the other *k-n* connections receive no bandwidth.

Fig. 2. Jain's Fairness Index for each algorithm-LU pair

From figure 2, class-based queueing had the best results for the fairness index, but all algorithms did better than standard DropTail scheduling except RED which had similar results to DropTail. The lower bars represent the situations you can expect either the TCP flows or the UDP flows to acquire an unfair amount of bandwidth on the gateway link.

Next, the simulation results of each node's average throughput according to the algorithm-LU pair. All flows only try to accomplish the target rate of 0.8Mbps with no excess bandwidth sharing. Comparing figure 3 and figure 4 you can see the throughput difference of the average TCP and UDP traffic. The fairness measure modeled the results well because the average throughputs were mostly similar while using CBQ with one outlier (LU 160%).

Fig. 3. TCP Average Node Throughput for each algorithm-LU pair

Fig. 4. UDP Average Node Throughput for each algorithm-LU pair

The class-based queueing and deficit round robin algorithms punished the UDP flows the most. Notice how DropTail does not punish UDP for being nonreactive to congestion, and so this is how UDP connections are able to monopolize the available bandwidth.

For the last scenario we visualize the burst data through nam with LU 200%. The burst and idles times are taken from exponential distributions so as to seemingly randomize On/Off times. Here is a list of videos of nam: https://www.youtube.com/watch?v=30ynu4PWfxY FQ, https://www.youtube.com/watch?v=z9EtSr_SqN0 CBQ, https://www.youtube.com/watch?v=pufv6gIBbMQ DropTail, https://www.youtube.com/watch?v=DSxr0xRbs1c SFQ, https://www.youtube.com/watch?v=rLbKGtONXek RED, https://www.youtube.com/watch?v=HUWJqTj7kb8 DRR.

Conclusion

We have described a simulation of interaction of TCP and UDP traffic using the six different algorithms we mentioned above. We compared the algorithms using Raj Jain's Fairness Index and the average throughput of TCP and UDP flows. CBQ and DRR were the most fair by a sizeable margin because they punished the unresponsive UDP traffic. If we were to further continue our experiments, we would also analyze burst traffic more indepthly.

References

Demers, A., Keshav, S., & Shenker, S., *Analysis and Simulation of Fair Queueing Algorithm,* Internetworking: Research and Experience, Vol. 1, 2-26. 1990

Floyd, S., & Jacobson, V., (1993) *Random Early Detection Gateways for Congestion Avoidance.* Online Article. Retrieved from http://www.icir.org/floyd/papers/early.twocolumn.pdf

Floyd, S., & Jacobson, V., *Link-sharing and Resource Management Models for Packet Networks,* IEEE/ACM Transactions on Networking, Vol.3, No.4, 1995

Lee, S., Seok S., Lee, S., & Kang C. *Study of TCP and UDP in a Differentiated Services Network using Two Markers System.* Online Article. Retrieved from https://pdfs.semanticscholar.org/29f1/bc0c1aaa03008af0a97238a0ef5ea798b7c8.pdf

McKenney, P., (2002) *Stochastic Fairness Queuing* .* Online Article. Retrieved from http://www2.rdrop.com/~paulmck/scalability/paper/sfq.2002.06.04.pdf

Zhao, Y., & Lindahl, C. (2008) *Method of Performing Deficit Round Robin Scheduling and Structure for Implementing Same.* Patent US7342936 B2. Retrieved from https://www.google.com/patents/US7342936