WPI-CS-TR-00-15 May 2000

Class-Based Router Queue Management for Linux

by

Nicolas Leazard
Miguel Maldonado
Edwin Mercado
Jae Chung and Mark Claypool

Computer Science
Technical Report
Series

WORCESTER POLYTECHNIC INSTITUTE

Computer Science Department
100 Institute Road, Worcester, Massachusetts 01609-2280

Class-Based Router Queue Management for Linux

Nicolas Leazard - wiggins@wpi.edu
Miguel Maldonado - miguel@wpi.edu
Edwin Mercado - emerado@wpi.edu
Jae Chung - goos@wpi.edu
Mark Claypool - claypool@cs.wpi.edu

May 4, 2000

1 Introduction

The Internet has been faced with a huge increase in the volume of traffic and the number of simultaneous active flows that a router must handle. In addition, the increase in the power of computers has enabled new multimedia applications that have very different timing requirements than do traditional text-based applications [CT99, mm-95], such as file transfer or email. The growth in volume and diversity of traffic places new emphasis on congestion control and traffic fairness. Our work implements two router queue management mechanisms that address the problem of congestion and fairness, and improve multimedia performance on the Internet.

Random Early Drop (RED) was the first widely-accepted alternative router queue management algorithm as a replacement for traditional drop tail-based algorithms [FJ93]. RED provides a way to control the average queue size and avoid global synchronization by randomly dropping packets upon impending congestion. While RED handles congestion for TCP flows in a fair manner, non-TCP flows that are unresponsive or have greedier flow-control mechanisms than TCP can take more share of the output bandwidth than TCP flows [LM97].

Class Based Threshold (CBT) was proposed as a class-based reservation mechanism for RED [MKD99], allowing explicit protection of TCP and responsive multimedia flows. While protecting responsive flows from the effects of unresponsive flows, the static resource reservation mechanism of CBT can result in poor performance for rapidly changing traffic mixes and is arguably unfair since it changes the best effort nature of the Internet.

To eliminate the CBT limitations of explicit resource reservation while preserving the CBT strengths of class-based isolation, we propose Dynamic-CBT (D-CBT) [Chu00, CC00]. D-CBT extends CBT to allocate bandwidth fairly among the classes of flows according to the current traffic conditions. This allows protection of TCP and responsive multimedia flows, while strictly limiting unresponsive UDP flows to their fair share of the output bandwidth.

In this work, we describe our implementation of both CBT and D-CBT in the Linux kernel. We start with the RED queueing algorithm for Linux [Kuz99] and extend it to support both CBT and D-CBT. Then, we extend the `tc` command to load CBT and D-CBT, and create a GUI interface for `tc` that allows us to graphically monitor router queue management performance. We illustrate the benefits of class-based isolation on fairness by running experiments in a Linux network testbed, using combinations of responsive and unresponsive flows. Our experiments clearly demonstrate that CBT and D-CBT can provide a much fairer allocation of bandwidth than does RED, and that D-CBT can provides a fairer allocation of bandwidth than does CBT when traffic makeup is changing.

2 Approach

2.1 RED: Random Early Detection

In the Internet, TCP recognizes packet loss as an indicator of network congestion, and its back-off algorithm reduces transmission load when network congestion is detected. The first outstanding active queue management mechanism developed was Random Early Detection (RED), which calculates the impending congestion drop rate according to the weighted average queue size and randomly drops incoming packets according to the rate [FJ93]. RED uses two thresholds, a *min_thresh* and a *max_thresh* to determine congestion. If the queue average is under *min_thresh* then there is no congestion. If the queue average is above *min_thresh* and below *max_thresh* then the router is becoming congested and RED drops packets with increasing probability. If the queue average is above *max_thresh* then congestion is imminent and incoming packets are dropped with a maximum probability (sometimes 1). Suggested guidelines for setting *max_thresh* and *mix_thresh* can be found in [FJ95].

RED revealed the critical problem that unresponsive flows without a proper application level flow control take more share of the output bandwidth then responsive TCP flows. In the worst case it is possible for unresponsive flows to monopolize the output bandwidth while TCP connections are forced to transmit at their minimum rates. This unfairness occurs because the unresponsive flows do not reduce transmission load while TCP flows do, and the same drop rate is applied to every flow at each state. Moreover, in addressing the problem of fairness, there has been a strong argument that unresponsive or misbehaving flows should be penalized to protect well-behaved TCP flows.

2.2 CBT: Class Based Threshold

Class-Based Threshold (CBT) is another active queue management technique that extends RED. CBT categorizes flows into three classes: TCP, UDP, and Multimedia-UDP. It applies a static, predetermined threshold to each of the UDP classes and then uses RED for queue management for all flows. The threshold of each of the UDP flows is the maximum output allowed during congestion. This allows CBT to protect both the TCP and the multimedia-UDP flows from the misbehaving UDP flows.

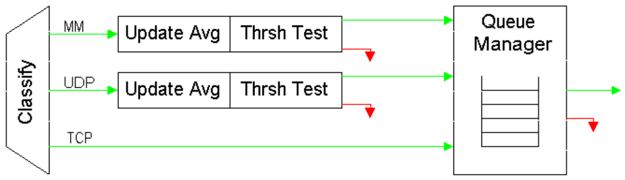

Figure 1: CBT Algorithm

Figure 1 depicts how CBT functions. When a packet arrives at the router it gets classified into one of the three classes. If the packet is a non-TCP packet, it's class' queue average gets updated and then a threshold test is performed with the fixed threshold for that class. If the test fails *(if the classes queue average > the classes threshold)* then the packet is dropped, else it is passed to the RED queue where the RED drop test is performed. If that packet falls within the TCP class, then it gets passed to the RED queue directly.

2.3 D-CBT: Dynamic Class Based Threshold

Although CBT protects both TCP flows and Multimedia-UDP from misbehaved UDP flows, it is inefficient in the way that it allocates bandwidth due to their static thresholds. Dynamic CBT(D-CBT) tries to eliminate the static thresholds and reduce the bandwidth allocation inefficiency found in CBT.

Dynamic CBT (D-CBT) builds on CBT and introduces the notion of flow counting into the queueing discipline field. D-CBT calculates the threshold for each UDP class dynamically such that the sum of the bandwidth assigned to each class represents a fair share of the total number of flows. This method allows for UDP flows not to be subjected to the threshold test while there is no congestion. This method also allows the protecting of the multimedia-UDP and TCP flows while offering fair bandwidth assignment to all the classes.

2.3.1 D-CBT The algorithm

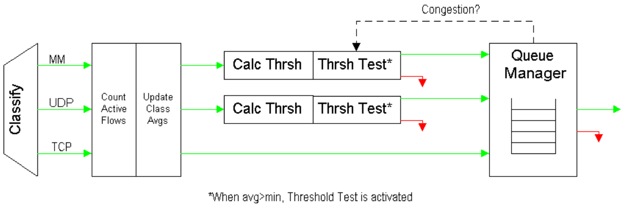

Figure 2: D-CBT Algorithm

Figure 2 depicts how D-CBT functions. This queueing discipline essentially works in the same way that CBT does in that it has the notion of classes introduced by CBT, and has the thresholds for these classes. The difference lies in that the thresholds for D-CBT are dynamic to fairly distribute the bandwidth, as well as protecting well behaved flows from the misbehaved ones. In addition, D-CBT must count the number of flows in each class in order to calculate the dynamic thresholds.

3 Implementation

CBT

CBT was implemented in the Linux kernel as a new module for the QoS networking section. The code for RED, written Alexey Kuznetsov [Kuz99], was used as the base for CBT. CBT recognizes UDP and TCP flows from the TOS field in the IPv4 packet header. Multimedia UDP flows must use the setpriority field, too, for IPv4, or the flow priority field in IPv6.

DCBT

D-CBT directly extends CBT as a new module for the QoS networking section. However, with the added requirement of counting flows for the three classes, D-CBT requires the use of IPv6, and he flows are identified by their IPv6 flow label. A sorted linked-list to keep track of all active flows.

Traffic Controller

The Traffic Controller (`tc`) command is the dynamic link loader for the QoS services in Linux. It allows users to load the different QoS modules from user space, and also allows the user to pass the necessary parameters for the services. In order to allow our new queueing disciplines to be loaded, we needed to extend `tc`. It was necessary to create the required structures for our queueing disciplines and all the functions on how to parse the arguments for loading the queue and the functions to retrieve the desired statistics from it.

Traffic Window

In order to monitor the traffic of flows at the kernel level, we created a graphical interface Traffic Window (`tw`), using GTK+ (the GIMP Toolkit) and GLADE (a user interface builder). The `tw` program uses the traffic controller (`tc`) program as a medium to communicate with the kernel level program, (i.e., RED, CBT, D-CBT, etc). This link allows `tw` to both send data to and receive data from the current router queue management policy.

The first window that pops up when `tw` is executed is the Configure page, as shown in the Figure 3.

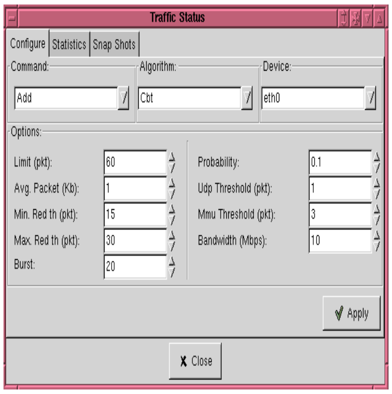

Figure 3: TW Configure Page

This is where the user can either Add an algorithm to a device or delete a device. Also in this page, the user can set up the parameter values that are being passed to the corresponding algorithms. Once this has been setup, the user can press the Apply button to start the process.

4 Experiments

We set up a network testbed consisting of 6 PC's running Linux, where 5 are used for generating traffic and 1 in the middle is as a congested router. For each router queue management discipline, RED, CBT and D-CBT, we ran a 180 second test consisting of 6 multimedia UDP flows sending 210 Kbps, 240 ftp-tcp flows, and 1 UDP blast sending at 10 Mbps. Traffic fairness is measured by the aggregated throughput in Kbytes/sec.

In figure 4 is the aggregated TCP throughput for the RED algorithm under our test. Throughput is acceptable during the first 40 seconds of the test, but at time 40 there is a complete collapse in aggregated TCP throughput. This is when the greedy UDP blast client sends traffic through our router, and exemplifies the great unfairness that can occur in RED with unresponsive UDP flows. When the UDP flow starts, TCP observes the high levels of congestion (indicated by dropped packets), it reduces its window sizes and slows down the rate at which it send packets, while the UDP blast continues to send at its max rate, thus monopolizing the router's queue.

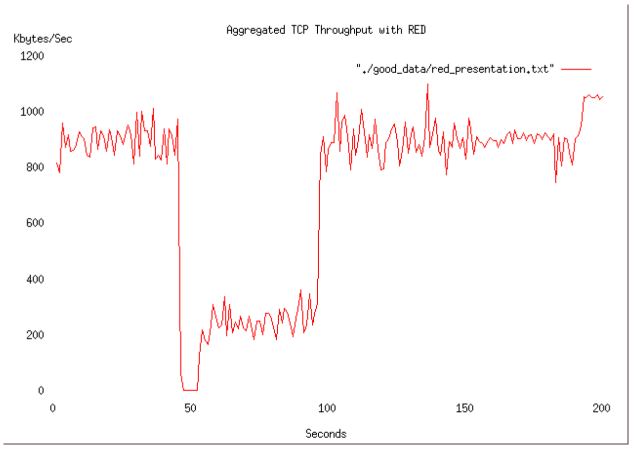

Figure 4: RED aggregate TCP throughput

Figure 5 depicts the aggregated TCP throughput for our CBT implementation. Notice that in this test when the UDP blast client begins the collapse in TCP throughput does not occur. This is a direct result of the class-based thresholds in CBT that limit the UDP client from getting more that its fair share of throughput during the test. However, there is still a slight drop in throughput because the thresholds that are being used are static. They do not actively respond to the current traffic situation at the router.

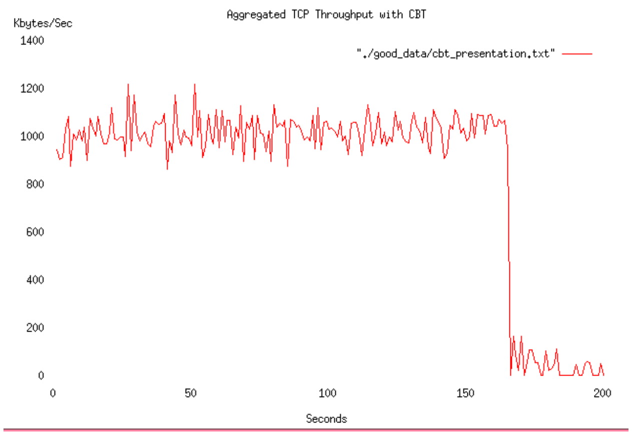

Figure 5: CBT aggregate TCP throughput

Figure 6 shows the aggregated TCP throughput for the D-CBT algorithm. This queueing discipline provides a further improvement over CBT in its existence of dynamic thresholds. Since the "untagged" UDP flow represents only 1/247 of the total number of flows, it is even more limited under D-CBT than in CBT. Also, the threshold test for non-responsive UDP gets turned on whenever its average is 10% above RED's `min_thresh`, unlike "tagged" UDP, whose threshold test gets turned on when whenever its average is above the RED average.

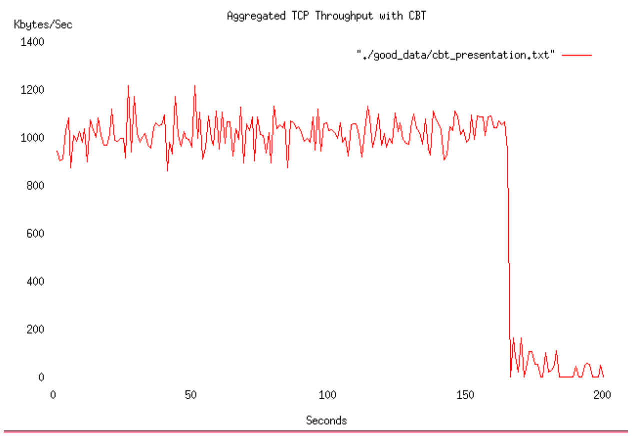

Figure 6: D-CBT aggregate TCP throughput

5 Conclusion

In this project we successfully implemented CBT and D-CBT, which previously have only been simulated, into the Linux kernel. We made use of kernel modules and extended `tc` to load parameters for each router queue management policy. Furthermore, we created a GUI (`tw`) to act in conjunction with `tc` and used IPv6 flow labels for the D-CBT implementation. Our experiment-based results show that our D-CBT implementation under Linux provides a significant improvement in fairness over the existing RED algorithm, and a better allocation of resources over the CBT algorithm. Thus, our D-CBT implementation protects TCP flows from unresponsive UDP flows, does not punish well behaved multimedia flows, and manages resources efficiently.

References

[CC00] Jae Chung and Mark Claypool. Dynamic-CBT - Better Performing Active Queue Management for Multimedia Networking. Technical Report WPI-CS-TR-00-10, Worcester Polytechnic Institute, April 2000.

[Chu00] Jae Chung. Dynamic CBT - Router Queue Management for Improved Multimedia Performance on the Internet. Master's thesis, Worcester Polytechnic Institute, May 2000. Advisor: Mark Claypool.

[CT99] Mark Claypool and Jonathan Tanner. The Effects of Jitter on the Perceptual Quality of Video. In *Proceedings of the ACM Multimedia Conference*, volume 2, November 1999.

[FJ93] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, August 1993.

[FJ95] S. Floyd and V. Jacobson. Link-sharing and Resource Management Models for Packet Networks. *IEEE/ACM Transactions on Networking*, August 1995.

[Kuz99] Alexey Kuznetsov. "ipv6 flow labels in linux-2.2". 1999.

[LM97] D. Lin and R. Morris. Dynamics of Random Early Detection. In *Proceedings of ACM SIGCOMM Conference*, September 1997.

[MKD99] Paris Mark, Jeffay Kevin, and Smith Donelson. "lightweight active router-queue management for multimedia networking". In *Proceedings, Multimedia Computing and Networking 1999*, pages 162–174, 1999.

[mm-95] Multimedia Communications Quality of Service. Technical Report MMCF/95-010, Multimedia Communications Forum, Inc., 1995.