PROVIDING FAIRNESS THROUGH DETECTION AND PREFERENTIAL DROPPING OF HIGH BANDWIDTH UNRESPONSIVE FLOWS

by

Gwyn Chatranon

B.Eng., KMITL, Thailand, 1995

M.S. in Telecommunications, University of Pittsburgh, 1997

Submitted to the Graduate Faculty of

the School of Information Sciences in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2004

UNIVERSITY OF PITTSBURGH

SCHOOL OF INFORMATION SCIENCES

This dissertation was presented

by

Gwyn Chatranon

It was defended on

August 8, 2004

and approved by

Dr. Sujata Banerjee, Associate Professor, School of Information Sciences

Dr. David Tipper, Associate Professor, School of Information Sciences

Dr. Richard Thompson, Professor, School of Information Sciences

Dr. Jose Carlos Brustoloni, Assistant Professor, School of Computer Sciences

Dr. Miguel Labrador, Assistant Professor, University of South Florida

Dissertation Advisors: Dr. Sujata Banerjee, Associate Professor, School of Information Sciences,

Dr. David Tipper, Associate Professor, School of Information Sciences

PROVIDING FAIRNESS THROUGH DETECTION AND PREFERENTIAL DROPPING OF HIGH BANDWIDTH UNRESPONSIVE FLOWS

Gwyn Chatranon, PhD

University of Pittsburgh, 2004

Stability of the Internet today depends largely on cooperation between hosts employing TCP (Transmission Control Protocol) in the transport layer and network routers along an end-to-end path. However, in the past several years, the incidence of various types of non-TCP traffic over the Internet, including streaming media applications, has increased. These traffic flows typically are based on UDP (User Datagram Protocol), and they usually do not employ end-to-end congestion or flow control mechanisms. Such applications can unfairly consume greater amount of bandwidth than competing, responsive TCP traffic. In this manner, the unfairness problem and congestion collapse can occur. To avoid substantial memory requirements and computational complexity, fair Active Queue Management (AQM) schemes requiring no (or partial) flow state information have been proposed over the past several years to solve these problems. However, these schemes have several problems under different circumstances.

This dissertation presents two fair AQM mechanisms, BLACK and AFC, that overcome the problems and the limitations of the existing schemes. Both BLACK and AFC need to store only a small amount of state information in order to maintain and use their fairness mechanisms. Extensive simulation studies show that both of these schemes outperform other schemes in terms of throughput fairness in a large number of scenarios. Not only are they able to handle multiple flows of unresponsive traffic, but they also improve fairness among TCP connections with different round trip delays. AFC, with little more overhead than BLACK, provides additional advantages, including an ability to achieve good fairness in conditions of different-sized and bursty traffic and to provide smoother transfer rates for unresponsive flows, which are usually transmitting real-time

traffic.

This research also includes a comparative study of the existing techniques for estimating the number of active flows, a crucial component of some fair AQM schemes, including BLACK and AFC. A further contribution of this dissertation is the first comprehensive evaluation of fair AQM schemes in the presence of various types of TCP-friendly traffic.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1.0 INTRODUCTION

1.1 BACKGROUND

The Internet is a vast network that connects many millions of computers used by a large, widespread population. A majority of the Internet traffic today, including traffic from world wide web (HTTP), file transfer (FTP), TELNET, and email (SMTP) processes, is carried by TCP (*Transmission Control Protocol*), a transport layer protocol. A key to the success of TCP's deployment has been its *Congestion Avoidance* algorithm, introduced by V. Jacobson in 1988 [33]. This mechanism helps an end host to determine the amount of available on the network so that its transmission rate can be adjusted accordingly. Basically, this approach aims to have the source gradually increase its congestion window, reflecting the amount of data allowed for transmission on the network at a given time, until congestion is indicated by one or more packet drops. In response, the TCP source backs off the amount of transmitting data by reducing the congestion window by half or more. This responsive mechanism, referred to as the additive increase/multiplicative decrease algorithm, keeps the network from being overloaded. It has become a critically important factor in maintaining the robustness and stability of the Internet.

Today, various types of traffic are increasingly deployed over the Internet. Traffic caused by streaming media applications usually relies on UDP (*User Datagram Protocol*). UDP typically does not employs either end-to-end congestion or flow control mechanisms; if these mechanisms are employed, it is on a very limited basis.. Rather, the sending rate is chosen based on the value appropriated for the applications, and no consideration is given to network congestion during the transmission. The lack of end-to-end congestion control on the flows from these applications can results in two serious problems: *unfairness* and *congestion collapse*. An unfairness problem occurs when traffic with no end-to-end congestion control unfairly consumes a greater amount of

bandwidth than competing responsive flows, such as TCP traffic. In some situations, responsive flows may even be shut down due to excessive reduction of traffic. A congestion collapse problem occurs when a large amount of bandwidth is wasted by packets that are discarded before reaching their destination because they were transmitted by a source with no congestion control. Both unfairness and congestion collapse can trouble the current Internet. These problems are discussed in greater detail below.

1.2 PROBLEM OF UNFAIRNESS

An unfairness problem occurs when unresponsive and responsive traffic share the same network and compete for scarce bandwidth. Unresponsive applications do not have congestion control mechanisms and, therefore, have no way of detecting and reacting to network congestion properly. When they are competing with responsive applications for bandwidth , the packets from both types of traffic may be dropped at the onset of congestion. All responsive flows, including TCP, reduce their transmission rates to alleviate the congestion, while unresponsive flows continue tosend their data at the original rate. As a result, a large proportion of the bandwidth can be consumed by the unresponsive flows unfairly.

The danger of the unfairness problem is illustrated in Figure 1 [25]. Under FIFO (First-In First-Out) scheduling, UDP flow (S2 to S4) share the same bottleneck link with three TCP flows (S1 to S3). As shown in the simulation, only when the arrival rate of UDP traffic is low can TCP traffic flows grasp the portion of bandwidth they deserve. As the arrival rate of UDP increases, the UDP flow begins to receive a larger proportion of the bandwidth than the combined bandwidth of the three TCP flows. Eventually, as UDP's arrival rate approaches the bottleneck link bandwidth, this situation almost effectively shuts down responsive TCP traffic.

The unfairness problem occurs even with traffic that has implemented some level of congestion control, but is not TCP-compatible. An example is RTP (Real-Time Transport Protocol). RTP employs an adaptive algorithm to control the amount of outgoing data in a manner similar to the additive increase/multiplicative decrease algorithm used to determine the congestion window in TCP's congestion control. However, although feedback information for the RTP adaptive data control mechanism is provided at a minimum of 5-second intervals, this is much longer than the

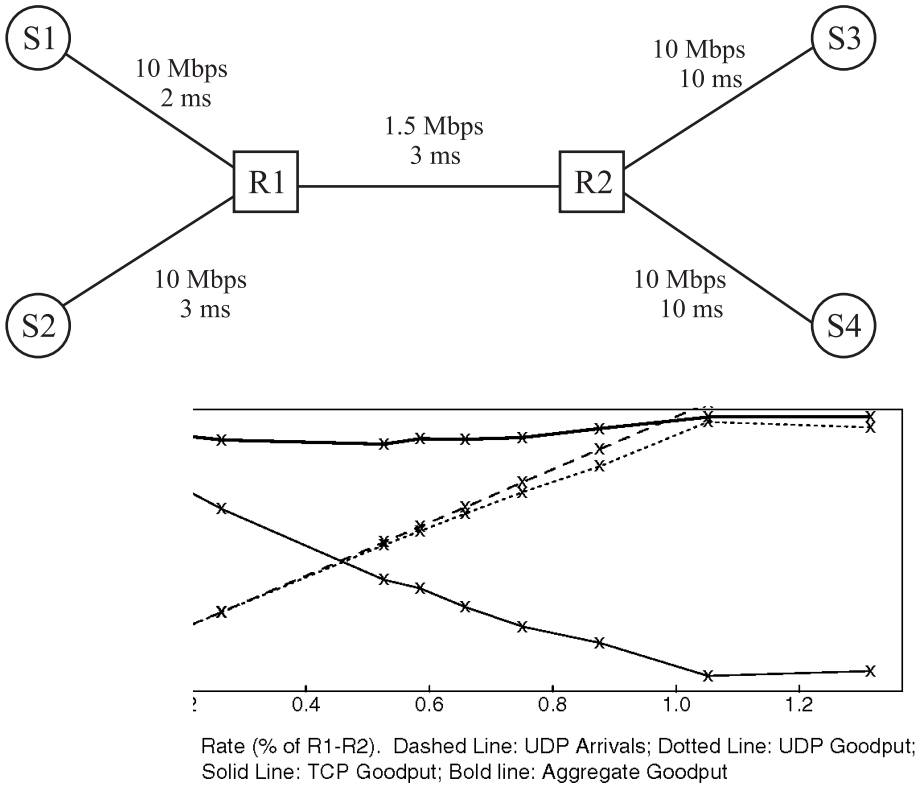


Figure 1: Simulation showing an unfairness problem [25].

time required for the TCP sender to be informed of congestion by a packet drop. As a result, while TCP sources lower their transmission rates, RTP connections continue to transmit at a higher rate and, eventually, they consume most of the bandwidth on the shared link [62].

One well-known solution to an unfairness problem is the per-flow fair scheduling mechanism [14, 60]. Because the bandwidth is allocated almost equally to all different flows passing through a router, per-flow scheduling eliminates the unfairness problem effectively. This merit, however, comes with the cost of increased state maintenance and greater complexity in the routers. This can result in a scalability problem at very high speeds, especially for a high-speed backbone router serving thousands of traffic flows. In addition, that the literature shows that most of the traffic flows on the Internet are short-lived. (I think you need to have reference numbers here.) A fair queueing mechanism that requires a router to maintain a separate queue for every passing connection, even the small flows which together comprise the majority of network traffic, is not intended to be widely implemented.

1.3 PROBLEM OF CONGESTION COLLAPSE

The congestion collapse problem arises when senders continue to transmit packets that will be dropped downstream before reaching their final destinations, resulting in wasted bandwidth from undelivered packets. The primary factor behind this problem is the increasing deployment of applications without end-to-end congestion control. When packets are dropped in these applications, they do not reduce their sending rate. Thus, the network bandwidth can be continuously consumed by undelivered packets due to the unresponsive behavior even if per-flow scheduling is deployed. This situation is illustrated in Figure Figure 2 [67].

The available bandwidth for each of the links in the figure is 8 Mbps, with the exception of the link from router R1 to node S4, which has 2 Mbps of bandwidth. Two flows, each with a constant rate of 6 Mbps, traverse from node S1 to node S3 and from node S2 to node S4, respectively. Assuming that router R1 uses per-flow fair scheduling, each flow receives 4 Mbps of bandwidth on the R1-R2 link. Since the final link of flow 2 has an available bandwidth of just 2 Mbps, about half of the packets from flow 2 are dropped at R2. Hence, the throughput of flow 2 is limited to 2 Mbps while the throughput of flow 1 remains at 4 Mbps. In this case, the flow 2 packets that are

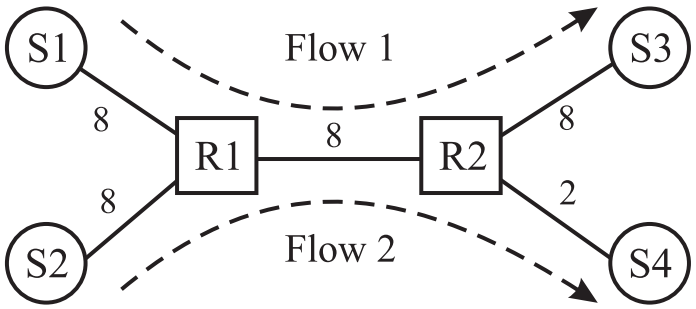


Figure 2: Example of congestion collapse [67].

discarded before reaching their destination node (S4) block the throughput of flow 1 to 4 Mbps, which causes congestion in spite of per-flow scheduling.

The impact of congestion collapse could be more severe if there were more unresponsive traffic flows in the same scenario, as demonstrated by Floyd and Fall's simulation [25]. Using the same topology as Figure 1, with the R2-S4 link bandwidth now set to 128 kbps, ten flows traverse from the left nodes to the right nodes of the network. TCP traffic flows from node S1 to node S3, and a number of UDP flows move from node S2 to node S4. The result is shown in Figure 3. The x-axis shows the number of UDP flows as a fraction of the total flows from 1 to 9; the y-axis shows the aggregate goodput of TCP flows as a fraction of the bandwidth on the R1-R2 link. The Figure provides evidence that a congestion collapse problem worsens as the number of unresponsive UDP flows increases, regardless of scheduling type.

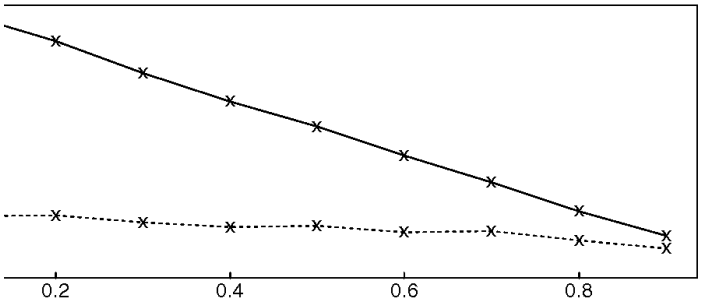


Figure 3: Congestion collapse as the number of UDP flows increases [25].

Consequently, scheduling type is not the key factor for either the congestion collapse or the bandwidth use; rather, the absence of end-to-end congestion control for unresponsive traffic is the key. To date, the current Internet paradism has been without an effective approach to regulating unresponsive flow applications and prevent the congestion collapse problem [32]. However, Floyd and Fall [25] propose that penalizing unresponsive high bandwidth flows would be a concrete incentive for users to implement end-to-end congestion controls that would solve unfairness and congestion problems.

1.4 OTHER CAUSES OF UNFAIRNESS PROBLEM

The unfairness problem is not only caused by long-lived, high-bandwidth applications lacking proper end-to-end congestion control mechanisms. Responsive traffic flows such as TCP connections with round-trip times (RTT) can also lead to problems of unfairness. A TCP connection with a small round-trip time can secure a large portion of bandwidth since it tends to receive acknowledgement packets faster, resulting in rapid congestion window increases. Ott et al. [47] and Padhye et al. [50] separately illustrate this fact in their models of TCP congestion control behavior, which show that the achievable throughput of TCP is inversely proportional to the RTT. Hence, fairness among TCP connections can deteriorate as a result of different round-trip times. A related problem is a flash crowd, which can occur even when traffic flows are transmitted over a responsive TCP protocol. A flash crowd is a large surge in the amount of traffic to a particular server (e.g., HTTP requests for breaking news), which cause an immense increase in the traffic load of the server or routers along the path. During a flash event, a large number of packets from other connections may be dropped because they are not protected from the flash crowds that compete for the bandwidth to access the same server.

Some types of short-lived traffic can also introduce unfairness. One such type is a short-lived UDP traffic flow that is transmitted at a very high peak rate. An example of this is a short duration of video traffic that is later canceled. Another example is any traffic from a source that intentionally transmits unresponsive data in a series of short-lived bursts to avoid detection by a router's fairness mechanism and at a very high peak rate to grasp as much bandwidth as possible. A UDP pulse will show up at the router for a short period of time, then leave. Most routers do

not respond quickly enough to be able to manage this type of traffic. Once detected, this traffic pulse may already cease after a short period of time. As a result, the connection can gain a large amount of bandwidth without being effectively controlled by a router. During the time that the traffic dominates a buffer space, packets from low-bandwidth or responsive flows can potentially be dropped in a large numbers. The situation can be even worse if there are multiple non-responsive UDP flows that, together, arrive at a router for a briefly before leaving; in this case, a number of responsive flows could be totally shut down. Even routers with some existing active queue management (AQM) mechanisms to combat the unfairness problem are unable to cope well with this type of traffic, as described in Chapter 2.2.

1.5 THE PROBLEM STATEMENT

Today's Internet architecture is vulnerable to the unfairness and congestion collapse problems because of traffic without conformant end-to-end congestion control. This misbehaving traffic can consume large amounts of bandwidth, resulting in inferior service to responsive users or instability of network operations. Although most of the current Internet traffic is TCP, there is clear evidence of an increasing number of real time and streaming media applications based on UDP protocol that have no congestion control mechanism or are unresponsive in nature. The negative impact of these unresponsive traffic flows could range from a mild effect to an extreme danger, as discussed in this chapter.

One major challenge in mitigating the unfairness problem is that most mechanisms, such as fair scheduling [14, 60] or per-flow active queue management schemes [68, 45, 4] usually involve some form of per-flow state information at routers along a network path. The merit of these mechanisms thus comes at the expense of memory resources and increased complexity; these aspects can prevent them from being widely implemented, especially for core network routers that carry thousands of traffic flows. In response, several newer fairness mechanisms, such as CHOKe [52], Stochastic Fair Blue (SFB) [19], and CApture-REcapture (CARE) [8] have been proposed. These mechanisms, which will be covered in Chapter 2.2, either have unique, less complex active queue management mechanisms that require no per-flow state information, or they have a small data structure to hold partial per-flow state information. However, these less complex AQM schemes

still have many limitations in solving the unfairness problem.

The goal of this research is to develop a router mechanism that requires no per-flow or only partial state information and aims to solve the problem of unfairness caused by unresponsive applications. The scheme is designed in light of the additional challenge of minimizing complexity to enable practical employment. Therefore, per-flow state information maintained for all active flows passing through the router should be kept minimal. The use of a preferential dropping mechanism is considered as an approach for a penalizing scheme that could be operated in a FIFO scheduling scenario, the simplest queueing discipline widely implemented in the Internet.

The remainder of this thesis is organized as follows. Chapter 2 provides a background of the Internet's congestion control concept. The chapter also discusses the well-known fair AQM schemes that contain no or partial state information, as well as their merits and limitations. The first scheme to combat the problem of unfairness, BLACK, is proposed and compared to other schemes, in Chapter 3. At the end of the chapter, the limitations of BLACK are discussed. Chapter 4 details approaches to solve one of the limitations - the estimation of the number of active flows. Then, AFC, the second scheme to combat the problems with some advantages over BLACK, is proposed in Chapter 5. A comparative evaluation of AFC, BLACK, and the other fair AQM schemes in an extended range of scenarios is made in this chapter. Conclusions of the research are presented in Chapter 6. A discussion of alternatives for controlling traffic in best-effort IP networks is provided in Appendix B.

2.0 LITERATURE REVIEW

This chapter is a review of the literature about existing solutions to the problem of unfairness. While there are several strategies for approaching the problem, the scope of this dissertation is limited to the router-based AQM approach. The first part of this chapter provides background information and focuses on the well-known congestion control mechanisms currently employed in the Internet. The second part of this chapter classifies existing solutions to the unfairness problem and discusses the router-based AQM approach, its design issues and its goals. The remainder of the chapter, which motivates most of the work in this dissertation, reviews and compares the most important router-based AQM mechanisms, giving special consideration to the limitations and problems of these schemes.

2.1 CONGESTION CONTROL MECHANISMS IN THE INTERNET

The Internet was originally designed to connect heterogeneous, time-sharing systems at several locations (e.g., universities and military systems) using a connectionless, packet-switching technique. Packet-switching technology is both flexible and fault-tolerant, as data can be routed in more than one direction, and it can be re-routed, as the need arises. The network can continue to operate even when some parts fail or when it is faced with a military attack or a disaster. Information sent as a message across the Internet may be broken up into several small chunks of data packets, called datagrams. Each datagram is attached to the header of the message, which contains information about where the packet is from, where it should be routed to, and other control details. Each packet may be individually routed across a different sub-network before it reaches its ultimate destination, where it is reunited with other packets and reassembled into a single message.

Despite the advantages of connectionless, packet-switching technology, the network must be well designed if it is to deliver good performance under high-load circumstances. If bandwidth demands exceed available network resources, network performance will decline as packets are dropped or delayed. Lost packets may require retransmission, especially for traffic under the control of a reliable transport protocol, such as TCP; this causes congestion from additional traffic on the network. Figure 4 depicts the problem of congestion. When the network load is light, most packets will be delivered to their destinations properly, and the number of delivered packets will be almost proportional to the number of packets sent from the senders. However, when the network traffic load increases beyond a certain level, the number of delivered packets drops towards zero, indicating that some network queues may be loaded with traffic beyond their capacity.

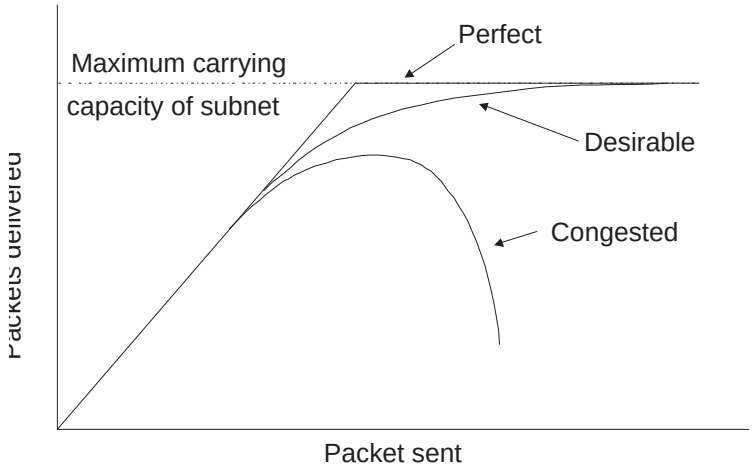


Figure 4: When too much traffic occurs, congestion happens and performance degrades sharply [69].

Unlike a telephone network, a best-effort network such as the Internet does not reserve network resources such as bandwidth or buffer space prior to data transmission. It is not economically feasible for a best-effort network to avoid congestion by over-dimensioning the network, especially when traffic tends to be sporadic in nature. Therefore, the network can experience congestion when many senders simultaneously transmit data at rates exceeding the network's capacity. The Internet's congestion problem poses particular challenges due to the users' widespread locations the difficulty in observing the entire network, and the inability of end hosts to control the network.

Based on the current congestion control paradigm, the network must provide some sort of feedback to its users in order to indicate that congestion has developed somewhere in the network. In response, users should adjust their transmission rates appropriately. This feedback mechanism is a fundamental concept of Internet congestion control architecture and has been the subject of intense research study. A number of techniques have been proposed in the literature, and some of them have became Internet standards. The following section presents major congestion control mechanisms in the current Internet.

2.1.1 TCP congestion control mechanism

Transmission Control Protocol (TCP), the de facto standard reliable transport protocol, is designed to provide reliable, in-order, connection-oriented communication between a pair of hosts, through reliable and unreliable internetworks.

The TCP congestion control mechanism was developed by V. Jacobson in 1988 [33]. At that time, a source could transmit as much data as advertised by the receiver, called the *advertised window*, without considering the network's condition. However, use of an advertised window alone is not enough to prevent a source from sending too much data into the network. Thus, a TCP source was intentionally designed to have an ability to determine the network's available capacity and to use that information to limit its data transmission.

In this manner, the sender must maintain a *congestion window* (cwnd), which is the amount of data that can be transmitted at a given time according to the network congestion level. The maximum window size, indicating the maximum number of bytes of unacknowledged segments at any given time, is then set to be a figure between the *congestion window* and the *advertised window* which reflects both the network capacity and the receiver capacity, respectively.

Generally, there is no direct way for the TCP source to have knowledge about a network. Therefore, the sender slowly probes the network by setting a series of congestion windows until either a timeout occurs or the advertised window is reached. During the exponentially increasing period or *slow-start* phase, a source increases its window by 1 every time it receives an acknowledgment packet (ACK) and sends two packets. In effect, the source doubles the congestion window size with each round trip time (RTT). Before or after a congestion window reaches the advertised

window value, a packet may be lost and a timeout might occur. If the underlying links are reliable, this is an implicit sign that congestion has occurred. In this instance, since the severity of the congestion is unknown, a conservative approach dictates that the congestion window be set back to 1 segment and the slow-start phase be restarted.

In some situations, congestion may take a long time to resolve. In this case, the exponential growth of the congestion window immediately after congestion has been detected may be overly aggressive and may worsen the condition. Therefore, Jacobson proposed a new algorithm for handling congestion. This algorithm begins with the slow-start phase, which is followed by a linear or *congestion-avoidance* phase, in which the sender increases its congestion window by 1/(*current congestion window*) each time it receives an ACK. Once congestion occurs,the source sets its *slow-start threshold* value to be half of the current congestion window, sets the congestion window to a value of 1, and restarts the slow-start phase. When the congestion window reaches the threshold value, the source switches to the linear phase, in which the congestion window is increased linearly, as illustrated in Figure 5 [65].

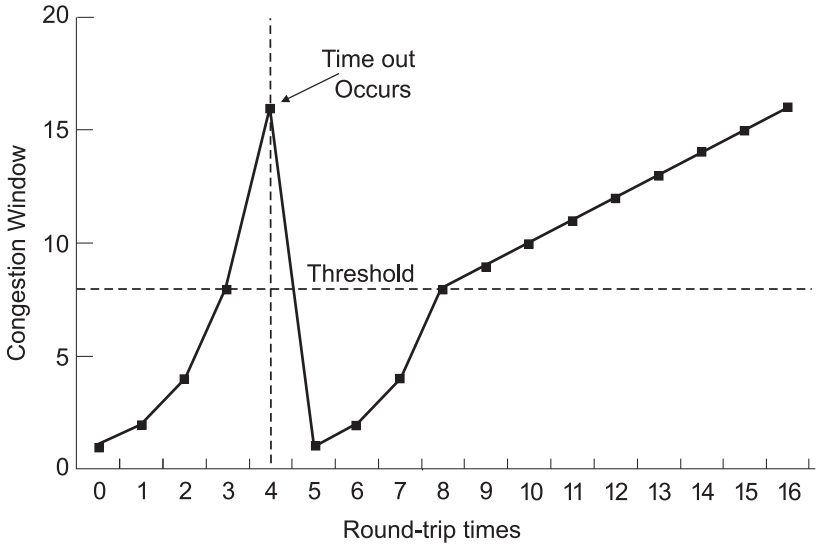


Figure 5: TCP slow-start and congestion-avoidance phases [65].

2.1.1.1 Adaptive retransmission timer As described in section 2.1.1, a traditional TCP implementation uses a timeout mechanism to alert the TCP sender to the loss of data packets. After each

packet is transmitted, the sender waits for an ACK from the receiver. If the ACK is not received within a *retransmission timeout* (RTO) interval, the sent packet is considered to have been lost and, therefore, to be in need of retransmission. The key factor here is the length of the RTO interval. If the interval is set too long the sender may end up wasting time waiting for lost packets. If the interval is too short, the packet may be retransmitted unnecessarily while its ACK is still in transit. This situation implies that timeout interval is related to RTT; thus TCP utilizes a smoothed round trip time (SRTT) to estimate a round trip delay time as:

In a traditional TCP implementation, TCP sender detects the loss of the packets using a timeout mechanism. After transmitting each packet, the sender waits for an ACK from the receiver. If the ACK is not received within a *retransmission timeout* (RTO) interval, a corresponding packet is considered as a lost packet and hence needed to be retransmitted. The key factor here is the length of RTO. If the value is too large, the sender may end up wasting time waiting for the lost packet. If the interval is too small, the packet may be retransmitted unnecessary as its ACK may still be in transit. This concern implies that timeout interval is related to RTT, thus TCP implements a SRTT (smoothed round trip time) to estimate a round trip delay time as:

$$SRTT(K+1) = \frac{1}{8}SRTT(K) + \frac{7}{8}RTT(K+1) \tag{2.1}$$

In the original implementation of TCP, the RTO timer is simply a multiplication of SRTT and a constant value. However, this approach can result in poor network performance because a connection may have a relatively high RTT variance. In a low variance environment, the RTO may be too high, and in a high variance environment, it may not be able to guard against unnecessary retransmissions. Taking into account RTT variance, Jacobson proposed a more effective approach for estimating RTO. This approach, based on a mean deviation of the RTT, is as follows

$$\begin{aligned} SDEV(K+1) &= \frac{1}{4} \mid RTT(K+1) - SRTT(K) \mid + \frac{3}{4} \times SDEV(K) \\ RTO(K+1) &= SRTT(K+1) + 4 \times SDEV(K+1) \end{aligned} \tag{2.2}$$

In this equation, the standard deviation of RTT is approximated by the mean deviation of the RTT samples (SDEV). In addition, if the ACK of the retransmitted packet fails to arrive within the

timeout interval, the TCP sender will double the RTO value every time it tries to retransmit the packet. This technique is referred to as a binary exponential back-off.

2.1.1.2 Variants of TCP To date, there are three well-known variants of TCP, namely *Tahoe*, *Reno* and *Vegas*. TCP Tahoe, first implemented in 1988, has all the features of traditional TCP, including a refined timeout calculation and the *Fast Retransmission* algorithm [34] that helps to expedite the retransmission process. Specifically, Fast Retransmission ensures that when a receiver gets an out-of-order packet, it issues an ACK for the last in-order packet it received. Then, it continues transmitting the ACK with the same sequence number for each incoming packet, until an expected packet arrives. When the sender receives a duplicate ACK, it is able to assume that an unacknowledged packet was either delayed or lost. To ensure that the packet was actually lost rather than simply delayed, a sender may wait until it receives three duplicate ACKs, before retransmitting the missing packet. Since the RTO timer is typically set to a much higher value than a round trip time, it is likely Fast Retransmission speeds up retransmission efficiently.

TCP Reno, first implemented in 1990, behaves in much the same way as TCP Tahoe, but it acts more aggressively with the *Fast Recovery* algorithm. For fast recovery at the onset of congestion, instead of reducing the congestion window to 1 and beginning a slow-start phase, the sender cuts the congestion window to half of its current value and uses incoming duplicate ACKs to send subsequent outgoing packets. The sender then proceeds with the linear growth congestion-avoidance phase, without wasting time exponentially increasing the congestion window again.

TCP Vegas enhances the congestion avoidance of TCP Reno by adjusting the congestion window not only according to packet loss but also according to observation of RTTs of packets that have been sent before. Larger RTTs indicate a congested network which results in a decreased window size. On the other hand, smaller RTTs indicate a reduction in network congestion and signal that a traffic source can increase its congestion window size. Theoretically, congestion window size will eventually converge on an appropriate value.

2.1.1.3 Mathematical models of TCP congestion control mechanism Several variants of TCP have been proposed during the past decade, and most of these preserve two fundamental components of the congestion control mechanism [47]:

1. The TCP source uses a dropped packet as an indication of network congestion and responds by decreasing the congestion window by at least half. This decrease is meant to reduce the effective sending rate, thereby relieving network congestion.
2. In the congestion-avoidance phase, the TCP sender increases the congestion window by, at most, one packet per round trip time.

TCP variants may differ in their responsiveness to congestion and aggressiveness in obtaining available bandwidth. For example, TCP Tahoe decreases the congestion window down to one, rather than by half as TCP Reno does. Some TCP variants send an ACK packet for every two data packets, increasing the congestion window by less than one packet per round trip time during the congestion avoidance phase. Nevertheless, these two components usually enable the determination of the upper limit for the sending rate of TCP.

Several TCP models have been developed to explain TCP's flow and congestion control mechanisms [49, 51, 3, 72, 61]. The simplest one, *The Stationary Behavior of Ideal TCP Congestion Avoidance* [49], has been used as a fundamental concept in the development of several mechanisms addressing the unfairness problem. This model assumes the following:

- Only a single packet is dropped when the congestion window reaches W packets.
- Uniform (non-bursty) average packet drop rate of p.
- TCP segments are size B bytes.
- TCP runs over a path with sufficient bandwidth and a fairly constant round trip time of RTT seconds.
- The sender always has data to transmit and the receiver has infinite buffers.
- The details of TCP data recovery and retransmission are neglected.

Under these assumptions the congestion window has a periodic saw-tooth shape, as shown in Figure 6.

As shown in the figure, once the congestion window reaches W packets, it is reduced by the sender to half ($W/2$). If every segment is acknowledged, the congestion window increases by one packet per round trip; therefore each cycle of the saw-tooth shape equals $W/2$ round trips or RTT * $W/2$ seconds. The total data delivered is depicted as the area under the saw-tooth, equal to $(\frac{W}{2})^2 + \frac{1}{2}(\frac{W}{2})^2 = \frac{3}{8}W^2$ packets per cycle. An alternate way to calculate this number is by accepting

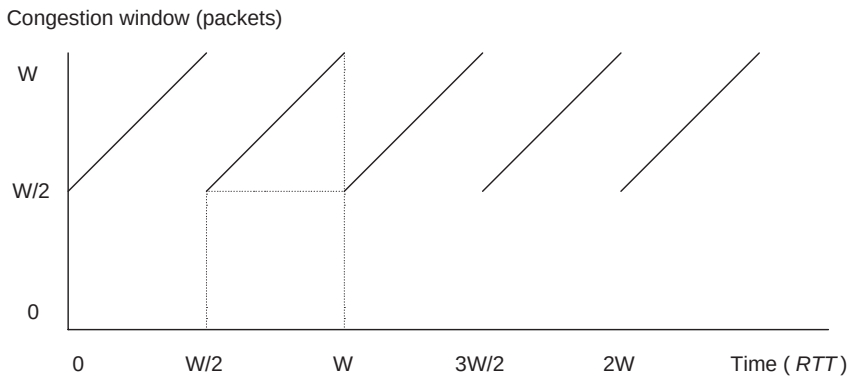


Figure 6: TCP congestion window behavior in steady state of a simplified model.

that, in each cycle, the congestion window starts at $W/2$ and increases by at most one per round trip time until it reaches W. Therefore, the sender transmits at least $\frac{W}{2}+(\frac{W}{2}+1)+(\frac{W}{2}+2)+...+W \approx \frac{3}{8}W^2$ packets per cycle. Hence, the fraction p of the packets that are dropped is then bounded by $p \leq \frac{8}{3W^2}$ which yields:

$$W \leq \sqrt{\frac{8}{3p}}. \tag{2.3}$$

As Equation (2.3) demonstrates the upper bound of a congestion window W for a TCP connection in which single packets are dropped; in this situation, the maximum throughput over a single cycle of the steady-state model then equals:

$$T = \frac{\text{Data per cycle}}{\text{Time per cycle}} = \frac{B * \frac{3}{8}W^2}{RTT * \frac{W}{2}} = \frac{B/p}{RTT\sqrt{\frac{2}{3p}}} = \frac{1.5\sqrt{2/3} * B}{RTT * \sqrt{p}} \tag{2.4}$$

Padhye, et al. [51], use stochastic analysis in their proposal of a more precise model of TCP congestion control behavior in steady state, including the effects of timeouts and retransmissions as shown below:

$$T \approx \min\left(\frac{W_{max}}{RTT}, \frac{1}{RTT\sqrt{\frac{2Bp}{3}} + T_0 \min(1, 3\sqrt{\frac{3Bp}{8}})p(1+32p^2)}\right) \tag{2.5}$$

In this case, T_0 is a retransmission timeout, and W_{max} is the maximum congestion window as limited by the receiver's buffer size.

In a TCP connection with delayed acknowledgments, in which the receiver sends only one ACK for every two received packets, the sender's congestion window increases more slowly. Since the ACK triggers the sender to increase its window and, in a delated-ACK situation, the sender receives ACK packets at a slower rate, then there is a slower rate of congestion window increase. In this case, the fraction p of the sender's packet drop rate is:

$$p = \frac{1}{\sum_{i=0}^{W}(W/2 + i/2)} \approx \frac{1}{(3/4)W^2} \tag{2.6}$$

Applying the same method to the throughput equation from Equation 2.6, the upper bound for the arrival rate when the receiver uses delayed-ACK is:

$$T = \frac{1.5\sqrt{1/3} * B}{RTT * \sqrt{p}} \tag{2.7}$$

These models provide a way to calculate the theoretical throughput of a TCP connection for a particular set of assumptions and compare it against the actual throughput of a specific flow as measured to draw conclusions about its friendliness to TCP. Therefore, these formulas have been used in the design of TCP-friendly protocols for streaming media applications [46, 27] as well as the development of router-based mechanisms that provide fairness between TCP and non-TCP traffic.

2.1.2 Active Queue Management

Traditionally, the Internet router uses drop-tail queueing to manage network traffic by accepting packets until the queue is full, then dropping any additional arriving packets until there is available space in the queue again. However, a disadvantage of drop-tail queueing is that a full queue is sustained most of the time and feedback about congestion is sent to the end host (via a packet drop) very late. Since packet dropping happens only once the queue is full, end hosts have no opportunity reduce their transmission rate before congestion seriously develops. It is also possible that packets from several sources are dropped at the same time when the queue overflows. These traffic sources then reduce their transmission rates simultaneously, which could lead to periods

of extreme link under-utilization alternating with periods of congestion. This phenomena, *global synchronization*, is a serious problem of drop-tail queueing.

Active queue management is designed to solve this problem by detecting congestion before the buffer overflows and providing feedback about congestion to the end hosts before a serious problem occurs. Typical active queue management mechanisms mainly focus on reducing packet transfer delay, keeping the steady state buffer size at low levels, and avoiding global synchronization problem.

Random Early Detection (RED) is the most well-known active queue management. RED, designed to detect the onset of congestion as indicated by average queue size, also aims to maintain a low average queue size. RED requires no state information concerning bandwidth usage of individual flow. Arriving packets are randomly dropped, or marked, when the potential of a buffer overflow is determined to exist according to monitoring of the average queue size. The dropping of packets is a means of notifying the transport-level user about the impending congestion. RED mechanisms also prevent a global synchronization problem by detecting congestion early and randomly dropping the packets of various users. Since congestion is likely be caused by a burst of traffic from one or few sources, the dropping of arriving packets by drop-tail queueing is biased against bursty traffic over non-bursty traffic having the same average rate. RED avoids this bias.

To achieve these goals, the RED router must maintains two thresholds, max_{th} and min_{th}, with a weighted moving average formula that estimates the average queue size, avg. To detect the onset of congestion, the average queue size is compared to two thresholds, as shown in Figure 7. If the average queue size is below the minimum threshold, minth, congestion is assumed to be minimal and all packets are accepted. If the average queue size is greater than the maximum threshold, max_{th}, congestion is assumed to be serious and all incoming packets are discarded. If the average queue size is between the two thresholds, it might indicate developing congestion. In this case, the arriving packets are discarded with a probability P_a and accepted with probability $1 - P_a$. This probability depends on two factors:

- The dropping probability increases as the average queue size approaches the maximum threshold, max_{th}.
- When the average queue size is between the two thresholds, a counter $count$ is incremented every time an arriving packet is queued. The higher the value of $count$, the higher the probability

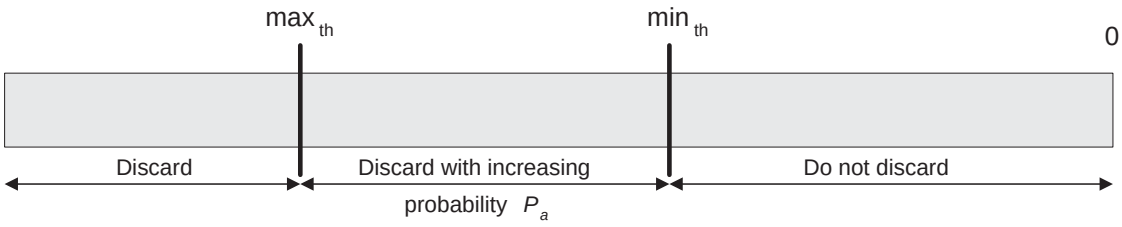


Figure 7: RED Buffer. (Reproduced from [65])

of a packet being discard.

Determining dropping probability P_a begins with the calculation of a temporary probability P_b, where

$$P_b = P_{max}(avg - min_{th})/(max_{th} - min_{th}) \tag{2.8}$$

This value increases linearly from 0, when the average queue size is equal to min_{th}, to a maximum value of P_{max} when the average queue size is at max_{th}. The final dropping probability P_a is given as

$$P_a = P_b/(1 - count \cdot P_b) \tag{2.9}$$

All incoming packets have the same dropping probability; thus, RED drops packets in proportion to the connections' share of the bandwidth. Through random dropping, RED is able to avoid the global synchronization problem.

The average queue size is used to filter out transient congestion that may occur at the router. The average queue size (avg) can be calculated using a weighted moving average formula $avg = (1 - w_q)avg + w_q * q$, where q is an instantaneous queue size and w_q is a weight to determine how fast the algorithm will respond to changes in the queue size. If w_q is set too high, RED may not filter out the transient congestion. If w_q is too low, avg may respond too slowly to changes in average queue size. The min_{th} should be set to a considerably large value if the incoming traffic is bursty, in order to maintain high link utilization.

2.1.3 Explicit Feedback Schemes

With TCP congestion control mechanism, the congestion is implicitly indicated to the source through packet drops at the router. This model works well with best-effort traffic that requires no delay or loss restriction. However, some applications, such as Telnet, are delay-sensitive, while some applications, such as voice and video transfer, are loss-sensitive. In some situations, unnecessary packet drops may result in retransmissions and cause increasing delays for the users. This could have a negative effect on both loss-sensitive and delay-sensitive applications. Consequently, several explicit feedback schemes have been developed so that routers can detect incipient congestion and provide explicit feedback to the source rather than just begin droping packets. In this way, the delay-sensitive or loss-sensitive sender can adapt to network conditions without experiencing the impact of dropped packets or delay from retransmissions [55]. Examples of such mechanisms include ICMP (Internet Control Message Protocol) Source Quench messages, DECbit congestion avoidance scheme [56], and ECN (Explicit Congestion Notification) [21, 55].

ICMP Source Quench messages were initially designed to inform data sources that they were sending packets too fast to be processed. As the buffer at the router fills up, further arriving packets are dropped and a source quench message is returned to the source. The source then slows down its transmission rate. However, this technique is rarely used because the Source Quench messages themselves consume addition bandwidth, thus increasing network congestion [64].

In the DECbit scheme [56], the router sends a congestion-indication bit in the packet header to inform the sender about network congestion. When the packet arrives at the router, the router computes the mean aggregate queue length of all sources. If the average queue size exceeds a certain threshold, the router sends the bit in the packet header. The receiver copies the congestion-notification bit into the header of its acknowledgment packet and sends it to the source, as illustrated in Figure 8. In contrast to the ICMP Source Quench message, the amount of feedback caused by congestion notification in the DECbit scheme is minimal. The sources use window flow control in which the window size is adjusted dynamically according to the series of congestion-indication bits it receives.

Based on the DECbit scheme, Explicit Congestion Notification (ECN) was introduced in 1994 [21] and proposed as RFC 2481 in 1999 [55]. ECN scheme mandates that routers provide conges-

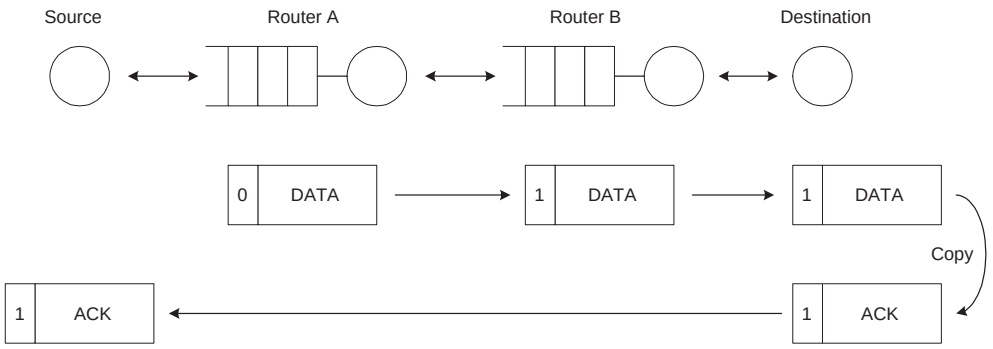


Figure 8: DECbit scheme [37].

tion indication for incipient congestion prior to the buffer overflows, such as is done by the RED mechanism. Two bits of ECN data are needed in the IP header: the *ECN-Capable Transport* (ECT) bit, to indicate whether the end-points of the transport protocol are ECN-capable, and the *Congestion Experienced* (CE) bit, set by the router to notify the end nodes about incipient congestion. The arriving packet can be either marked by setting a CE bit or dropped at the router in response to the congestion. ECN, however, requires support from the transport protocol. At the present time, the proposal defines new mechanisms for ECN operation only for TCP protocol and leaves the modification of ECN for other transport protocols to further research. In ECN mechanisms for TCP, the two endpoints have to verify that they are both ECN-capable during the connection setup phase. In the TCP header, an ECN-Echo flag is reserved for the receiver to use when informing the source about congestion when a CE packet has been received. In addition, a Congestion Window Reduced (CWR) flag is reserved to inform the receiver when the congestion window has been reduced. Congestion is indicated by a single packet drop in non-ECN-Capable TCP. ECN mechanisms have been tested and shown to improve throughput over NON-ECN Reno TCP for bulk transfer as well as transactional transfer [59], due to the need for fewer retransmissions.

2.2 ROUTER-BASED AQM MECHANISMS TO SOLVE THE UNFAIRNESS PROBLEM

2.2.1 Taxonomy

During the past several years, a number of mechanisms have been proposed to solve the problem of unfairness. Some of these mechanisms aim at solving the problem from inside the network, and some focus on developing a responsive or TCP-congestion-control-compatible protocol for use at traffic sources and destinations. These mechanisms can be classified in several possible ways. In this research, the algorithms are classified as either *router-based* or *end-system-based*, based on their implementation placement.

Router-based algorithms are deployed inside networks (i.e., in the routers) to regulate the traffic flow that causes the unfairness problem. Some of these schemes introduce a level of punishment that provides an incentive for a protocol implementer to deploy end-to-end congestion control mechanisms, a crucial factor in solving the congestion collapse problem. End-system-based algorithms are aimed at defining the end-to-end flow and congestion control mechanism to be implemented by the end systems. In either case, if the protocol is designed to be compatible with TCP, it is called a TCP-friendly protocol. Although both of these approaches are important, having the Internet relying purely on end users' decisions about flow and congestion control would be a potential risk to network performance, at least in terms of the unfairness problem. Therefore, development of a router-based mechanism which addresses a fairness problem becomes a crucial and necessary task and is the main focus of this research.

Many different protocols and algorithms have been based on each approach. Several TCP-friendly end-system-based solutions have been developed; these typically use Equations 2.4 or 2.5 to adjust the sender's transmission rate or rely on other strategies to increase or decrease the *congestion window* of TCP. Some of the most important rate-based examples of TCP-friendly transport layer protocols are TCP-Friendly Rate Control Protocol (TFRC) [27], Rate Adaptation Protocol (RAP) [57], Loss-Delay Based Adaptation Algorithm (LDA) [63], and TCP Emulation At Receivers (TEAR)[58].

The main goal of each of these protocols is to transmit data smoothly in a TCP-friendly manner

so that users of streaming applications perceive the handling of network traffic to be more effective than when they are using TCP. Windows-based TCP-friendly end-to-end protocols have also been devised, such as the family of Binomial algorithms, including Inverse-Increase/Additive-Decrease (IIAD) and SQRT [6], and the General Additive Increase Multiplicative Decrease (GAIMD) [37] algorithm. These generalize the same TCP protocol using different values for the parameters a and b that define the increase and decrease strategy[1]. For a detailed description of these schemes, the reader is directed to the references cited and survey papers that already exits in the topic, such as has been written by Widmer and colleagues [70].

Router-based mechanisms are presented in a greater detail in this chapter[2]. The following two sections address the main issues to consider when designing such algorithms and the most important available schemes. In particular, *TCP Model-based Identification Scheme* [25], *Longest Queue Drop (LQD)* [68], *Fair Random Early Detection (FRED)* [45], *Balanced RED (BRED)* [4], *Stabilized RED (SRED)* [48], *CHOose and Keep for responsive flows (CHOKe)* [52], *Stochastic Fair Blue (SFB)* [19], and *CApture-REcapture (CARE)* [8] are discussed.

2.2.2 Main Design Issues and Goals

One of the most important aspects of a router-based scheme is its ability to achieve fairness among competing flows, particularly when unresponsive high-bandwidth flows share the bottleneck link. For instance, these flows can be identified and some sort of preferential dropping policy or similar active queue management scheme applied, or another unique technique may be employed, as will be discussed in the following section. Some schemes provide a restriction or punishment for identified unresponsive flows that consume more bandwidth than their fair share, either at the time of congestion or during light loads, as a mean of promoting the use of end-to-end congestion control mechanisms. From this discussion, it can be inferred that two important issues are the identification of unresponsive flows and the calculation of fair share. Routers have to accurately identify which flows should be regulated so that they do not accidentally punish well-behaved flows; once identified, the flows in need of regulation should be penalized fairly. These and other important issues to consider during router-based mechanism design are explained below.

[1] For TCP, $a = 1$ (additive increase) and $b = 0.5$ (multiplicative decrease).
[2] Most of the materials in this section are to be published in [10]

- **Ability to identify and regulate unresponsive flows.** Identification of unresponsive flows may or may not be necessary to solving the unfairness problem. Nonetheless, the identification of traffic that is harmful to other traffic provides extra flexibility and, usually, better results than when such identification does not occur. For example, a router may want to punish unresponsive traffic as an incentive for a well-behaved end-to-end congestion control, and it may wish not to punish well-behaved flows at all. As another example, one may want to arrange flows in a less priority queue and to restrict them based on flow or another method so that they can co-exist with good-behaved flows. Flow identification is not for free, though, and care must be taken to make the identification scheme both simple and scalable.
- **Fairness.** Although fairness is a desirable goal for all of the schemes of this kind, its meaning need not be as strict and complex as that of a fair scheduling mechanism. Since flows with no end-to-end congestion control could gain increasing amounts of bandwidth and might even completely shut down all responsive flows on the link, placing some regulation or preferential dropping on these flows may be enough to limit the effect of this problem. This could be done without using any complex scheduling algorithm. In other words, the degree of fairness could be relaxed, allowing unresponsive flows that need more bandwidth to obtain the resources they need as long as they are not harmful to other flows and the network is not congested.
- **Need to estimate the number of active flows.** Several schemes require a good estimate of the number of active flows traversing the router without a large amount of memory space required in order to calculate the fair share and achieve other goals. This is an important area of active research closely related to router-based schemes. In Chapter 4, a brief overview of the most important mechanisms for estimating the number of active flows and an evaluation scheme are provided.
- **Simplicity of Operation.** One approach to solving the unfairness problem is to use per-flow scheduling mechanisms that separately regulate the bandwidth of each flow, as stated by Keshav [37]. However, as demonstrated through a series of simulations conducted by Floyd and Fall [25], while this mechanism may alleviate the problem of unfairness, the congestion collapse problem can still occur, regardless of the scheduling type. Per-flow scheduling also introduces a high cost in terms of state information and complexity in order to achieve fairness. A simple, yet effective mechanism, rather than a computationally complex one, becomes one

of the most important goals in a router-based approach. This is one of the main reasons why most schemes still use FCFS (First-Come First-Serve) as the scheduling discipline.

- **Scalability.** Scalability is another important issue. Since these schemes are part of the packet forwarding process and core routers handle thousands or even millions of flows, the mechanisms must scale well. For instance, the complexity of the algorithm and the use of router resources should not be proportional to the number of flows. Simplicity and scalability are related and important issues to consider when implementing the algorithms in practice.

2.2.3 Description of the Most Important Schemes

Router-based AQM schemes can be categorized into schemes that need full per-flow state information and schemes that do not need full per-flow state information. The latter can require no per-flow state information, or they can require only partial state information. At the same time, the schemes that need no per-flow state information can be further divided in two groups depending on whether or not they need to estimate the number of active flows. Figure 9 is a diagram showing this classification. This research focuses on those mechanisms that do not need full state information, primarily for scalability reasons. Even though the state-full AQM schemes usually have less computational complexity than fair scheduling mechanisms, they still have high space complexity. As a result, these mechanisms have rarely been implemented in high speed or backbone routers.

Note that there is another mechanism, called Core-Stateless Fair Queueing (CSFQ) [66], that aims at providing fairness without maintaining full per-flow state information at the core routers. However, the edge routers under this scheme need to calculate the arrival rate of every passing flow and append this value to the header of every packet such that the core routers could provide fairness based on this information. Because CSFQ's implementation is much different from the other schemes, i.e. the need for a tight coordination between edge routers and core routers, and additional overhead in packet headers, which turn to be its drawbacks, CSFQ is thus not included in this section.

The rest of this chapter provides a description of each of these AQM schemes. Table 1 provides a qualitative comparison of the main design issues discussed previously.

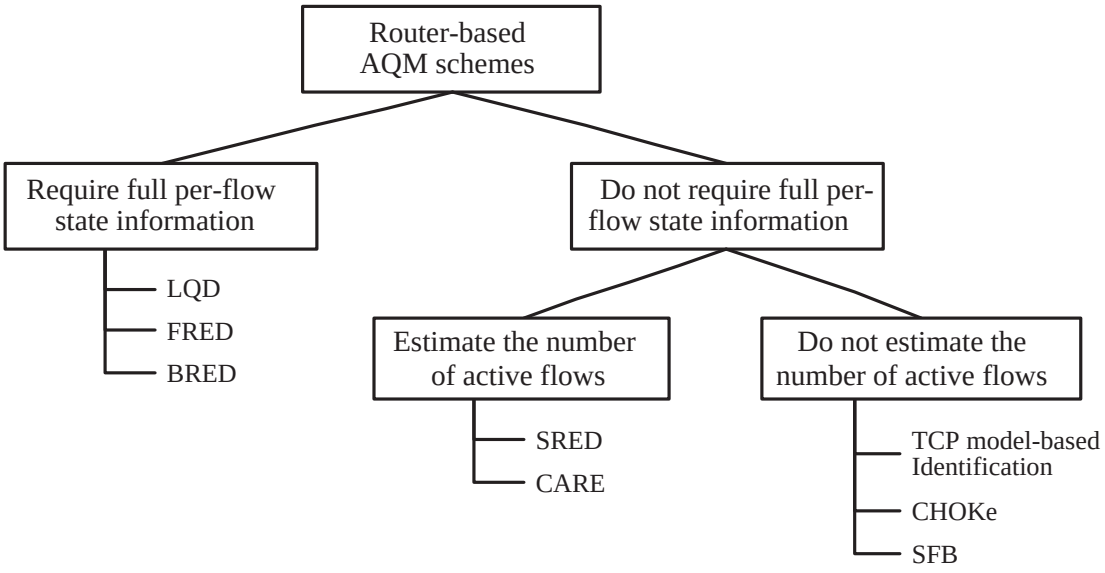


Figure 9: Classification of router-based solutions.

2.2.3.1 TCP Model-based Identification Scheme Floyd and Fall's 'Promoting the Use of End-to-End Congestion Control in the Internet' [25] is well-known for its extensive demonstration of the danger of unfairness and congestion collapse. The authors propose a router-based solution to identify unresponsive flows and punish them by limiting their rates or putting them in a low priority queue. Unresponsive flows are identified by comparing the flow's arrival rate with the throughput (T) obtained from Equation 2.10, which represents an approximation of the throughput that a TCP connection would have received under the same circumstances as the considered flow.

$$T \leq \frac{1.5\sqrt{2/3} * B}{RTT * \sqrt{p}} \tag{2.10}$$

This equation is derived from a simple TCP model that captures the behavior of the TCP congestion window in steady state, as shown in Figure 6 [49] and previously discussed in Section 2.1.1.3. The variable W in the picture represents the size of the congestion window when a packet is dropped with probability p. Notice that, in order to determine the TCP throughput from this equation, the flow's round-trip time (RTT), packet size (B), and dropping probability (p) should be known. If the flow's arrival rate, as estimated from RED's packet drop history, is greater than that obtained from the equation, the flow is identified as being unresponsive, and it should be

punished by putting it into another class of queue. In this case, the authors suggest Class Based Queue (CBQ) as a method of partitioning the buffer space into a queue for responsive traffic and a queue for unresponsive traffic. The flow's rate restriction is then removed once the flow's arrival rate decreases to less than $1.22B/(RTT * \sqrt{p})$ for a new packet drop probability p.

This mechanism is quite simple and can successfully identify unresponsive flows if the router has accurate values for all parameters. Since monitoring all flows' arrival rates may incur an excessive CPU processing overhead, the estimation of an arrival rate using RED packet drop history is suggested [24]. This estimation results in a rough approximation that may be somewhat higher or lower than the actual flow's arrival rate. In addition, a router has no easy way to determine the flow's round trip time; hence, it is unlikely to determine expected throughput precisely. In many circumstances, this leads to misidentification of unresponsive flows. Overall, the TCP model itself is overly simplified and unrealistic for actual implementation.

2.2.3.2 Longest Queue Drop (LQD) Suter et al. propose Longest Queue Drop (LQD), a buffer management scheme to solve the unfairness problem [68]. The basic concept of this scheme is that the flow with the largest number of packets that is waiting in the queue should be the first flow to be penalized or dropped. In LQD, buffer space is partitioned so that each flow has the same normalized buffer in an effort to achieve a fair bandwidth share. If the size of the entire buffer is noted as B, each connection i has a nominal buffer allocation b_i, which can be thought of as the connection i's guaranteed buffer size. Initially, b_i is set to B/n for all i, in which n is the number of backlogged connections.

When a connection i needs more than b_i buffers, two scenarios are possible. First, if the global buffer occupancy is less than B, the connection i is allocated part of the available buffer space, as long as the new global buffer occupancy does not exceed B. Second, if the global buffer occupancy at that time is equal to B and the connection i has a current occupancy q_i that is less than b_i, LQD makes room for the incoming packet by dropping the front packet from a connection with a current occupancy exceeding its allocation. The packets are dropped from the front instead of the back because this triggers TCP's retransmit/recovery mechanisms faster, helping TCP to increase its throughput [43]. Three methods are proposed for selecting which connection packets should be dropped from:

1. **Longest Queue Drop (LQD)**: This option selects the connection that uses the higher share of the bandwidth. This can be done by choosing connection j with the largest value of $(q_j - b_j)$. This method offers the router some level of isolation and protection, as if there is only one misbehaving flow then it should experience a higher loss rate.
2. **Dynamic Soft Partitioning with Random Drop (RND)**: This option selects a connection at random from amongst the connections for whom $q_j > b_j$. The goal of this scheme is to reduce the amount of bursty loss, as some TCP versions (e.g., TCP Reno), are known to behave badly in these circumstances.
3. **Approximated Longest Queue Drop (ALQD)**: One drawback of the LQD method is that it requires a search operation be run on the backlogged queues in order for the connection with the longest queue to be identified. ALQD improves LQD by maintaining information about the length and identity of the longest queue.

Simulation has shown that LQD performs quite well in terms of flow isolation even when TCP flows have different round trip delays. Its drawback is that a router needs to keep state information for every backlogged flow, otherwise a searching procedure must be performed on the entire buffer space very time a push-out decision is necessary.

Although the ALQD method was designed to reduce this cost, it may lead to bursty loss since it tends to drop packets only from the flow with the longest queue. In addition, LQD drops packets only when the global buffer is full. This results in a lack of early congestion notification, a prominent feature of active queue management schemes such as RED and its derivations.

2.2.3.3 Fair Random Early Detection (FRED) Even though RED active queue management has the ability to drop packets from connections in proportion to their bandwidth, the scheme does not guarantee fair bandwidth sharing among multiple connections, nor does it have a mechanism for handling unresponsive flow competing with adaptive flows such as TCPs. Fair Random Early Detection (FRED) [45], a modification of RED, also aims to solve the unfairness issue. Essentially, FRED operates similarly to RED but with additional features, explained below.

In FRED, two global parameters, min_q and max_q, maintain the minimum and maximum numbers of packets that may comprise each flow in a queue, respectively. For each active flow, two variables must be maintained on a per-flow basis: the number of packets in the buffer ($qlen$) and the

number of times the flow has failed to respond to congestion notification ($strike$). Additionally, FRED has an extra global variable, average per-flow queue size ($avgcq$).

FRED guarantees a minimum buffer space ($minq$) to protect flows from having low-speed connection. An arriving packet is buffered if the connection has fewer packets ($qlen$) than min_q and if the average buffer size is less than max_{th}. In order to manage a non-adaptive flow, FRED enforces a per-flow queueing limit. The flow buffer occupancy is maintained at a maximum leve (max_q) and the number of times each flow tries to go beyond max_q is noted as a $strike$ variable. The buffered packets from flows with high $strike$ values are not allowed to be more than average per-flow queue size ($avgcq$), and this prevents unresponsive flows from consistently dominating the entire buffer space.

Overall, FRED could achieve fairness in many situations as a result of its minimal differences from RED. However, a main drawback of the scheme is the need to track per-flow information for every active flow passing through the router, which is both costly and undesirable.

2.2.3.4 Balanced-RED (BRED) The fundamental idea of Balanced-RED or BRED [4] is to provide fairness through a RED-like mechanism acting upon each active flow that passes through the router. BRED maintains $qlen_i$ as the number of packets of each active flow i within the global buffer of size B, and it maintains $Nactive$ as the number of total active flows. Three thresholds are determined for the size of each $qlen_i$ in order to perform their dropping policy as follows:

- l_1: The minimum number of packets that may comprise a flow in the buffer before its packets start being dropped with a probability of p_1.
- l_2: The number of packets that may comprise a flow in the buffer before its packets start being dropped with a probability of p_2, where $p_2 > p_1$.
- W_m: The maximum number of packets that may comprise the flow in in the buffer.

If the flow state from each arriving packet is not present, it is initialized with $qlen_i$ = 0, and the number of active flows ($Nactive$) is incremented by one. If the information of this flow is already in memory, then preferential dropping is applied to the arriving packet according to the number of the flow's packets already in the queue ($qlen_i$), as shown above. For any departing packets from

flow i, the $qlen_i$ is decremented by one packet. If $qlen_i$ reaches zero, the number of active flows is decreased by one.

BRED has a well-demonstrated ability to isolate unresponsive flows from responsive flows. However, unresponsive flows are still able to claim much higher bandwidth than responsive flows. This effect is significant when the buffer size is insufficient for making the effective buffer size of each flow less than its bandwidth-delay product. There are many parameters that may effect the performance of BRED as they do RED. However, Anjum and Tassiulas did not explore whether or not those parameters would have a significant impact on the different types of scenarios. Lastly, BRED also requires per-flow state information to make decisions about which packet to be dropped.

2.2.3.5 Stabilized RED (SRED) Stabilized RED (SRED) [48] aims to stabilize the occupancy of an FCFS buffer, independently of the number of active flows. Therefore, the dropping probability depends on both the buffer occupancy and the estimated number of active flows.

The TCP bandwidth equation (Equation 2.10) shows that

$$cwnd \sim p^{-\frac{1}{2}} \tag{2.11}$$

where $cwnd$ is the flow's congestion window and p is the packet drop probability. With N flows, the sum of the N congestion windows is in the order of $N \times p^{-\frac{1}{2}}$ (MSSs, assuming all flows have the same Maximum Segment Size). Ott and colleagues argue that the target buffer occupation, Q_0, must be of the same magnitude as $N \times p^{-\frac{1}{2}}$, and they assume $Q_0 = N \times p^{-\frac{1}{2}}$ MSSs. Therefore, p must be in the order of N^2. This becomes part of SRED's dropping policy, as follows:

$$p_{zap} = p_{sred(q)} \times \min(1, \frac{1}{(256 \times (P(t))^2}) \tag{2.12}$$

with

$$p_{sred(q)} = \begin{cases} p_{max} & \text{if } \frac{1}{3}B \leq q < B, \\ \frac{1}{4} \times p_{max} & \text{if } \frac{1}{6}B \leq q < \frac{1}{3}B, \\ 0 & \text{if } 0 \leq q < \frac{1}{6}B. \end{cases}$$

where $1/P(t)$ is an estimate of the number of active flows in the time shortly before the arrival of packet t, B is the buffer size, and q is the instantaneous buffer occupancy. According to the

SRED dropping policy, as long as the number of active flows N does not exceed 256 (an arbitrary number), the dropping probability is also dependent upon the number of active flows to maintain the target buffer occupancy. That is

$$p_{zap} = \frac{p_{sred}}{256^2} \times \frac{1}{(P(t))^2} \sim \frac{p_{sred}}{65,536} \times (\text{number of flows})^2.$$

Once N exceeds 256, the dropping probability is equal to p_{sred} only in order to prevent TCP sources from spending a great deal of time in a time-out status due to excessive dropping probability.

Rather than maintaining per-flow state data, in order to estimate the number of active flows, SRED maintains a *Zombie List* which includes the M recently seen flows. Each flow in the list contains $Count$, a variable that represents how often its packets arrive. If the Zombie List is full, it's a packet's flow identifier is added to the list as it arrives and the $Count$ is set to zero. Then, the packet is compared with a randomly chosen item in the Zombie List:

- If they match, the $Count$ of that flow in the list is increased by one. This event is called a *Hit*.
- If they don't match, the flow identifier that was randomly picked from the list is replaced with that of the arriving packet, and the $Count$ is reset to zero, with a probability of p. This event is called a *No Hit*.

When packet t arrives, let

$$Hit(t) = \begin{cases} 0 & \text{if } nohit, \\ 1 & \text{if } hit \end{cases}$$

and $P(t)$ be an estimate for the hit frequency around the time of arrival of the t-th packet at the buffer, according to

$$P(t) = (1 - \alpha)P(t - 1) + \alpha Hit(t) \tag{2.13}$$

with $0 < \alpha < 1$. Finally, according to Ott et al., the number of active flows N is estimated by $1/P(t)$.

This estimation is based on the assumption that an arriving packet belongs to flow i with a probability of π_i and that a zombie represents flow i with a probability of π_i. So, for each arriving packet, the probability of causing a hit $P(t)$ is $\sum_i \pi_i^2$. For N flows with identical traffic intensity

$\pi_i = \frac{1}{N}$, for $1 \leq i \leq N$, this means that $P(t) = \sum_i \pi_i^2 = \frac{1}{N}$. In general, this estimation is accurate if all flows have the same traffic intensity as $\frac{1}{N} \leq \sum_{i=1}^{N} \pi_i^2 \leq 1$.

One of the drawbacks of SRED is that it assumes that all N flows have the same traffic intensity. In reality, this is not the case. In addition, although the SRED scheme can use its hit mechanism to identify unresponsive flows, it does not provide an effective penalty.

2.2.3.6 CHOKe A design goal of CHOKe (*CHOose and Keep for responsive flows, CHOose and Kill for unresponsive flows*) [52] is to offer a simple mechanism for controlling unresponsive flows. To achieve this goal, a small modification is made to the plain FCFS queue with RED active queue management. The CHOKe algorithm is illustrated in Figure 10 in which extra functions of CHOKe are shown in gray while RED functions are shown in white.

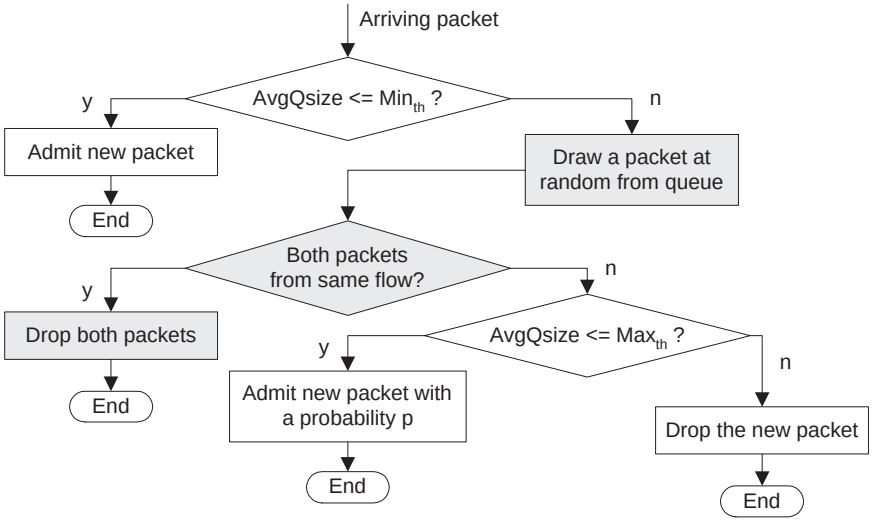


Figure 10: The CHOKe algorithm [52].

When a packet arrives, if the average queue size is greater than min_{th}, CHOKe draws a packet randomly from the buffer (*drop candidate*) and compares it to the arriving packet. If they are from the same flow, then both are dropped; otherwise, the arriving packet is accepted into the queue with a drop probability that is computed by RED. The basic idea behind CHOKe is that a FIFO queue is more likely to have packets that belong to unresponsive flows than are other non-FIFO queues, and they are more likely to be chosen for comparison. Therefore, packets from unresponsive flows

are likely to be dropped more often.

This scheme works well if there is only one unresponsive flow in the network. Therefore, Pan, et al. proposed a modification that would deal with multiple unresponsive flows. In their modification, $m > 1$ packets are randomly chosen from the queue per each packet arrival. To minimize the complexity of multiple random samplings per each packet arrival, the buffer is partitioned between min_{th} and max_{th} into k regions, and the number of drop candidates (m) to be randomly chosen from the buffer is set to $2 \cdot i(i = 1...k)$ according to the region of the average buffer occupancy

CHOKe is very simple to implement, maintains minimum state information and controls unresponsive flows. However, it can control unresponsive flows only if there are enough packets from the flows in the buffer at the time of congestion. Pan,et al. demonstrate in their simulation study [52] that with CHOKe, the high-bandwidth UDP flows still consume much greater bandwidth than do TCP flows. In addition, CHOKe operates on a per-packet basis; therefore, in some scenarios, a flow with twice the packet size can consume almost twice the bandwidth of other flows [54].

2.2.3.7 Stochastic Fair Blue (SFB) The main novelty of the Stochastic Fair Blue (SFB) [19] scheme is that it attempts to manage unresponsive flows without relying on queue occupancy statistics. SFB maintains a set of L hash tables that detail different hash functions, with N items in each table. Each item, referred to as a bin, tracks both the number of times flows are hashed into that item and a dropping probability p_m. Each arriving packet is hashed, according to a string such as the flow ID, into one of the N bins in each of L hash tables, and the number of packets stored in that bin is increased by one. If the number of packets in a particular bin is higher than a certain number, the dropping probability (p_m) for the bin increases by a certain amount. On the other hand, p_m is decreased when the number of packets in a bin drops to zero. In the SFB scheme, for each flow, there are L values of p_m from the L tables to which it is hashed. The final dropping probability is, however, determined as the minimum value of these p_m.

According to this algorithm, an unresponsive flow consuming high bandwidth would ramp up p_m to 1 in all of the bins of L tables. In this way, the flow is then identified as a high-bandwidth unresponsive flow that should be penalized with rate limitations. Because The use of multiple hash tables makes it unlikely that two different flows would be hashed into the same items on every table. In other words, a perfect hashing for every active flow that passes through the router is not

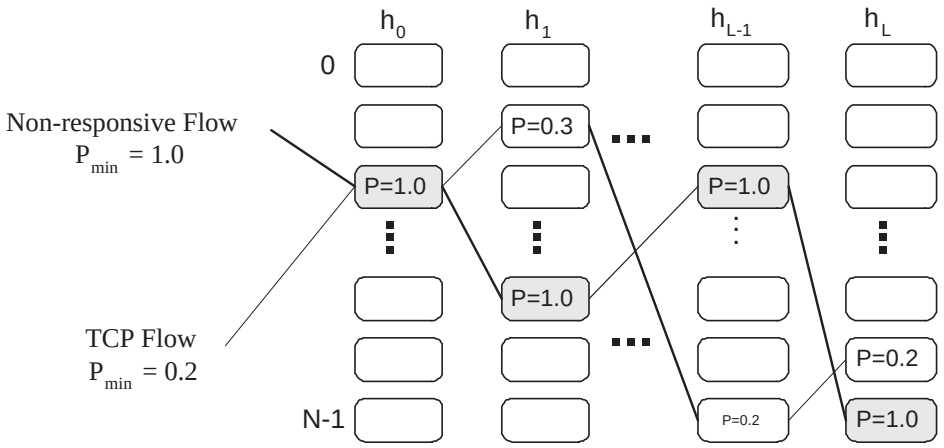


Figure 11: Example of SFB [19].

required. Consequently, the possibility of penalizing a responsive flow that hashed into the same item as any unresponsive flow is reduced.

Even with multiple hash tables, however, SFB can incorrectly identify a responsive flow as an unresponsive one, if there are many unresponsive flows at the router. In this instance, the flow would be continually penalized without being reclassified. In addition, an identified unresponsive flow that later become responsive would also be penalized without being reclassified. To correct these problems, Feng et al. propose that SFB have a moving hash in which the hash function is changed and reset at regular intervals (e.g., every two seconds). The authors also suggest the use of double sets of tables that work at different times, so that one set can warm up a mechanism before another set is reset. This prevents unresponsive flows from getting more bandwidth during the time after the reset.

Nonetheless, the size of the tables should be carefully pre-determined because the higher the number of unresponsive flows, the higher the probability of misidentification. Also, the higher the number of responsive flows - even short-lived ones - the higher the number of responsive flows being hashed into the same bins as unresponsive flows and, therefore, penalized. In addition, the identified unresponsive flows are punished by rate limits where the maximum rate allowed must be manually determined and static. Since the mechanism is independent of queue occupancy, some unresponsive flows could still be punished despite an ample amount of available buffer space.

2.2.3.8 CApture-REcapture (CARE) CARE [8] is based on the Capture-Recapture (CR) model that has been used widely by ecologists and biologists to estimate the number of animals in a population and by software developers to estimate the number of defects in software during the inspection process. The model is applied here to estimate the arrival rate of traffic flows and the number of active flows using the M_0 CR model and the M_h CR model, respectively.

The basic idea of the M_0 model is that the proportion of the number of packets from flow i (m_2) among t captured packets is the same as the proportion of the number of packets from flow i (n_2) in a buffer of size B. If CARE captures t incoming packets, the incoming rate of flow i can be estimated using the equation $B * m_2/t$. On the other hand, the M_h model is more suitable to a scenario in which the captured probabilities are different among flows; thus, it is used for estimating the number of active flows (N). For this task, CARE uses the *Jackknife estimator* to estimate N. Because the derivation of the Jackknife estimator is difficult, the estimation of the number of active flows is briefly summarized as follows:

1. Capture an incoming packet with probability p_{cap} and store it in the *capture list*, which can store t packets.
2. After capturing t packets, construct the capture frequency to see how many unique flows have been seen once (f_1), twice (f_2), and more, up to t times (f_t).
3. Based on these capture frequencies, use the Jackknife estimator to estimate the number of active flows (N_{JK}) using the following equation:

$$N_{JK} = a(t,K)_1 f_1 + a(t,K)_2 f_2 + ... + a(t,K)_t f_t \tag{2.14}$$

where $a(t,K)_i$ are the coefficients in terms of the number of capture occasions (t) and the order of estimation (K). An optimum value of K must be determined because as K increases, the bias of N_{JK} decreases while the variance of N_{JK} increases. To determine the optimal K, the coefficients for $K = 1$ to $K = 5$ must be derived. Then, combined with frequency data, $N_{J1}, N_{J2}, ..., N_{J5}$ are calculated. Next, an interpolated estimator between $m-1$ and m, where m is the first order of estimation for the significance level $P_m > 0.05$ is computed. If $m = 1$, then N_{J1} can be taken as the estimator; otherwise, the interpolation between $N_{J(m-1)}$ and N_{Jm} is used to estimate the number of active flows.

Once the number of active flows has been estimated, CARE can drop packets from the flows that consume more than a fair share of bandwidth with a dropping probability of $1.0 - (t/(N_{JK} \cdot m_2))$.

CARE has several drawbacks. First, in order for the estimate to be more accurate, the model must make a large number of captures (t) compared to the number of flows. This has a direct impact on the scalability and simplicity of the scheme's operation. The need for many captures means running the capture process t times, building a capture frequency table of up to t values, and calculating the coefficients to apply Equation 2.14 and find the estimated number of flows. This might require a great deal of CPU processing power, making CARE unsuitable for high-speed routers. Second, there is no mechanism described by Chan and Hamdi [8] to reduce the size of the capture list, which can grow to the number of flows (not a scalable solution). Finally, that the number of flows is considerably larger than the size of the capture list is assumed and no evaluation is provided to assess CARE's performance. In other words, CARE works well when the memory space available is adequate, although it does not require full per-flow state information. Otherwise, the estimate of the number of active flows could be inaccurate. Further discussion about estimating the number of active flows is provided in Chapter 4.

2.2.4 Summary

The characteristics of the existing router-based mechanisms presented in this chapter are summarized in Table 1 based on their design criteria.

Given the widespread nature of the current Internet architecture, deploying a router-based mechanism that requires per-flow state information is impractical, since it would require excessive memory resources and CPU processing overhead. In addition, it has been demonstrated that a large amount of traffic is actually carried by a small number of connections, and the many remaining connections are short-lived or low-bandwidth flows [29, 38, 12]. Generally, short-lived flows may be idle most of the time, and some flows may transmit only a few bytes of data. Holding per-flow information to provide a fair share of bandwidth for every active flow (e.g., such as would be done by LQD, FRED, and BRED), may result in wasted memory resources and unnecessary CPU processes.

Several schemes recently proposed focus instead on regulating only high-bandwidth flows so as not to harm small or short-lived flows which are the majority of the Internet traffic. TCP model-based identification schemes and CHOKe are examples of such schemes. In comparison, SRED, SFB, and CARE attempt to solve the unfairness problem by using a data structure that holds partial information about the flows and is used to identify unresponsive flows at the time of congestion. However, as discussed in this section, these schemes have certain limitations.

To this end, none of the schemes presented offer a well-round solution, and a thorough research investigation still is needed in this area.

	Holds per-flow information	Estimates number of active flows	Explicitly identifies unresponsive flows	Penalizes unresponsive flows	Early congestion notification	Factors of mis-identification
TCP Model-based Identification[25]	○	○	●	●	N/A	Unknown parameters especially flows' RTTs. Very simplified TCP model
LQD [68]	●	●	○	●	○	N/A
FRED [45]	●	○	○	●	●	N/A
BRED [4]	●	○	○	●	●	N/A
SRED [48]	▽	●	●	○	●	Its probabilistic mechanism
CHOKe [52]	○	○	○	●	●	N/A
SFB [19]	▽	○	●	●	●	Size of bins and number of flows
CARE [8]	▽	●	●	●	○	Large amount of memory for CR model

● = Yes, ○ = No, N/A = Not Applicable

▽ Maintain a data structure to partially store flows' information.

Table 1: Comparison of router-based fairness active queue management mechanisms

3.0 BLACK: A NEW FAIRNESS BUFFER MANAGEMENT SCHEME

As discussed in the previous chapter, when examined in light of the main design issues and goals of router-based solutions provided in section 2.2.2, the existing buffer management or AQM schemes either fail to solve the unfairness problem, or they suffer from serious limitations, especially in terms of scalability and simplicity. Several per-flow packet scheduling mechanisms originally proposed to ensure fairness of network service have the same problem. These have not been employed widely in high speed networks, but have been limited to premium services, as a result of both the need to maintain per-flow state information and the computational complexity.

Packet dropping through AQM mechanisms such as those discussed in the previous chapter is a more practical approach, as they are rather less complex and thus becomes a main research in this field. Early fair AQM techniques such as LQD, FRED, and BRED alleviated the unfairness problem effectively, but not efficiently; the techniques still required per-flow accounting of buffer usage for every single active flow.

Rather than monitoring every active flow, several newer packet dropping schemes focus only on high-bandwidth or misbehaving flows, tracking and regulating them to ensure that they do not steal bandwidth or shut down small and short-lived flows. TCP Model-based Identification, CHOKe and CARE are the examples of such schemes. Other packet dropping schemes, such as SRED, SFB and CARE, hold partial information of traffic flows and rely on that information to penalize unresponsive flows when congestion occurs. However, all of these schemes have several limitations, such as penalizing misidentified flows, lacking protection for small flows, or needing a high amount of memory and CPU processing power. Moreover, none of these schemes has achieved fairness in handling TCP flows with varied implementations or round trip delay. These limitations are discussed in Section 2.2.

The purpose of this chapter is to introduce BLACK[1], a novel AQM mechanism that ensures throughput fairness while keeping memory resource low by holding only partial per-flow state information. In addition, BLACK eliminates all of the problems faced by the existing light-weight fair AQM schemes. This discussion presents the basic idea of the new scheme, then explains the structure and each component of the scheme in greater detail. A thorough evaluation comparing the new scheme with existing schemes is also provided. The limitations of BLACK are summarized at the end of this chapter.

3.1 PRELIMINARIES

The goal of the new scheme is to avoid holding per-flow state information for every passing flow. However, tracking and penalizing flows that consume more resources than their fair share may necessitate that partial flow state information be maintained. In order to manage memory resources more efficiently, the proposed scheme is designed to keep only partial information about each flow's share of resource, such as its buffer occupancy information, and it is designed to keep this information only for those flows that consume more than their fair share. When congestion occurs, packets from these flows are subjected to preferential dropping, according to the proportion of their consumed resource. Furthermore, according to this scheme, packet dropping occurs only when necessary; the probability that dropping will occurs increases with the buffer size since the latter is an indicator of the amount of network congestion that exists. As a result, not only do large flows gain enough bandwidth when the network's traffic load is light, but most of the small flows are also protected. It is expected that managing the queue according strategie of fairly sharing available buffer space reduces the bias against round trip time for TCP traffic. On the other hand, those traffic flows that consume less buffer space than their fair share and are usually carried by a responsive protocol like TCP are handled by a global active queue management, such as RED, that is able to control the queue size and, thus, the delay. This active queue management function also provides early notification of congestion to responsive TCP sources.

One concern about keeping limited per-flow state information relates to how the router determines the fair share that each flow should be granted. This research addresses the question as it

[1]Most of the materials in this chapter were published in [9]

seeks a mechanism to approximate each flow's fair share at the time of congestion. One way this is done is by estimating the number of active flows. With this information, a theoretical dropping policy functioning at the router is developed to achieve fairness among active flows.

3.2 BLACK

The idea of a new fairness scheme that reduces use of memory space by holding data only for large flows results from several studies which show that most of the bytes of Internet traffic are actually carried by a small number of connections and that the remaining large number of connections are low-bandwidth flows. This phenomenon is referred to as *mice and elephants*, with the mice being the small flows and the elephants being the large flows that contribute the most bytes to the network. Using data collected from OC-12 links in the core of the Sprints Tier-1 IP backbone network, Papagiannaki et al. find that the proportion of large flows to total flows is approximately 60% [53]. These findings are shown in Figure 12. Kim's analysis of 20 traces of length 60-120 seconds shows that 50% of all flows are single packet flows and 80% of the flows contain less than 20 packets [38], and this conclusion is similar to that of other studies [29, 30]. Figure 13 shows the cumulative sum of per flow throughput where the top three flows total 50% of all bytes counted and the top 12 flows produce 67% of all bytes counted. Kim also suggests that the amount of memory space allocated to control the congestion could be greatly reduced by accounting for this phenomenon [38].

From these results, only few large flows contribute to most portions of bytes in the network and they are usually a cause of congestion. In addition, several streaming media applications usually consume large amount of bandwidth while providing no end-to-end congestion control mechanism, or very limited, and become the main factor of unfairness problem because they do not reduce the transmission rate on the advent of congestion. Therefore, dropping the packets from these types of flows at the router could reduce the level of congestion more effectively than dropping the small flows. And because those large flows are small in number, the number of per-flow state information could be greatly reduced by keeping track of and control only this type of flows that feed a larger portion of traffic to the queue. When the congestion occurs, these flows should be more responsible and thus their packets should be dropped first. On the other hand, small flows usually come from

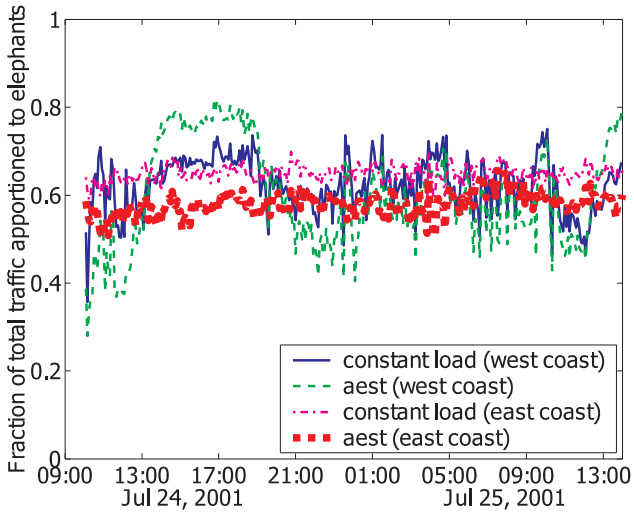


Figure 12: Proportion of large flows to the total traffic [53].

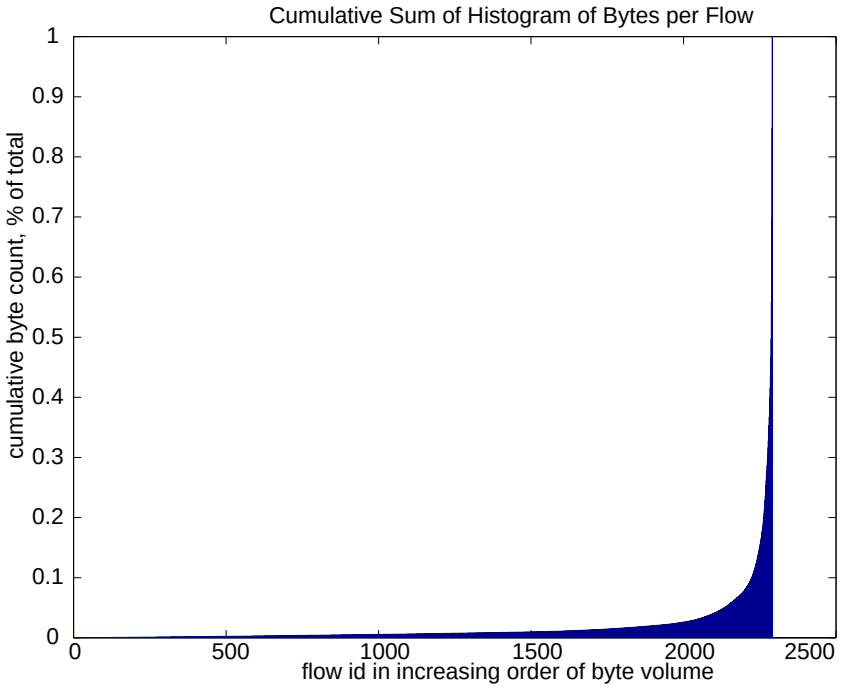


Figure 13: Cumulative sum of per-flow throughput from traffic traces [38].

the delay sensitive applications like HTTP or Telnet, where packet dropping is not preferable. These types of flows are also typically not the main factor of the congestion and should not be the first priority to be dropped. The idea of blacklisting and control those unresponsive or high-bandwidth flows becomes a name of the new fairness scheme – *BLACK (Blacklisting unresponsive flows)*.

These results show that only a few large flows contribute most of the bytes in the network and that they are usually the cause of congestion. In addition, several streaming media applications usually consume a large amount of bandwidth while providing no - or limited - end-to-end congestion control mechanism, and these applications become the main cause of the unfairness problem because they do not reduce their transmission rates on the onset of congestion. Therefore, the router's dropping of packets from these types of flows could reduce the level of congestion more effectively than dropping packets from the small flows. In addition, because there are few large flows, the amount of per-flow state information could be reduced greatly by tracking and controlling data for only the flows that feed a large proportion of traffic into the queue. When congestion occurs, these flows are likely more responsible; thus, their packets should be dropped first. Also, small flows usually come from delay-sensitive applications like HTTP or Telnet, in which packet dropping is not preferable. Since these types of flows are typically not the main cause of congestion, they should not be the first priority to be dropped. The strategy of blacklisting and controlling unresponsive and high-bandwidth flows is the source of the name of the new fairness scheme: *BLACK (Blacklisting unresponsive flows)*.

The basic approach of BLACK is to provide a fair bandwidth allocation mechanism that aims to achieve fairness among responsive and unresponsive flows as well as among responsive TCP flows with different round trip delays. It does this by tracking and controlling high bandwidth flows. The information BLACK keeps includes the amount of buffer consumed by these flows, or a buffer occupancy fraction, as an indicator of a flow's bandwidth share at a network link. Suter et al. and Laksham and Madhow have shown that, at a FIFO router, the bandwidth allotted to different active connections is roughly proportional to their share of buffer space. Hence, if the router allocates the buffer evenly among all active flows, fair bandwidth allocation can be achieved at a high degree [68, 42].

In one of our the pilot studies, five constant-bit-rate (CBR) traffic flows with arrival rates of

1 Mbps, 2 Mbps, 3 Mbps, 4 Mbps, and 5 Mbps were fed into a single link with a bandwidth of 5 Mbps. The queueing discipline applied was a modified version of tail dropping. In this case, the router kept track of all five traffic flows and controlled them so as not to allow any one flow to consume more buffer space than its fair share according to the buffer occupancy fraction. Even though the arrival rates of these five flows were different and the sum of the arrival rates was much greater than the bottleneck link bandwidth, the throughput achieved by these flows was well controlled at the fair rate at 1 Mbps, as shown in Figure 14. An intuitive reason for this result is that, under a FIFO system in which no packets in the queue are dropped, the amount of the packets queued in the buffer is the amount of the packets that would be served eventually, if the queue were never empty. Therefore, the fair amount of buffer occupancy indicates the fair amount of achievable throughput.

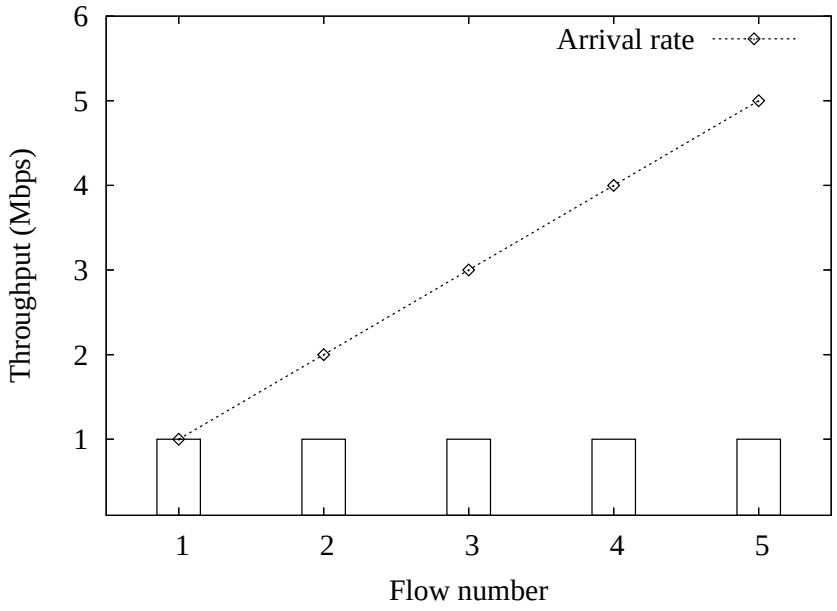


Figure 14: Simulation result showing fair throughput achieved by controlling the fair buffer occupancy fraction.

However, instead of directly counting the exact amount of packets that enter and leave the queue for all the flows, a sampling technique should be used to approximate a buffer occupancy fraction for the flows that are recorded. This information of only high bandwidth flows are managed to be stored, using some memory management, in a limited cache memory called *HBF cache*

memory. The packets from the flows that occupy more than their fair share of buffer space should be dropped according to the proportion of buffer space they consume relative to their fair share. Furthermore, packet dropping should be performed only when necessary, such as with the increase of dropping probability according to the buffer size. In this way, large flows gain additional bandwidth when the network has a light load and most of the small flows are also protected. When the queue is managed according to fair share of buffer space, it is expected that the bias against round-trip time experienced by TCP traffic also is reduced, too.

In addition, to keep both the average queue size and the queueing delay low, while providing early congestion notification to responsive traffic sources, a global active queue management scheme should work in conjunction with BLACK. RED [28] is a good choice for this task since it is one of the most well-known schemes and has been implemented by a large number of router manufacturers already. In this scenario, all packets pass through the preferential packet dropping policy of BLACK before going to the RED function for the benefits discussed above.

The design of BLACK then requires four components as shown in the Figure 15. These will be discussed in detail in the following sections.

1. Buffer occupancy fraction approximation.
2. Packet dropping function.
3. HBF cache memory management.
4. Estimation of the number of active flows.

3.2.1 Buffer occupancy fraction approximation

It is desirable to keep as few per-flow states as possible. Therefore, only an approximation of what fraction of buffer space each flow occupies or a list of possible high-bandwidth flows is needed. To meet this goal, a sampling technique has been developed.

When each packet arrives, if the queue size exceeds a certain threshold, the router randomly selects one packet from the queue. A flow ID [2] of the randomly chosen packet is recorded. When

[2] A flow ID could be a combination of source and destination address, source port and destination port, or only a FlowID field for an IPv6 packet.

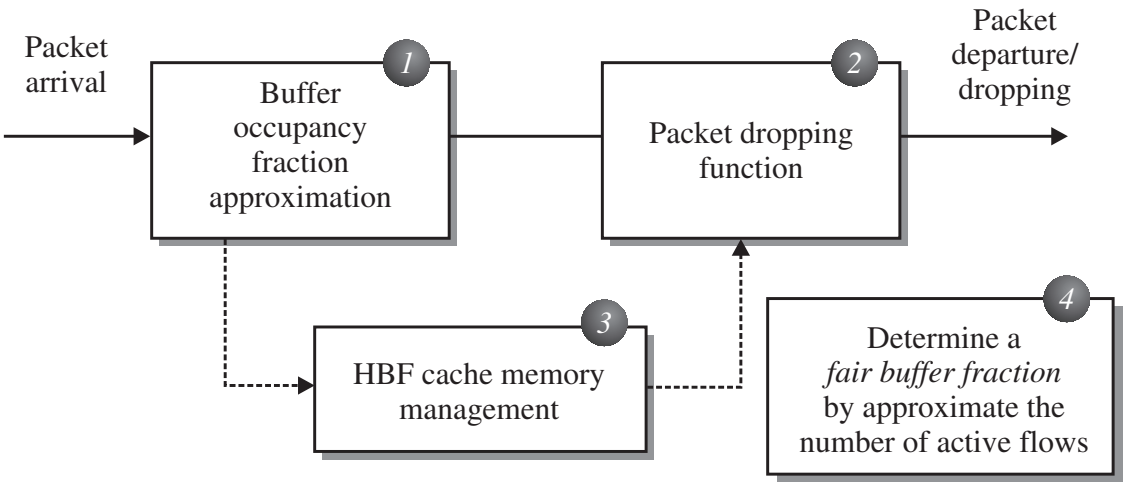


Figure 15: Components of BLACK.

this happens, a *Hit* ($\hat{h}$) for that flow is declared. Next time, if the randomly selected packet is from the same flow, the value of its $\hat{h}$ is increased by one.

After m packet samplings, the router checks a number of *Hits* or $\hat{h}$ for each recorded flow ID. For flow i, a result of the number of i divided by m is called a *HitFraction* (H_i), which is an approximation of the average flow's buffer occupancy fraction.

To reduce an approximation error of H_i, an exponential weighted moving average can be used to smooth out the value of H_i for flow i over multiple sampling periods as

$$H_i = (\alpha * H_i) + (1 - \alpha) * \hat{H}_i \tag{3.1}$$

where $\alpha < 1$ and $\hat{H}_i$ is an instantaneous value of *Hit Fraction* on the last sampling period. A flow that has a larger *HitFraction* than a fair buffer occupancy fraction (B_f) is considered to be a possible high-bandwidth flow.

The advantage of the *HitFraction* method is that it is simple enough to identify the buffer occupancy fraction of multiple high-bandwidth flows through m random packet selections. In addition, because packets from high-bandwidth flows in the queue are more likely to be picked up, partial information can be collected and there is no need to maintain per-flow states for all active flows. This protects small or short-lived flows from being penalized as well.

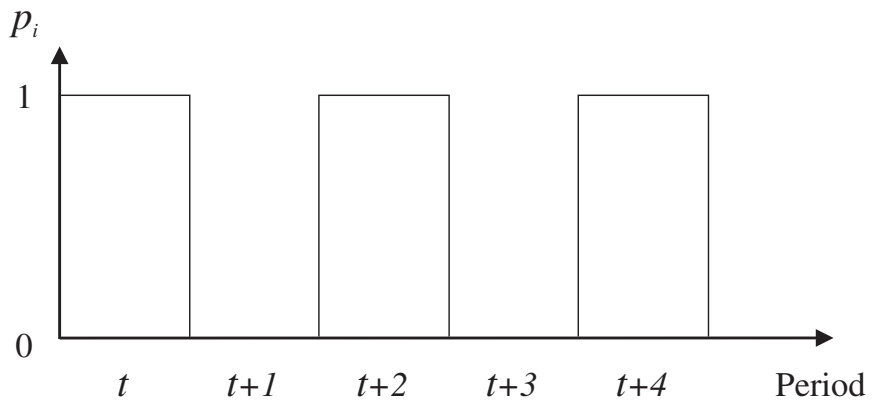


Figure 16: Example of oscillation in dropping probability according to Equation 3.2.

3.2.2 Packet dropping function to control high-bandwidth flows

Once buffer occupancy fraction information is obtained, the flows with *HitFractions* greater than their fair buffer fraction (B_f) are subject to dropping. The dropping probability is set to the percentage of extra buffer space that each flow consumes in excess of B_f, which is

$$p_i = \begin{cases} \min\left(1, \frac{H_i - B_f}{B_f}\right) & \text{if } H_i > B_f \\ 0 & \text{if } H_i \leq B_f \end{cases} \tag{3.2}$$

However, if the dropping probability is set according to the Equation 3.2 for the entire next sampling period, it may result in an oscillation of the flow's achievable throughput.

For example, assume that unresponsive flow i, with an arrival rate that is much greater than its fair share, is fed into the queue using a dropping policy according to the Equation 3.2. At the end of the current sampling period t, the queue would have information about buffer occupancy of flow i, or H_i, which is much greater than B_f, and the dropping probability calculated using Equation 3.2 is set to 1. In the next period, $t + 1$, no packets from flow i are allowed to get into the queue due to their high dropping probability. This blocking is in effect for the entire period $t + 1$. As a result, the probability p_i at the end of period $t + 1$ would be zero, leading to no packet dropping in period $t + 2$, and this fluctuation would repeat endlessly as long as flow i maintains its traffic stream. This problem is illustrated in Figure 16.

3.2.3 Fine tuning buffer occupancy approximation for adaptive packet dropping

The oscillation of achievable throughput may occur because the dropping probability is set purely according to buffer occupancy information in the past and does not reflect an instantaneous fraction from the current period.

To minimize the oscillation, the dropping probability should be adjusted periodically so that rather than being constant after each sampling period, it is dynamically adjusted according to the current buffer occupancy fraction. Instead of using the *HitFraction* parameter as calculated based on past information alone, the number of *Hit* ($\hat{h}_\tau$) collected in the current period also can be used to calculate the dropping probability through the average *HitFraction* ($\bar{H}_i$) that accounts for both past and current information, as follows:

$$\bar{H}_i = \frac{\hat{h}_\tau + (H_i * m)}{m_\tau + m}, \tag{3.3}$$

where m_τ is the number of sampling packets so far in the current period, and $m_\tau \leq m$. In this manner, if $\bar{H}_i$ is higher than the fair buffer fraction, then this packet will be dropped with a probability of

$$\hat{p}_i = \begin{cases} \min\left(1, \frac{\bar{H}_i - B_f}{B_f}\right) & \text{if } \bar{H}_i > B_f \\ 0 & \text{if } \bar{H}_i \leq B_f \end{cases} \tag{3.4}$$

This strategy is used in place of Equation 3.2. Note that the term ($H_i * m$) in Equation 3.3 is the average number of *Hit* ($\hat{h}$) from the past sampling periods. A mathematical derivation confirming the correctness of Equation 3.3 is provided in an Appendix A.

3.2.4 HBF cache memory management

HBF cache memory is designed to hold only information about some connections that are likely to consume more bandwidth than their fair share. Thus, BLACK needs some mechanism to keep only large flows in the limited cache memory size. In this way, it does not need to hold the per-flow state information of every single passing flow. The HBF cache memory management is adapted from the LRU mechanism [38, 39] as follows.

This type of memory management is based on a web-caching technique in which frequently accessed web objects are stored in cache memory so as to be more accessible to web clients in the future. In general, if the web clients request those objects that are in the cache memory, they have shorter delays because the objects are sent from the memory rather than from a web server further away. The key mechanism of web-caching is web cache replacement which determines which objects should stay in the memory for future access and which objects should be swapped out of this limited memory. Least Recently Used (LRU) mechanism is one of the simplest and widely used web cache replacement strategies. LRU is based on the assumption that the least recently used objects are not likely to be used again in the near future and, therefore, should be swapped out of the memory.

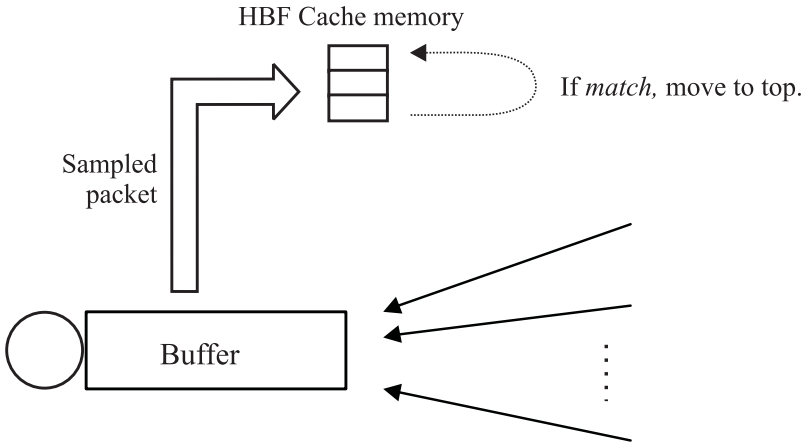


Figure 17: HBF cache memory management.

The modified LRU mechanism provided by Kim and, separately, Kim and Whang is adapted from the LRU web cache replacement mechanism in order to identify and track long-lived high bandwidth flows using a limited cache memory [38, 39]. Having analyzed the traffic trace, the authors found that most bytes were from only 1-2% of all flows. In addition, they determined that once a flow's packet arrives at the router, there is a very high probability that the next packets from the same flow will reach the router in the near future. These results show a good situation for web caching to be used as a mechanism for tracking long-lived high bandwidth traffic. With this approach, a flow ID, rather than a web object, may be kept in the memory. Once a flow's packet

arrives at the queue, the router looks through the cache, arranged as a linked list, to determine if the flow ID has been recorded. If a match is found, a counter tracking the number of packets from this flow is increased by one and the item is moved to the top of the cache. If no record of this flow exists in the memory, the item's counter which had been set to zero is replaced with a flow ID of the incoming packet's flow. However, if no match is found and no item has a counter set to zero, then the last item will be replaced, since it is the one that was least recently seen. If a counter exceeds a certain threshold, the flow associated with that counter is declared to be a long-term high-bandwidth flow.

HBF cache memory management in BLACK is adapted from the above operation to track the approximation of the buffer occupancy fraction. For each packet arrival, one packet is sampled from the queue and the router looks up in the cache to determine if the packet's flow ID has been recorded. If a match is found, its Hit value will be increased by one and the item will be moved to the top of the cache. If no record of the flow is found, then the last item will be replaced with the record of the incoming packet and this will be moved to the top of the cache. However, in order to prevent the record of a large flow being removed from memory, such a replacement will occur only if the Hit Fraction of the last item is less than B_f.

In addition, as described by Kim, to improve accuracy by filtering out small flows, or which previously refferred to as mice flow, a random decimator is added [38]. If the cache size is full and no match is found, the last item in the cache is replaced by the incoming packet with a probability of p_r. A small value, such as 0.05, is suggested for p_r; this implies that only about one of twenty packets would be able to make a replacement. Hence, it is more difficult for short flows to overwhelm the cache. As Kim explains, use of this random decimator increases the accuracy of identifying long-lived high bandwidth flows, or elephant flows, from 76.36% and 48.67% to 92.55% and 81.73%, respectively, in two experiments [38]. In BLACK, the last item is replaced with the probability p_r only if the last item has a *HitFraction* of less than Bf.

3.2.5 Estimation of the number of active flows

One of the parameters necessary for networks to provide fair service is the value of each flow's fair share of network resource. In BLACK, the buffer occupancy fraction is kept in the HBF

cache memory and used by the preferential packet dropping function. Therefore, the fair buffer occupancy fraction is estimated. One estimate of the fair fraction is based on the number of active flows ($Nact$). In other words, the fair fraction (B_f) is simply $1/Nact$.

Because per-flow state information should be minimized, an exact count of the number of active flows passing through the router is both undesirable and impossible, especially for high speed routers. BLACK estimates the number of active flows using a simple operation based on the information provided by the packet sampled during the first stage, as follows.

Upon a packet's arrival, BLACK compares its flow ID to that of a packet randomly sampled from the buffer. If the arriving packet and the sampled packet are from the same flow, then a *match* is declared.

Assuming N flows arrive at the router, labeled as flows numbered $1, 2, ..., N$. Let $P_{a,i}$ be the probability that an arriving packet belongs to flow i, and let it be equal to π_i. If the incoming flows have the same traffic intensity, then $\pi_i = \frac{1}{N}$ for all $1 \leq i \leq N$.

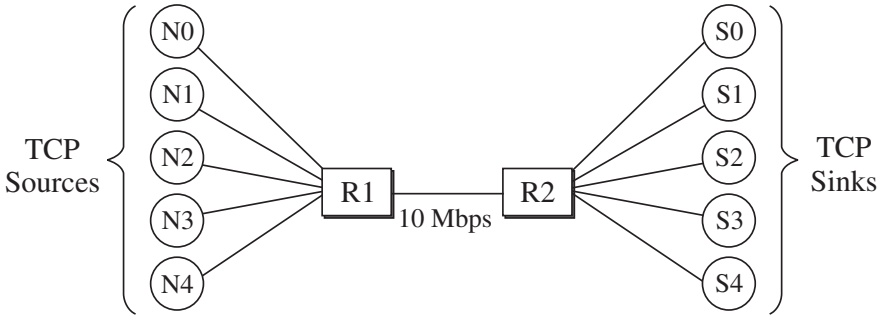


Figure 18: Simulation topology for pilot experiment on the estimation of the number of active flows.

Let $P_{s,i}$ be the probability that a randomly chosen packet from the buffer belongs to flow i. In a similar manner to $P_{a,i}$, $P_{s,i}$ is approximately $\pi_i = \frac{1}{N}$. Based on the definition of a *match* event, the probability that a *match* will occur for an arriving packet from flow i, $P_{match,i}$, is then equal to

$$\begin{aligned} P_{match,i} &= P_{a,i} \times P_{s,i} \\ &= \frac{1}{\pi_i} \times \frac{1}{\pi_i} \end{aligned}$$

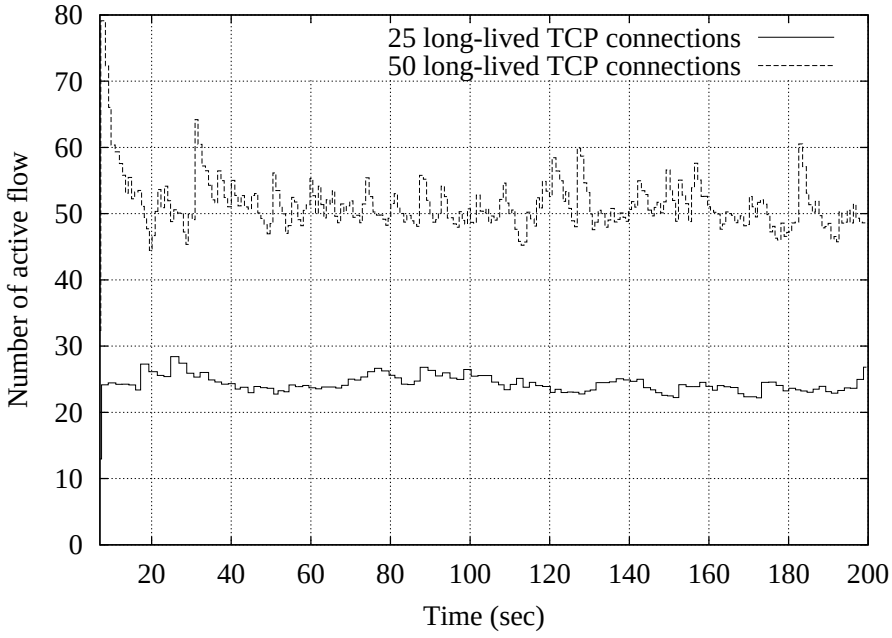


Figure 19: Results from a pilot study of BLACK's estimation of the number of active flows.

$$= \frac{1}{\pi_i^2} \\ = \frac{1}{N^2} \tag{3.5}$$

given that $P_{a,i}$ and $P_{s,i}$ are independent of each other. As a result, that probability that a *match* will occur for an arriving packet from any of the N flows, P_{match}, if the N flows have the same traffic intensity, is

$$P_{match} = \sum_{i=1}^{N} P_{match,i} = \sum_{i=1}^{N} \frac{1}{N} = \frac{1}{N}. \tag{3.6}$$

For m arrival packets, a probability P_{match} can be estimated by simply counting the number of *matches* events over m packet arrivals. Hence the estimated number of active flows is

$$Nact = E[N] = E\left[\frac{1}{P_{match}}\right] = \frac{m}{\text{(a number of } \textit{match} \text{ events)}} \tag{3.7}$$

during the m arriving packets interval.

In a pilot study, BLACK's estimation of the number of active flows was tested in a simulation based on the topology shown in Figure 18. Two sets of experiments were conducted with 25 TCP

flows and 50 TCP flows, respectively. The results are plotted in the Figure 19 and show a close approximation of the actual number of passing flows.

One of the problems encountered by the above simplified method is a decrease in accuracy when traffic intensity varies greatly. To relieve this effect, this estimation must be performed after the packet dropping function. Since the queue accepts packets from active connections in a rather fair manner through the dropping policy, it is expected that variance of the traffic intensity can be minimized.

3.3 COMPARATIVE EVALUATION OF BLACK

In this section, several scenarios are used to evaluate BLACK and some of the other important AQM schemes listed in Chapter 2.2. The mechanisms that require full per-flow state information are not included here because of the practical limitations posed by scalability problems, as is especially the case for high-speed routers handling several thousands of flows. Specifically, CHOKe, SFB, BLACK and CARE, are evaluated. SRED and the TCP Model-based Identification mechanism are not evaluated because, although both could identify misbehaving flows, SRED does not provide a mechanism to control them and the TCP Model-based Identification requires an additional mechanism (e.g., a separate CBQ queue) to handle them. For cpmparative purposes, RED is included, since it is the most widely used AQM scheme.

3.3.1 Simulation setup

NS-2 [1] is used as a simulation tool with the dumbbell topology shown in Figure 20 to assess the performance of RED, SFB, CHOKe, BLACK and CARE under four different scenarios. The first scenario is a comparison of the effectiveness of the AQM schemes in achieving fairness when TCP sources compete with one unresponsive flow. In the second scenario, the first scenario is repeated but includes multiple unresponsive flows. In the third set of simulations, only TCP sources are considered but with different round-trip times. The fourth scenario compares the effect of the AQM mechanisms on TCP-friendly protocols. Common simulation settings are as follows.

Constant-bit-rate UDP flows compete against a number of TCP flows over a 5-Mbps R1-R2

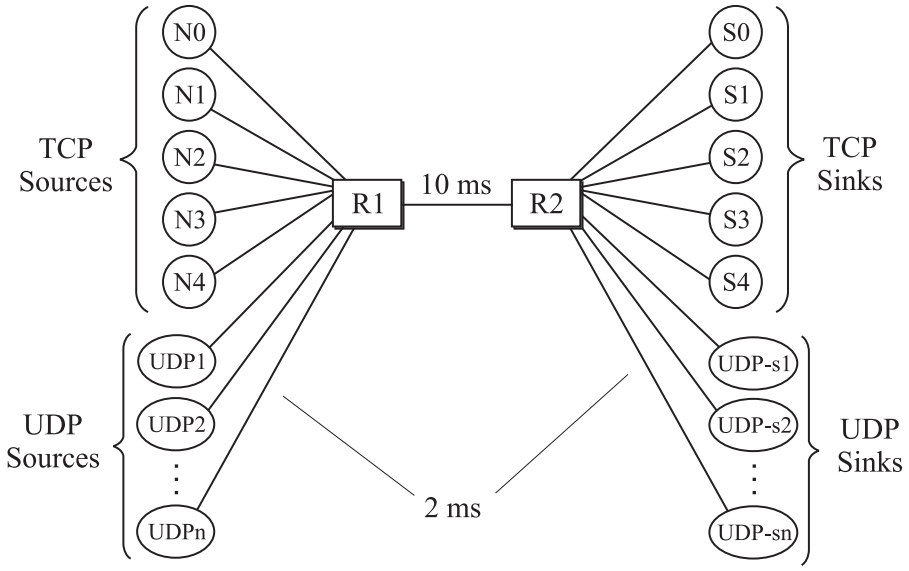


Figure 20: Simulation topology for an evaluation of BLACK.

link with a propagation delay of 10 ms. End nodes are connected to the routers at 100 Mbps with 2 ms delay. Both UDP and TCP flows transmit data with a packet size of 1 Kbyte. The maximum buffer space at the router R1 is set to 300 packets. Each experiment lasts for 200 seconds of simulation time, and it is repeated 20 times. The statistics are collected from 50 seconds to 200 seconds of simulation time.

SFB is set with a default configuration of two levels of hash functions of 23 bins with double set of hash tables for moving hash functions (total of 46×2 bins) using the NS code provided by [2]. The min_{th} and max_{th} threshold settings for BLACK, CHOKe, and RED are 50 and 150 packets respectively which are the Gentle RED parameters [23]. For the case of CARE, the number of capture occasions (t) is 200, where 50 out of 200 can be used for the estimation of the number of flows, and the probability p_{cap} is 0.04 according to [8]. BLACK uses the sample size (m) of 3,000 packets and the size of HBF cache memory of 20. The choice of BLACK parameters are discussed in Section 3.3.5 and 3.3.6. The parameter settings are summarized in Table 2.

RED	min_{th} = 50 packets, max_{th} = 150 packets.
CHOKe	min_{th} = 50 packets, max_{th} = 150 packets.
SFB	Two levels of hash functions of 23 bins with double set of hash tables for moving hash functions (total of 46×2 bins).
CARE	Capture occasion (t) = 200 with 50 of this value is used for the estimation of the number of active flows
BLACK	min_{th} = 50 packets, max_{th} = 150 packets, sample size (m) = 3,000 packets, HBF cache size = 20.
Common parameters	Maximum buffer size = 300 packets.

Table 2: Parameters of different queues for the evaluation of BLACK.

3.3.2 Single unresponsive flow

In the first experiment, only one UDP source transmits packets at the rate of 5 Mbps from node N5 to S5. The R1-R2 link is set with a low rate of 5 Mbps with 10 ms delay. When the UDP's arrival rate is equal to the R1-R2 link bandwidth, the R1-R2 link becomes a severe bottleneck link. An amount of 100 TCP traffic is randomly selected to originate from one of the source nodes N0 - N4 and flow to one of the respective sink nodes S0 - S4. The transmission delay of the access links are set to 10 ms. BLACK queue is equipped at the router R1 with the HBF cache size of 20, a size roughly equal to only 1/5 of the number of long-lived flows. For SFB, according according Feng et al. [19], the suggested mechanism for handling misbehaving flow is to limit the rate so that it would not exceed a certain defined level. Thus, three rate limit thresholds for SFB are set – twice the fair rate, half of the fair rate, and at the fair rate.

The individual throughput of each of the 100 TCP traffic and UDP traffic under RED, CHOKe, SFB, and CARE are summarized in Table 3.

As shown in the Table, for RED, the unresponsive UDP flow grasps almost all of the bandwidth leaving the remaining TCP connections with an extremely low throughput of 5.046 Kbps, compared to the fair share of 49.5 Kbps (5 Mbps divided by 101 as the number of flows). The result is not unexpected. Although RED is designed to drop packets from the flows in proportion to their

	Average UDP throughput (Kbps)	**Average TCP throughput (Kbps)**	**Jain's fairness index among TCP flows**
RED	4,374.82	5.046	0.5551
CHOKe	1187.301	38.160	0.9842
SFB, rate limit ≈ twice the fair rate	98.99	49.046	0.9836
SFB, rate limit ≈ fair rate	49.57	49.538	0.9826
SFB, rate limit ≈ half the fair rate	24.95	49.78	0.9817
BLACK	74.04	49.033	0.9871
CARE	172.66	48.307	0.9862

Table 3: Results from the single unresponsive flow scenario. Bottleneck link speed of 5 Mbps. UDP has a constant bit rate of 5 Mbps and competes with 100 TCP flows.

arrival rates, it can only deal with responsive traffic according to its dropping policy. Traffic flows that do not back off in response to the packet drops continue to pass through the queue with a very small drop rate until the queue is full or the average queue length reaches the maximum threshold. This is why RED could not prevent high bandwidth unresponsive traffic from consuming almost all of the bandwidth and leaving only a small portion for responsive traffic.

CHOKe, although it utilizes RED as a core mechanism, uses its own packet matching and dropping mechanism to provide better throughput for those TCP flows than does RED, by reducing the bandwidth of unresponsive UDP traffic. However, with a UDP throughput of 1187.301 Kbps, CHOKe still fails to achieve fair sharing since there are not enough packet matches to control the unresponsive flows effectively.

CARE is another mechanism that provides better fairness than both RED and CHOKe. However, the estimation of the number of active flows was not as good in the scenario with a high bandwidth UDP source[3]. Therefore, the UDP throughput is still relatively higher than the fair share of bandwidth in this case with high bandwidth UDP traffic. In addition, CARE needs more memory resources and computational complexity than the other schemes. Smaller capture size would affect the performance of CARE as demonstrated using the experimental setup above and shown in Figure 23, plotted with 95% confidence interval.

For SFB, the UDP connection receives a fair throughput only with a good adjustment of the rate limiting threshold. Note that, in practice, an inability to automatically set the fair rate limit is a big disadvantage of SFB. In addition, with the simulation code provided by the authors of SFB [19], the rate limiting is performed through a single parameter that controls how long the packets from the unresponsive flows are prevented from getting into the queue once detected. Setting up this parameter to achieve the fair rate, which is assumed to be known, is difficult since the flow's arrival rate was not known in advance.

The effect of the UDP arrival rate on the performance of these schemes is illustrated in Figure 22. RED has almost no control over UDP throughput when the bandwidth allotted to the connection is about the same as the arrival rate. CHOKe has better control of UDP throughput, but clearly is still far from the fair share of 49.5 Kbps. BLACK has the best overall control over UDP traffic even when its arrival rate is twice as much the bottleneck link. CARE also provides good

[3]A detail analysis of different methods to estimate the number of active flows will be discussed in Chapter 4.

fairness when the portion of UDP traffic is low, but its performance is not prolonged under a high load of UDP traffic due to the inaccuracy of the estimation of the number of active flows under this condition.

On the other hand, BLACK provides the UDP's with throughput that much closer to their fair share of 49.5 Kbps. In addition, while it provides more than adequate bandwidth to the TCP connections, the UDP throughput is bounded at some level regardless of its arrival rate, as shown in the Figure 21 (plotting with 95% confidence interval). For comparison, this figure is replotted with RED, CHOKe, and CARE as Figure 22.

For calculating fairness of TCP connections, Jain's fairness index [35] is used as an indicator with the following formula:

$$Fairness = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2} \tag{3.8}$$

where x_i is the effective throughput of the $i-th$ TCP flow and n is the number of connections, where the scheme with better fairness would have a fairness index closer to 1. From the results in the table for this scenario, the fairness among TCP connections for CHOKe, SFB, BLACK, and CARE are about the same under this symmetric topology with a little impact from single unresponsive UDP flow. In the case of SFB, the result also indicates that setting the rate limit to different value does not interfere with the fairness of TCP throughput.

Another experiment is conducted to see the comparative performance of BLACK in a high speed network under single unresponsive scenario. The same topology in Figure 20 is used but with a bottleneck link rate of 45 Mbps. Single CBR traffic is generated from node UDP1 to node UDP-s1 with an arrival rate of 10 Mbps. Hundred TCP traffic are configured in the same way in the previous experiment. The maximum buffer size is set to 800 packets, where the min_{th} and max_{th} are set to 200 packets and 400 packets respectively according to Gentle RED parameters setting. The result is shown in Table 4.

Even with a CBR rate of roughly 22% of the bottleneck link rate, the results is having the same trend as those obtained in the previous experiment shown in Table 3. With SFB (with fair rate limit), and BLACK, both UDP and TCP traffic gain the bandwidth of almost equal to the fair share of 445 Kbps. CARE could control UDP at some level where the UDP connection obtains the throughput of less than twice the fair share. However, for CHOKE and RED, each TCP traffic

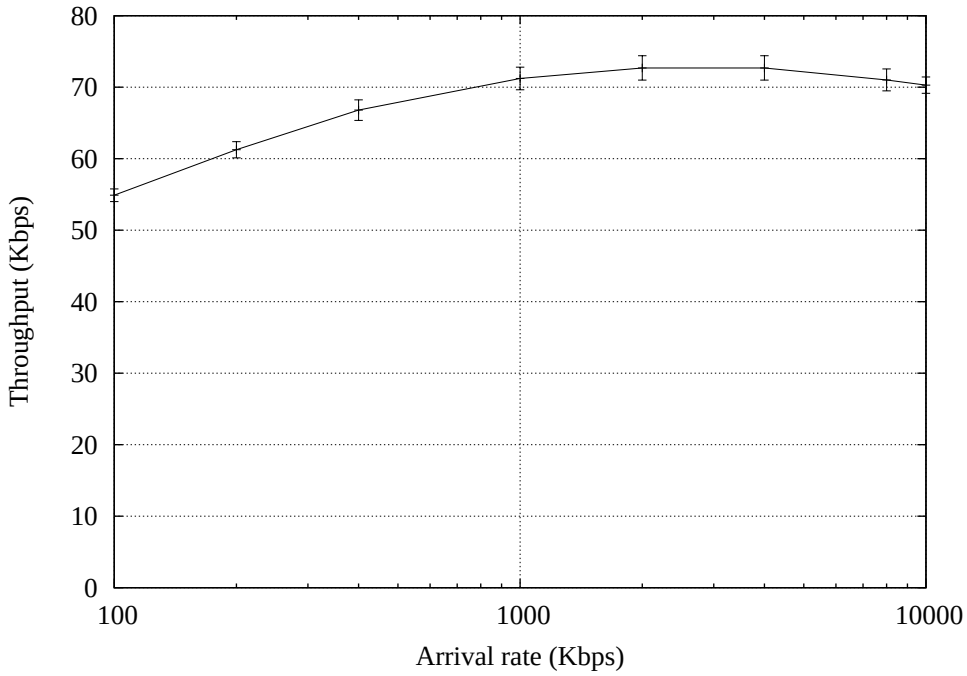


Figure 21: UDP throughput vs. arrival rate for BLACK; plotted with 95% confidence interval.

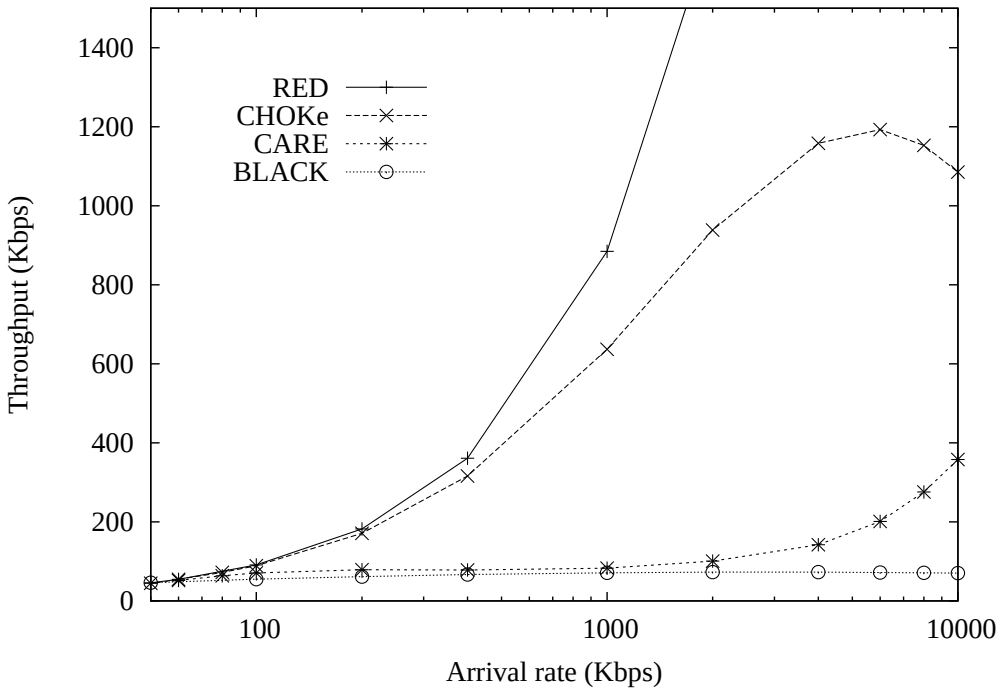


Figure 22: UDP throughput vs. arrival rate for RED, CHOKe, BLACK and CARE; Each point is an average value calculated for 20 runs.

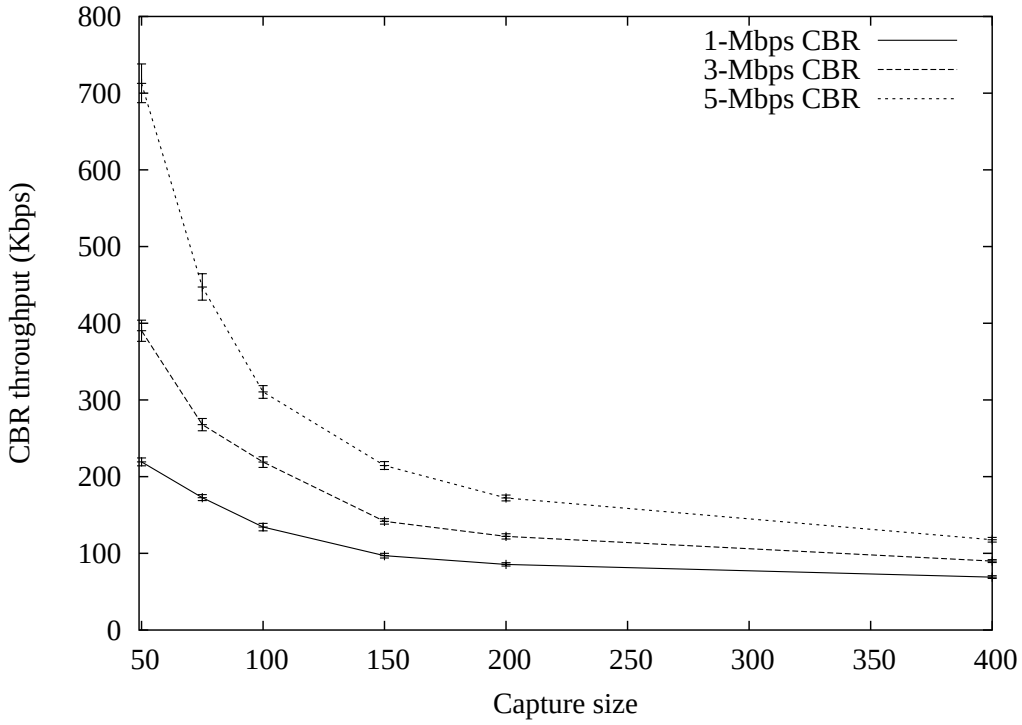


Figure 23: Effect of capture size on CARE performance.

	Average UDP throughput (Kbps)	**Average TCP throughput (Kbps)**	**Jain's fairness index among TCP flows**
RED	9,768.23	352.77	0.9952
CHOKe	6,632.55	384.12	0.9964
SFB, rate limit ≈ twice the fair rate	909.85	450.37	0.9792
SFB, rate limit ≈ fair rate	454.92	445.91	0.9815
SFB, rate limit ≈ half the fair rate	222.39	448.15	0.9807
BLACK	426.21	446.19	0.9948
CARE	825.80	442.19	0.9944

Table 4: Results from the single unresponsive flow scenario. Bottleneck link speed of 45 Mbps. UDP has a constant bit rate of 10 Mbps and competes with 100 TCP flows.

achieve only 86% and 79% respectively of what they should have been obtained based on the fair share (a lost of about 100 Kbps and 70 Kbps), leaving the bandwidth of 21.9 and 14.9 times the fair share to the UDP connection.

3.3.3 Multiple unresponsive flows

To demonstrate how well each different scheme could handle multiple unresponsive flows which is more similar to a real word scenario, the experiments with multiple CBR traffic are conducted based on the same simulation topology from Figure 20. Five unresponsive UDP flows are fed into network through the bottleneck R1-R2 link which has the capacity of 5 Mbps. Two experiments are arranged for high bandwidth CBRs and low bandwidth CBRs where each of five UDP traffic has a constant bit rate of 500 Kbps and 5 Mbps respectively. In addition, CHOKe is implemented with its *self adjusting mechanism* to handle multiple unresponsive flows. With this mechanism, the region between the minimum threshold (min_{th}) and the maximum threshold (max_{th}) are divided into 8 subregions (k) and the number of packet matching in CHOKe's dropping policy performs $2 \times i$ times per each packet arrival where $i = 1..k$ is the region where the current average queue size is falling into.

The results are tabulated in the Table 3.3.3 and Table 3.3.3. The average throughput shown in each column is average per-flow throughput. Both CHOKe and RED clearly show an inability to protect responsive TCP flows in either case. For the case of low bandwidth CBRs, each of the TCP connections gains only 71% and 60% of the fair share of 48 Kbps. In the case with high bandwidth CBR traffic, TCP traffic are completely shut out in the case of RED and receive only 5.1 Kbps per connection in the case of CHOKe. These results show that even CHOKe is equipped with the self adjusting mechanism, with the expense of up to eight packet samplings and matchings per single packet arrival, it could reduce the UDP throughput at some level and still hardly achieve fairness.

For the rest of the schemes, SFB, BLACK, and CARE still perform well under the scenario with low bandwidth CBRs. Although the average per-flow throughput of UDP connection is higher than the fair share of 48 Kbps, it is bounded to some small value leaving enough bandwidth for each of the TCP connections that receive around 97%. This performance, however, is no longer achieved with CARE under multiple high bandwidth CBRs case. The data from the simulation

(not shown) indicates that the precision of CARE's estimation of the number of active flows is altered by a heavy load of multiple unresponsive flows. On the other hand, BLACK still maintain a good performance under the heavy load from these unresponsive flows, even with a small HBF cache memory size of only 20. The Jain fairness index shown in the third column of the table also demonstrates that the fairness among TCP connections is unaltered under this extreme condition in the BLACK scheme.

3.3.4 TCP with different round-trip times

It is well known that even with cooperation among TCP sources exercising congestion control mechanism, fairness among TCP can be deteriorate when the connections have different round-trip times ($RTTs$). A TCP connection with a smaller round-trip time can grasp larger portion of bandwidth because it usually receives acknowledgment packets faster which results in an increasing of congestion window more rapidly. [47, 50] illustrate this fact through a model of TCP congestion control behavior, which shows that the achievable throughput of TCP is inversely proportional to RTT.

It is expected that the use of BLACK would reduce unfairness among TCP connections even without full per-flow state information. Connections with smaller round-trip times consume larger portions of buffer space, but they are more likely to be captured by the HBF cache memory and have a higher probability of being dropped, so they do leave some buffer space for smaller connections. In this manner, bias against connections with shorter round-trip times can be reduced. Figure 24 shows a simulation topology illustrating this advantage of BLACK, with different values of propagation delay denoted. In this experiment, two hundred TCP traffic flows were randomly selected to originate from nodes N0, N1, N2, N3 or N4 and traverse through a 45-Mbps link to randomly selected sink nodes S0, S1, S2, S3 and S4.

To conduct a fair comparison with SFB maintaining two levels of hash function of 23 bins, the BLACK scheme is configured at router R1 with a cache size of 23×2 or 46, and CARE is still set with a capture size of 200. The results are shown in Figure 26 with plotting at 95% confident interval. As shown, BLACK achieves better fairness than do RED, SFB, CHOKe, and CARE.

Table 7 shows the standard deviation and Jain's fairness index [35] achieved by each of the

	Average UDP throughput (Kbps)	**Average TCP throughput (Kbps)**	**Jain's fairness index among TCP flows**
RED	420.32	29.021	0.9565
CHOKe	318.52	34.107	0.9836
SFB, rate limit ≈ twice the fair rate	100.01	45.032	0.8863
SFB, rate limit ≈ fair rate	42.23	47.920	0.8595
SFB, rate limit ≈ half rate	17.66	49.150	0.8611
BLACK	67.60	46.653	0.9901
CARE	107.92	44.241	0.9852

Table 5: Results from the multiple unresponsive flow scenario. Bottleneck link speed of 5 Mbps. Each of the five UDPs has a constant bit rate of 500 Kbps and compete with 100 TCP flows.

	Average UDP throughput (Kbps)	**Average TCP throughput (Kbps)**	**Jain's fairness index among TCP flows**
RED	1000.00	0.000	N/A
CHOKe	841.29	5.092	0.1108
SFB, rate limit ≈ twice the fair rate	95.29	45.285	0.9594
SFB, rate limit ≈ fair rate	47.84	47.658	0.9611
SFB, rate limit ≈ half rate	22.94	48.903	0.9593
BLACK	65.02	46.378	0.9967
CARE	1000.00	0.000	N/A

Table 6: Results from the multiple unresponsive flow scenario. Bottleneck link speed of 5 Mbps. Each of the five UDPs has a constant bit rate of 5 Mbps and compete with 100 TCP flows.

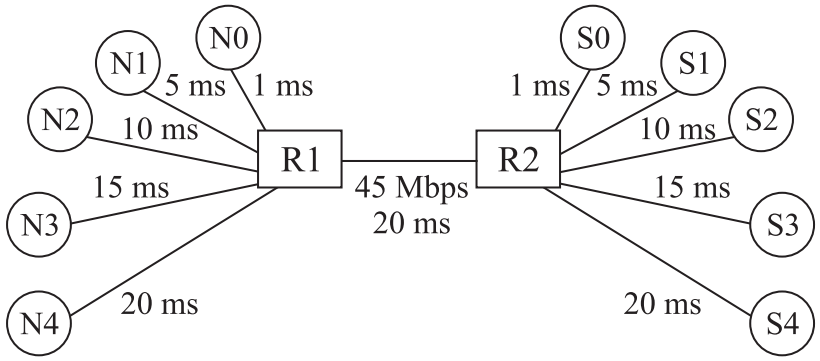


Figure 24: Simulation topology for TCP with different round-trip times.

four schemes. These standard deviation values are also plotted with 95% confidence interval over multiple runs in Figure 25. It can be seen that the BLACK scheme presented better fairness results than the other schemes, with a smaller standard deviation and higher fairness index. Note that SFB did not provide the good fairness for TCP traffic flows that was stated by Feng et al. [19].

3.3.5 Effect of the sample size

The sample size (m) or the number of packets to be sampled during each period effects how well BLACK can estimate the flows' buffer occupancy fractions. If the sample size is too small, not enough data is captured for a precise estimate to be made. In Section 3.2.3, the flow's buffer occupancy fraction is modeled as a binomial proportion with m trials of packet samplings. In this way, the minimum value of m can be chosen to be $100(1-\alpha)$ percent confident that the error is less than a specified value E, through

$$m = H(1-H)\left[\frac{z(\alpha/2)}{E}\right]^2 \tag{3.9}$$

where H is the buffer occupancy fraction. The value of H can vary from a very small fraction up to as large as 1 depending on the proportion of the flow's sampled packets over the number of total sampled packets in the period. Based on the fact that the value of $H(1-H)$ reaches a maximum when $H = 0.5$, the value of $H = 0.05$ can then be used to obtain the upper bound of m.

Therefore, if we want to be 95% confident that the error in using H to estimate the buffer occupancy fraction is less than E, the required sample size (m) is

$$\begin{aligned} m &= H(1-H)\left[\frac{z(\alpha/2)}{E}\right]^2 \\ &= H(1-H)\left[\frac{1.96}{0.02}\right]^2 \\ &= (0.5)(1-0.5)\left[\frac{1.96}{E}\right]^2. \end{aligned}$$

The required size (m) under this condition according to the different values of E is shown in Table 8. For $100(1-\alpha)$% confidence, where α = 0.01, 0.05, and 0.1, the required sample size is plotted as a function of error E, as shown in Figure 27.

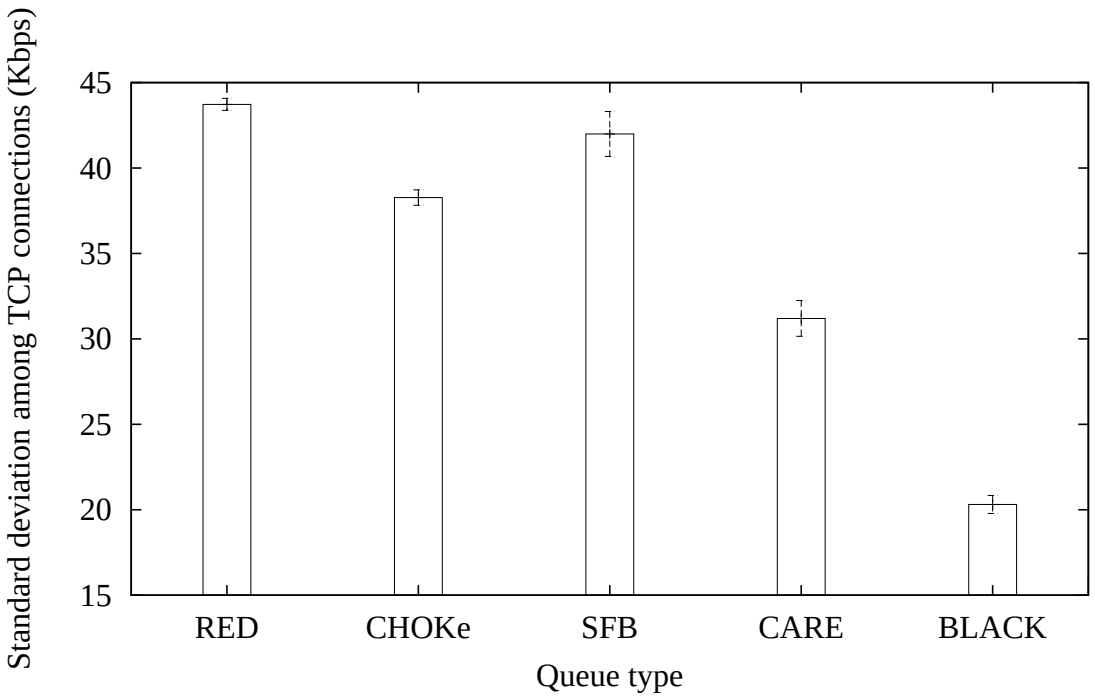


Figure 25: Standard deviation of TCP throughput under different queue types; plotted with 95% confidence interval.

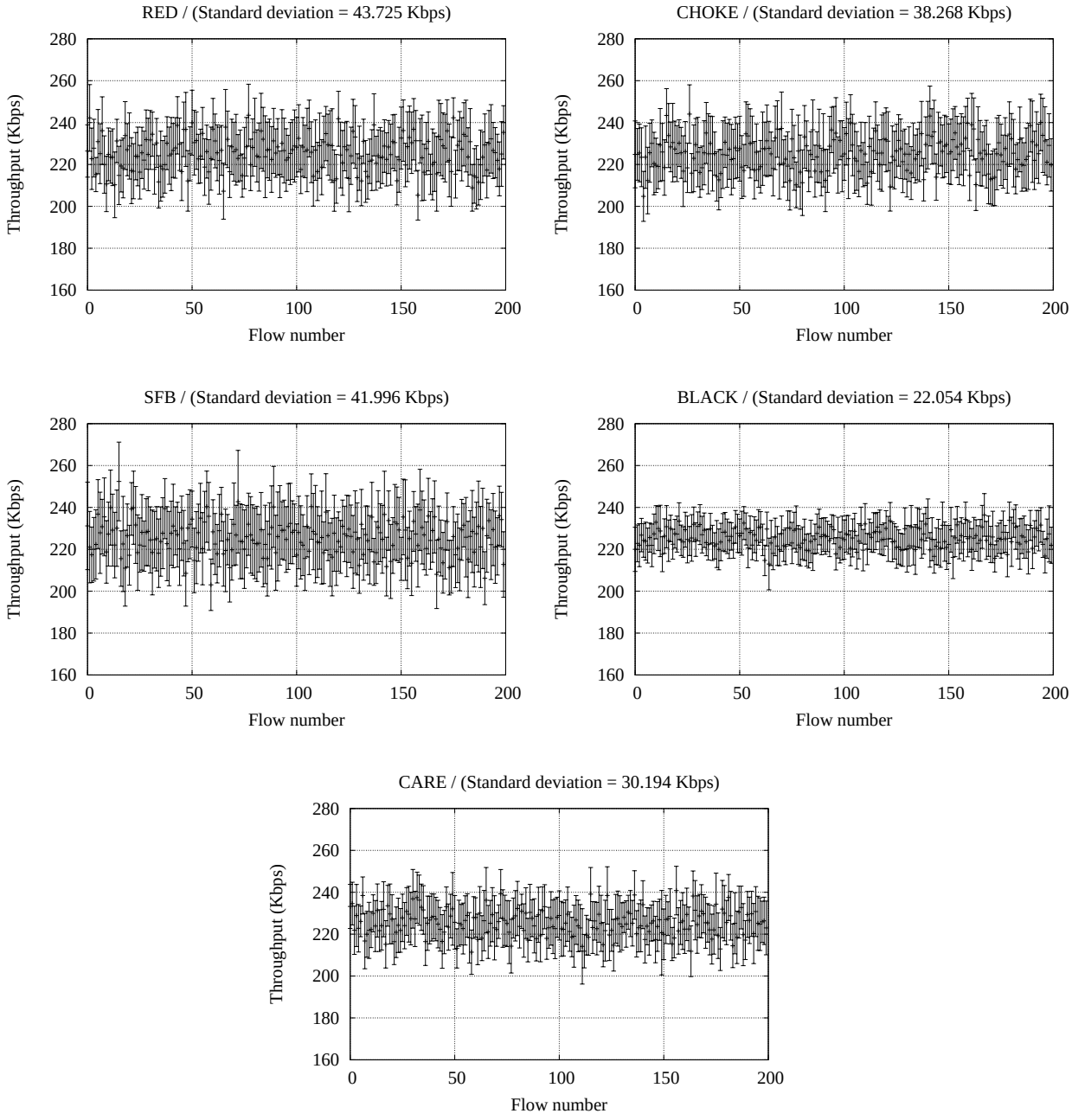


Figure 26: Per-flow TCP throughput for 200 TCP connections with different RTTs over a 45-Mbps link; plotted with 95% confidence interval.

Flow throughput (Kbps)	RED	CHOKe	SFB	BLACK	CARE
Standard deviation	43.725	38.268	41.996	22.054	30.194
Jain's fairness index	0.9769	0.9721	0.9665	0.9905	0.9815

Table 7: TCP connections in the different round-trip times scenario.

As shown in Table 8, if we want to be 95% confident that the error in using H to estimate the buffer occupancy fraction is less than 0.02, the sample size should be at least 2,401. An experiment using the same simulation setting as is described at the end of Section 3.2.3 was conducted to visualize what accuracy the estimation of the buffer occupancy fraction could achieve at different sample sizes. Results for sample sizes of 100, 600, 2,000, and 4,000 are depicted in Figure 28 and 29.

The sample size in all of the experiments described in this chapter was chosen to be 3,000 because this was more than the minimum sample size required to be 95% confident that the error of the estimation would be less than 0.02, a reasonable range of tolerable error. As an example, another experiment was performed using the simulation settings for a single CBR traffic flow as is described in Section 3.3.2 and the same topology as is shown in Figure 20. The performance of BLACK was measured for two scenarios of 1-Mbps and 5-Mbps CBR arrival rates. The results in Figure 30 show clearly that when the sample size is small, BLACK does not have enough data to perform good bandwidth control on CBR traffic as well as it does when the sample size is at least 3,000 packets.

3.3.6 Determining the size of HBF cache memory

The size of HBF cache memory should be large enough to ensure that the records of high bandwidth flows are contained in the memory according to the LRU operation. Two approaches are possible.

The first approch is a probabilistic approach based on an observation of measured data. Suppose the proportion of the rate of unresponsive flow over an aggregate arrival rate is known as γ, then the probability that any arrival packet is from the unresponsive flow is γ. Therefore, the cache

E	m
0.02	2,401
0.03	1,067
0.04	600
0.05	384

Table 8: Sample size (m) as a function of E with 95% confidence.

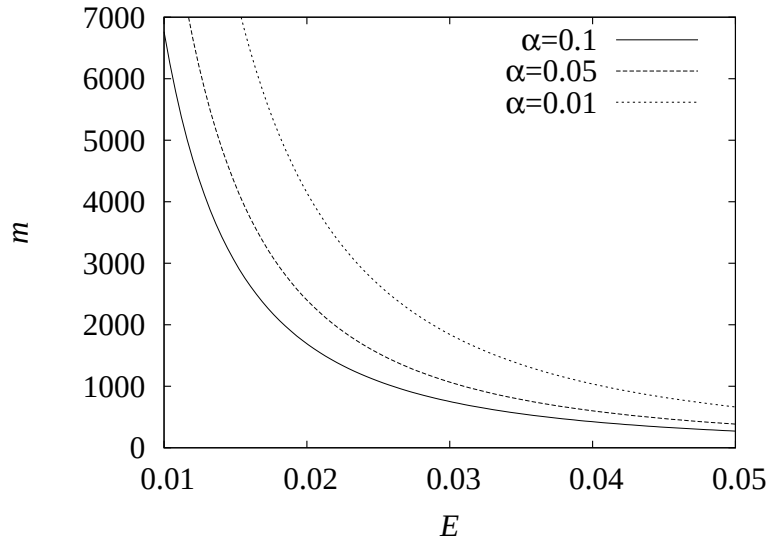
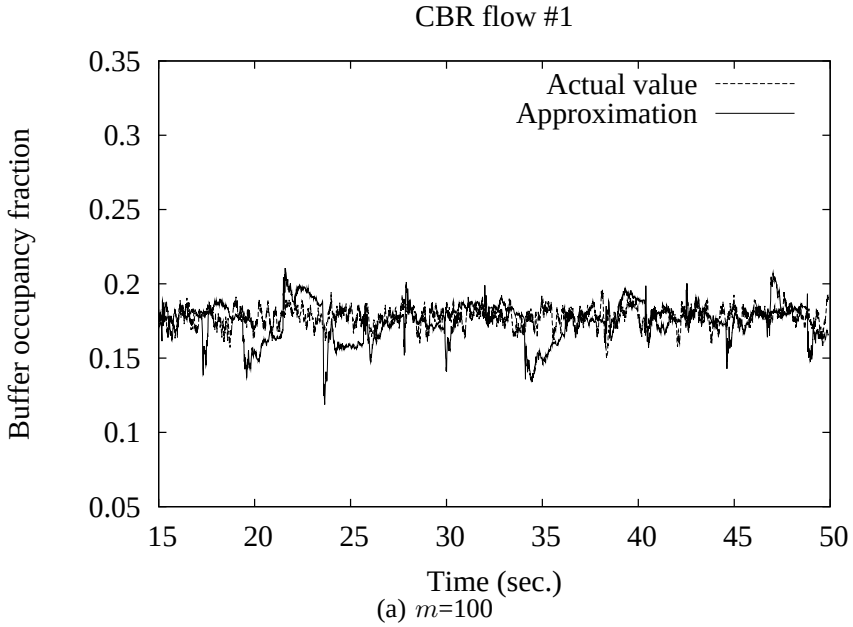


Figure 27: Sample size (m) as a function of E with $100(1-\alpha)$ confidence.

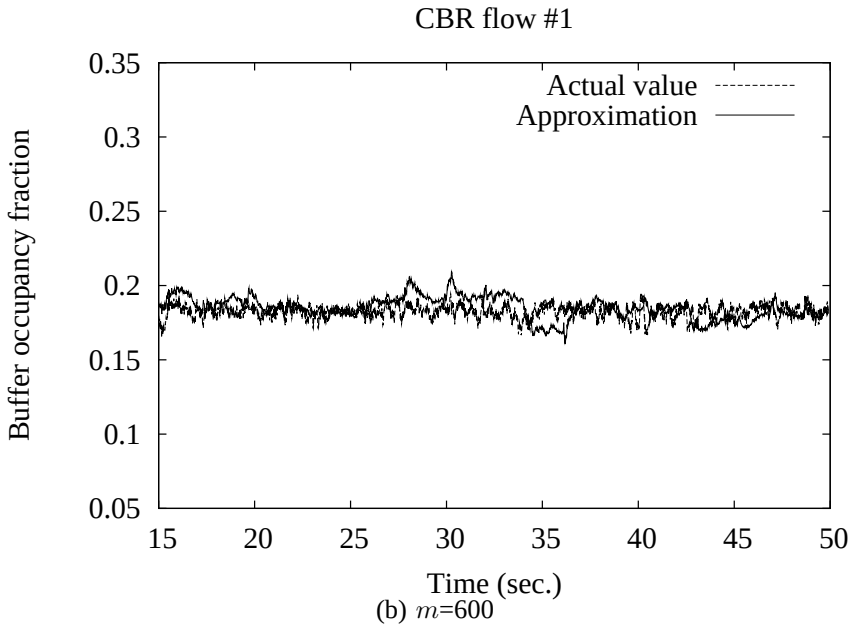
size should be approximately $1/\gamma$. In practice, γ could be observed from measured data in advance by a network operator to determine an appropriate cache size.

To demonstrate this strategy, a simulation is set up with one CBR traffic competing with 100 TCP traffic over a 5-Mbps link. Three experiments are conducted for three different arrival rates of CBR traffic – 200 Kbps, 500 Kbps, and 1 Mbps. The result is illustrated in Figure 31. For 1-Mbps traffic, the arrival rate is roughly around 1/4 of the aggregate arrival rate, therefore $\gamma \approx 1/4$. Consequently, a cache size of about $1/\gamma$ or 4 would be enough to regulate the 1-Mbps traffic effectively. From the figure, even a size of three is enough to make BLACK performed as designed. On the other hand, with the same calculation, a cache size of three is not enough for 200-Kbps and 500-Kbps traffic and CBR traffic gain much higher bandwidth than the fair share (dashed line).

For a second approch to set the size of HBF cache, if the memory is not scarce, the size could be dynamically allocated using to the number of long-lived active flows. Note that in this manner the cache size is still small comparing to the total number of active flows due to the fact that most bytes from the Internet is from only the small number of flows [53, 38, 29, 30] and BLACK only manages to regulate those elephant (large) flows rather than mice (small) flows, according

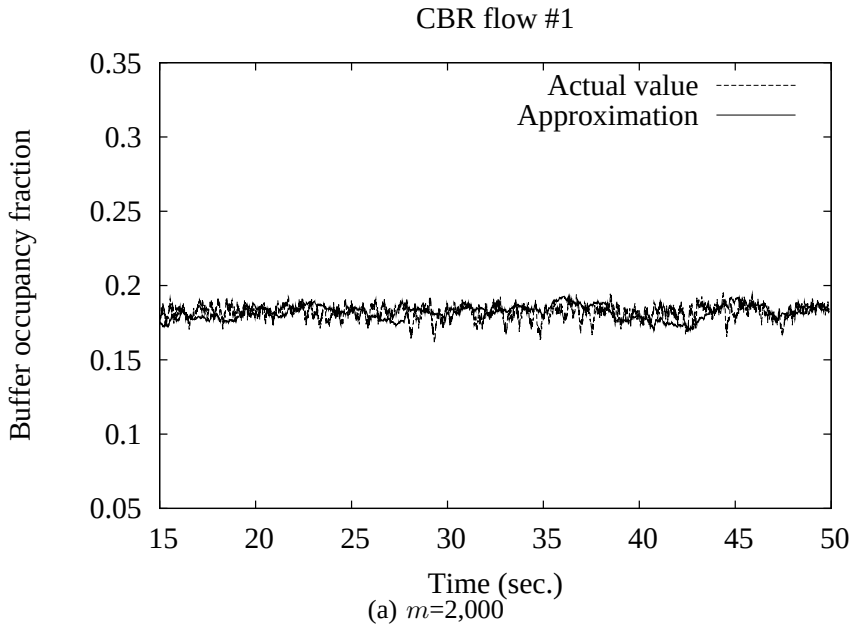


(a) m=100

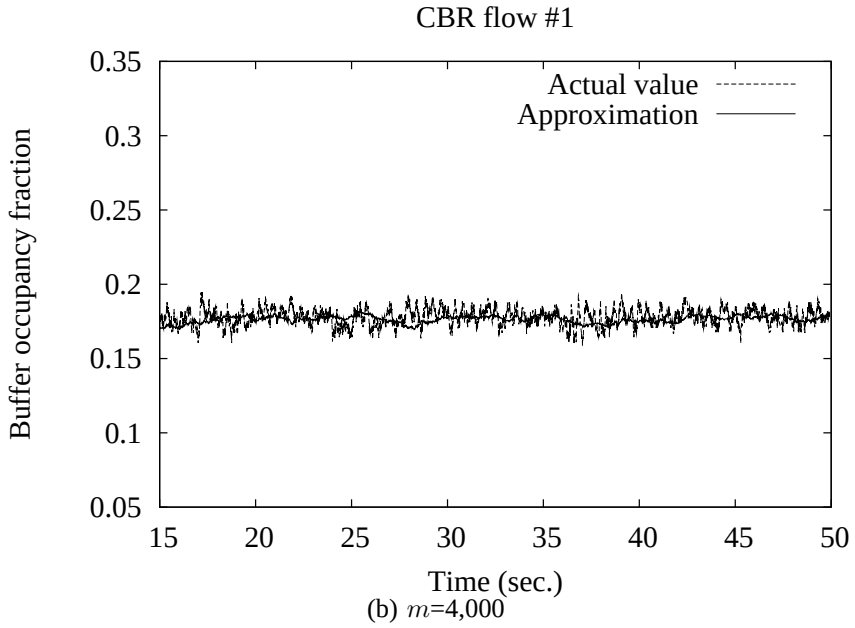


(b) m=600

Figure 28: The effect of sample sizes (m) on approximation of the buffer occupancy fraction.



(a) m=2,000



(b) m=4,000

Figure 29: The effect of sample sizes (m) on approximation of the buffer occupancy fraction (continue).

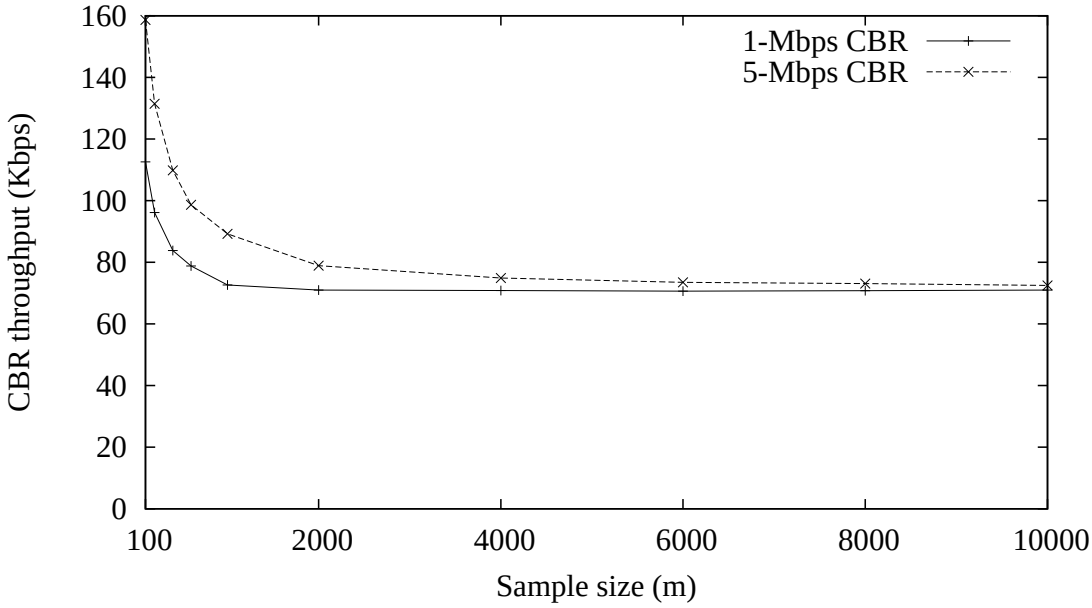


Figure 30: The effect of sample size on the performance of BLACK.

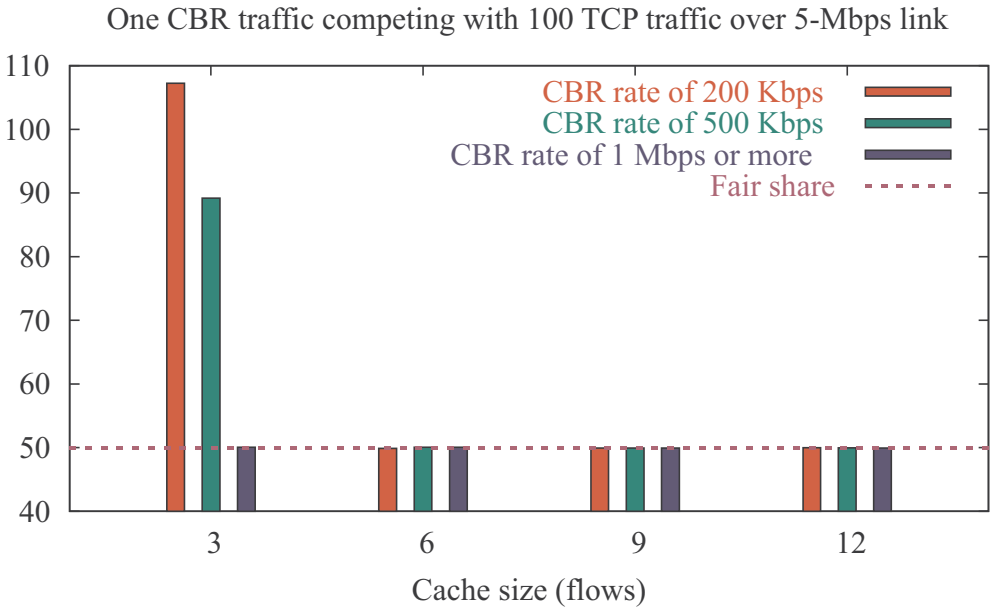


Figure 31: Performance of BLACK according to different cache sizes and CBR arrival rates.

to *mice and elephant* model described in Section 3.2. In this way, it is still scalable and practical to dynamically allocate the cache size equals to the number of large flows. In addition, not only unresponsive traffic is regulated, fairness among long-lived TCP connections with different round-trip delays would be improved.

3.4 LIMITATIONS OF BLACK

Throughout the series of simulation experiments described in this chapter, BLACK has shown an ability to both protect responsive TCP traffic from high bandwidth unresponsive flows and improve fairness among TCP connections with different round-trip times. It requires a very small amount of per-flow state information and is relatively less computationally complex than other schemes. Upon the arrival of each packet, BLACK samples a packet from the queue and updates some information in an HBF cache, which could be a high speed memory. Less frequently, it calculates the buffer occupancy fraction and number of active flows. Only a packet from a flow that is consuming more buffer space than its fair share is subjected to dropping, as shown in Equation 3.4. However, BLACK still has some limitations, as described below.

1. BLACK's estimation of the number of active flows could be inaccurate in some scenarios, such as when the size of the queue buffer is small or when the intensity of traffic is largely unequal. high bandwidth unresponsive traffic could be out of control if BLACK underestimated the number of active flows.

   This problem implies that BLACK needs a better estimation of the number of active flows. This problem is analyzed in greater detail and alternative methods for estimating the number of active flows are discussed in the following chapter.

2. BLACK could not perform well in the scenario in which different packet sizes are injected into the queue from different flows. Two problems could occur. The first problem is with the way BLACK estimates the number of active flows when the estimation assumes equal traffic intensity. Different packet sizes from incoming traffic flows imply different possibilities of *match* events for flows. In this way, two connections with different packet sizes could be perceived as having different traffic intensities, and this estimation would be inaccurate.

The second problem could occur because BLACK collects information about each sample packet through a *Hit* variable without accounting for the packet size. Smaller packets have a higher possibility of being sampled than do larger packets, which may appear to have a higher buffer occupancy fraction even through the number of bytes from these connections are the same. As a result, BLACK could over-penalize a traffic flow whose packets have a higher tendency to be sampled from the queue because of their small packet sizes. The inferiority of fairness performance due to different packet sizes is not unique to BLACK; it is also a problem for CHOKe and CARE. For CHOKe, it is clear that a flow with smaller packet sizes would have more packet matches than a flow with larger packet sizes, assuming the same arrival rate. A flow with more packet matches would be penalized more. For CARE, the mechanism only accounts for the number of packets, regardless of the packet sizes.

This problem could be illustrated by the same simulation experiment as discussed in Section 3.3.2. However, instead of one CBR traffic flow, two 2.5-Mbps CBRs with packet sizes of 500 bytes and 1,000 bytes are fed to the queue. The results, plotted in Figure 32, show that the CBR traffic with 500-byte packet size receives about only half of the bandwidth gained by the CBR traffic with 1000-byte packet size even through they have the same arrival rate.

3. 3. CBR throughput fluctuates more after passing through a BLACK queue. This occurs even if the arrival rate of CBR is not very high. Adapting the simulation setting described in Section 3.3.2 so that the CBR arrival rate is only 1 Mbps (20% of total bandwidth), the throughput of CBR over time is shown to have a lot of fluctuation in Figure 33. This behavior is not less than preferable, especially for media applications.

Despite these limitations, BLACK exhibits superior fairness performance with low overhead and is preferable to the other lightweight fair AQM schemes in various environments. However, the problem of the estimation of the number of active flows may significantly degrade the performance of BLACK in some scenarios. In the next chapter, this problem is addressed in greater detail and alternate mechanisms that could replace BLACK's original estimation mechanism are discussed.

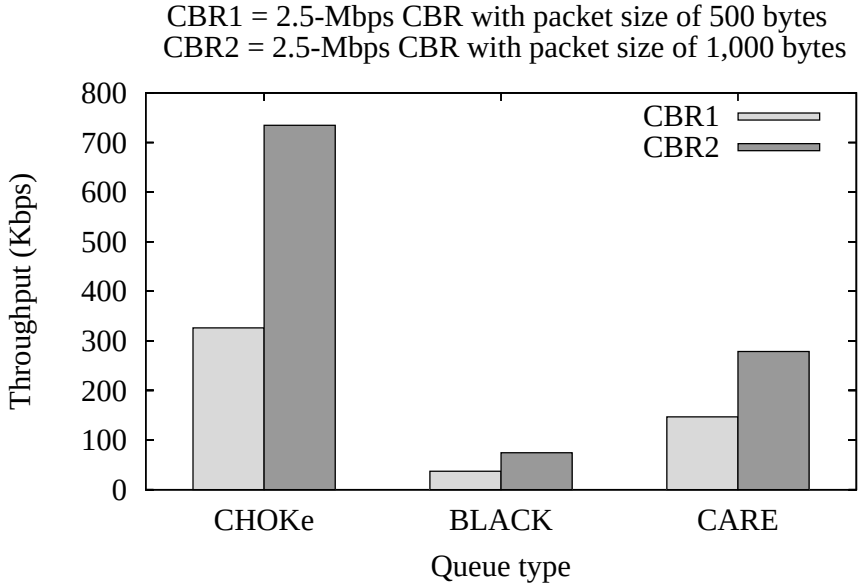


Figure 32: Average throughput of two CBR traffic flows with packet sizes of 500 bytes and 1,000 bytes, under CHOKe, BLACK, and CARE.

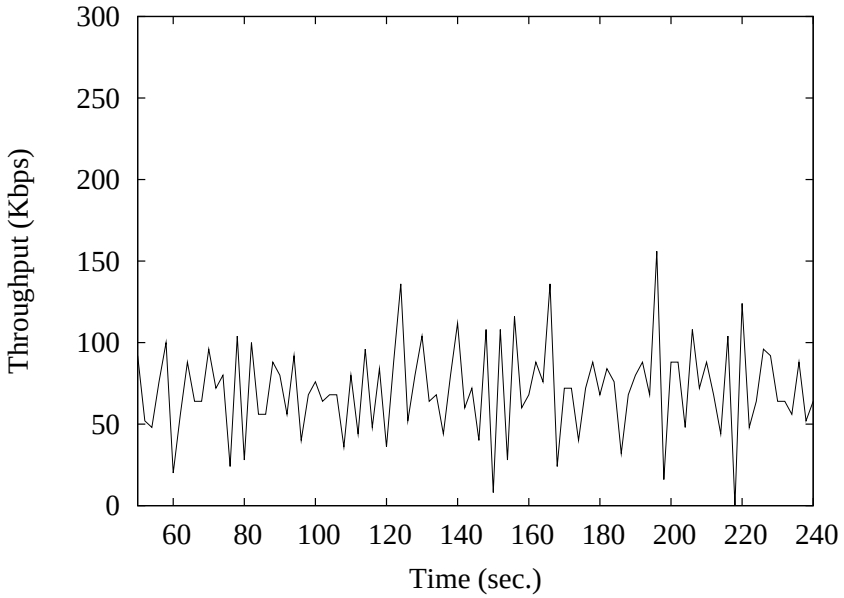


Figure 33: Fluctuation of CBR throughput after passing through BLACK queue.

4.0 THE ESTIMATION OF THE NUMBER OF ACTIVE FLOWS

The estimation of the number of active flows is an important component of BLACK in order to effectively provide a fair share of bandwidth. As shown through a pilot experiment in Section 3.2.5 and a series of simulation experiments in the previous chapter, BLACK's estimation of the number of active flows has shown the promising results in several scenarios. However, there are some situations, which are not rare, that the mechanism could not produce an accurate estimation. This inaccuracy could lead to an inferior performance of BLACK as it might under-punish or over-punish the misbehaving traffic or even a responsive traffic. A degree of performance penalty depends on the inaccuracy obtained the estimation.

The problems of the mechanism that BLACK incorporates to estimate of the number of active flows are covered in great details in the following section. Then, the alternative mechanisms to estimate the number of the active flows available in the literatures are reviewed and discussed in the rest of this chapter[1].

4.1 PROBLEMS OF BLACK'S ESTIMATION OF THE NUMBER OF ACTIVE FLOWS

An inaccuracy of BLACK's mechanism to estimate the number of active flows could occur in a number of situations which is basically a result of two fundamental factors below:

1. **Small buffer size:** An inaccuracy occurs when a buffer size is not large enough, comparing to a bandwidth-delay product, where there could be a correlation between a sampled packet and an incoming packet. Under this circumstance, at a given instantaneous time, packets from one or few connections may occupy large portion of a buffer space due to a relatively large pipe

[1] Some parts of the materials in this chapter are to be published in [10]

of transmission line, giving a higher chance that the sampled packet from the buffer and an incoming packet are from the same connection.

2. **Unequal traffic intensity:** The estimation model assumes the same traffic intensity for those N flows, an inaccuracy can occur when the traffic intensity is very different.

Both factors are discussed along with the supporting simulation results as follows.

4.1.1 Problem of small buffer size

The problem of small buffer size can demonstrated through a following experiment. Using a ten-node dumbbell topology as shown in Figure 34, A hundred TCP traffic are randomly assigned to originated from one of the five source nodes on the left associated with one of the five destination nodes on the right. These TCP connections inject the packets, each of 1,000 bytes in size, to a bottleneck link ($R1 - R2$) of 5 Mbps with 5 ms delay. All the access links have 2 ms delay.

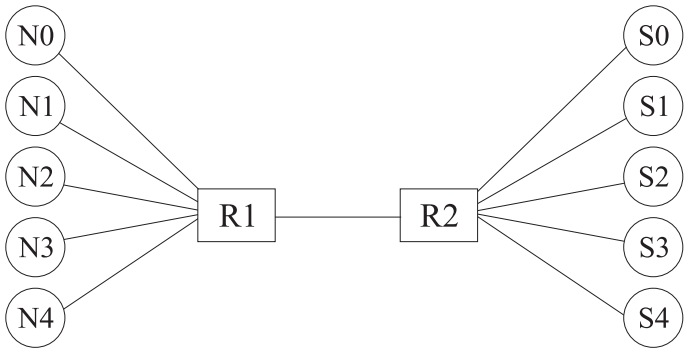


Figure 34: Ten-node dumbbell Topology.

With a setting for a maximum buffer size of 300 packets, it is more than enough to get an estimation that comes close to the actual number of active flows, for the pipe of 10Mbps $*$ 5ms $=$ 6.25 packets in bandwidth-delay product, as shown in Figure 35.

Now, when the bandwidth-delay product of the link is increased with the link bandwidth of 45 Mbps and a delay of 20 ms, the same buffer space of 300 packets becomes too small for an accurate estimation. In this scenario, a bandwidth-delay product becomes 45 Mbps $*$ 20ms = 112.5 packets. With a buffer size of at most 300 packets and a larger burst of traffic at high speed

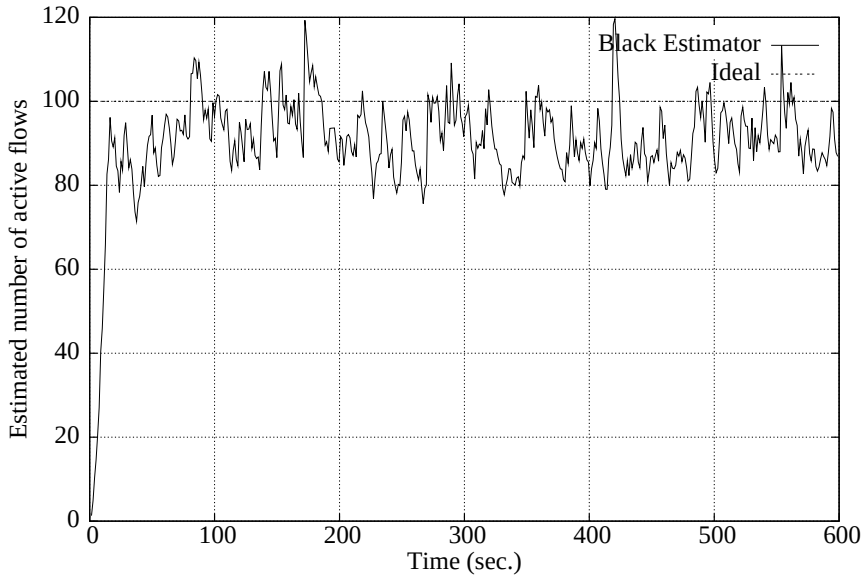


Figure 35: BLACK's estimated number of active flows for 100 TCP flows over a 10-Mbps link with 5ms delay. Maximum buffer size of 300 packets.

in nature, it provides a higher chance of having a correlation between a sampled packet and an incoming packet. Consequently, a probability P_{match} is higher than the actual value. With an estimated number of active flows $Nact$ that is proportional to $1/P_{match}$ according to Equation 3.7, the result is then underestimated as shown in Figure 36.

To relief this problem, the maximum buffer size should be set with a higher value to cope with a higher bandwidth-delay product. With a new setting of 3000 packets of maximum buffer size, the estimation provides a much more accurate value of the number of active flows. This new result is illustrated in Figure 37.

Although this problem could be resolved by using a large buffer size, the solution introduces a higher queueing delay. As a large buffer size becomes a requirement of BLACK's estimation of the number of active flows, it is not advised to use this method when queueing delay is a critical factor. One way to overcome this buffer space requirement is to employ a virtual buffer that records a flow ID of the packets that have been queued in the actual buffer, where a virtual buffer size is much larger than an actual buffer, as depicted in Figure 38. If the virtual buffer is full, the packet at the head of the virtual buffer will be discarded leaving one available slot at the end for a new record.

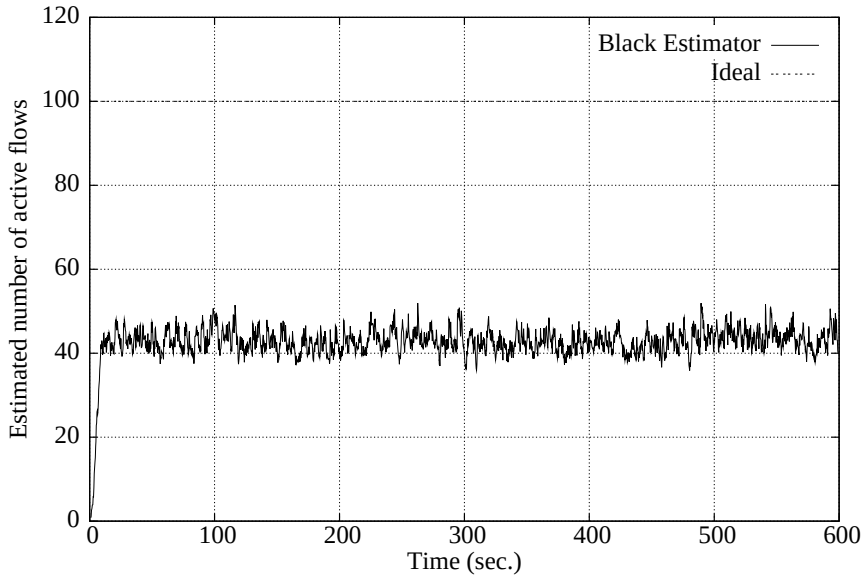


Figure 36: BLACK's estimated number of active flows for 100 TCP flows over a 45-Mbps link with 20ms delay. Maximum buffer size of 300 packets.

In this way, a small buffer space problem should be kept minimized, given that memory space is not scarce.

4.1.2 Problem of unequal traffic intensity

Even with a large buffer size, BLACK's estimation method could still be suffered when the traffic intensity is much different. This problem could occur when the round trip delay are largely different among TCP connections or when packet sizes are different which would bias the value of $P_{a,i} \cdot P_{s,i}$, where flow i is a flow with smaller packet size. The effect of this problem becomes more compelling when high-bandwidth UDP connections co-exist with TCP traffic. This situation can be demonstrated through a simulation as described below.

Using the same symmetric topology in Figure 34 with the bottleneck link of 10MBps bandwidth and 20ms delay, 100 TCP traffic are now competing with 10 UDP traffic with aggregate arrival rate equal to half of a bottleneck link bandwidth. UDP traffic starts at 300 seconds of simulation time. The buffer size is set to 3,000 packets to avoid the problem of small buffer size. Without any bandwidth fairness control mechanism, the estimation is shown to be largely distorted

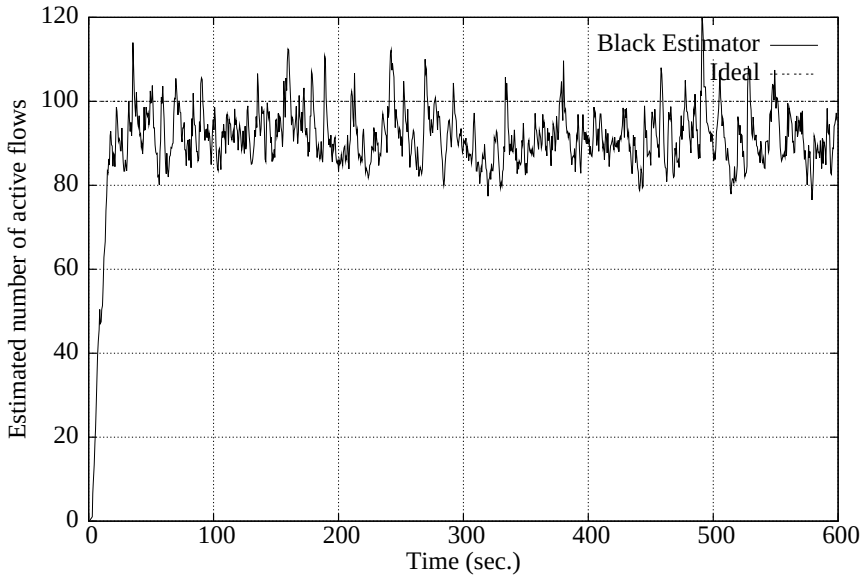


Figure 37: BLACK's estimated number of active flows for 100 TCP flows over a 45-Mbps link with 20ms delay. Maximum buffer size of 3,000 packets.

in Figure 39.

Without any queueing mechanism or dropping policy for traffic rate control, traffic with distinctively higher intensity such as UDP traffic in this scenario would have a higher chance of being sampled from the buffer ($P_{s,i}$) as well as a higher chance of arriving at the buffer ($P_{a,i}$) at any given time. Since $P_{match} = \sum_{i=1}^{N} P_{a,i} \cdot P_{s,i}$ and estimated number of active flows ($Nact$) is proportional to $1/P_{match}$, $Nact$ would be underestimated and approach a value of one as the proportion of bottleneck bandwidth consumed by UDP traffic approaches 100%. This phenomenon is illustrated

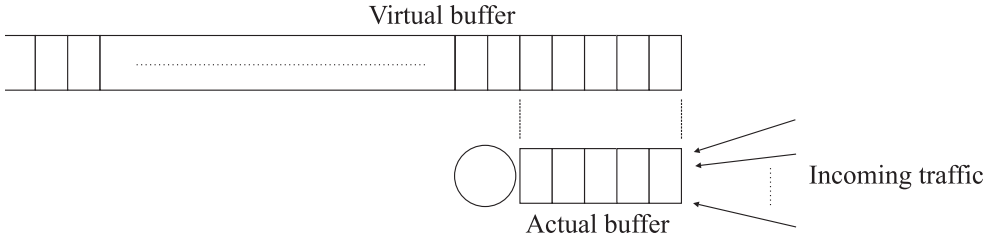


Figure 38: BLACK with virtual buffer to resolve small buffer problem.

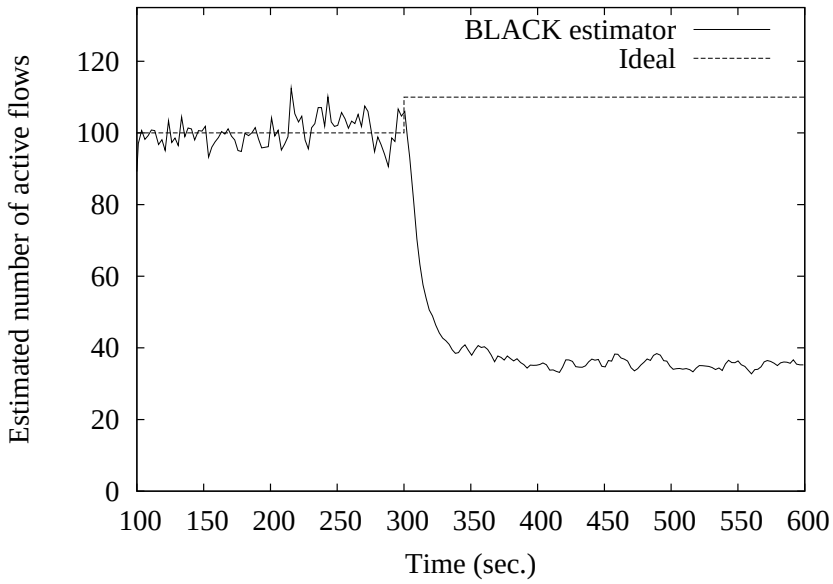


Figure 39: BLACK's estimated number of active flows for 100 TCP flows and 10 UDP flow over a 10-Mbps link with 20ms delay. UDP traffic start at 300 seconds and consume half of the bottleneck link bandwidth.

in Figure 40 through the same simulation settings but with varying UDP arrival rate, plotted with 95% confidence interval.

Consequently, an alternative method to improve the accuracy of estimation with no requirement of huge buffer space and equal traffic intensity is advised.

4.2 ALTERNATIVE METHODS TO ESTIMATE THE NUMBER OF ACTIVE FLOWS

In this section, two alternative approaches for estimating the number of active flows are discussed and evaluated – **(1) Bitmap Approach** [15, 16] and **(2) Capture-Recapture (CR) model approach** [8].

Bitmap approach replies on a probabilistic algorithm in conjunction with hashing at bit level. As a flow ID of an incoming packet hashes into only a single bit of memory size combined with its own hashing technique to reduce a size of hash table, Bitmap approach is claimed to be able to estimate the number of active flows with relatively small amount of memory space requirement.

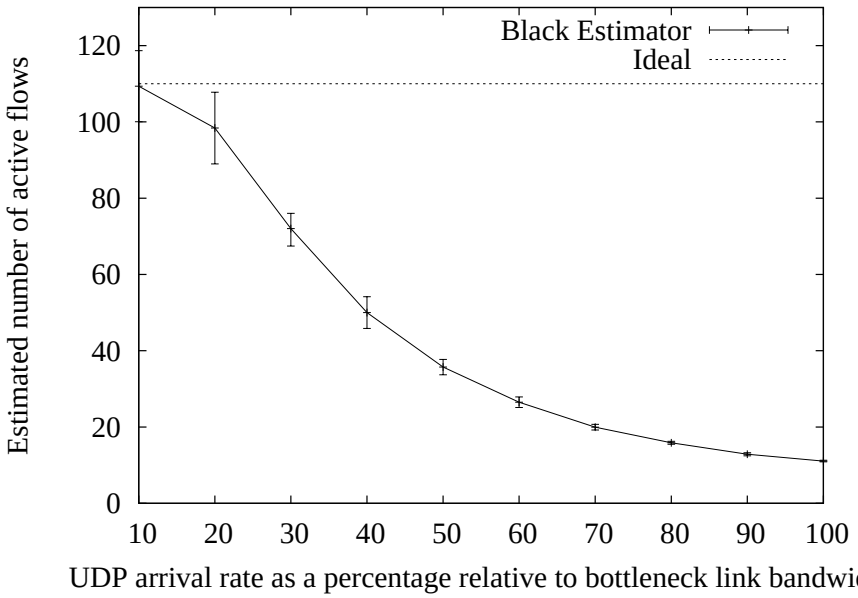


Figure 40: BLACK's estimated number of active flows for 100 TCP flows and 10 UDP flow over a 10-Mbps link with 20ms delay. UDP arrival rate varies from 10% to 100% of bottleneck link bandwidth.

The simplest form of bitmap scheme is known as *Direct Bitmap* technique is included in this section.

On the other hand, CR model approach borrows an estimation technique from Capture-Recapture model that is widely used to estimate the number of animals in a population by ecologists and to estimate the number of defects in software inspection process. By constructing a frequency histogram containing which flow IDs have been seen once, twice, and up to t times, after capturing a certain amount of incoming packets, *Jackknife Estimator* [7] can then be used as an estimator to calculate an approximated number of active flows for this CR model. CR model with Jackknife Estimator is the estimation mechanism that is utilized by CARE as discussed in Section 2.2.3.8.

In addition, according to an evaluation of different estimators for CR model in [36], apart from Jackknife estimator, *First-order Sample Coverage* and *Second-order Sample Coverage* [44] are shown to have superior performance in their estimation of number of females yellowstone grizzly bear population. Therefore, in this section, both Sample Coverage techniques will be applied to the problem of estimating the number of active flows through a CR model and compare with the other schemes.

Therefore, the following four estimating techniques will be discussed in details below and evaluated through a series of simulation in Section 4.3:

- Direct Bitmap
- CR model with Jackknife estimator
- CR model with First-order Sample Coverage
- CR model with Second-order Sample Coverage

4.2.1 Direct Bitmap

Direct Bitmap [15, 16] is the simplest form of Bitmap approach to estimate the number of active flows that are passing through to a network node. The idea of Direct Bitmap is to probabilistically count the number of bits that are hashed according to the flow ID, and calculate the number of active flows at the end of a measurement interval according to the formula developed from the probability that the hashed collisions would occur.

Direct Bitmap begins with resetting all the bits in a bitmap of size b bits to zero. For each incoming packet, a hash function is applied on the flow ID to map each flow to a bit of the bitmap. In this way, once a packet arrives, a bit that the flow ID hashes to is set to 1. Note that the method assumes that the hash function distributes the flows randomly.

At the end of a measurement interval, an estimation of the number of active flows during that period is calculated using a formula that takes into account collisions from hashing. For a bitmap of size b bits, the probability that a given flow hashes to a particular bit is $p = 1/b$. Assuming that N is an actual number of the number of active flows, the probability that no flow hashes to a particular bit is therefore $p_z = (1-p)^N \approx (1/e)^{N/b}$. Then, the expected number of bits not set in the bitmap at the end of the measurement interval can be calculated by $E[z] = bp_z \approx b(1/e)^{N/b}$. Here, z is the number of $zero$ bits found in the hash table during the interval. Finally, the estimated number of active flows $\hat{N}$ is then equal to

$$\hat{N} = b \ln \left(\frac{b}{z}\right) labeleq : directbitmap \tag{4.1}$$

The advantage of this scheme, apart from using only small amount of memory space, is that the estimation is not effected by the intensity of incoming traffic as all packets belonging to the same flow map to the same bit.

There are several variants of Direct Bitmap – *Virtual Bitmap, Multiresolution Bitmap, Adaptive Bitmap,* and *Triggered Bitmap* – that require much less memory space. However, the fundamental concepts of these variants are basically the same as Direct Bitmap, thus only Direct Bitmap is included for evaluation.

4.2.2 CR model with Jackknife estimator

CR model has been widely used to estimate the number of animals in a population by ecologists and biologists, and to estimate the number of defects in software inspection process. Later it has been adapted to estimate the number of active flows having their packets in a queue [8]. The basic idea of CR model, in terms of estimating the number of animal population, begins with an inspector randomly captures a small amount of animals from its habitat. Then he counts the number of these captured animals, marks and releases them. Later, he captures a small amount of animals again,

and counts the number of captured animal and the marked animals that are recaptured. Finally, using these capture-recapture data, an estimated number of animals is calculated.

Three basic models provided in capture-recapture method are (1) Model M_b: assume that the probabilities of capture among animals vary with the behavioral response of these animals, (2) Model M_h: allow the variance of the probability of capture for individual animals, and (3) Model M_t: assume that the probabilities of capture varies by time. In this manner, M_h model is suitable to estimate the number of active flows where the capture probability are different among traffic flows.

Because M_h model can have as many as $n+1$ parameters: N and $p_1, p_2, ..., p_n$, where p_i is the capture probability for an individual flow i and N is the actual number of active flows, while N is the only value needed to be known, the authors in [8] suggest the *Jackknife estimator* as a method to estimate N without having to estimate all the p_i [7].

For simplicity, the algorithm of M_h model will be shown through an example of estimating the number of animals in a population as follows:

1. Suppose animals are captured in 18 days with one capture occasion per day, so there are total 18 capture occasions (t).
2. Construct a capture frequency (f_i) for these animals as shown in the table below. In this table, i represents the number of times an animal has been (re)captured during these 18 days, while f_i represents the number of animal for each i. In this case, there are 43 animals that have been captured only once, 16 animals that have been captured twice, etc.

i	1	2	3	4	5	6	7	8
f_i	43	16	8	6	0	2	1	0

3. Then, based on these capture frequencies, use a Jackknife estimator to estimate the number of total population using the following equations:

$$N_{JK} = a(t,K)_1 f_1 + a(t,K)_2 f_2 + ... + a(t,K)_t f_t \tag{4.2}$$

where $a(t,K)_i$ are the coefficients in terms of the number of capture occasions (t) and the order of estimation (K). An optimum value of K has to determined because as K increases, the bias of N_{JK} will decrease while the variance of N_{JK} will increase. An estimated number of

animals in population along with a method to choose an optimum K are shown in the following procedure:

a. Calculate N_{JK} for $K = 1 - 5$, based on the capture frequencies in Table 1, using these equations

$$
\begin{aligned}
S &= \sum_{i=1}^{t} f_i \\
N_{J1} &= S + \frac{t-1}{t} f_1 \\
N_{J2} &= S + \frac{2t-3}{t} f_1 - \frac{(t-2)^2}{t(t-1)} f_2 \\
N_{J3} &= S + \frac{3t-6}{t} f_1 - \frac{3t^2-15t+19}{t(t-1)} f_2 \\
&\quad \frac{(t-3)^2}{t(t-1)(t-2)} f_3 \\
N_{J4} &= S + \frac{4t-10}{t} f_1 - \frac{6t^2-36t+55}{t(t-1)} f_2 \\
&\quad + \frac{4t^3-42t^2+148t-175}{t(t-1)(t-2)} f_3 - \frac{(t-4)^4)}{t(t-1)(t-2)(t-3)} f_4 \\
N_{J5} &= S + \frac{5t-15}{t} f_1 - \frac{10t^2-70t+125}{t(t-1)} f_2 \\
&\quad + \frac{10t^3-120t^2+485t-660}{t(t-1)(t-2)} f_3 - \frac{(t-4)^5-(t-5)^5}{t(t-1)(t-2)(t-3)} f_4 \\
&\quad + \frac{(t-5)^5}{t(t-1)(t-2)(t-3)(t-4)} f_5
\end{aligned}
$$

In this example, calculated N_{JK} for K = 1 to 5 are tabulated below.

Order(K)	Jackknife estimator (N_{JK})
S	76
N_{J1}	116.6
N_{J2}	141.5
N_{J3}	158.6
N_{J4}	170.3
N_{J5}	176.5

b. Compute an interpolated estimator between $m-1$ and m, where $m+1$ is the first order that the significance level $P_m > 0.05$. This calculation begins with calculating the statistic

$$T_m = \frac{N_{Jm+1} - N_{Jm}}{v\hat{a}r(N_{Jm+1} - N_{Jm}/S)^{1/2}} \tag{4.3}$$

where

$$v\hat{a}r(N_{Jm+1} - N_{Jm}/S) = \frac{S}{S-1}\left[\sum_{i=1}^{t}(b_i)^2 f_i - \frac{(N_{Jm+1} - N_{Jm})^2}{S}\right] \tag{4.4}$$

and $b_i = a(t, m+1)_i - a(t, m)_i \cdot T_m$. Then each of these statistic (T_m) will be evaluated at α = 0.05 using P_m values determined from the standard normal distribution. For the first index m that the significance level $P_m > 0.05$, if $m = 1$, N_{J1} is then taken as the estimator of the number of active flows. If $m > 1$, then compute an interpolated estimator between $m-1$ and m as $N_J = cN_{Jm} + (1-c)N_{J(m-1)}$, where

$$c = (0.05 - P_{m-1})/(1 - P_m) \tag{4.5}$$

In this example, m is calculated as three, such that the interpolation is performed on N_{J2} and N_{J3} with the resultant estimator of 142.

The example above of M_h model can be adapted to estimate the number of active flows. Instead of capturing n packets for each capture occasion (out of t capture occasions), only one packet is captured for each capture occasion for simplicity. Note that the accuracy then depends on t as n is reduced to one. The estimation process is as follows:

1. Capture t packets from a queue buffer.
2. Construct a set of capture frequency data by observing the flow ID of the captured packets.
3. Estimate the total number of active flows in the buffer using the Jackknife estimator.

The authors further modify the algorithm so that the capture is performed on the incoming packets instead of the packets from the buffer. Now assume that the estimation is performed on the number of flows having their packets in a virtual buffer of size B so each incoming packet is captured with a probability of p_{cap} for the total of t packets and store in a linked-list called *capture*

list. Hence, in this case, $p_{cap} = t/B$ and the size of the liked-list can be reduced to only t instead of B. The modified procedure is as follows.

1. Capture an incoming packet with the probability p_{cap} and store the packet in the *capture list*.
2. Construct a set of capture frequency data by observing the flow ID of the captured packets in the *capture list*.
3. Estimate the total number of active flows in the buffer using the jackknife estimator.

   Note that the accuracy of the estimation decreases with the decrease of p_{cap}.

4.2.3 CR model with First-order and Second-order Sample Coverage

Sample coverage estimator [44, 13] is an alternative estimator to the Jackknife estimator which can be used in M_h model to estimate the number of active flows. Both First-order and Second-order Sample Coverage begins with the same technique to capture packets and construct frequency data as in Section 4.2.2. After obtaining the frequency data, instead of using the Jackknife estimator, the estimation of number of active flows, $\hat{N}_{SC1}$ and $\hat{N}_{SC2}$, however continue with these equations:

- **First-order sample coverage:**

$$\hat{N}_{SC1} = \frac{S + f_1\hat{\gamma}^2}{\hat{C}_1} \tag{4.6}$$

  where $\hat{C}_1 = 1 - \frac{f_1}{t}$ and

$$\hat{\gamma}^2 = \max\left[\frac{S}{\hat{C}_1}\sum_{j=2}^{t}\frac{j(j-1)f_j}{t(t-1)}, 0\right]. \tag{4.7}$$

- **Second-order sample coverage:**

$$\hat{N}_{SC2} = \frac{S + f_1\hat{\gamma}^2}{\hat{C}_2} \tag{4.8}$$

  where $\hat{C}_2 = 1 - \frac{(f_1 - 2f_2/(t-1))}{t}$ and

$$\hat{\gamma}^2 = \max\left[\frac{S}{\hat{C}_2}\sum_{j=2}^{t}\frac{j(j-1)f_j}{t(t-1)}, 0\right]. \tag{4.9}$$

4.3 EVALUATION OF ALTERNATIVE METHODS

Alternative methods for an estimation of the number of active flows are evaluated through a simulation as described in this section.

A simulation is setup with a ten-node dumbbell topology as shown in Figure 34. There are n TCP flows competing for the bandwidth of 10-Mbps bottleneck bandwidth. All the access links are 2ms and the bottleneck link is 5ms in delay. The queueing discipline is simply tail dropping.

Four alternative methods for an estimation of the number of active flows are compared:

1. Direct Bitmap
2. CR Model with Jackknife estimator
3. CR Model with First-order Sample Coverage
4. CR Model with Second-order Sample Coverage

with BLACK estimation also shown in the results for comparison.

Due to the fluctuation of the estimation using CR-model as shown in Figure 41, the estimated number of active flows is further averaged over time with a factor of 0.3 to smooth out these values.

At the end, computation complexities of these methods are briefly discussed.

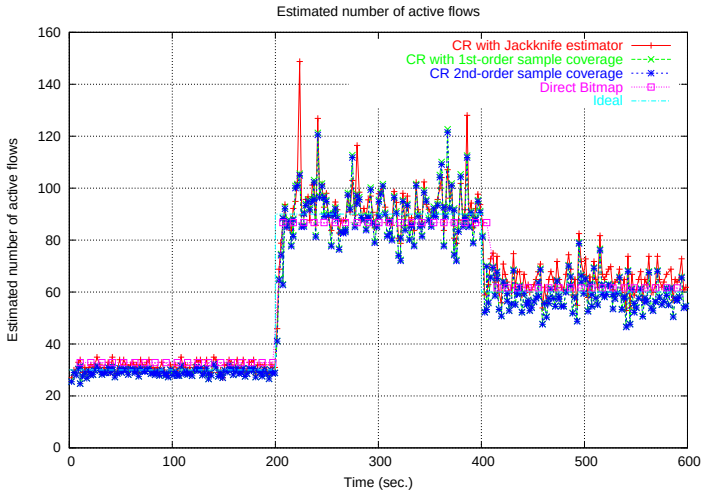


Figure 41: Estimated number of active flows for the experiment with 30, 90, and 60 flows over 600 seconds.

4.3.1 Different number of TCP flows with about the same traffic intensity

In this section, n TCP flows with very large received window are competing for the bandwidth at the bottleneck link. At the simulation time from 0 - 200 seconds, only 1/3 of total connections are active. For 200 - 400 seconds, all of the n connections are active. And from 400 - 600 seconds of simulation time, 1/3 of the total connections are terminated leaving 2/3 of connections being active.

Memory space for CR model estimation are set to t = 100 capture records. For Direct Bitmap, b = 100 bits are reserved as a bit map for the estimation.

The performance of the estimation methods are evaluated for n of less than 100, higher than 100, and much higher than 100 as shown in Table 9.

Set	Number of TCP flows at simulation time 0-200, 200-400, 400-600 seconds
1	30, 90, 60
2	60, 180, 120
3	150, 450, 300

Table 9: Number of TCP traffic in different simulation sets.

The results from Figure 42, 43, and 44 show that all of these methods perform well, when the number of actual flows are not much larger than the required memory space – that is when $t \leq n$ and $b \leq n$. The estimation begins to be unstable when t and b are larger than n, and the results are unpredictable when $t \ll n$ and $b \ll n$. Several other sets of simulations were also conducted and show a similar trend.

The result of Direct Bitmap is consistent with [15, 16] that Direct Bitmap requires a bitmap size (b) that scales almost linearly with the number of flows (n) in order to get an accurate estimation of number of active flows ($\hat{n}$), where the average error is bounded to Standard Deviation$(\hat{n})/n = (\sqrt{b}/n)\sqrt{e^{n/b} - 1}$. This bound is tighter with large n which implies a much lower estimation error when the number of flows is large, when b of about the same order of magnitude as n.

For CR model, there is no significant different between the Jackknife estimator and Sample Coverage in this case, where both estimators require large t for accurate estimation. A series of

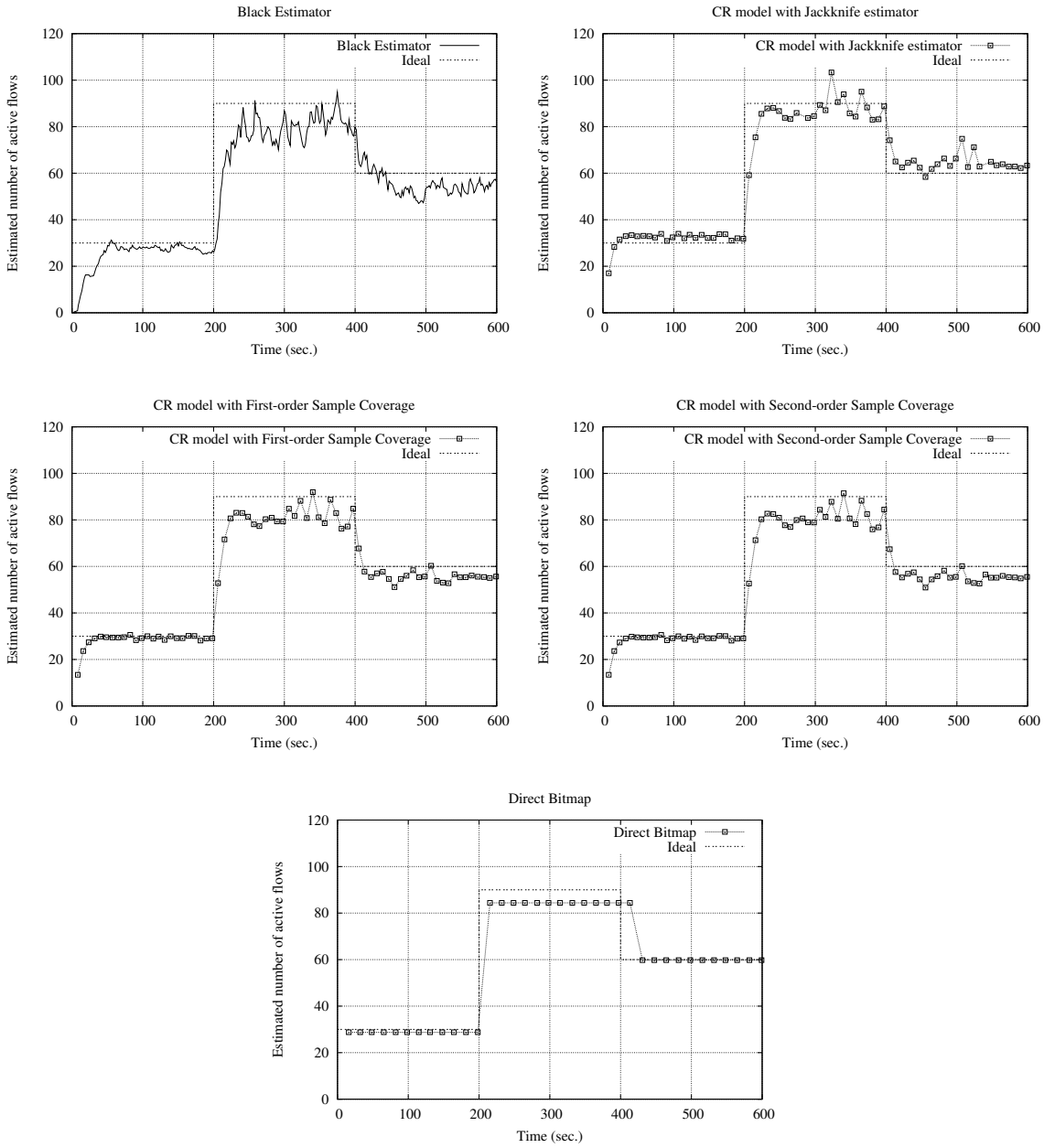


Figure 42: Estimated number of active flows for the experiment with 30, 90, and 60 TCP flows over 600 seconds.

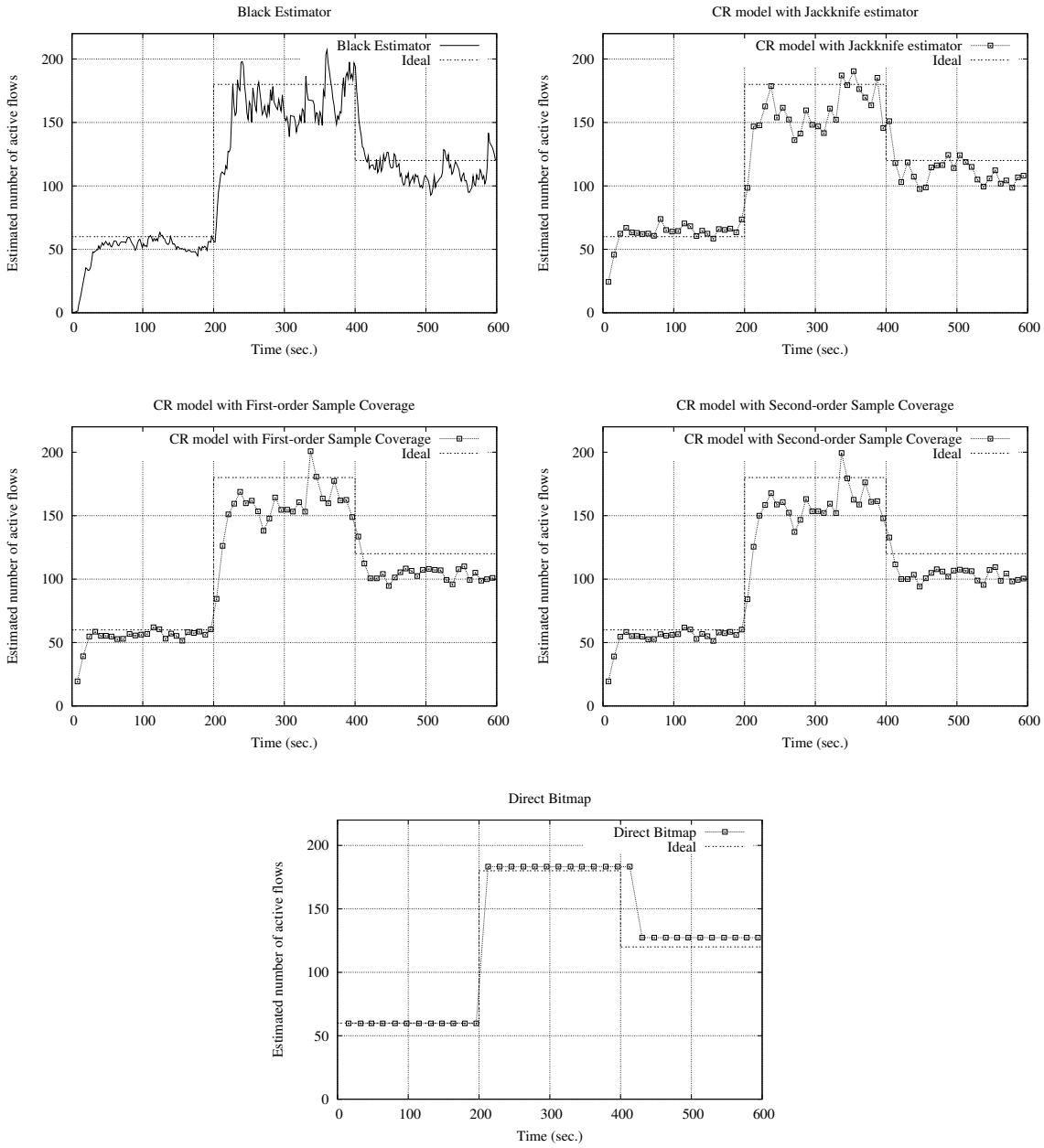


Figure 43: Estimated number of active flows for the experiment with 60, 180, and 120 TCP flows over 600 seconds.

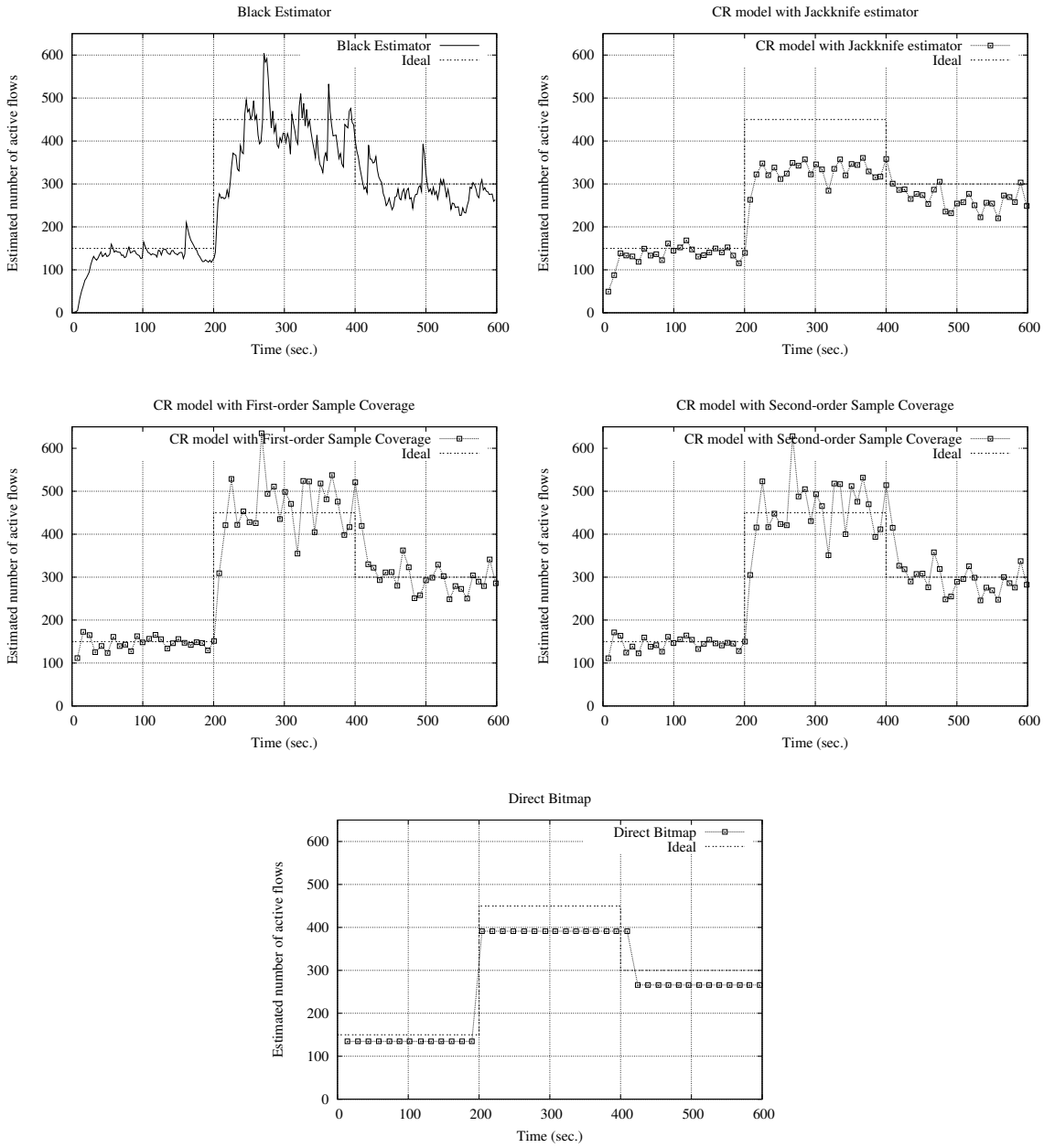


Figure 44: Estimated number of active flows for the experiment with 150, 450, and 300 TCP flows over 600 seconds.

extensive experiment in [36] also shows that a performance could be improved with higher t/n and recommends t to be relatively large compared to n. As a result, the memory required is quite high to have an accurate estimation.

If $t \ll n$, the performance is unpredictable and this also happens with Direct Bitmap. However, CR model needs more memory space than Direct Bitmap as each captured record stores a flow ID which could be a combination of source address, destination addresses, source port and destination port, while Direct Bitmap needs only one bit for an incoming flow to hash into.

4.3.2 Different number of TCP flows with different round-trip delay

To see any effect as a result of TCP with different round-trip delay to the accuracy of estimation, the topology in Figure 45 is used instead. The set of experiment is the same as shown in Table 9 in the previous section.

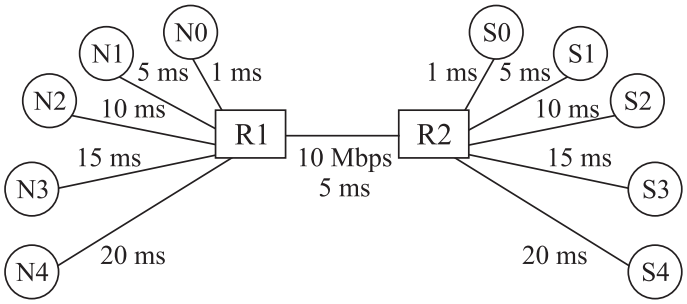


Figure 45: Ten-node dumbbell Topology with different access links' delay.

As can be seen from Figure 46, 47, and 48, when the n is not much larger than t or b all of these methods produce quite a good estimation. More variance is more noticeable when n becomes much larger than t in the case of CR model. With appropriate size of t and n, both Direct Bitmap and CR model show no problem with different traffic intensity as BLACK estimation as demonstrated in Section 4.1.2.

4.3.3 Different number of TCP flows with large UDP flow

A big difference in estimation accuracy appears when UDP traffic is presented. For comparison, the ten-node topology with 10-Mbps bottleneck link as used in Section 4.3.1, or as shown in Figure

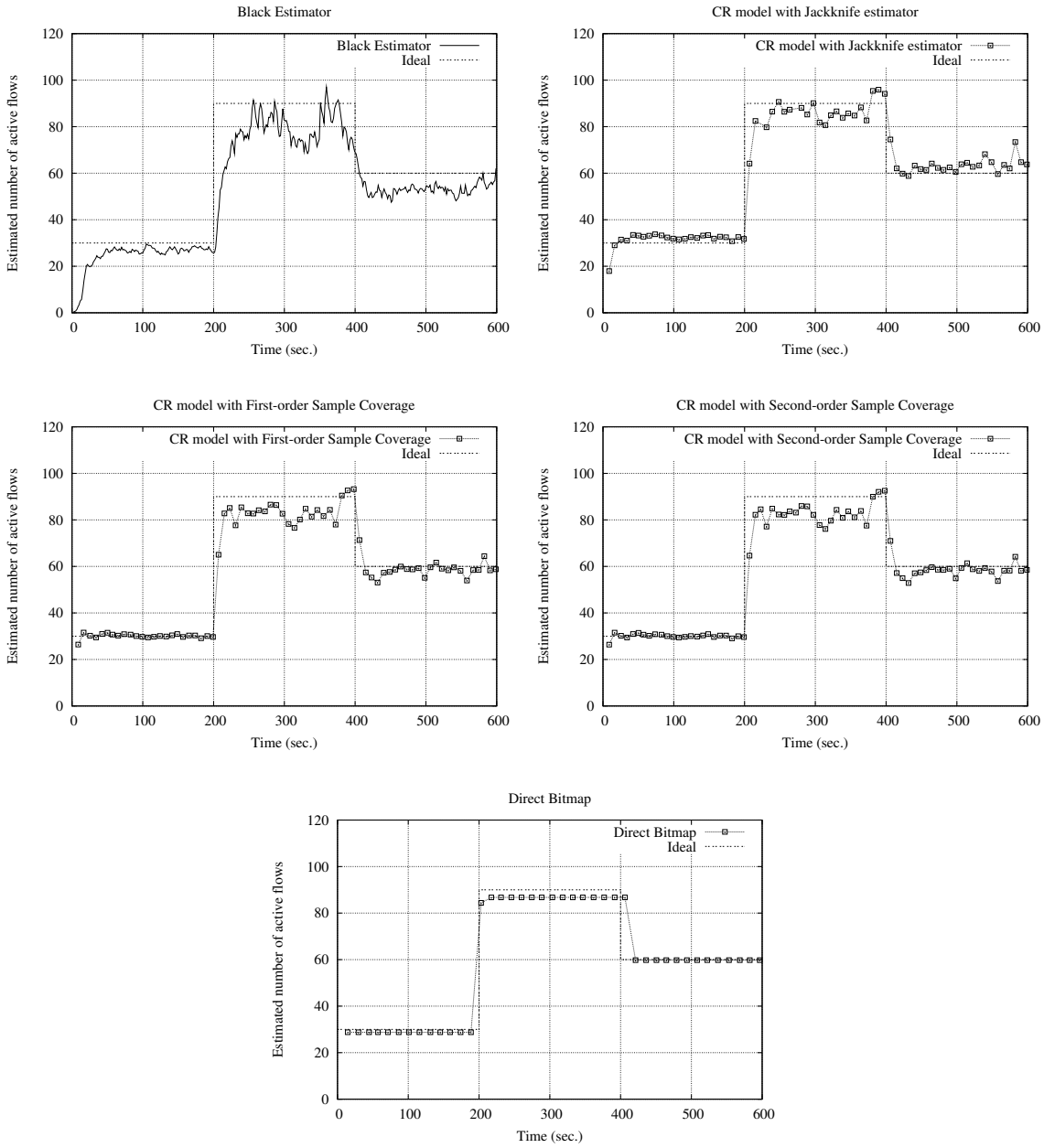


Figure 46: Estimated number of active flows for the experiment with 30, 90, and 60 TCP flows with different RTT over 600 seconds.

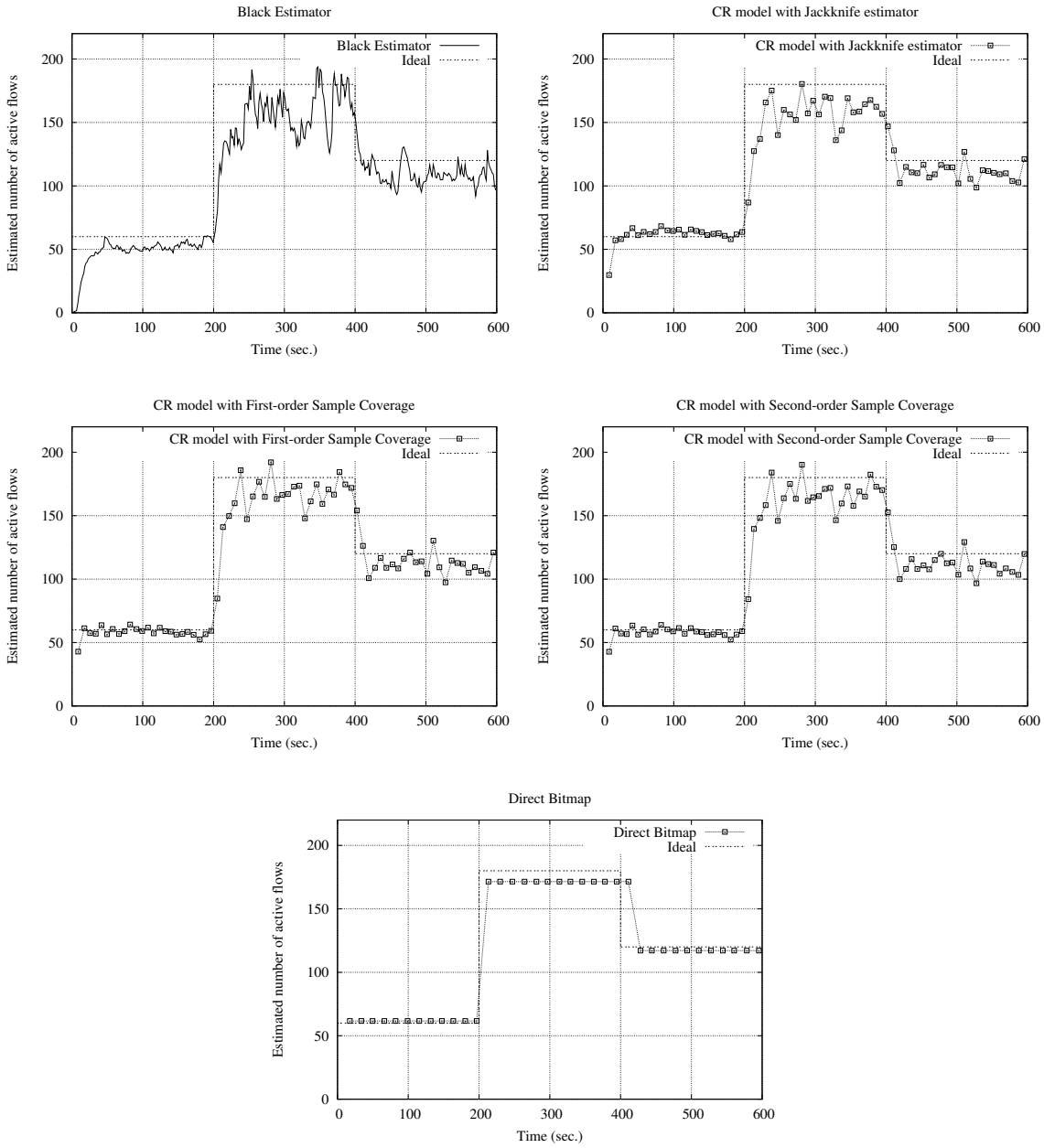


Figure 47: Estimated number of active flows for the experiment with 60, 180, and 120 TCP flows with different RTT over 600 seconds.

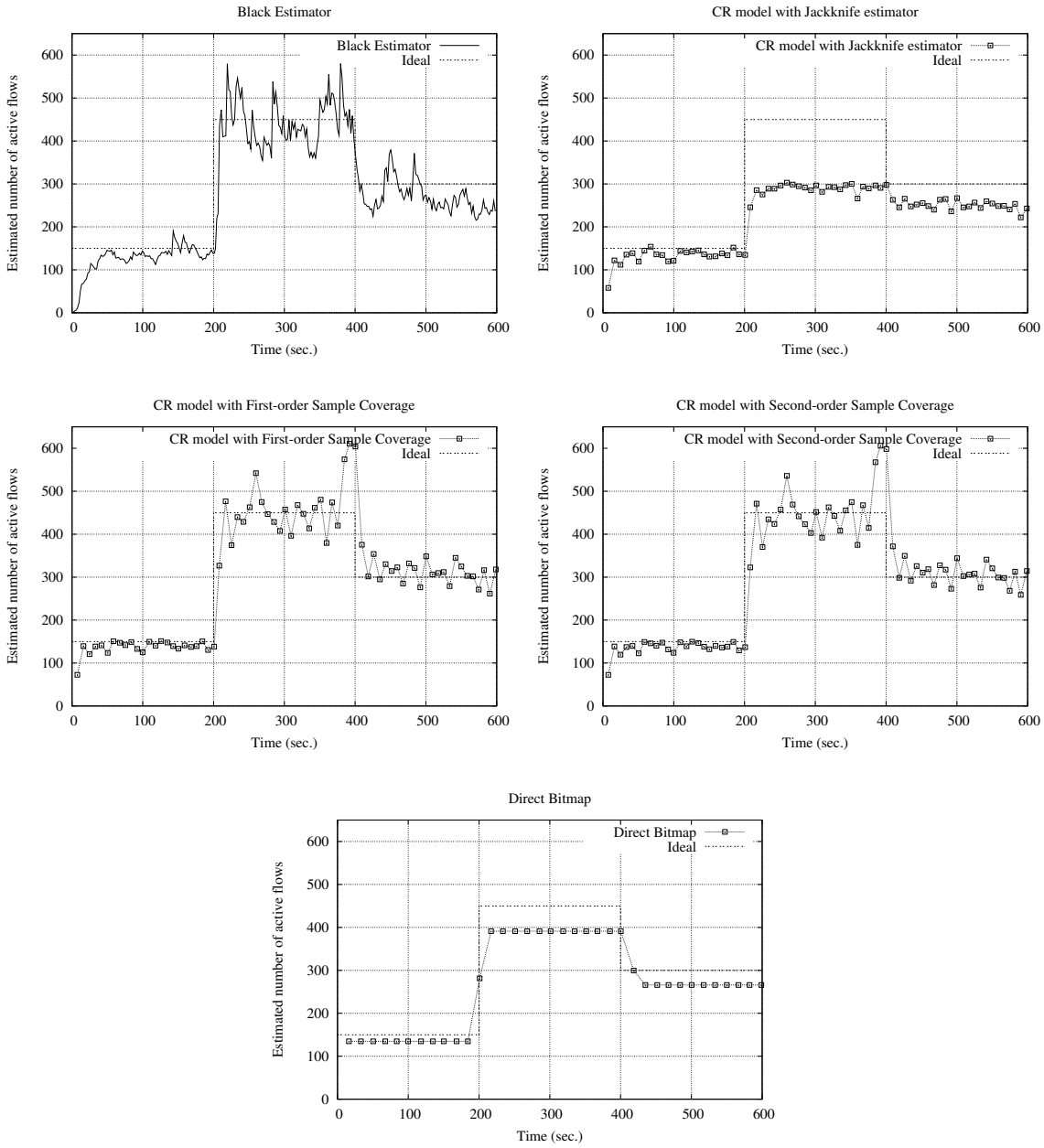


Figure 48: Estimated number of active flows for the experiment with 150, 450, and 300 TCP flows with different RTT over 600 seconds.

34, is maintained but with UDP running as a background traffic. Because the effect of UDP traffic is the only concern, so the number of TCP traffic is set not to be too large for $t = 100$ records in CR model and $b = 100$ bits in Direct Bitmap, with two different arrival rates of UDP traffic as shown in Table 10. UDP traffic start at 300 seconds of simulation time.

Set	Number of TCP flows at simulation time 0-200, 200-400, 400-600 seconds	UDP arrival rate
1	30, 90, 60	33% of bottleneck link bandwidth
2	30, 90, 60	66% of bottleneck link bandwidth

Table 10: Number of TCP traffic and UDP arrival rate in two simulation sets.

In Figure 49, even when UDP arrival rate is equal to 33% of the bottleneck link bandwidth, Sample Coverage show an extreme over-estimation of the number of active flows at after 300 seconds, which is contrasting to an extreme under-estimation from BLACK as explained in Section 4.1.2. On the other hand, both Jackknife estimator and Direct Bitmap are torelant to the interference of UDP background traffic.

For Sample Coverage, j-order Sample Coverage estimated N is

$$\hat{N}_{SCj} = \frac{m}{\hat{C}_j} + \frac{f_1}{\hat{C}_j}\hat{\gamma}^2 \qquad ; j = 1 \text{ or } 2$$

where $m/\hat{C}_j$ is an initial estimator that assumes n flows with equal traffic intensity. A bias correction term $(f_1\hat{\gamma}^2/\hat{C}_j)$ that increases with heterogeneity is added for a more accuracy when traffic intensity are unequal. In fact, $\hat{\gamma}$ is an estimated coefficient of variation (C.O.V.) of the probability **p** = $(p_1, p_2, p_3, ..., p_n)$, where p_i is a probability that flow i's packet being caught. The estimation N_{SCj} works well with small actual C.O.V. and provides less error than the Jackknife estimation as reported in [44]. However, when C.O.V. becomes large an inaccuracy appears and increases with C.O.V, as can be observed from Equation 4.7 and 4.9 that $\hat{\gamma}$ is calculated based on $\sum_{j=1}^{n} j(j-1)f_j$. In this experimental environment, C.O.V. of **p** is very large, i.e. 6.3335 for 60 identical TCP traffic and one UDP traffic consuming 33% of bottleneck link bandwidth, resulting in a large margin of error.

For another set of experiment, once the UDP arrival rate becomes 66% of the bottleneck link bandwidth, the Jackknife estimator can no longer provide an accurate estimation as illustrated in Figure 50. However, a better estimation from the Jackknife estimator can be obtained only if t is larger, however. Besides, in practice, it is a very rare chance that any one or two traffic would consume as much as 66% of an actual link bandwidth. On the other hand, direct Bitmap shows a superior performance even with a presence of large UDP traffic because a bit that a flow hashes to is irrelevant to the number of packets injected to the network by this flow.

4.3.4 Presence of short-lived background traffic

Performance of different estimation methods under a scenario with a presence of short-lived background traffic are evaluated through the same ten-node topology with 10-Mbps bottleneck link in Figure 34, but with five web-client and web-server nodes. Total of 250 HTTP sessions are setup as background traffic according to a parameter setting . Number of (long-term) TCP traffic are set according to set 1 in Table 10 – 30 flows at 0 - 200 seconds, 90 flows at 200 - 400 seconds, and 60 flows at 400 - 600 seconds of simulation time.

The results in Figure 51 show fairly good estimation for the number of long-term TCP traffic for the Jackknife estimator and both Sample Coverage methods. However, Direct Bitmap does not perform as well because it was designed to estimate the number of *all* active flows including even a flow with one packet arrived at the queue during the estimating period. Besides, additional error is also contributed from a small bitmap size b, which is set to 100, comparing to the number of all traffic (TCP and short-lived background traffic).

The problem of Direct Bitmap in estimating the number of long-term traffic can be solved with a trade off of larger memory space required. Instead of arranging a memory space as a bitmap of size b bits, a modified version of this scheme arranged memory space into b hash items, where each of them holds one small-size integer as a counter. Two alternative methods can be applied:

- **Direct Bitmap with low-pass filter:** A flow ID of an incoming packet hashes into an item in this memory space and increase its integer value by one. In this way, each hash item records the number of packets from a flow during the estimating period. Those short-lived flows can be filtered out by setting a minimum threshold of d packets on each counter. After an estimating

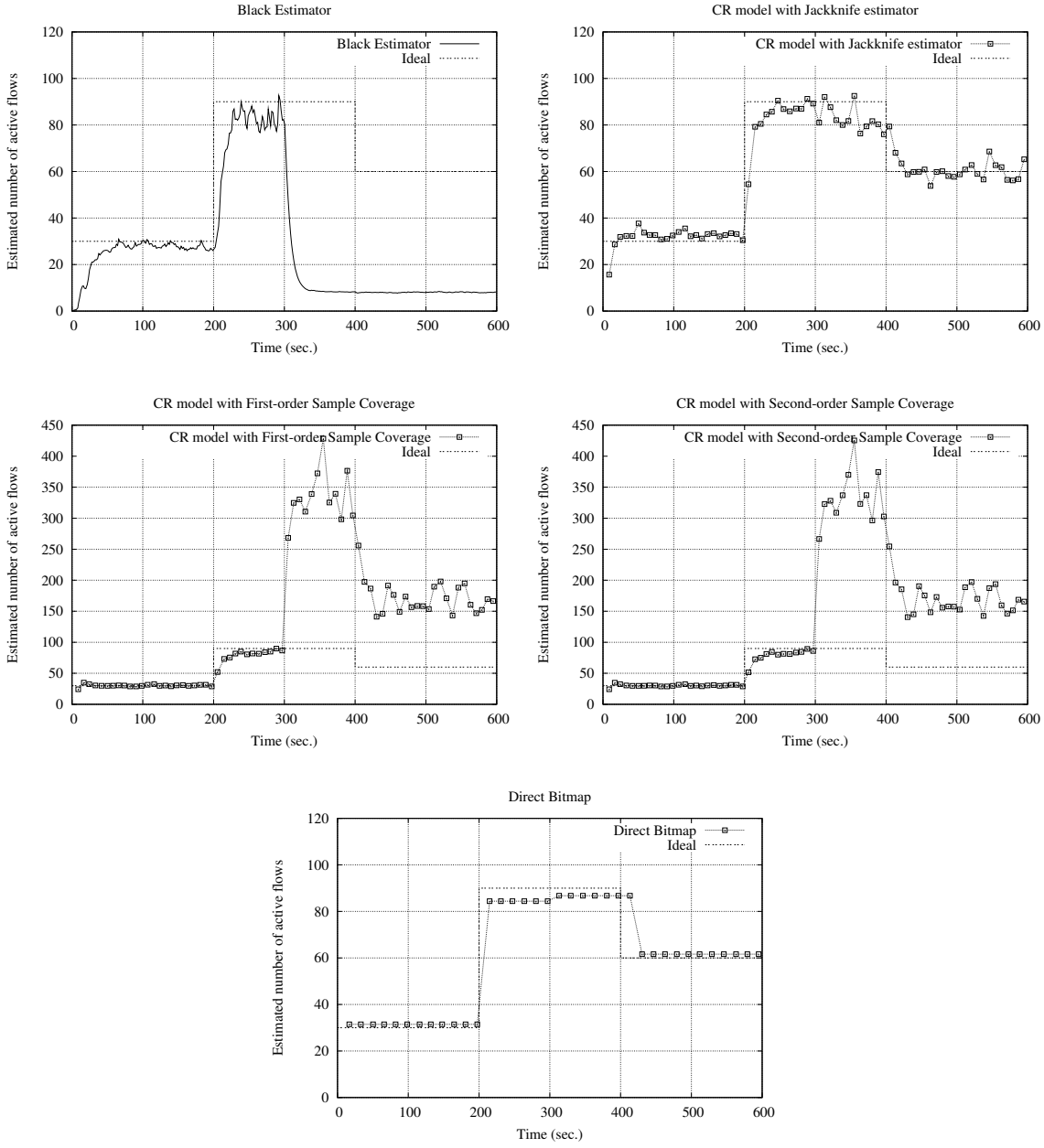


Figure 49: Estimated number of active flows for the experiment with 30, 90, and 60 TCP flows over 600 seconds, with UDP traffic consuming 33% of bottleneck link bandwidth starting at 300 seconds.

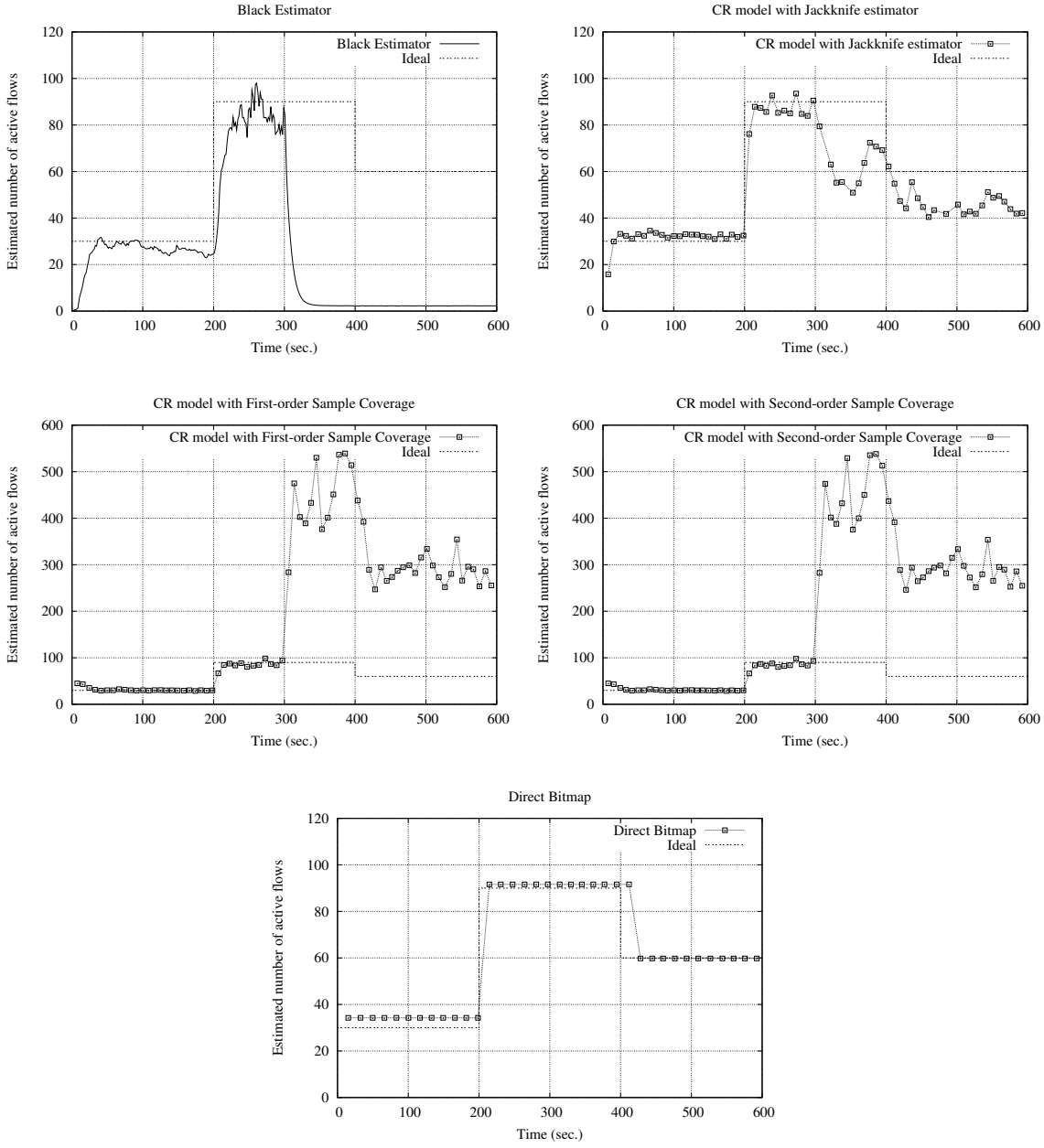


Figure 50: Estimated number of active flows for the experiment with 30, 90, and 60 TCP flows over 600 seconds, with UDP traffic consuming 66% of bottleneck link bandwidth starting at 300 seconds.

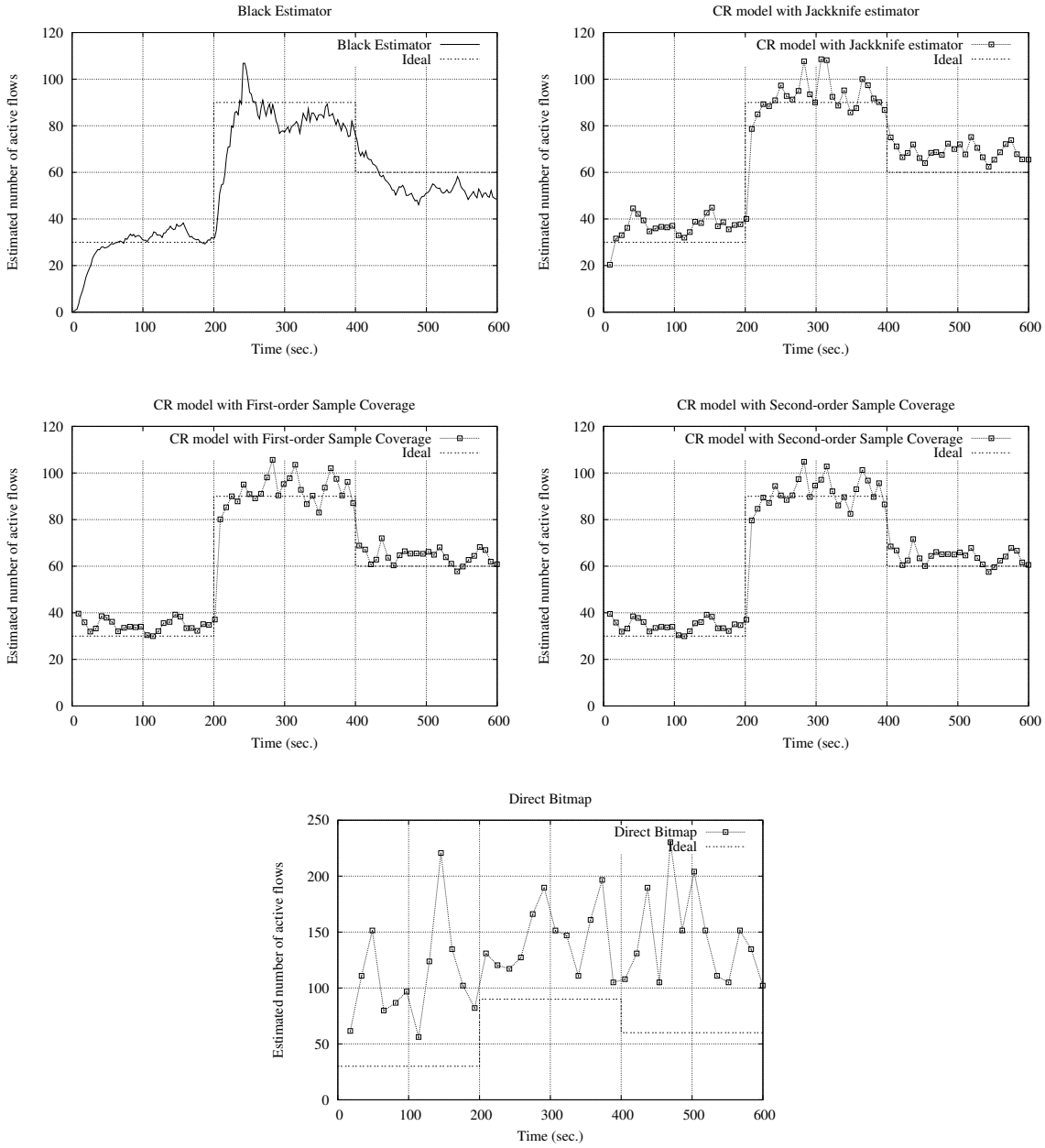


Figure 51: Estimated number of active flows for the experiment with 30, 90, and 60 TCP flows over 600 seconds, with 250 HTTP sessions as background traffic.

period, z', the number of hash items that holds the number of packets more than d packets, is counted. The value of z' is nearly equivalent to the number of 1 bits in the original Direct Bitmap method. Therefore, z is simply b minus by z', and the estimated number of active flows can be determined using the same equation, $b\ln(\frac{b}{z})$.

- **Direct Bitmap with packet removals:** In the same way, a flow ID of an incoming packet hashes into a hash item and increase its counter by one. However, when a packet is removed from the queue, the counter is decremented. If a queue is empty, all counters hold a value of zero. In this manner, the flows that have their life time shorter than an estimating period would be filtered out automatically. Only the zero bits are counted as z and the estimated number of active flows is simply $b\ln(\frac{b}{z})$ as the other methods. This approach is an extended version by the authors from the original Direct Bitmap paper [15, 16].

The results for both methods, illustrated in Figure 52 and 53 respectively, show a huge improvement as opposed to the performance of original Direct Bitmap in this circumstance shown in Figure 51. Although setting a threshold of d packets to decide which flows are long-lived traffic is somehow not difficult, Direct Bitmap with packet removals is easier to deploy because it involves no additional parameter tuning. However, the packet removals approach may have a little higher variance due to the backlogging packets in the queue from short-lived traffic at the time of estimation, especially when the queue is large as shown in Figure 54. Nonetheless, this problem should be minimized if a router is equipped with an active queue management that is usually designed to keep an average queue size low to achieve low delay.

4.3.5 Algorithm complexity

Since all of the estimation methods perform an estimation periodically, algorithm complexity can be considered based on per-packet arrival processing and estimation processing.

1. **Direct Bitmap:** For each incoming packet, perform a flow ID hashing to a bitmap. At the end of estimating period, count the number of zero bits (z) and determine the number of active flows from $b\ln(b/z)$.
2. **CR Model:** For each incoming packet, record a flow ID with a probability p_{cap}. After t flow ID have been recorded, construct frequency data (how many distinct flow IDs have seen once,

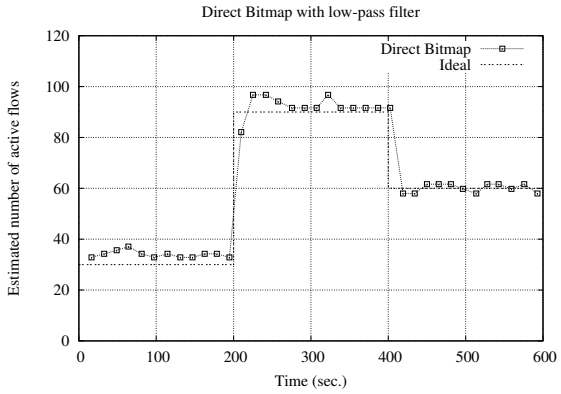


Figure 52: Estimated number of active flows using Direct Bitmap with low-pass filter for the experiment with 30, 90, and 60 TCP flows over 600 seconds, with 250 HTTP sessions as background traffic.

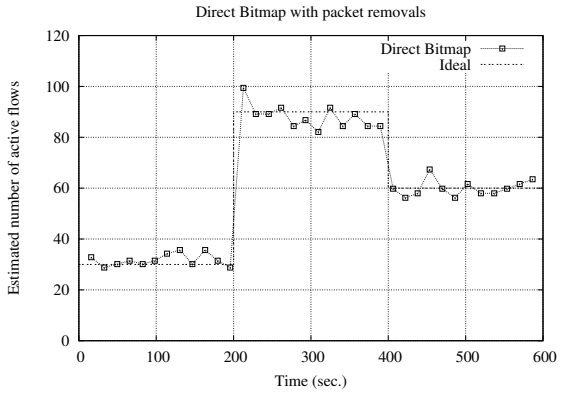


Figure 53: Estimated number of active flows Direct Bitmap with packet removals for the experiment with 30, 90, and 60 TCP flows over 600 seconds, with 250 HTTP sessions as background traffic. Max. buffer size = 300 packets.

twice, and up to t times). Then perform a series of calculation using either the Jackknife estimator or Sample coverage as discussed in Section 4.2.2 and 4.2.3.

It is clear that CR model consumes much higher processing power since the first step of constructing frequency data. This complexity increases as a size of capture list (t) becomes larger, which is needed if the number of actual flows is large. Although the estimation can be performed as a background task, it is still not as suitable as Direct Bitmap for a high speed router with very large number of flows passing through.

4.4 SUMMARY

According to the evaluation results through a series of simulation experiment, with a consideration of algorithm complexity, it is clear that Direct Bitmap requires least memory space with low computational complexity. For CR model, the Jackknife estimator is more robust than Sample Coverage in different scenarios, especially when a C.O.V. is high. Direct Bitmap does not suffer with a presence of large UDP traffic but may have a problem when there is a large number of short-lived background traffic as all of these flows are taken into account instead of only those long-lived ones. In this case, Direct Bitmap with low-pass filter or Direct Bitmap with packet removals can be used to filter out those small flows from the estimation, with the expense of higher memory usage. In the present days, however, memory becomes cheaper. Thus, the memory requirement of both modified versions of Direct Bitmap would not be a problem. In some cases that memory might be scarce, the amount of memory required can be dramatically reduced with variants of Direct Bitmap such as Virtual Bitmap, Multiresolution Bitmap, Adaptive Bitmap, and Triggered Bitmap.

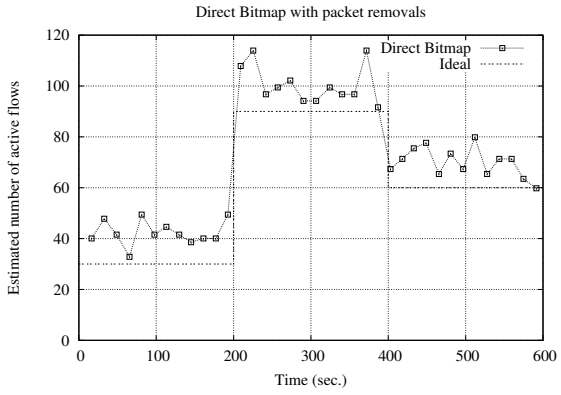


Figure 54: Estimated number of active flows Direct Bitmap with packet removals for the experiment with 30, 90, and 60 TCP flows over 600 seconds, with 250 HTTP sessions as background traffic. Max. buffer size = 1000 packets.

5.0 AFC: ACHIEVING FAIRNESS USING A CREDIT-BASED MECHANISM

At the end of Chapter 3.3, the limitations of BLACK mechanism were addressed – unfairness due to inaccuracies in the estimation of the number of active flows, unfairness when packets are of different sizes, and fluctuations (high variance) in the throughput of the flows. These problems, which might cause a serious degradation of fairness performance in some circumstances, lead to a development of new a scheme proposed in this chapter called *AFC, Achieving Fairness using a Credit-based mechanism*. Although sharing several conceptual ideas as BLACK, AFC contains several newly designed components. Apart from using Direct Bitmap, a more accurate method to estimate the number of active flows discussed in Chapter 4, AFC includes a new design of HitFraction approximation, dropping function, and a new *credit-based mechanism*. AFC not only overcomes these limitations but also provides a better fairness performance in a wide rage of scenarios, however, with some overhead expense over BLACK.

Also included in this chapter is the comparative evaluation of AFC, for its fairness performance and robustness, with the other fair AQM schemes appeared in Chapter 3 i.e. RED, CHOKe, SFB, CARE, and BLACK. The simulation scenarios are expanded to new cases with traffic with different packet sizes, short-flows, and TCP-friendly traffic. In these experiments, BLACK is also equipped with Direct Bitmap rather than its original estimation of the number of active flows, for a fair comparison. The performance metrics are average per-flow UDP and TCP throughput and fairness among TCP traffic. At the end, complexity of these schemes are briefly discussed which shows that AFC has a higher overhead than BLACK and it is a choice of a network operator to choose BLACK or AFC for implementation.

This chapter is organized as follows. The first half of this chapter, covered in Section 5.1, reviews the limitations of BLACK and discusses the solutions proposed by AFC along with its components. Then, the performance evaluation of AFC along with the other fair AQM schemes

are given in Section 5.2. The chapter summary is provided in Section 5.4.

5.1 AFC MECHANISM

Although BLACK has shown the promising results through a series of simulation experiments in providing throughput fairness in Chapter 3.3, there are still some limitations of BLACK as described in Section 3.4 as summarized below. In addition, because these limitations are not unique to BLACK, the AQM schemes that contain similar limitations are also discussed here.

1. *Inaccuracy in the estimation of the number of active flows:* This problem comes from the fact that simplistic assumptions are made to compromise the performance of the algorithms. For example, this is the case of BLACK, which bases the estimation on a simple probability model that assumes an equal traffic intensity of the incoming flows. As a result, the estimation could be inaccurate and high bandwidth unresponsive flows could be out of control. The estimation mechanism utilized in CARE, even with a complex model, has also been found to fail under certain scenarios especially with a presence of a traffic whose arrival rate is relatively large comparing to a link bandwidth.
2. *Unfairness due to traffic with different packet sizes:* Most performance evaluations of fair AQM schemes have been performed assuming packets of equal sizes and several schemes fail to achieve their promised goals otherwise. For instance, this is the case of BLACK, CHOKe and CARE. BLACK collects the information of each sample packet at packet level, not byte level. Smaller packets have a higher possibility to be sampled from the queue than larger packets, which may appear to have higher buffer occupancy fraction even through the number of bytes are equal, and over-penalizing could occur. In the case of CHOKe, it is clear that a flow with smaller packet sizes would have more number of packet matchings than a flow with larger packet sizes, given that they have the same arrival rate, and would be penalized more. Also for CARE, the mechanism only takes into account the number of packets regardless of the packet sizes and thus fairness in this case would not occur.
3. *Throughput fluctuation:* No performance evaluation of fair AQM schemes has looked at the traffic characteristics of the flows after passing through the scheme. Not only BLACK, as

it will be shown later, most schemes make the throughput of CBR flows to fluctuate more after passing through them. This behavior is not preferable especially by streaming media applications.

As a result, AFC, a new fair AQM scheme that addresses the limitations of the well-known schemes listed above is proposed, and discussed in the following sections. Although AFC shares similar conceptual idea as BLACK, it contains several newly designed components and incorporates a new concept, *Credit-based Mechanism*, which would enhance the fairness achieved by the scheme. These components are explained as follows.

5.1.1 The estimation of the number of active flows

The solution for the first problem has been addressed along with the comparative evaluation in the previous chapter where *Direct Bitmap* is chosen as the mechanism to estimate the number of active flows due to its computational simplicity and very small amount of memory requirement. The simulation results in various types of scenarios show that Direct Bitmap provides a high accuracy for the estimation with the computational complexity that is far less than the other schemes. There are also several variants of Direct Bitmap such as *Virtual Bitmap*, *Multiresolution Bitmap*, *Adaptive Bitmap*, and *Triggered Bitmap* which require much less memory space while keeping low estimating error; for example 2 Kbytes of memory is needed for Adaptive Bitmap to estimate the number of active flows up to 100 million flows with an average error of less than 1% [15, 16]. However, only Direct Bitmap is used because of its lower complexity. The other variants are left as a choice of the service providers if smaller amounts of memory are preferred at the expense of a little higher complexity.

5.1.2 Handling traffic with different packet sizes

Most fair AQM schemes fail to provide fairness when flows send packets of different sizes. For example, this is the case of BLACK since it computes a flow's buffer fraction based on the number of packets of that flow over the number of total sampled packets, rather than a byte count. To solve this problem, instead of counting the number of packets for the candidate flows in the cache memory, the information is updated with the size of the sampled packet.

In details, rather than increasing the Hit variable by one each time the flow ID of the sampled packet is found in the HBF cache memory, the Hit variable ($\hat{h}$) is updated with the size of the sampled packet. Furthermore, the total number of bytes are counted in each period rather than the number of packets being sampled (m). In this way, both $\hat{h}$ and m have a unit of byte. At the end of a sampling period, the *HitFraction* of a flow is calculated by $\hat{h}$ (in bytes) divided by the number of bytes being sampled (m in bytes). The *HitFraction* obtained using the byte count then has no problem even when the passing traffic have different packet sizes.

Note that CHOKe cannot prevent this problem without breaking a large packet into smaller packets about the same size as suggested by the authors of CHOKe in [54]. As CHOKe relies purely on packets matching, between a packet that is sampled from the queue and the arriving packet, a flow that has smaller packet size would be penalized more with the higher probability of matching.

5.1.3 Reducing throughput fluctuation

The trace of the results from the experimentation set in Chapter 3.3 indicates that throughput fluctuation under BLACK occurs mainly from a dropping function and adds up by a sampling error. A sampling error causes an inaccuracy in $HitFraction$ approximation and thus over- or under-penalization of high-bandwidth unresponsive flows from time to time. This issue will be explained in this section while the throughput fluctuation due to a dropping function will be covered in the next section.

In BLACK, packets are sampled from a queue buffer, using packet arrival as a trigger event for each sampling. After m sampled packets, we can estimate a fraction of packets from a particular flow occupying a buffer space, which referred to as $HitFraction$. However, it is possible that a sample size (m) could be greater than a maximum buffer size (B) to collect enough statistics for the estimation. In this way, a $HitFraction$ could be interpreted as a fraction of packets occupied in a virtual buffer of size m, as shown in Figure 55.

However, the way BLACK samples packets does not always resemble the idea of a sampling from a virtual queue. BLACK samples one packet as triggered by an arrival of a packet, no matter whether that arriving packet will be dropped or enqueued. At the advent of congestion, an ag-

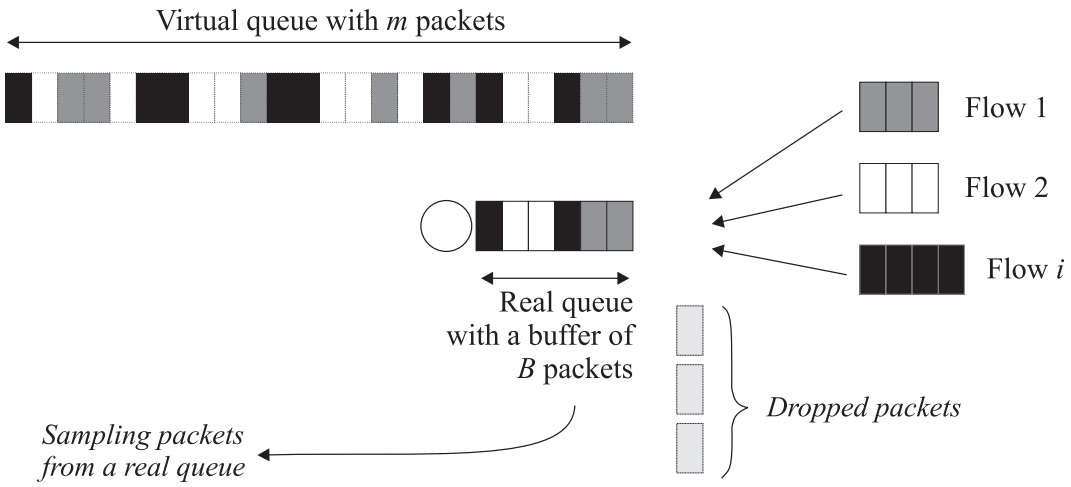


Figure 55: Virtual buffer idea to estimate a $HitFraction$.

gregate arrival rate might be high, and so a high level of packet drops, which is different from a serving rate. While packets are backlogging in the buffer, it is possible that the high sampling rate, due to high packet arrival rate, may cause the same packet(s) to be sampled more than once.

To reduce this possible error, AFC directly collects the $HitFraction$ statistics from the packets that are enqueued and tread them in the same way as sampled packets in BLACK. After a sampling period, a $HitFraction$ of each flow could be determined using the same idea of sampling packets from a virtual queue. Furthermore, since the statistics is now collected in byte according to Section 5.1.2, the variable m becomes the aggregate byte count of all packets entering the queue. In BLACK, a guide line of the value of m is 3,000 in a unit of packets for a single period. Thus, for AFC, the value of m becomes $3,000 \times B$ where B is an average packet size in byte. The average packet size B is normally between 576 bytes - 1,500 bytes for the connections that tend to generate long term traffic or most bytes to the network. It is, however, up to the service providers to choose a value of B as they can directly measure an average packet size for the traffic that pass through their network.

In this manner, the components of AFC along with a flow path of packets that pass through the queue is illustrated in Figure 56.

In addition, this new sampling method decreases the complexity in two ways. First, randomly

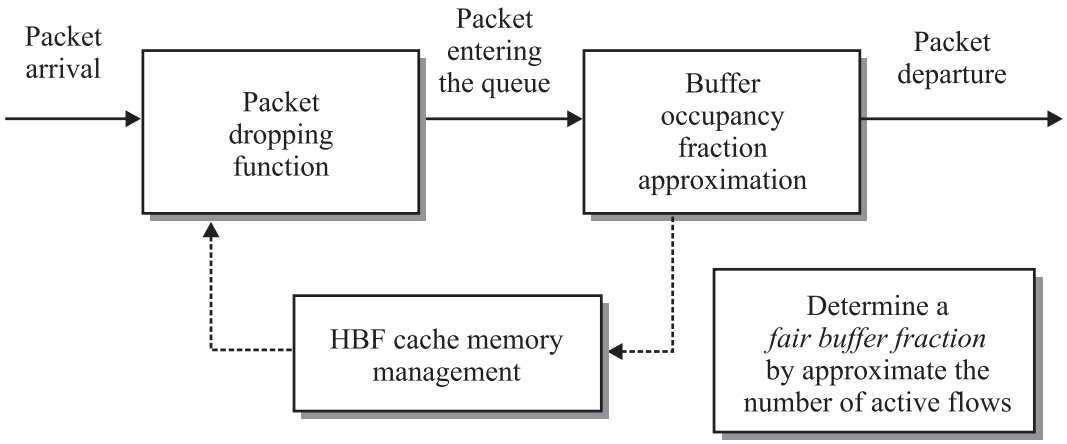


Figure 56: AFC components and a flow of packets that pass through the queue (shown in a solid arrow line).

sampling packet from the queue is no longer required as the information is directly collected from the packet that is enqueued. Second, the update frequency tends to be less because AFC collects the statistics only when there is a packet enqueued excluding those dropped packets, unlike BLACK that the statistics are collected per each packet arrival.

5.1.4 Dropping function

In BLACK, packets are dropped by the percentage of an extra buffer space the flow occupying more than a fair share, or as shown in a form of

$$p_{drop} = \frac{HitFraction - FairFraction}{FairFraction} \tag{5.1}$$

according to Equation 3.2.

Thorough experiments show that even though the exact number of active flows is known, which yields a perfect value of a $FairFraction$, unresponsive flows could still achieve somewhat higher bandwidth than a fair share. This problem could be explained through a following example.

For simplicity, assume an unresponsive flow i is a CBR traffic feeding its packets to the queue. If the current $HitFraction$ of this flow ($HitFraction_i$) is about the same as a $FairFraction$,

according to Equation 5.1 the dropping probability is zero. Now, if the $HitFraction_i$ becomes $1.25 \times FairFraction$, the dropping probability turns to be 0.25. That means 25% of the incoming packets from flow i would be dropped. When the $HitFraction_i$ becomes $1.5 \times FairFraction$, half of the incoming packets would be dropped. And when the $HitFraction_i$ becomes twice the $FairFraction$ or more, all of the incoming packets are dropped. This behavior is illustrated through a dropping probability showing in Figure 57.

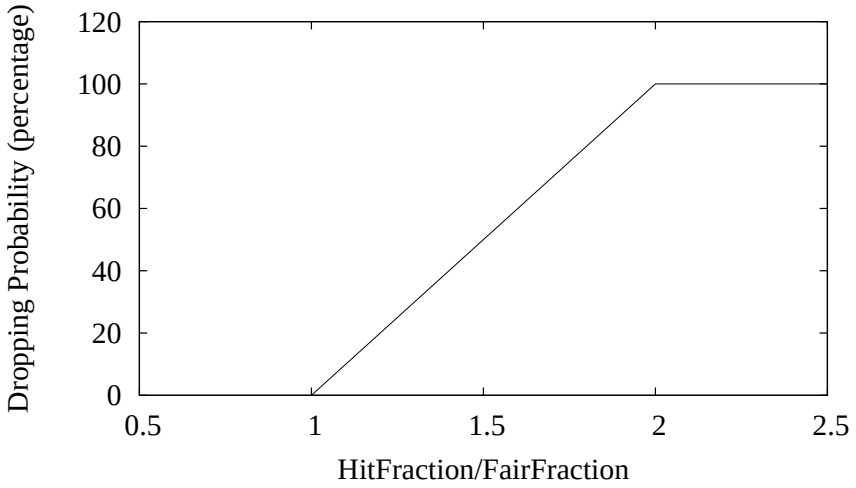


Figure 57: Dropping probability as a function of $HitFraction/FairFraction$ of BLACK.

Now, suppose the $HitFraction_i$ is $1.5 \times FairFraction$, which means that flow i is currently having more number of packets occupying the (virtual) queue than a fair share by as much as 50%. Intuitively, packets from flow i should be prevented from entering the queue for a while so that the queue could drain these extra 50% packets. However, because BLACK drops new incoming packets with a dropping probability of 0.5, at least 50% of the incoming packets would still occupy the buffer, whether or not the queue has drained the extra packets from the previous sample period yet.

If the queue is not capable of draining the old extra packets and the new extra packets, the new $HitFraction_i$ would go beyond $1.5 \times FairFraction$ iteratively and end up at twice the $FairFraction$. At this point, no more packets are allowed to be enqueued, and the real draining of the cumulative extra packets occurs here. After a short while, the $HitFraction_i$ would come down to about the $FairFraction$, and a new period of this fluctuation will continue.

It could be observed that this fluctuating behavior results in

1. By average, unresponsive CBR could gain more bandwidth than a fair share periodically, depending on the value of the $HitFraction_i$, from the extra packets that the dropping function lets them pass through the queue.
2. The throughput of CBR at the destination may be highly fluctuated.

Therefore, the dropping function should be more aggressive to the unresponsive traffic. Ideally, once the $HitFraction_i$ is higher than the $FairFraction$, no more incoming packets from flow i should be enqueued, which implies a dropping probability of one, until the extra packets are drained and the $HitFraction_i$ becomes lower than the $FairFraction$. This approach requires that the mechanism should be able to keep track of the $HitFraction_i$ fast enough so that the flow with higher $HitFraction$ than the $FairFraction$ would not be penalized longer than necessary. Since, a $HitFraction$ is dynamically adjusted with a mechanism that takes into account both past and current information according to Equation 3.3, so the queue does not have to wait for the end of the sampling period to update the $HitFraction$ and the new dropping policy should contain no problem in terms of responsive action.

However, dropping all the packets when a $HitFraction$ becomes higher than a $FairFraction$ might have a problem with responsive traffic like TCP traffic that backs off when its packets are dropped. As a demonstration, once the $HitFraction$ of TCP traffic reaches a $FairFraction$, its incoming packets are dropped which triggers a back-off period at a TCP source. After a short while, as the queue has drained some packets, the $HitFraction$ becomes lower than a $FairFraction$ once again and incoming packets are allowed to get in. However, the TCP source may still be backing off its data transmission, thus no or only few packets would arrive at the queue causing its $HitFraction$ to be even lower. Later, after the TCP source expands its congestion window, a burst of packets once again arrives at the queue and the $HitFraction$ eventually reaches the $FairFraction$ again. In other words, the $FairFraction$ becomes an upper limit of the $HitFraction$ of TCP traffic which has a sawtooth behavior, giving the average $HitFraction$ to be less than the $FairFraction$, as roughly illustrated to aid this explanation in Figure 58.

From the figure, it is clear that the average $HitFraction$ over time could be lower than the $FairFraction$ which should be the target for a long term $HitFraction$. Under this circumstance,

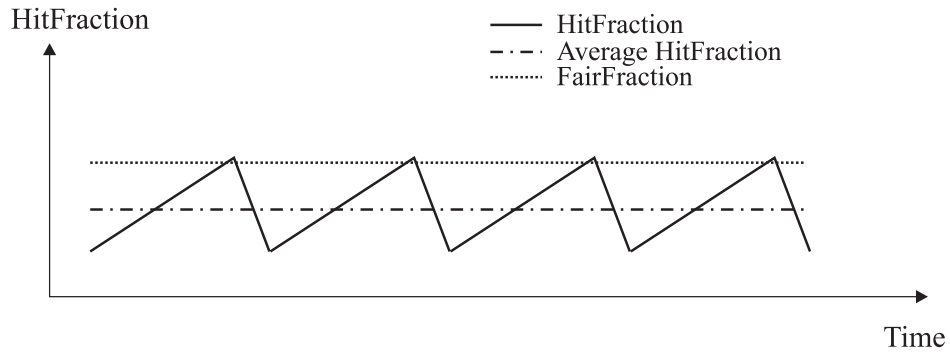


Figure 58: Simplified $HitFraction$ behavior of responsive traffic under AFC dropping policy with no credit-based mechanism.

an underutilization of the responsive traffic and the queue could occur if the HBF cache size is large comparing to the number of active flows ($Nact$).

To prevent this problem, a *credit-based mechanism* is introduced in AFC, so that a flow's $HitFraction$ can be higher than the $FairFraction$ if it has a credit available, e.g. a credit from a back-off period of responsive flows. Note that AFC's credit-based mechanism has no relationship with a large number of papers in the literature about credit-based flow control in ATM network such as that appeared in [41].

5.1.5 Credit-based mechanism

The idea of a *credit-based mechanism* in AFC is to allow a $HitFraction$ to go beyond a $FairFraction$ if a flow has a credit available, so that an average $HitFraction$ over time is about the same as a $FairFraction$. Here, a credit is defined as an area under the $HitFraction$ curve above or below the $FairFraction$ in Figure 58, which is referred to as $\Delta\mathcal{A}$. Precisely, a credit of any given flow can be approximated every time a $HitFraction$ is updated according to

$$\Delta\mathcal{A}_t = \Delta\mathcal{A}_{t^-} + \left[(HitFraction_t - FairFraction_t) * (t - t^-)\right], \tag{5.2}$$

as roughly illustrated in Figure 59, where t indicates a current update time and t^- indicates a previous update time. Obviously, the value of $\Delta\mathcal{A}_t$ should be kept as close to zero as possible. However, when a responsive flow is backing off, there is usually not enough packets to fill up

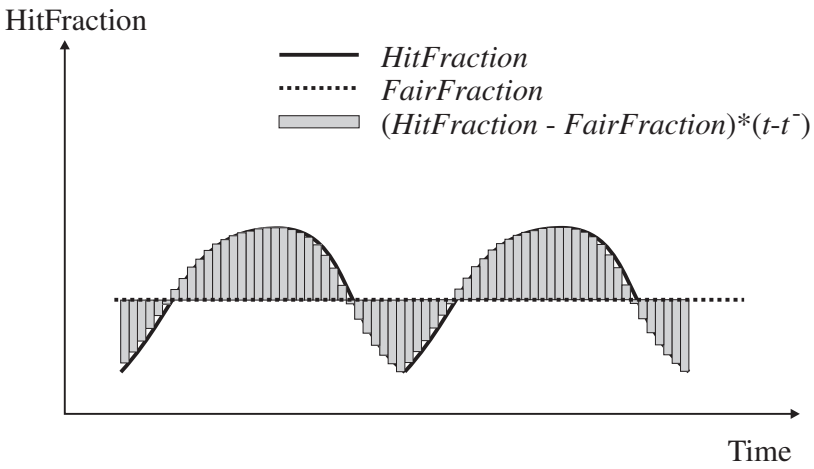


Figure 59: Simplified $HitFraction$ behavior of responsive traffic under new AFC dropping policy.

the queue to raise its $HitFraction$ to be as much as a $FairFraction$, and thus its $\Delta\mathcal{A}_t$ would become negative. The negative value of $\Delta\mathcal{A}_t$ means that this flow have this amount of credit for AFC to allow the packets of this flow to enter the queue even if its current $HitFraction$ is greater than the $FairFraction$. This dropping policy is contrast to the refined dropping function introduced in the previous section where the packets are dropped when a $HitFraction$ is greater than a $FairFraction$ only. By allowing a $HitFraction$ of a flow to be higher than a $FairFraction$ if it has available credit, e.g. from its previous back-off period that causes $\Delta\mathcal{A}_t$ to be negative, an underutilization of a responsive flow such as that in Figure 58 is prevented.

A simulation is setup, to see the evolution of a $HitFraction$ over time, with the same settings of 200 TCP traffic and the asymmetric topology in Section 3.3.4 except that the HBF cache is large enough to hold every passing flow and the number of active flows is assumed to be known. The result in Figure 60 shows that the $HitFraction$ of this sample flow could swing above or below the $FairFraction$, as its upper bound is not restricted to the $FairFraction$ when the method described in the previous section is used or that shown in Figure 58. Nevertheless, the fairness among these TCP connections with different round trip delays is also achieved with a standard deviation of only 6.17 Kbps around the average throughput of 225 Kbps, in this ideal case.

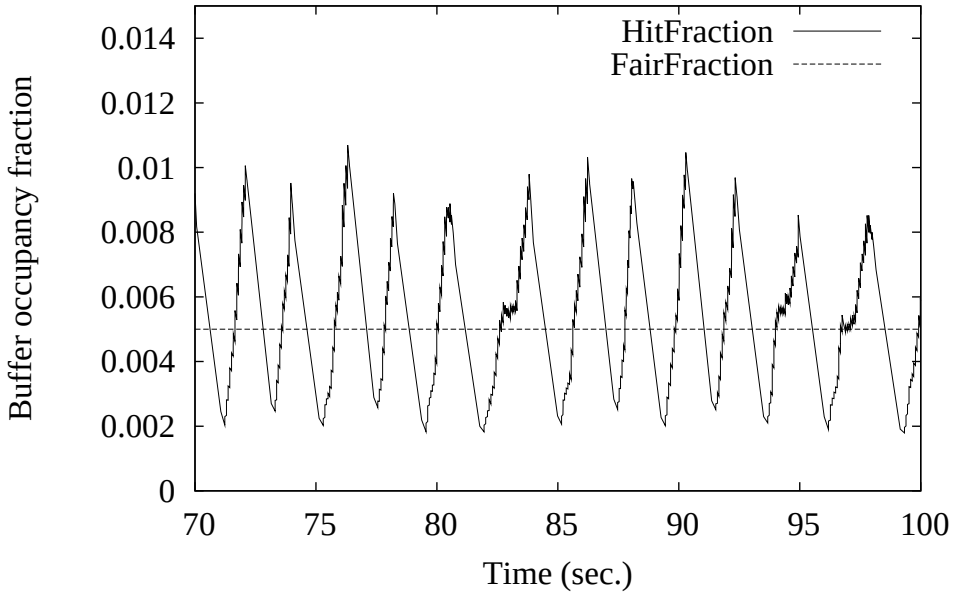


Figure 60: Evolution of $HitFraction$ over time of a sample flow under AFC with a credit-based mechanism.

5.2 PERFORMANCE EVALUATION

In this section, an evaluation of AFC begins with the same set of experiments conducted in Chapter 3.3 which is composed of (1) single unresponsive flow scenario, (2) multiple unresponsive flows scenario, and (3) TCP with different round-trip delays scenario. Because one of the design goals of AFC is to solve the problem of throughput fluctuation in BLACK, not only the throughput fairness is used as a performance metric but the instantaneous throughput over time of CBR traffic is also considered and compared with BLACK. Then, the more diverse scenarios are conducted to examine fairness and robustness of the fair AQM schemes, including a scenario with traffic with different packet sizes, a scenario with short-lived and bursty traffic, and a scenario when TCP-friendly traffic and TCP traffic sharing the same bottleneck link. In general, the scenarios could be categorized into four parts according to the types of traffic in consideration – long-lived unresponsive traffic, TCP with different round-trip times[1], short-lived traffic, and TCP-friendly traffic – as shown in Table 11. In addition, at the end of this section, the complexities of these schemes are

[1]This is the only category that only TCP traffic with the same settings are sharing the same bottleneck link.

also discussed.

Long-lived unresponsive traffic • Single unresponsive flow versus TCP traffic • Multiple unresponsive flows versus TCP traffic • Traffic with different packet sizes
TCP traffic with different round-trip times
Short-lived traffic • Effect of short-lived responsive traffic in background • Bursty unresponsive traffic
TCP-friendly traffic versus TCP traffic

Table 11: Simulation scenarios.

Network Simulator 2 (NS-2) [1] is again used as a simulation tool. BLACK queue is re-evaluated using *Direct Bitmap* as the estimation algorithm of the number of active flows. The results of RED, CHOKe, SFB, and CARE from Chapter 3.3 are shown or replotted for comparison purposes. All the parameters are set according to the respective experiments in Chapter 3.3, which are resummarized in each of the sections. Because AFC contain the same parameters as BLACK, all the settings are the same as BLACK, as previously discussed in Section 3.3.5 and 3.3.6. Each experiment runs for 200 sec. and it is repeated 20 times. The statistics are collected from 50 sec. to 200 sec. of the simulation time. The analysis of the other AQM schemes, except AFC, are not repeated in this chapter unless necessary.

5.2.1 Single unresponsive flow

A single unresponsive flow experiment utilizes the symmetric dumbbell topology shown in Figure 20. A large unresponsive CBR traffic shares the same bottleneck link with 100 TCP traffic. Two experiments were set up – (1) 5-Mbps bottleneck link rate with 5-Mbps CBR traffic (2) 45-Mbps bottleneck link rate with 10-Mbps CBR traffic. All the access links are 100 Mbps. The parameters of different queue types are set according to Table 12. The results of the simulation are

RED	min_{th} = 50 packets, max_{th} = 150 packets.
CHOKe	min_{th} = 50 packets, max_{th} = 150 packets.
SFB	Two levels of hash functions of 23 bins with double set of hash tables for moving hash functions (total of 46×2 bins).
CARE	Capture occasion (t) = 200 with 50 of this value is used for the estimation of the number of active flows
BLACK and AFC	min_{th} = 50 packets, max_{th} = 150 packets, sample size (m) = 3,000 packets, HBF cache size = 20.
Common parameters	Maximum buffer size = 300 packets.

Table 12: Parameters of different queues in the single unresponsive flow experiment.

tabulated in Table 13 and 14 for both experiments respectively.

The last row of Table 13 shows the per-flow throughput of different flows under AFC, in comparison to the per-flow throughput obtained from the other schemes in the upper rows. AFC is, not only better than BLACK in terms of per-flow throughput fairness, but also better than the other schemes and very close to SFB with the rate limit being set to the fair rate. In another experiment with varying CBR arrival rate, AFC still showing a superior performance in controlling unresponsive traffic be close to the fair throughput, even with the arrival rate of twice the bottleneck link rate, as shown with BLACK in Figure 61 and shown with the other schemes in 62. Note that although it is unlikely to have the unresponsive flows that consume this large portion of a link bandwidth for high-speed core routers, however it is possible that similar situation could happen on one of the end-to-end links that the traffic traverse through. The second experiment with the bottleneck link speed of 45 Mbps fed with 10-Mbps CBR and 100 TCP traffic also exhibits the same trend as shown in Table 14. Under AFC, per-flow throughput of both CBR and TCP traffic are about the fair rate, comparing to the other schemes, while the fairness among TCP traffic is not distorted as shown through Jain's fairness index in the third column.

In addition, in terms of CBR's throughput fluctuation that occurs with BLACK, this phenomenon is greatly reduced under AFC, as shown in Figure 63 from the first experiment (5-Mbps CBR with 5-Mbps link) which illustrates the throughput over time of the CBR traffic. Because of

	Average UDP throughput (Kbps)	**Average TCP throughput (Kbps)**	**Jain's fairness index among TCP flows**
RED	4,374.82	5.046	0.5551
CHOKe	1187.301	38.160	0.9842
SFB, rate limit ≈ twice the fair rate	98.99	49.046	0.9836
SFB, rate limit ≈ fair rate	49.57	49.538	0.9826
SFB, rate limit ≈ half the fair rate	24.95	49.78	0.9817
CARE	172.66	48.307	0.9862
BLACK	73.20	49.301	0.9918
AFC	50.46	49.529	0.9925

Table 13: Results from the single unresponsive flow scenario including AFC. Bottleneck link speed of 5 Mbps. UDP has a constant bit rate of 5 Mbps competing with 100 TCP flows.

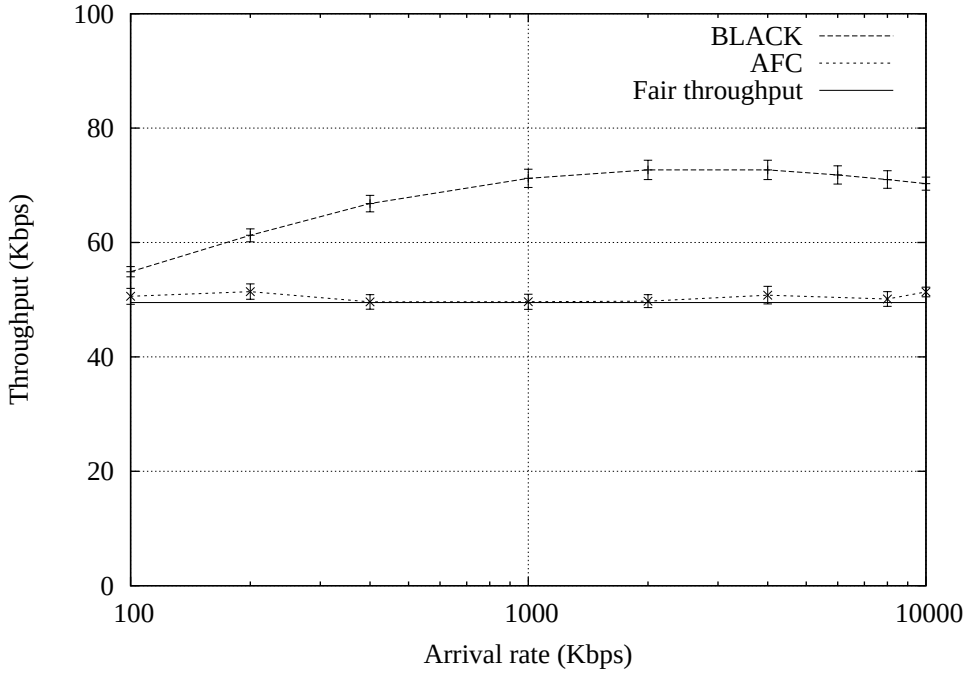


Figure 61: UDP throughput vs. arrival rate for BLACK and AFC; plotted with 95% confidence interval.

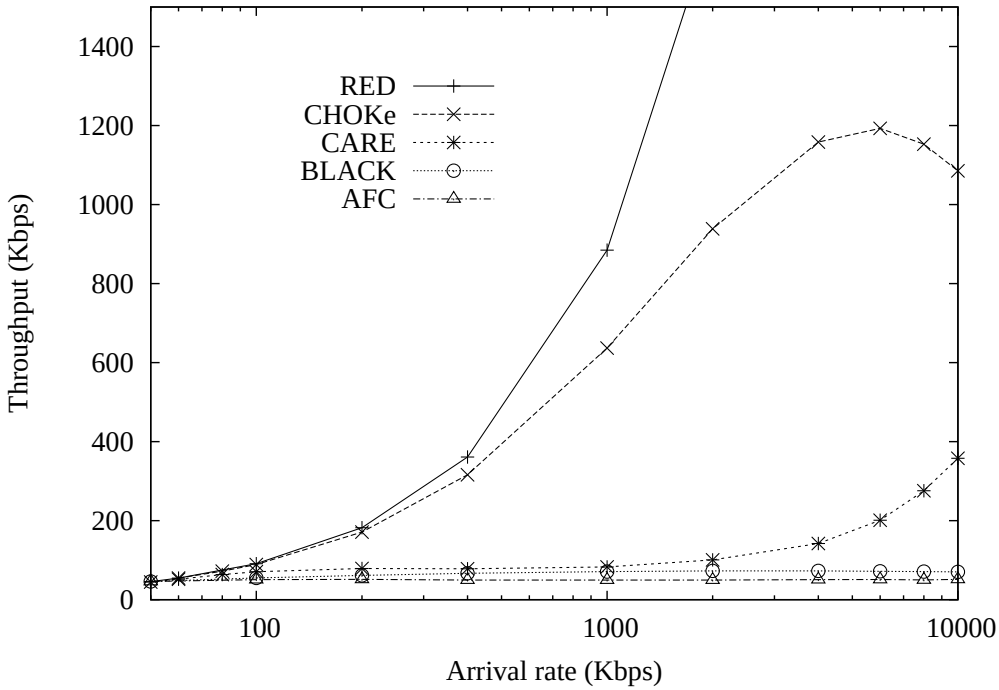


Figure 62: UDP throughput vs. arrival rate for RED, CHOKe, CARE, BLACK and AFC; Each point is an average value over 20 runs.

a large difference in UDP throughput obtained from different schemes, the UDP throughput over time of CHOKe is plotted on different vertical (throughput) scale. The figure clearly shows that CBR throughput is much smoother under AFC. Only SFB provides smooth throughput comparable to AFC, while the other schemes result in highly fluctuated throughput. For the second experiment with 10-Mbps CBR with 45-Mbps link, because of a high difference among CBR throughput under different schemes, only CBR throughput of BLACK and AFC are illustrated in Figure 64 to show the improvement of AFC in providing smoother transfer rates for the unresponsive flows.

	Average UDP throughput (Kbps)	**Average TCP throughput (Kbps)**	**Jain's fairness index among TCP flows**
RED	9,768.23	352.77	0.9952
CHOKe	6,632.55	384.12	0.9964
SFB, rate limit ≈ twice the fair rate	909.85	450.37	0.9792
SFB, rate limit ≈ fair rate	454.92	445.91	0.9815
SFB, rate limit ≈ half the fair rate	222.39	448.15	0.9807
CARE	825.80	442.19	0.9944
BLACK	426.21	446.19	0.9948
BLACK	511.45	445.24	0.9945
AFC	445.54	445.99	0.9956

Table 14: Results from the single unresponsive flow scenario including AFC. Bottleneck link speed of 45 Mbps. UDP has a constant bit rate of 10 Mbps competing with 100 TCP flows.

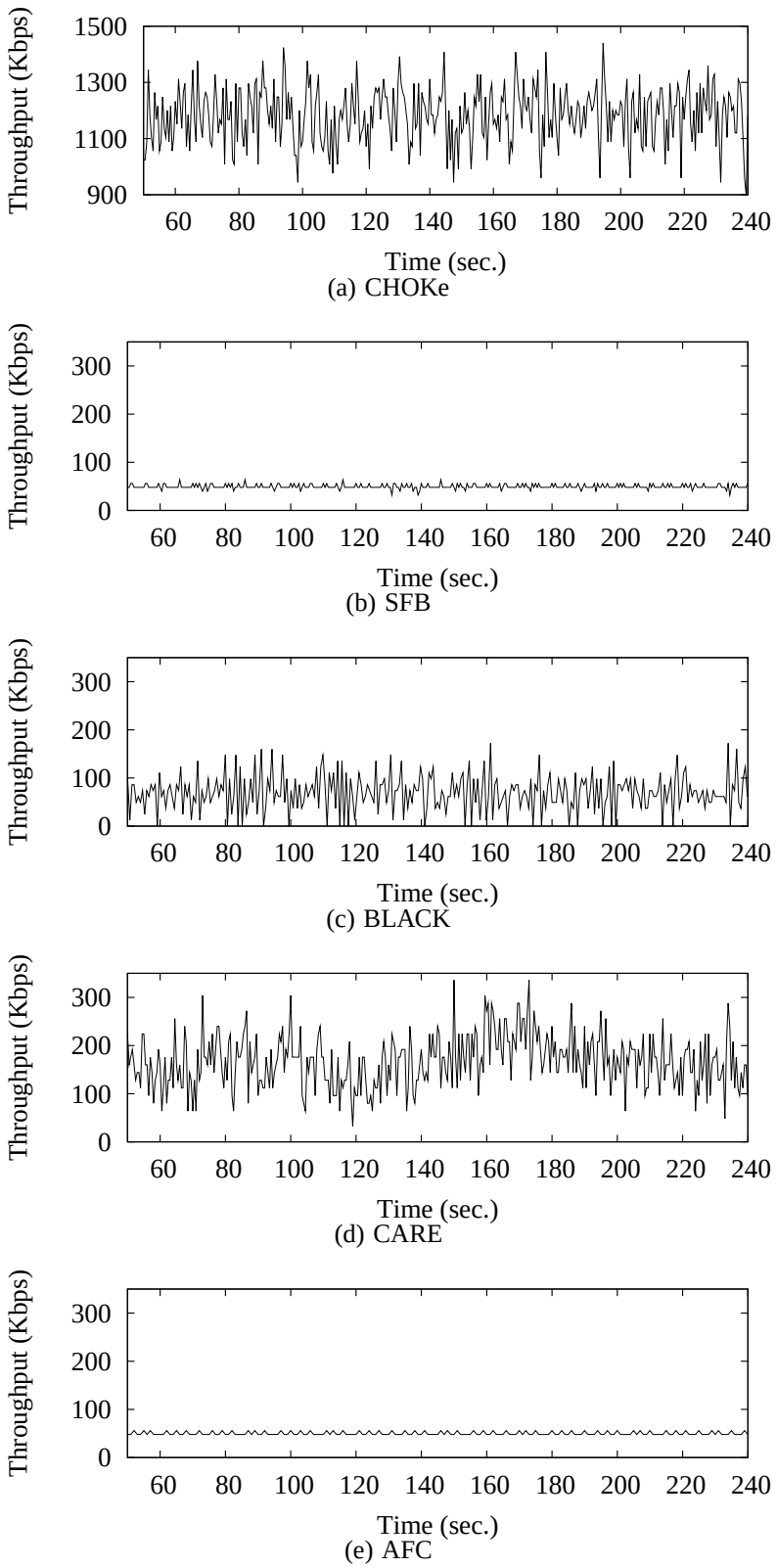


Figure 63: CBR throughput over time after passing through different fair AQM schemes. Bottleneck link rate is 5 Mbps and CBR arrival rate is 5 Mbps.

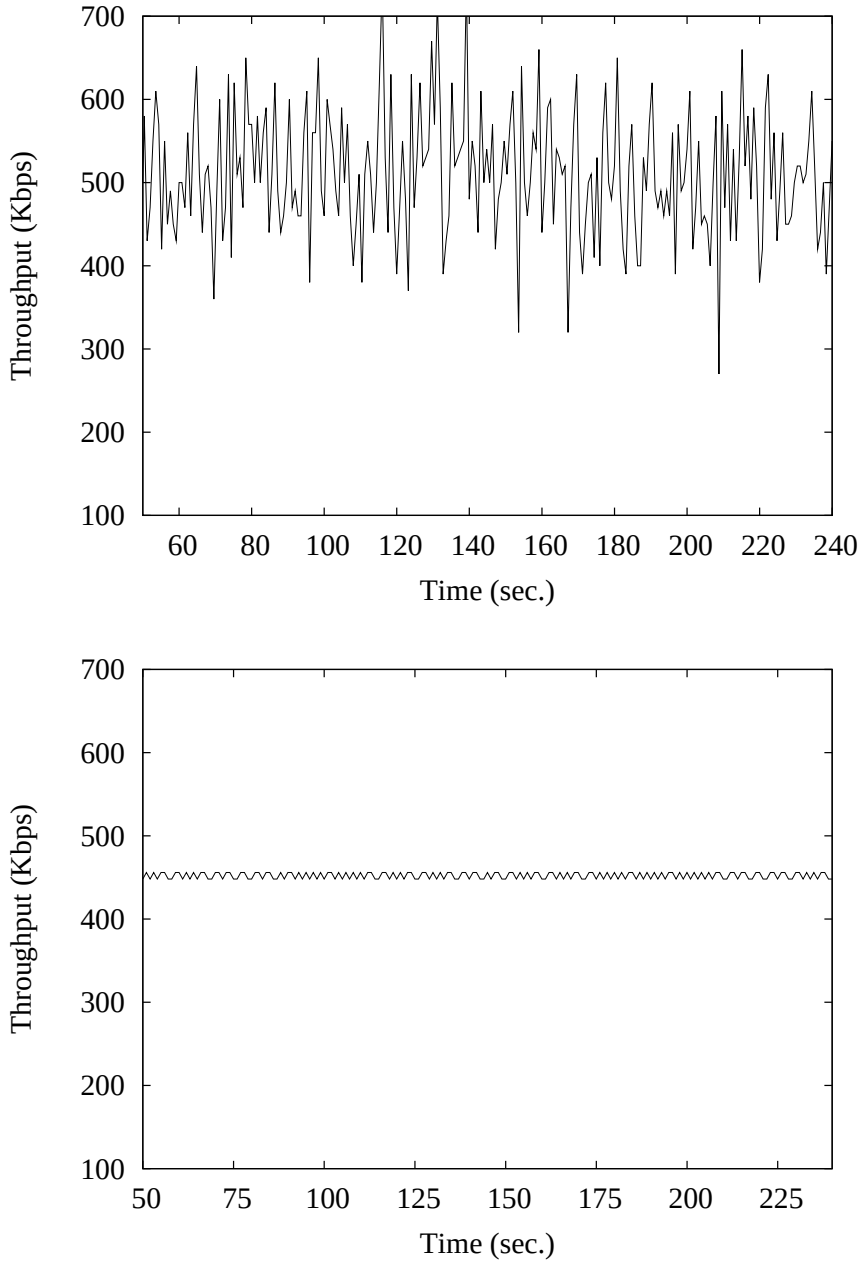


Figure 64: CBR throughput over time after passing through BLACK and AFC. Bottleneck link rate is 45 Mbps and CBR arrival rate is 10 Mbps.

5.2.2 Multiple unresponsive flows

In this section, the experiments with multiple unresponsive flows, which are closer to a real world scenario, are set up using the same topology with 5-Mbps and 10-ms delay of the bottleneck link as in the first experiment in the previous section. Two sets of experiments are conducted – (1) five CBRs (each of 500 Kbps) sharing the link with 100 TCP traffic, and (2) five CBRs (each of 5 Mbps) sharing the link with 100 TCP traffic. All the queue parameters are the same as those shown in Table 12 except that CHOKe is now equipped with its *self adjusting mechanism* to handle multiple unresponsive flows as discussed in Section 3.3.3.

	Average UDP throughput (Kbps)	**Average TCP throughput (Kbps)**	**Jain's fairness index among TCP flows**
RED	420.32	29.021	0.9565
CHOKe	318.52	34.107	0.9836
SFB, rate limit ≈ twice the fair rate	100.01	45.032	0.8863
SFB, rate limit ≈ fair rate	42.23	47.920	0.8595
SFB, rate limit ≈ half rate	17.66	49.150	0.8611
CARE	107.92	44.241	0.9852
BLACK	66.34	46.716	0.9904
AFC	49.33	47.567	0.9908

Table 15: Results from the multiple unresponsive flow scenario including AFC. Bottleneck link speed of 5 Mbps. Each of five UDP has a constant bit rate of 500 Kbps competing with 100 TCP flows.

The results of the two experiments with different CBR arrival rates in Table 15 and 17 show a superior fairness performance of AFC over the other schemes. Here, although SFB also achieves

Queue type	CBR per-flow throughput (Kbps)				
	flow 0	flow 1	flow 2	flow 3	flow 4
SFB, rate limit ≈ fair rate	18.08	96.90	16.37	104.69	6.61
AFC	47.08	48.05	48.74	48.00	47.52

Table 16: A sample result from a single run comparing per-flow throughput of CBR traffic under SFB and AFC. Each CBR arrival rate is 500 Kbps. Bottleneck link is 5 Mbps with 100 TCP traffic in background.

about the same level of fairness, a network operator needs to manually set a rate limit threshold as SFB has no prior knowledge of a fair throughput nor the number of active flows. Besides, without a special per-flow treatment or a separate queue to serve the unresponsive flows, although the average per-flow throughput of unresponsive traffic is limited to about the rate limit threshold, SFB cannot guarantee fairness among these unresponsive flows. With the code for NS simulator provided by the authors of SFB [18], all the misbehaving traffic, once detected with its hashing technique, are prevented from entering the queue for a certain period of time (or *hold time*) to achieve a desired throughput. Basically, a concept of hold time has been introduced since its predecessor, BLUE [20], in a form of time interval between the updates of a dropping probability. In one of the simulations of SFB under this scenario, even through the average per-flow throughput of the five CBR traffic is about the about the fair share, the individual throughput are extremely varied as shown in comparison with AFC in Table 16. This result indicates that the SFB's double moving hash function alone is not enough to provide fairness, but a special per-flow treatment is needed in order to improve its fairness performance.

Again, even with multiple large unresponsive flows, AFC still provide very small fluctuation of CBR throughput. Figure 65 show the results from one of the run in the multiple unresponsive flows experiment with CBR arrival rate of 5 Mbps each.

	Average UDP throughput (Kbps)	**Average TCP throughput (Kbps)**	**Jain's fairness index among TCP flows**
RED	1,000.00	0.000	N/A
CHOKe	841.29	5.092	0.1108
SFB, rate limit ≈ twice the fair rate	95.29	45.285	0.9594
SFB, rate limit ≈ fair rate	47.84	47.658	0.9611
SFB, rate limit ≈ half rate	22.94	48.903	0.9593
CARE	1,000.00	0.000	N/A
BLACK	77.04	46.181	0.9903
AFC	48.81	47.609	0.9972

Table 17: Results from the multiple unresponsive flow scenario including AFC. Bottleneck link speed of 5 Mbps. Each of five UDP has a constant bit rate of 5 Mbps competing with 100 TCP flows.

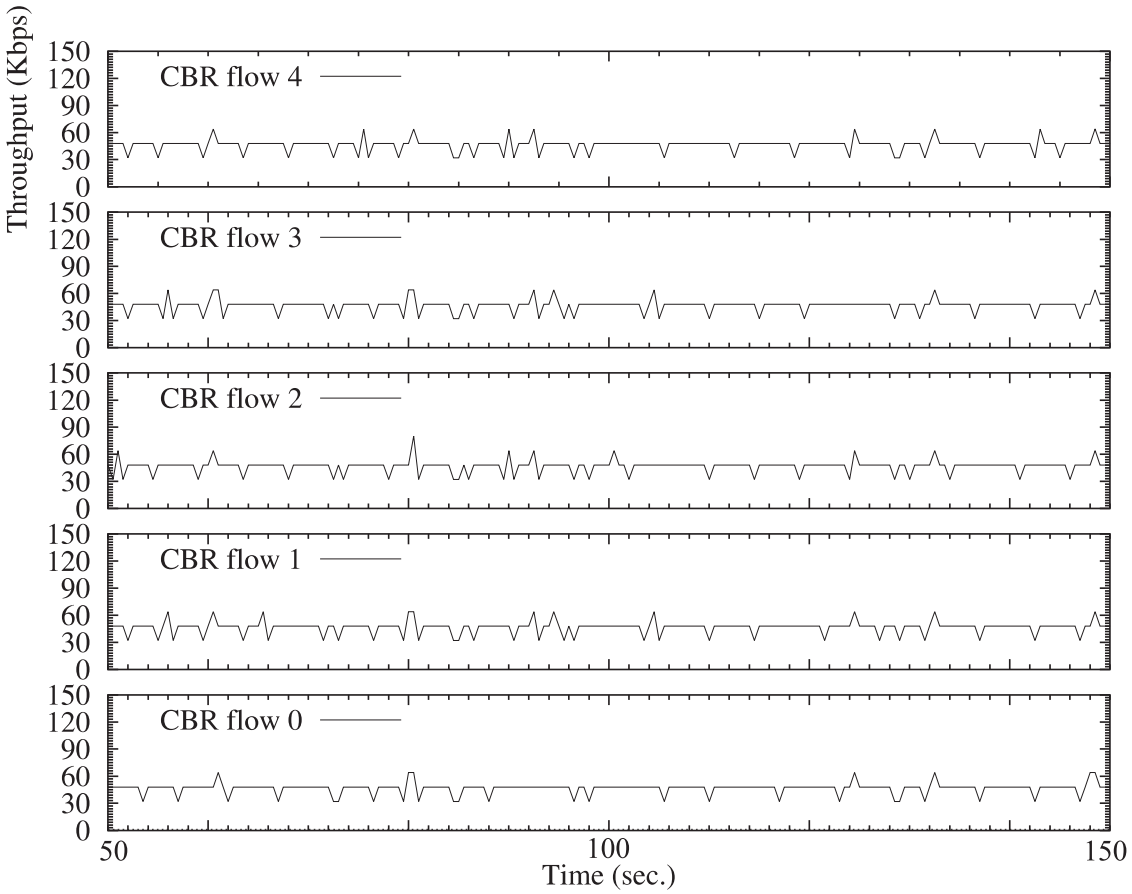


Figure 65: CBR throughput over time after passing through AFC. Bottleneck link rate is 5 Mbps and CBR arrival rate is 5 Mbps each.

5.2.3 Traffic with different packet sizes

As described with the pilot study in Section 3.4 at the end of Chapter 3.3, most fair AQM schemes were not designed to provide fairness for traffic with different packet sizes. Some schemes such as CHOKe currently has no better solution rather than breaking a large packet into few smaller packets of the same size to handle this situation [54]. AFC is designed to solve this problem by having a byte count, as opposed to a Hit count in BLACK, to provide a better fair share of throughput under this circumstance.

An experiment was set up with the same symmetric topology shown in Figure 20 with a bottleneck link of 5 Mbps and 10-ms delay. One hundred TCP traffic are competing with three 1-Mbps CBR traffic. However, these three CBR traffic have different packet sizes where CBR1 has a packet size of 100 bytes, CBR2 has a packet size of 500 bytes, and CBR3 has a packet size of 1,000 bytes. The average per-flow throughput of CBR traffic under different fair mechanisms are plotted along with a fair share of bandwidth in Figure 66. The figure clearly shows much superior fairness performance of AFC over the other schemes where the per-flow throughput of three CBR with totally different sizes of packets are provided in a fair manner.

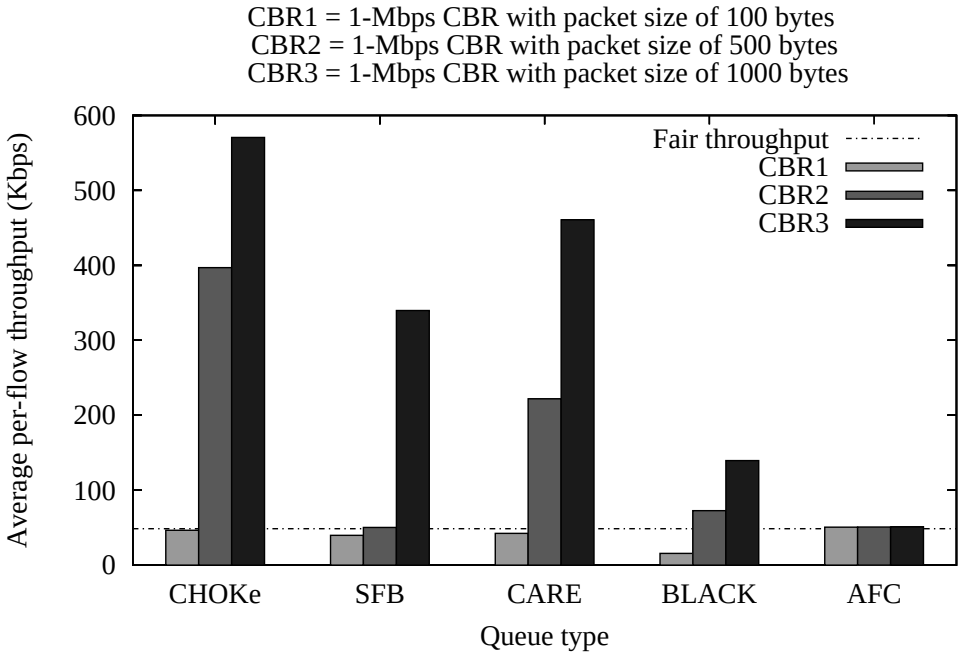


Figure 66: Average per-flow throughput of CBR traffic with different packet sizes.

5.2.4 TCP with different round-trip times

With the same simulation settings as in Section 3.3.4, 200 TCP traffic are randomly assigned to originate from node N0, N1, N2, N3 or N4 which are linked to the randomly selected sink node S0, S1, S2, S3 and S4 with the asymmetric topology in Figure 24. The cache size of BLACK and AFC is 46 which is the same as SFB, or only a quarter of the number of these long-lived TCP traffic. The result of the simulation under AFC is presented in Figure 68; in this figure, the results from the other schemes are repeated for a purpose of comparison. The standard deviation values and Jain's fairness indexes from these this same experiment are tabulated in Table 18, while these standard deviations are also separately plotted with 95% confidence interval over multiple runs in Figure 67. It is clear that AFC still achieves good fairness performance among TCP connection with different round-trip times, apart from providing fairness when unresponsive traffic exist. SFB shows much larger standard deviation among TCP throughput, which is about double of those under BLACK and AFC. The reason is that SFB provides rate limit only on those flows that are detected as unresponsive traffic, or when the dropping probability in all the bins that these flows hashed to reach one. All the other flows would be controlled by a dropping function similar to its predecessor, BLUE [20], which provides no fairness guarantee.

Flow throughput (Kbps)	**RED**	**CHOKe**	**SFB**	**CARE**	**BLACK**	**AFC**
Standard deviation	43.725	38.268	41.996	30.194	20.308	16.149
Jain's fairness index	0.9769	0.9721	0.9665	0.9815	0.9917	0.9948

Table 18: TCP connections with different round-trip time scenario.

5.2.5 Effect of short-lived traffic

In all of the previous experiments in this Chapter and in Chapter 3, the fairness performance is evaluated under the ideal scenarios where all of the sources transmit only long-lived traffic. In this section, short-lived traffic are introduced in two different types of scenario – (1) low-bandwidth

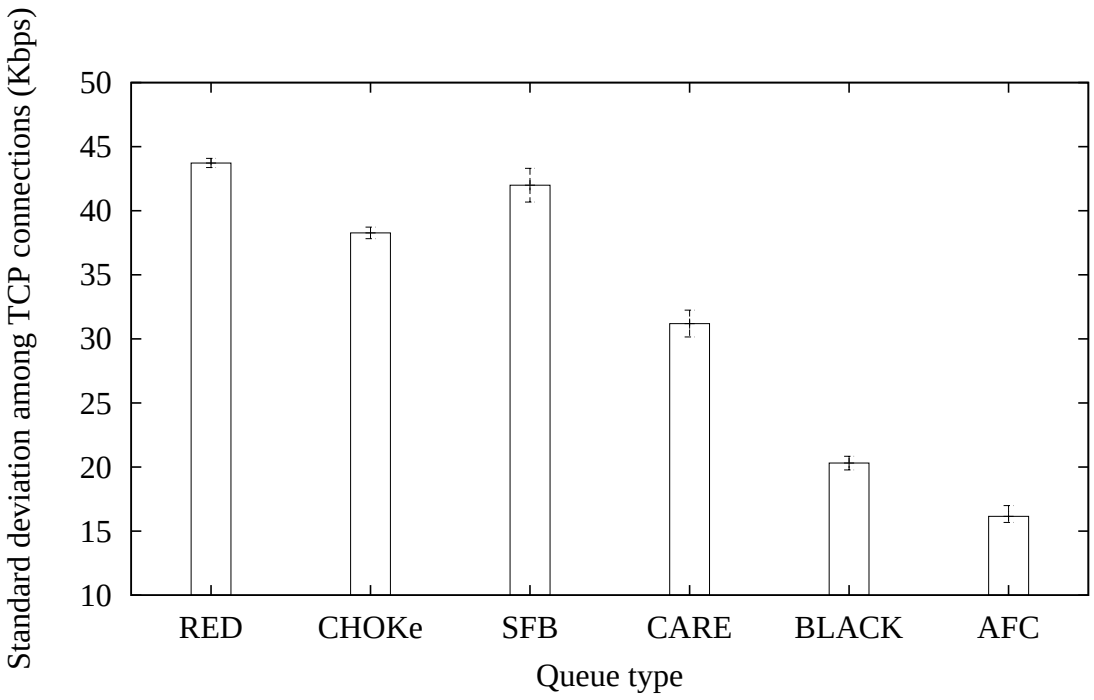


Figure 67: Standard deviation of TCP throughput under different queue types; plotted with 95% confidence interval.

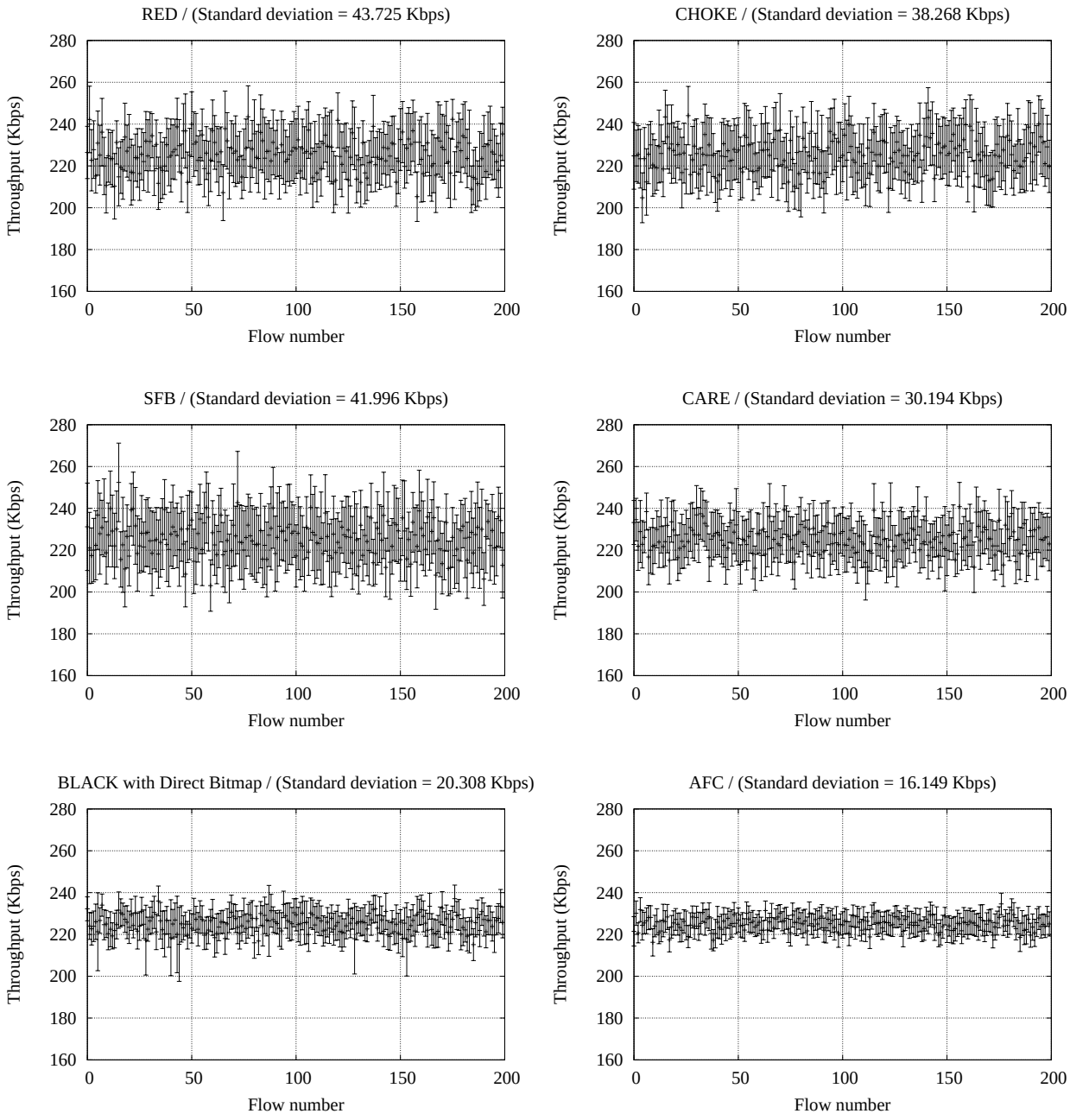


Figure 68: Per-flow TCP throughput for 200 TCP connections with different RTTs over a 45-Mbps link; plotted with 95% confidence interval. The result from AFC is included.

short-lived traffic such as web traffic (2) bursty traffic or high-bandwidth short-lived traffic such as a malicious traffic that aims to saturate a link by sending an impulse of traffic to escape a fair mechanism of an AQM scheme. A number of experiments are set up to see whether these short-lived traffic have any negative effect on the fairness performance of the AQM schemes.

5.2.5.1 Low bandwidth short-lived traffic The first set of experiments are conducted to evaluate how well the AQM scheme could still provide fairness when there are a different loads of web traffic in background (or mice flows according to mice and elephants model discussed in Section 3.2). The same dumbbell topology in Figure 20 is set up with a bottleneck link of 10 Mbps and 10 ms delay. Five CBR traffic of 2.5 Mbps each and 100 TCP traffic are sharing the same bottleneck link along with w sessions of web traffic. Each web session contains default parameters recommended in the NS2 script [1] where the traffic model is based on [17]. A Pareto distribution is used for flow lengths of web traffic where the average number of packets per flow is 15 with a shape parameter of 1.2. Starting time of each of w web sessions are randomly set during 250 seconds of simulation time. With a fixed simulation time, large w implies high web traffic load scenarios or more number of web traffic that would arrive at the queue than small w. In this experiments, the number of web sessions (w) are set to 0, 2,500, 5,000, 7,500 and 10,000 sessions to be generated during this 250 seconds of simulation time. CHOKe, SFB (with rate limit set to the fair fraction), CARE, and AFC are evaluated whether their fairness performance is interfered with different web traffic loads.

The results of this experiment are shown in Figure 69 in a form of an average per-flow CBR throughput over an average per-flow TCP throughput. The average per-flow CBR throughput is an average over five CBR traffic and over 20 runs of simulations. Ideally, an average CBR throughput over an average TCP throughput should be close to one to ensure fairness among CBR traffic and long-lived TCP traffic. The upper image in Figure 69 shows that AFC, CARE, and SFB could achieve their fairness performance with minimal interference from different web traffic loads. In this case, average CBR throughput under CARE is higher than AFC and SFB because of multiple CBR traffic with a high arrival rate that causes an underestimation of the number of active flows, however the average CBR throughput comparing to average TCP throughput is only slightly higher with more number of background web sessions. On the other hand, CHOKe shows a different

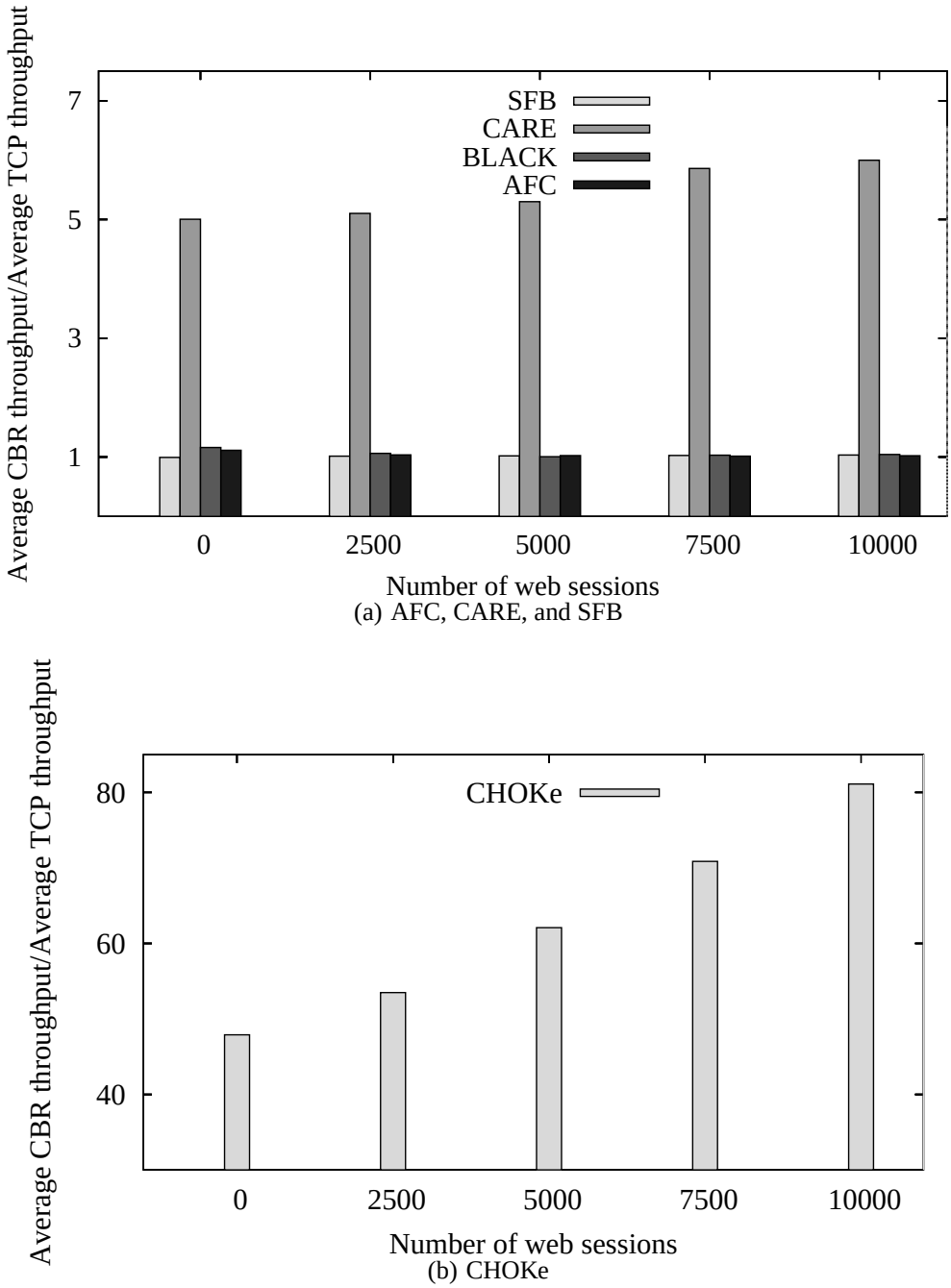


Figure 69: Average per-flow CBR throughput over average per-flow TCP throughput (average over 20 runs) at different background web traffic loads.

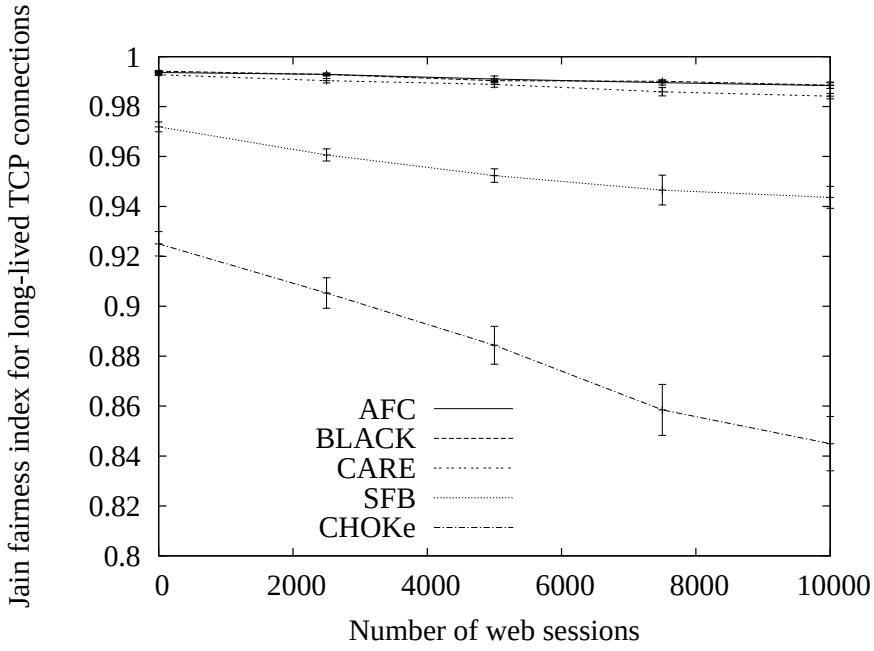


Figure 70: Jain's fairness index among long-lived TCP connections at different background web traffic loads. The legend represents the lines in the figure from top to bottom respectively.

result in the lower image of Figure 69. When there is no background web traffic exists, the average per-flow CBR throughput is about 50 times the average long-lived TCP throughput. However, the average CBR throughput are getting significantly higher than the average TCP throughput as the number of web sessions increases, or approximately 80 times higher when the number of web sessions becomes 10,000 sessions. The rationale behind this poorer performance of CHOKe under high a load situation is that when more number of packets from the web traffic get in to the queue, with the average length that is controlled by underlying RED mechanism, less number of packets from the same CBR connection are in the queue at the same time which leads to less chance of a packet matching and less control of unresponsive traffic. On the other hand, in terms of the fairness among only long-lived TCP connections, AFC, BLACK, and CARE (top three lines respectively) perform well regardless of the web traffic load according to Jain's fairness index showing in Figure 70. SFB provides less fairness to long-lived TCP traffic, and this value is degraded as more number of web sessions turn on. For CHOKe, fairness on TCP traffic is highly interfered with higher load of background web traffic.

5.2.5.2 High bandwidth short-lived traffic or bursty traffic Although several fair AQM mechanisms achieve average long-term fairness in different scenarios, none has been evaluated in terms of their performance to detect and control short-lived misbehaving traffic or bursty traffic. In a real network, not all of high bandwidth traffic are CBR, which have been used as a misbehaving traffic in all of the previous experiments. There are a number of media applications that feed traffic that is bursty to a network. Providing an average long-term fairness for bursty traffic or high bandwidth short-lived traffic may not be perfect for the light-weighted fair AQM schemes that do not maintain flows state information and might not have enough long term information about average arrival rate. Some AQM schemes need some amount of time to collect some information before identifying and controlling misbehaving traffic, and may erase those information after a short while. Although not common, it is possible that some bad people may use this fundamental knowledge to generate high-bandwidth short-lived traffic that might escape a fair AQM mechanism to saturate a link by pumping high speed impulse traffic. A key question here is how well the fair AQM schemes can detect and control these types of traffic.

A series of experiment are set up with the misbehaving traffic are represented by a very high-bandwidth short-lived UDP traffic with fixed ON and OFF periods or bursty traffic with ON and OFF periods are drawn from exponential distribution. Again, 100 long-lived TCP traffic are passing through the same bottleneck link of 10 Mbps. Different settings of short-lived traffic are shown in Table 19. Two types of ON and OFF periods are tested – (1) traffic with equal mean ON and OFF period, and (2) traffic with longer OFF period. A purpose of the latter type is to see if there is any different fairness performance when a traffic is paused for a longer period of time. Peak rate during an ON period also comes with two settings – 1 Mbps and 10 Mbps. A 1-Mbps peak rate is used in an environment with low bandwidth bursty traffic. A peak rate of 10 Mbps is used to simulate a scenario with an attack of short-lived high bandwidth traffic.

Because the results of fixed ON-OFF traffic and exponential ON-OFF traffic are very similar, only those obtained from the experiments with exponential ON-OFF traffic are exhibited. The results under different queue types are separately tabulated in Table 20, 21, 22, 23, 24, and 25, for RED, CHOKe, CARE, BLACK, and AFC respectively.

Even through the average per-flow throughput of both CBR and TCP traffic are shown, the fairness performance could be compared by the average per-flow throughput of TCP traffic alone

Peak rate	10 Mbps				1 Mbps			
Mean ON/OFF period	= 1 sec./1 sec.		= 2 sec./4 sec.		= 1 sec./1 sec.		= 2 sec./4 sec.	
Number of UDP traffic	5	15	5	15	5	15	5	15

Table 19: A combination of the experiments on high-bandwidth short-lived traffic. These bursty traffic share the same bottleneck link with 100 long-lived TCP traffic.

as it indicates how well the AQM schemes could protect responsive flows. As a guideline, a fair throughput under the scenarios with 5 bursty traffic is 95.2 Kbps, and a fair throughput under the scenarios with 15 bursty traffic is 86.9 Kbps.

For bursty traffic with 10-Mbps peak rate, RED mechanism, a scheme that provides least fairness, obviously cannot protect TCP traffic at all in all the scenarios that the peak rate of bursty traffic is 10 Mbps (Table 20). All TCP connections receive about 10 Kbps, or even shut out in two cases of fifteen 10-Mbps bursty traffic. CHOKe (Table 21) and CARE (Table 23) could provide some protection when there are five bursty flows, but cannot prevent fifteen 10-Mbps bursty traffic from grasping almost all of the bandwidth. It is not unexpected for CHOKe as the results in the previous experiments so far show that CHOKe does worse in providing fairness under a higher number of unresponsive flows. CARE, however, is a little worse than CHOKe and the reason behind this behavior is its inability to estimate the number of active flows correctly with a presence of unresponsive traffic with higher rate. BLACK could provide some level of protection as it could reserve about 60% - 70% of a fair throughput to each TCP connection by average (column 1 and 3), when there are five 10-Mbps bursty traffic. A trace file indicates that BLACK's HitFraction mechanism, that randomly samples packets from the queue, could not perform as well as when the unresponsive traffic is non-bursty. On the other hand, AFC which utilizes a direct counting of HitFraction provides much better fairness as each TCP connection gains more than 92% of the fair share. However, both BLACK and AFC performance are degraded when there are fifteen 10-Mbps bursty traffic coming to the queue because of the cache size of only 20. In this case, bursty traffic does have an impact on the mechanism of BLACK and AFC. When the number of bursty traffic is comparable to the cache size, those traffic could be replaced easily during the OFF period of the

traffic. So their HitFraction statistics has to be restarted once the traffic is in the ON period again. BLACK turns to be largely distorted as each TCP connection receives only around 30% - 40% of the fair share (column 2 and 4). AFC, however, is still far better as each TCP connection gains 72% - 90% of the fair share. On the other hand, SFB performs very well in these scenarios (Column 1-4 of Table 22, but with a fine tuning of the rate limit threshold given that the arrival rates of bursty traffic must be known in advance.

For a low-bandwidth bursty traffic case (column 5-8), although all of the schemes leave the bandwidth to each bursty traffic at least 1.8 - 3 times the fair share, all of the schemes still provide some level of protection to TCP connections, particularly when there are less number of bursty traffic. However, when there are fifteen bursty traffic, RED, as the based line for a comparison, clearly provides least fairness as all bursty traffic altogether take up to 60% of total bandwidth and leave each TCP connection with a bandwidth equals to only about half of the fair throughput. Note that there are 100 long-lived TCP connections sharing the portion of the bandwidth left from UDP, and if there are less number of TCP connections, a proportion of per-flow TCP throughput to a fair share would be even less. CHOKe (Table 21) also provides similar fairness performance to RED even with lower-bandwidth bursty traffic. CARE, BLACK and AFC (Table 23, 24, and 25) achieve better fairness than RED and SFB on both five bursty flows case and fifteen bursty flows case. Here, CARE estimates the number of active flows much better when the bursty traffic has a peak rate of 1 Mbps, as opposed to 10 Mbps. Nevertheless, even with the best case of AFC, CBR still grasp 1.8 times the fair share of bandwidth. SFB case (Table 22) is different from the other schemes because SFB needs to know a rate limit threshold in advance which should be manually configured. Using the same setting that achieves very good fairness performance in the 10-Mbps peak rate case (column 1-4), SFB turns to provide poor fairness performance in 1-Mbps peak rate case (column 5-8). This result shows that although SFB could detect unresponsive traffic, it cannot use the same setting to provide fairness for different scenarios.

5.2.6 Fair AQM schemes with TCP-friendly traffic

In this section, the problem is moved from the fairness between TCP traffic and the traffic that is not responsive or non-TCP-friendly traffic, to the fairness between TCP traffic and TCP-friendly

Peak rate	10 Mbps				1 Mbps			
Mean ON/OFF	= 1 sec./1 sec.		= 2 sec./4 sec.		= 1 sec./1 sec.		= 2 sec./4 sec.	
Number of UDP	5	15	5	15	5	15	5	15
Average UDP throughput (Kbps)	1698.9	666.3	1651.5	665.9	456.9	405.0	297.6	287.4
Average TCP throughput (Kbps)	9.8	≈ 0.0	13.3	≈ 0.0	77.2	39.4	85.1	56.9
Jain fairness index for UDP	0.975	0.895	0.937	0.788	0.993	0.978	0.976	0.971
Jain fairness index for TCP	0.849	0.221	0.461	0.168	0.990	0.883	0.992	0.977

Table 20: A result of high-bandwidth short-lived UDP traffic sharing a link with 100 TCP traffic under RED.

Peak rate	10 Mbps				1 Mbps			
Mean ON/OFF	= 1 sec./1 sec.		= 2 sec./4 sec.		= 1 sec./1 sec.		= 2 sec./4 sec.	
Number of UDP	5	15	5	15	5	15	5	15
Average UDP throughput (Kbps)	893.9	653.3	880.3	645.0	385.2	364.4	254.3	245.16
Average TCP throughput (Kbps)	55.3	1.9	56.0	3.3	80.8	45.5	87.3	63.2
Jain fairness index for UDP	0.987	0.980	0.976	0.967	0.996	0.979	0.975	0.970
Jain fairness index for TCP	0.962	0.364	0.963	0.353	0.992	0.928	0.992	0.986

Table 21: A result of high-bandwidth short-lived UDP traffic sharing a link with 100 TCP traffic under CHOKe.

Peak rate	10 Mbps				1 Mbps			
Mean ON/OFF	= 1 sec./1 sec.		= 2 sec./4 sec.		= 1 sec./1 sec.		= 2 sec./4 sec.	
Number of UDP	5	15	5	15	5	15	5	15
Average UDP throughput (Kbps)	86.1	115.6	86.0	102.4	321.1	375.6	206.9	232.6
Average TCP throughput (Kbps)	95.7	82.7	95.7	84.7	84.0	43.8	89.7	65.1
Jain fairness index for UDP	0.984	0.958	0.984	0.969	0.993	0.978	0.974	0.966
Jain fairness index for TCP	0.969	0.808	0.974	0.854	0.943	0.682	0.964	0.889

Table 22: A result of high-bandwidth short-lived UDP traffic sharing a link with 100 TCP traffic under SFB.

Peak rate	10 Mbps				1 Mbps			
Mean ON/OFF	= 1 sec./1 sec.		= 2 sec./4 sec.		= 1 sec./1 sec.		= 2 sec./4 sec.	
Number of UDP	5	15	5	15	5	15	5	15
Average UDP throughput (Kbps)	1098.7	666.4	995.8	664.8	213.0	264.8	182.0	189.5
Average TCP throughput (Kbps)	44.4	≈ 0.0	49.5	≈ 0.0	89.4	60.4	90.9	71.6
Jain fairness index for UDP	0.989	0.925	0.981	0.868	0.994	0.986	0.984	0.981
Jain fairness index for TCP	0.989	0.235	0.888	0.235	0.990	0.973	0.989	0.990

Table 23: A result of high-bandwidth short-lived UDP traffic sharing a link with 100 TCP traffic under CARE.

Peak rate	10 Mbps				1 Mbps			
Mean ON/OFF	= 1 sec./1 sec.		= 2 sec./4 sec.		= 1 sec./1 sec.		= 2 sec./4 sec.	
Number of UDP	5	15	5	15	5	15	5	15
Average UDP throughput (Kbps)	595.7	385.7	774.8	452.5	257.1	264.2	220.1	214.1
Average TCP throughput (Kbps)	68.9	37.2	59.4	26.1	87.2	60.5	89.0	67.9
Jain fairness index for UDP	0.760	0.572	0.753	0.775	0.924	0.835	0.937	0.921
Jain fairness index for TCP	0.928	0.540	0.890	0.617	0.968	0.925	0.963	0.962

Table 24: A result of high-bandwidth short-lived UDP traffic sharing a link with 100 TCP traffic under BLACK.

Peak rate	10 Mbps				1 Mbps			
Mean ON/OFF	= 1 sec./1 sec.		= 2 sec./4 sec.		= 1 sec./1 sec.		= 2 sec./4 sec.	
Number of UDP	5	15	5	15	5	15	5	15
Average UDP throughput (Kbps)	231.4	191.1	150.2	130.9	265.7	245.1	170.8	156.4
Average TCP throughput (Kbps)	87.7	68.4	92.1	78.7	86.7	63.4	91.5	76.5
Jain fairness index for UDP	0.981	0.963	0.960	0.939	0.986	0.959	0.972	0.968
Jain fairness index for TCP	0.989	0.971	0.991	0.981	0.992	0.956	0.993	0.991

Table 25: A result of high-bandwidth short-lived UDP traffic sharing a link with 100 TCP traffic under AFC.

traffic[2]. As previously discussed in section 2.2, TCP-friendly protocol or TCP-compatible protocol is an end-system-based approach which is an alternative way to combat the unfairness problem, given that all of the end users would eventually deploy these protocols. These TCP-friendly protocols have been recently proposed as alternative protocols that are responsive to network congestion and more suitable for media applications as they provide much smoother traffic, but are also aimed to achieve the same long term throughput as TCP. Because these TCP-friendly protocols are rather new, the fair AQM mechanisms that have been focused in this chapter so far have never been evaluated in scenarios with TCP traffic and TCP-friendly protocols, which utilize different back-off mechanisms to cope with network congestion. For example, because TCP-friendly protocols are designed to provide a smoother congestion control, they usually react in a less responsive manner to individual packet drops. Unfairness has been reported in a number of scenarios such as IIAD versus TCP but using drop-tail queueing [6]. Furthermore, these TCP-friendly protocols may get penalized unfairly by the various AQM schemes and sometimes may even impose a negative impact in terms of fairness; however, this aspect has not been studied yet.

However, the key question considered in this section is therefore whether the fair AQM mechanisms, which fall under a router-based approach in solving the unfairness problem, actually provide fairness when different variants of TCP compete with TCP-friendly traffic.

Well known TCP-compatible protocols include GAIMD [71], Binomial Algorithms [6], and TFRC [27]. These protocols are used for evaluation of the fair AQM mechanisms in this section, and are briefly described as follows.

- GAIMD or Generalized AIMD is based on the same additive increase multiplicative decrease (AIMD) window-based mechanism of TCP provided in section 2.1.1. However, the congestion window is adjusted through an increase parameter a and a decrease parameter b, and denoted as AIMD(a,b). Under no packet loss conditions, the congestion window increases from W to $W+a$ packets per round-trip time. On the other hand, in response to a packet loss, the congestion window decreases from W to $(1-b)W$. With this notation, current TCP implementations can be referred to as AIMD(1,1/2). Since TCP's halving of the congestion window as a result of a single packet drop is not suitable for media applications, GAIMD variants has been proposed to provide a smoother change in the sending rate through a decreasing parameter b of less than

[2]Most of the materials in this section are to be published in [11]

1/2 such as GAIMD(1/5,1/8) or GAIMD(1,1/8).

- The family of Binomial algorithms are TCP variants that use nonlinear congestion control by means of four control parameters k, l, a and b. Both k and l are additional parameters introduced to express TCP-compatible congestion control such that the congestion window is increased from W to $W + a/W^k$ and decreased from W to $W - bW^l$ in the case of no loss and a single loss respectively. It has been shown that as long as $k + l = 1$, all Binomial algorithms will achieve fairness with competing TCP. Two Binomial algorithm examples that are well-suited to multimedia applications are IIAD ($k = 1$, $l = 0$) and SQRT ($k = l = 0.5$).
- TFRC (TCP Friendly Rate Control), a rate-based TCP-compatible protocol, utilizes TCP's throughput equation to adjust the sender's rate based on feedback from the receiver. TFRC changes the sender's rate in a much smoother manner than TCP. Upon detecting a packet loss, the receiver estimates and sends back a loss event rate back to the sender. With this information combined with the calculated round-trip time (R), TCP retransmission timeout value (t_{RTO}), and the packet size s, the sender can adjust the transmission rate according to the steady-state TCP bandwidth equation $T = s/(R\sqrt{2p/3} + t_{RTO}(3\sqrt{3p/8})p(1 + 32p^2))$ [51].

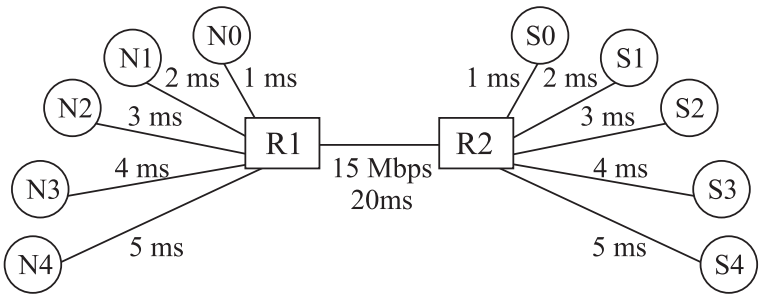


Figure 71: Simulation topology for TCP-friendly traffic experiment.

For a simulation setting to evaluate fairness performance of the fair AQM mechanisms when TCP variants compete with these TCP-compatible traffic, an asymmetric topology shown in Figure 71 has a bottleneck link has a bandwidth of 15 Mbps with a propagation delay of 20 ms. All the traffic, with 1-Kbyte packets, are originated randomly from one of the source nodes N0 - N4 to one of the sink nodes S0 - S4. All of the access links are connected to the routers R1 and R2 at 100 Mbps with the delay denoted in the figure. Maximum buffer space at the router R1 is set to 300

packets. A simulation script and the settings are adapted from [27] which evaluates TFRC with TCP traffic under RED queue.

The AQM mechanisms under evaluation are RED, CHOKe, SFB, CARE, BLACK and AFC. Drop Tail scheme is also included as a baseline case for comparison. SFB is configured with double sets of two levels of hash functions, each of 23 bins, which comes to a total of 46 bins. BLACK and AFC are equipped with a cache size of 46. The min_{th} and max_{th} threshold settings for RED, CHOKe, BLACK, and AFC are 50 and 150 packets respectively [3]. CHOKe is implemented with its self adjusting mechanism, where the region between min_{th} and max_{th} are divided into k=8 subregions and the number of drop candidates is set to $f(i) = 2 \cdot i$ ($i = 1..k$).

The TCP-friendly protocols included in the evaluation are TFRC, representing the case of rate-based TCP-friendly protocols, and the window-based protocols IIAD, SQRT, and GAIMD. Since the fair AQM schemes are designed to combat unfairness, our experiments then focus on a less responsive GAIMD(1,1/8), which has been reported to achieve about 2.23 times the bandwidth of competing TCP flows [26]. In this experiment, variants of TCP, i.e. Tahoe, Reno, Newreno, and Sack TCP are included, to see an interaction between several types responsive traffic.

From the total combination of the experiments, the most important results are illustrated as follows.

5.2.6.1 TCP vs. aggressive TCP-friendly protocol Figure 72 shows simulations of n Sack TCP traffic competing with n GAIMD(1,1/8) under the different AQM mechanisms. The graphs illustrate the flows' throughput over the last 60 seconds of the total simulation time. Each mark on the graph represents the throughput of one flow normalized to the value of the fair share of the bottleneck link. As both types of traffic have an equal number of flows on each set run, the x-axis only shows the number of flows for each type of traffic.

The results with Drop Tail and RED show that GAIMD(1,1/8) receives much higher bandwidth than SACK. Under CHOKe, BLACK, and AFC fair AQMs, both the mean and the variance of the normalized throughput among flows are narrower indicating a better fairness over RED. The reason behind this observable performance is from the fact these AQMs drop packets from large flows according to the buffer space occupied by these flows. As GAIMD(1,1/8) reduces its

[3] These are the Gentle RED parameters [23].

window size at a much slower pace on each packet drop (to smooth the congestion control for media applications) than TCP(1,1/2), GAIMD(1,1/8) tends to feed more packets into the buffer than TCP, and thus faces higher packet drop probability. However, GAIMD(1,1/8) still gains a slightly higher share of the bandwidth than competing TCP flows even under CHOKe or BLACK, due to the lack of complete knowledge of per-flow information. In one of the experiments (not shown here) when BLACK is assumed to have a large enough cache size to maintain state of every flow, almost perfect long-term fairness between GAIMD(1,1/8) and TCP can be achieved.

An interesting point is the fairness performance of SFB. Although the mean normalized throughput of both types of traffic under SFB are close to 1.0, the variance among all flows are spread out rather widely. One observation we found was that, even though SFB can attain fairness among the same type of TCP traffic with or without unresponsive flows [19], it is very difficult to tune SFB's parameters to achieve close-to-fairness among different TCP-congestion control flows. The results shown in Figures 72, 73, and 74 were obtained using the suggested parameter settings in [2, 18]. A better tuning of the parameters that updates the dropping probability more rapidly provides better results as shown in the last image in Figure 73. However, this difficulty occurs in the entire set of our simulation experiments. Therefore, we do not recommend using SFB without having a sensitivity analysis on the tunable parameters of the environment under consideration.

In addition, because SFB was designed to provide fairness by controlling high-bandwidth unresponsive traffic, we could not expect to see a high degree of fairness among different type of responsive flows. According to the SFB mechanism [19], it is rare that any responsive flow would be classified as a misbehaving traffic. In this way, the fairness performance of TCP-friendly traffic and TCP variants under SFB falls under the control of the packet dropping function and the hashing which does not guarantee fairness.

5.2.6.2 TCP vs. comparable TCP-friendly protocol In contrast to an aggressive protocol, TCP achieves about the same long-term throughput than the Binomial protocols IIAD and SQRT. In Figure 73, it is shown that with the exception of Drop Tail and SFB, TCP and comparable TCP-friendly protocols receive their fair share no matter the AQM scheme utilized.

A queue with a fair AQM mechanism usually allows similar flows to experience similar drop rates, which is the basic assumption for the TCP equation model at steady state. Hence, TCP and

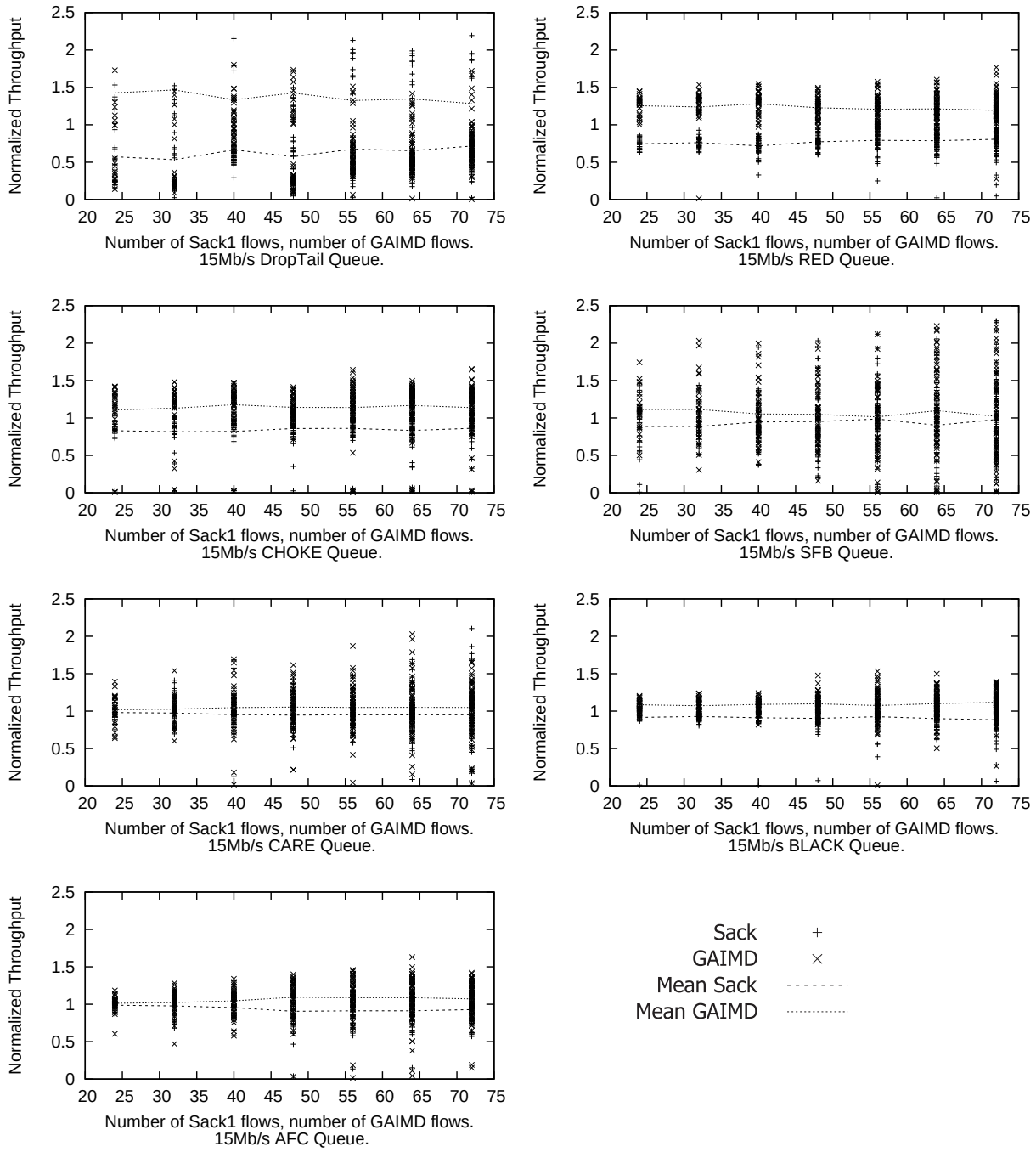


Figure 72: Normalized throughput under the scenario of Sack vs. GAIMD(1,1/8), with drop-tail queueing, RED (first row), CHOKe, SFB (second row), CARE, BLACK (third row), and AFC (bottom) where x-axis shows the equal number of flows for each type of traffic.

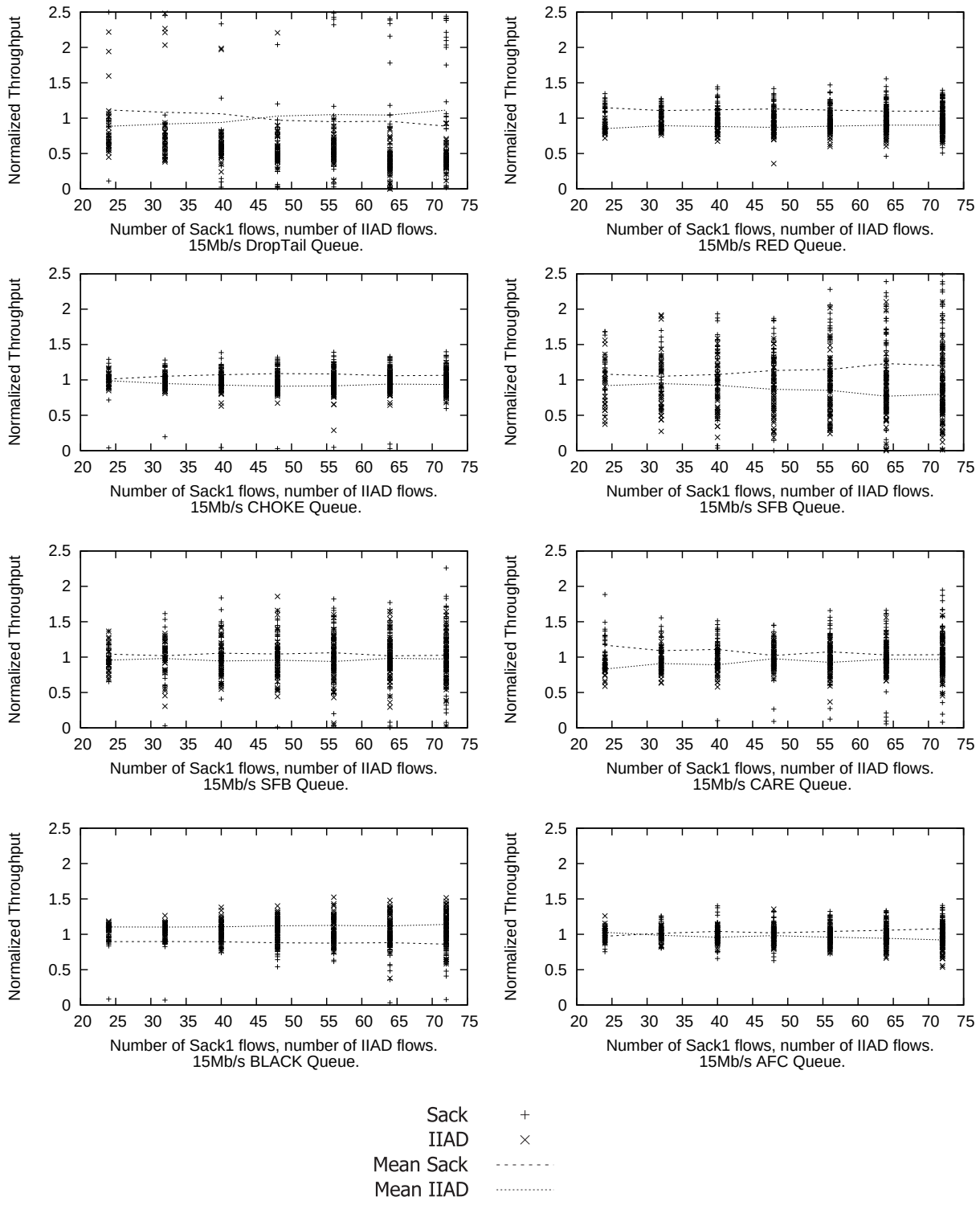


Figure 73: Normalized throughput under the scenario of Sack vs. IIAD, with drop-tail queueing, RED (first row), CHOKe, SFB (second row), SFB with tuned parameters and CARE (third row), BLACK and AFC (fourth row) where x-axis shows the equal number of flows for each type of traffic.

Binomial protocols can share the same bottleneck link in a fair manner as expected. From the figure, it can be seen how CHOKe, BLACK and AFC improve fairness over Drop Tail and RED. SFB however can provide good fairness only if its parameters are well tuned, otherwise a poor fairness performance close to Drop Tail could occur. In the case of the rate-based TFRC protocol, TCP variants receive about the same bandwidth share as TFRC under RED, CHOKe and BLACK AQM schemes, except some cases of Reno TCP which will be explained in the following section.

5.2.6.3 Negative impact of AQM's aggressive dropping Although fair AQM schemes could provide fairness when responsive TCP traffic compete for the bandwidth with unresponsive traffic, some of these schemes could provide a much inferior results when some TCP variants share a bottleneck link with TCP-friendly traffic.

It is well known that TCP Reno typically suffers performance problems when multiple packets are dropped from the same window of data [22]. AQM mechanisms using aggressive dropping policies may introduce multiple packet drops which will deteriorate TCP Reno's performance. The simulation results shown in Figure 74 show that CHOKe, although gaining better fairness than RED in various scenarios, can lead us back to the unfairness problem when Reno competes with some TCP-friendly traffic. This degraded performance occurs because CHOKe drops *both* the incoming packet and the sampled packet from the buffer if they are from the same flow, therefore resulting in a higher tendency of multiple packet drops from a single window, and eventually timeouts. In Figure 77 when Reno TCP is replaced with Sack, although the problem of CHOKE exists but the effect is not as negative as in the case of Reno TCP.

Figure 75 illustrates this problem comparing the congestion window behavior of a sample Reno TCP connection under RED, CHOKe and CARE during 40 - 60 seconds of simulation time. Although increasing the number of sampled packets (drop candidates) per each packet arrival provide a better handling of multiple unresponsive flows as suggested in [52], it does the opposite for TCP Reno. Another experiment, as shown in Figure 76, confirms that unfairness between TCP Reno and TFRC increases with the number of drop candidates. This behavior also occurs even under Sack TCP vs. TFRC, but with much less degree of severity.

CARE faces another problem as TFRC increases and decreases its data rate at much slower pace than TCP traffic. CARE performs the calculation of flows' arrival rate and set a dropping

probability only at the end of a measuring period (after the capture list is full). If a flow consumes more bandwidth than a fair share, a high dropping probability is set for that flow for a whole next measuring period. On the other hand, if a flow consumes the bandwidth that is less than a fair share, almost all of its packets could enter the queue in the whole next measuring period. TFRC has no problem with this behavior as it decreases and increases its data rate in a slower pace, and it considers a burst of packet drops as a single loss event. However, TCP is not only more responsive and more sensitive to a burst of packet drops. For example, from the Reno-TFRC experiment, if Reno TCP has an arrival rate that is greater than a fair share calculated through the Jackknife estimator, a high dropping probability is set for this flow for the next whole measuring period. Because of the high drop probability, a congestion window of the flow in the figure is dropped sharply after 45 seconds. During this period, new packets from this flow could hardly get into the queue becuase the high dropping probability is set constantly during the whole period of time, causing a decline of TCP throughput. After this period, the number of packets from this flow collected in the capture list would be small, resulting in a very low dropping probability calculated for the next period. Consequently, the congestion window is expanded until the queue is full or a new dropping probability is set, as shown in the period of 48 - 56 seconds, and the cycle repeats.

On the other hand, BLACK drops packets less aggressively and the dropping probability is dynamically adjusted according to a flow's HitFraction. This type of performance degradation does not occur under BLACK, except in the case of BLACK mechanism when it highly overestimates the number of active flows and over-punishes multiple packets. However, with the modification that incorporates Direct Bitmap, AFC controls the fairness performance very well. AFC also achieves similar fairness performance in this case because of the dynamically adjusted dropping function.

In conclusion, the results in this TCP-friendly section show the better performance of CHOKe, BLACK and AFC even when traffic with less responsive congestion control such as GAIMD (1,1/8) co-exist with TCP, despite the fact that GAIMD still gains a slightly larger portion of the bandwidth. For TCP-friendly protocols with congestion control comparable to TCP, the fair AQM mechanisms provide only a small improvement over RED if at all, in a number of cases. This shows that RED is sufficient to maintain adequate fairness for a number of TCP-compatible protocols. However, with little extra complexity, better fairness is obtained by deploying CHOKe,

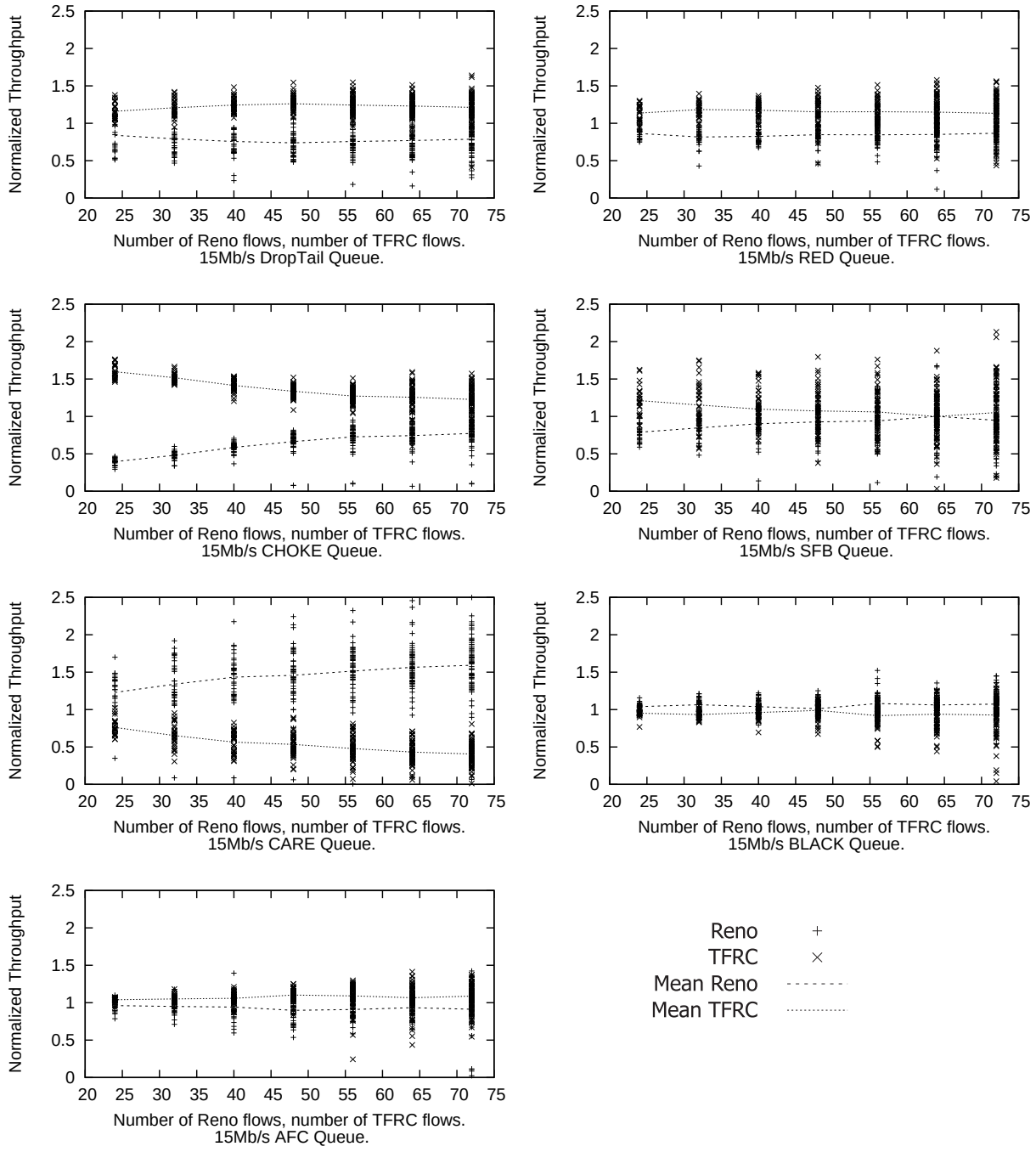


Figure 74: Normalized throughput under the scenario of Reno vs. TFRC, with drop-tail queueing, RED (first row), CHOKe, SFB (second row), CARE, BLACK (third row), and AFC (bottom) where x-axis shows the equal number of flows for each type of traffic.

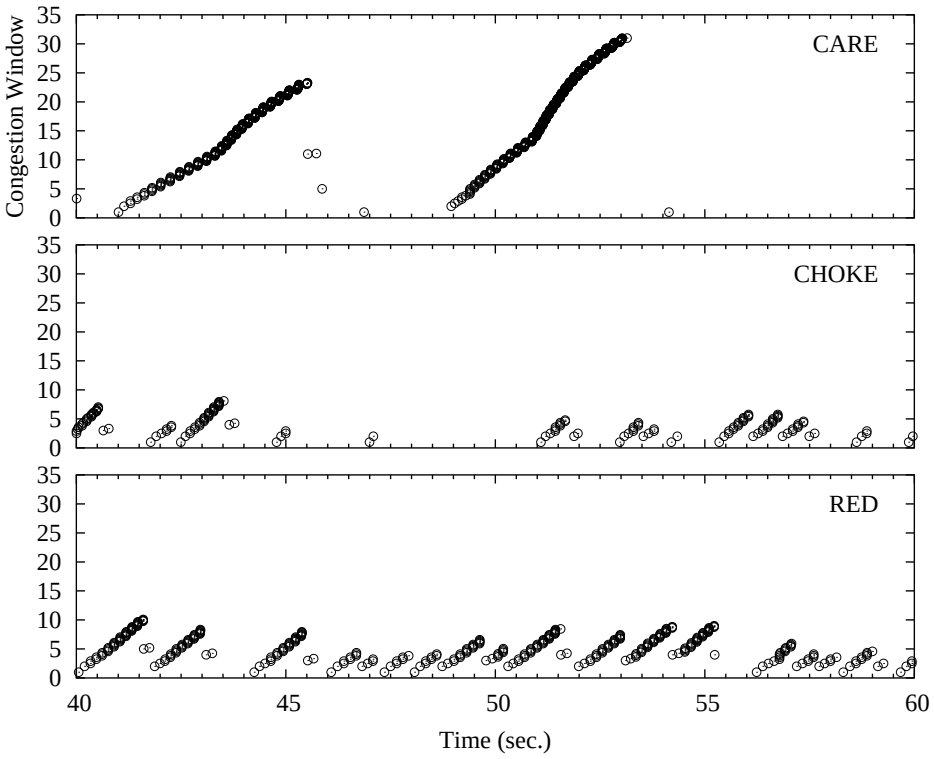


Figure 75: Congestion window behavior of TCP Reno flow number 5 under RED, CHOKe, and CARE during a time frame of 40-60 seconds.

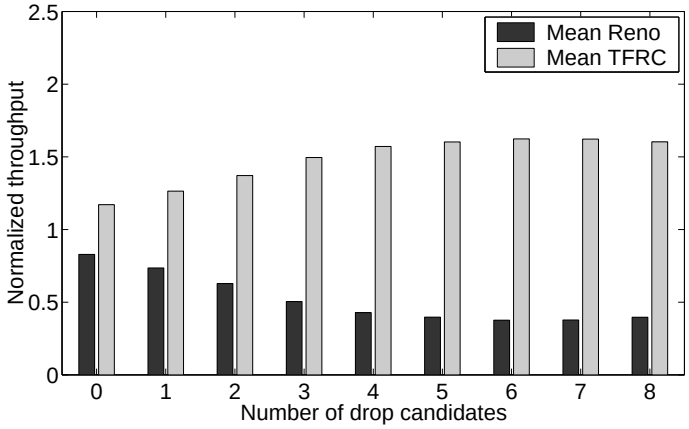


Figure 76: Comparing average normalized throughput of 40 TCP Reno traffic and 40 TFRC traffic under CHOKe with different number of drop candidates.

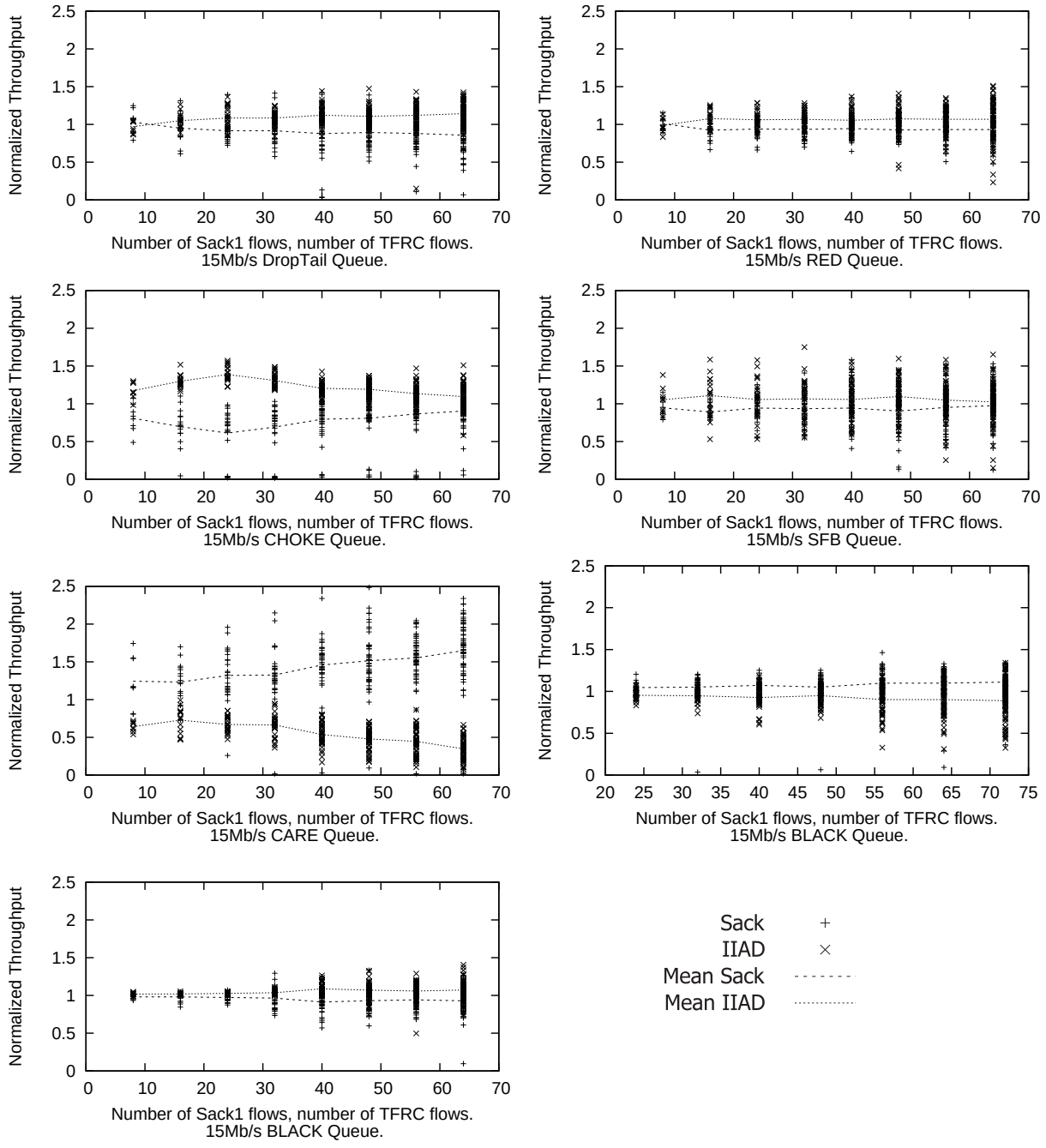


Figure 77: Normalized throughput under the scenario of Sack vs. TFRC, with drop-tail queueing, RED (first row), CHOKe, SFB (second row), SFB with tuned parameters and CARE (third row), BLACK and AFC (fourth row) where x-axis shows the equal number of flows for each type of traffic.

BLACK, and AFC, especially in the case of the less responsive AIMD protocol. In addition, these AQM mechanisms shield the system against unresponsive traffic. Care should be taken in those scenarios where TCP variants sensitive to a burst of losses are used, such as TCP Reno. CHOKe and CARE are found to be too aggressive in dropping packets, since it drops multiple packets at the same time, and introduces bursts of losses. Lastly, the fairness performance obtained with SFB is sensitive to the parameters chosen. Without proper parameter tuning, SFB produces unexpected inferior results, even in a scenario with TCP-friendly protocols with congestion control comparable to TCP. The results in this section indicate that an AQM scheme must be chosen not only based on its performance or capability to deal with unresponsive flows as usually done but also considering its performance when other types of flows are included, such as TCP-friendly sources.

5.2.7 Complexity

In terms of space complexity, although AFC requires additional fields to store credit information and more memory to store byte values rather than packet values, overall it still needs only small size of a HBF cache memory to hold the candidates of high bandwidth unresponsive flows, in a similar way as BLACK. The experiments show that with a small memory, enough to hold the information of 20 long-lived flows, AFC could achieve very good fairness performance even with the presence of 10,000 web sessions in background (Section 5.2.5.1).

Computational complexity is also not high and practical to be deployed. Rather than searching through the HBF cache memory to access any recorded item, the cache memory can be implemented as a linked list with a hash table as an external index. In this manner the complexity to access the item in the cache memory would be only O(1). All the computations, i.e. $HitFraction$ and a dropping probability, are only performed once per an arrival of a packet *that only are from a flow whose record is stored in the cache memory*. This could be less complex than multiple hashing with different functions of SFB and significantly less complex than the calculation of CARE. For AFC, it requires one extra computation for the credit information according to Equation 5.2. However, because the calculation is only needed for those flows that have their information stored in the HBF cache memory which is, in general, merely a fraction of the total number of active flows in the real network.

Besides, with the Direct Bitmap method, the number of active flows is calculated with small amount of memory as previous described in Chapter 4 and low CPU overhead using one simple equation (Equation **??**) once after each period.

For SFB, substantial amount of memory is required, although relatively low, as not only multiple levels of hash tables are required, double moving hash function that prevents high-bandwidth traffic from being classified as misbehaving traffic forever would double the amount of memory. SFB also requires a search through multiple levels of hash tables for the minimum dropping probability to be dropped for each flow. On the other hand, CARE is the most complex scheme. As discussed in the previous chapter, CARE requires a large amount of memory (large capture size) to store enough information in order to have an accurate estimation of the number of active flows. In addition, a large capture size implies a much higher complexity in order to build the capture frequency table and calculate the coefficients with complex equations using the Jackknife estimator.

5.3 MULTI-HOP FAIRNESS

This section provides a brief discussion that BLACK or AFC not only provide per-hop fairness discussed in all of the previous sections, but also provide fairness in multi-hop scenario. Multi-hop fairness is to ensure that when different traffic flows are from different sources and to different destinations, a fair AQM scheme does not blindly regulate traffic flows to a fair share while resources may be plentiful in some link of the end-to-end paths associated to those flows. A parking lot scenario according to RIAS (Ring Ingress-Aggregated with Spatial Reuse) fairness reference model [40] can be used to demonstrate multi-hop fairness.

A parallel parking lot scenario is shown in Figure 78. The flows in dash line share the bandwidth of the same bottleneck link from node 4 to node 5, thus each of them receive an equal share or 25% of the link bandwidth. Since one of these flows to node 5 is originated from node 1, it shares the link connecting node 1 and node 2 with the traffic showing in a solid line flowing from node 1 to node 2. If node 1 strictly provides fairness to these two flows (one from node 1 to node 5 and another one from node 1 to node 2) without considering the available resource, the flow in a solid line should also receive only 25% of the 1-2 link bandwidth. However, providing fairness in that way would only result in a link underutilization. Ideally, a flow in a solid line should receive

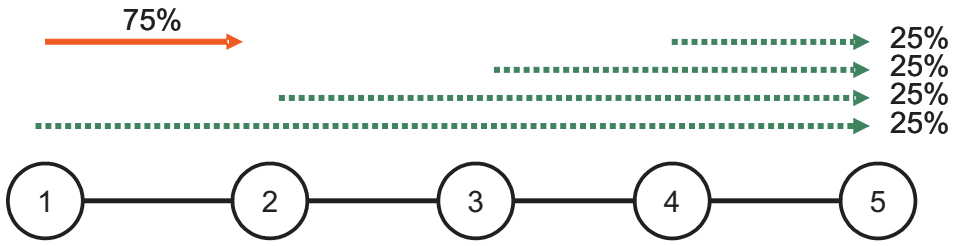


Figure 78: Parallel parking lot scenario.

75% of a link bandwidth.

BLACK's and AFC's operation fit well in this scenario as both of them only exercise their fairness mechanism only when congestion occurs; no packets are dropped if the link is not fully utilized or with empty queue. To support this statement, a simulation is set up using the parking lot scenario showing in Figure 78, where the links' bandwidth are set to 10 Mbps. A flow from node 1 to 5 has an arrival rate of 2.5 Mbps, or 25% of the link bandwidth. A flow from node 1 to 2 has an arrival rate of 10 Mbps or 100% of the link bandwidth. The result in Figure 79 shows that with AFC, the flow from node 1 to 2 receive a bandwidth of approximately 7.5 Mbps, or 75%, according to the ideal case described above[4]. This experiment is to show that BLACK and AFC could perform effectively, not only in terms of per-hop fairness, but also multi-hop fairness.

5.4 SUMMARY

In this chapter, AFC is proposed as an alternative fair AQM solution apart from BLACK scheme previously discussed in Chapter 3. AFC aims to provide better fairness and overcome the limitations of BLACK, and the other schemes, by including several newly designed components. First, AFC collects the packet size information so a byte count is performed rather than a Hit count, so that the unresponsive flows with different packet sizes would receive an equal amount of bandwidth. Second, Direct Bitmap or its variants, an alternative light-weighted estimator for the number of active flows discussed in Chapter 4, is used to improve the estimation accuracy. Third, the buffer occupancy fraction approximation is improved by collecting $HitFraction$ statistics when

[4]BLACK achieves the same result as AFC, so only the result of AFC is shown here.

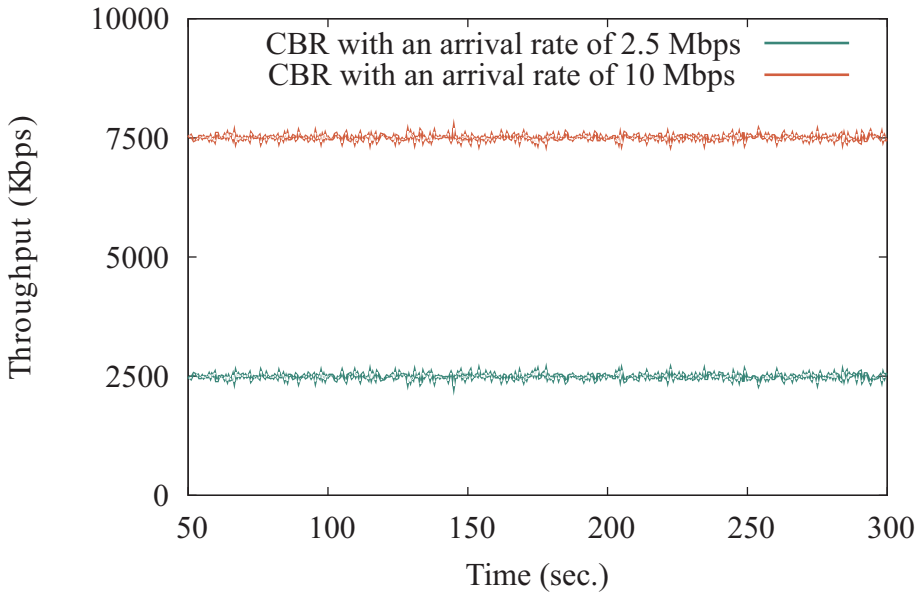


Figure 79: Simulation result of the parallel parking lot scenario; showing the throughput obtained by the flow from node 1 to node 2 (10 Mbps) and the flow from node 1 to node 5 (2.5 Mbps).

the packets enter the queue, rather than sampled from the queue. Lastly, a new dropping policy using a credit-based mechanism is introduced to provide a better fairness. With the credit-based mechanism, the packets are allowed to enter the queue even if a $HitFraction$ is higher than a $FairFraction$, if a flow has available credit left, such as from a previous back-off period of a responsive traffic. In this way, responsive flows would not be over-penalized with a more aggressive dropping function, while fairness is achieved.

The second half of this chapter presents the the simulation study on the performance of AFC comparing to the other AQM schemes including RED, CHOKe, SFB, CARE, and BLACK. Here, BLACK is modified such that the estimation of the number of active flows is replaced with Direct Bitmap, for a fair comparison with AFC. The comparison was conducted under different scenarios and the results can be summarized as follows.

In a scenario with streaming unresponsive traffic, AFC outperforms all the other AQM schemes in providing throughput fairness when high-bandwidth unresponsive traffic compete with long-lived TCP traffic over a bottleneck link. Even when the queue is attacked by multiple high-bandwidth unresponsive traffic, AFC still protect TCP traffic from being shut down and in fact

give each TCP connection a bandwidth approximately equal to a fair throughput. CHOKe and CARE could only provide some level of fairness when the arrival rate of unresponsive is mild or moderate. CARE's estimation of a number of active flows is inaccurate when an arrival rate of unresponsive traffic is high comparing to the aggregate arrival rate. SFB, on the other hand, could achieve fairness only when its rate limit threshold is well adjusted which could only be done manually, or a separate queue is needed to treat the detected misbehaving traffic. Although BLACK provides similar fairness performance as AFC, its dropping policy usually causes detected flows to gain somewhat more than a fair share of bandwidth. In contrast, AFC provides better fairness and does not introduce high throughput fluctuation to streaming traffic, particularly CBR traffic. Besides, AFC is the only scheme that could provide fairness when different unresponsive traffic come with different packet sizes.

For the unfairness problem among TCP connections with different round-trip delay, BLACK and AFC help reducing per-flow throughput difference among flows. However, SFB does the opposite because its mechanism provides a rate limit only to those flows that are detected as being unresponsive. Undetected flows would be controlled by a dropping function that does not guarantee fairness.

For a short-lived traffic scenario, SFB, CARE, BLACK and AFC fairness performance are not interfered with different web traffic loads in the background. In other words, different elephant (large) flows (both unresponsive and responsive flows) receive equal share of bandwidth without any interference from mice (small) flows, according to mice and elephants model previously discussed in Section 3.2. CHOKe, on the other hand, has inferior performance as not only CBR traffic gain higher throughput with more web traffic load, but fairness among TCP traffic are also significantly degraded. The results for high-bandwidth short-lived traffic show that all of these schemes cannot keep the bursty traffic down to a fair share because of their limited memory space and the way they keep flow information. However, these AQM schemes could still protect TCP traffic by providing adequate amount of bandwidth if the arrival rate of the bursty traffic is not very high. This experiment also confirms the problems of CARE and SFB – CARE's estimation of the number of active flows is inaccurate when the arrival rate of unresponsive traffic is high, and SFB needs a manual adjustment of a rate limit threshold for each scenario, which is more difficult for bursty traffic, in order to provide a good fairness performance.

In the last section of experiment, many TCP-friendly protocols which aims to achieve about the same amount throughput as TCP connection, while providing less throughput fluctuation for media applications, the fair AQM schemes work well in their design environment such as under RED or Tail Dropping. But under some schemes, fairness between TCP-friendly traffic and TCP traffic may not be achieved. SFB has a problem selecting an appropriate setting for its parameters under different scenarios to provide good fairness. CHOKe and CARE have found to be too aggressive for some sensitive TCP variants such as Reno TCP, and thus TCP-friendly traffic which is usually more robust gain much higher bandwidth under these AQM schemes.

In summary, BLACK with Direct Bitmap and AFC are the best overall fair AQM schemes that provide throughput fairness in a large number of different scenarios, both per-hop and multi-hop case. Comparing to BLACK, AFC provides better fairness in almost all scenarios, much less throughput fluctuation, and an ability to handle bursty traffic and traffic with different packet sizes. Nevertheless, AFC requires higher overhead than BLACK as it needs to calculate a flow's credit and needs an additional field to hold this information for the flows that are stored in the HBF cache. An implementer has a choice to deploy BLACK or AFC based on the trade-off between the advantages of AFC and its overhead.

6.0 CONCLUSIONS

6.1 CONTRIBUTIONS

In this dissertation, the Internet's current problems of potential bandwidth unfairness and congestion collapse have considered. Without any network mechanism to prevent it, unresponsive traffic can gain much more bandwidth than is a fair share, and it could even shut down responsive traffic sharing the same bottleneck link. To avoid high complexity and memory consumption, Active Queue Management (AQM) with no or partial state information is considered. Existing fair AQM schemes of this type have limitations which could be problematic for the provision of fairness in any of several scenarios or if widely deployed. To address the drawbacks of these mechanisms, two AQM schemes are proposed – BLACK and AFC. Both of them utilize only a small amount of memory to handle high-bandwidth unresponsive flows and keep their bandwidth usage to about a fair share. The superiorit of the two schemes have been demonstrated through a wide range of simulation experiments. Specifically, the major contributions of this dissertation are as follows:

- This paper provides an investigation of the existing fair AQM schemes that combat the unfairness problem using no or partial state information. These schemes provide some level of fairness in some scenarios, but they fail to achieve anything close to fairness in others. Both qualitative and quantitative information are provided in this examination of their limitations [10].
- This paper develops a novel fair AQM mechanism, called BLACKlisting unresponsive flows (BLACK). This mechanism aims to provide fairness by enabling equitable sharing of buffer space by the active flows. Based on the fact that most Internet traffic is carried by a small number of flows, BLACK uses only a small amount of memory to detect and control unresponsive

traffic as well as to achieve better fairness among TCP traffic flows with different round-trip delays [9].

- This paper presents a comparative study of the means of estimating the number of active flows. This estimation is crucial for some fair AQM schemes, including BLACK. In a large number of simulation experiments performed for several estimation algorithms, Direct Bitmap utilizes the lowest amount of memory with very low computational complexity. The results of this study could apply not only to a fair AQM scheme; it would also benefit other applications that operate with low memory resource and CPU overhead [10].
- This paper develops the Achieve Fairness using a Credit-based (AFC) mechanism. While AFC shares some of the concepts of BLACK, it has several newer components that overcome drawbacks and limitations of existing schemes including BLACK. Not only does AFC handle heavy unresponsive flows better, it also improves fairness of network bandwidth distribution among TCP connections through different round-trip delays. It achieves good fairness even under conditions of bursty traffic and when handling traffic with different packet sizes. In addition, AFC provides smoother transfer rates for unresponsive flows that are usually transmitting real-time traffic.
- This paper presents the first comprehensive performance evaluation of the fairness of different fair AQM schemes for TCP-compatible protocols and TCP variants. We found that several AQM schemes that provide good fairness performance when unresponsive traffic co-exists with TCP traffic might cause inferior fairness performance when TCP-friendly traffic designed to fairly share the bandwidth with TCP traffic are included [11].

6.2 SUMMARY

At present, TCP is the de facto standard protocol that is widely deployed on the Internet due to the success of its congestion control mechanism which enables end hosts to cooperatively adjust their transmission rates according to network conditions and, thus, share the available bandwidth fairly among a large number of users. However, streaming media traffic is experiencing tremendous growth. Real-time applications using the UDP protocol are typically considered to be *unresponsive traffic* because UDP provides no end-to-end congestion control. These applications tend not only to

generate more traffic, but they also do not back off in response to network congestion. As a result, when TCP and UDP-based applications share the same bottleneck link, the unfairness problem may arise. In a severe case, a congestion collapse problem can occur [25].

Recently, a wide variety of applications and congestion control mechanisms have emerged, and these may create an uncooperative environment for end hosts. As a result, the Internet's reliance on end-host mechanisms to prevent unfairness and congestion is potentially risky for network performance and stability. Consequently, router-based mechanisms that address the fairness problem have been widely investigated over the past several years.

Several mechanisms have been proposed to ensure fair shares of bandwidth to competing flows. Among the most important ones are fair scheduling mechanisms or fair per-flow packet dropping techniques such as Longest Queue Drop (LQD) [68], Fair Random Early Detection (FRED) [45], and Balanced-RED (BRED) [4]. These schemes are based on per-flow information. However, they usually require a considerable amount of memory and CPU processing power; these demands have prevented them from being widely deployed due to scalability and complexity concerns.

Instead, recent trends to solve the unfairness problems focus on fair active queue management (FAQM) schemes that maintain no or partial state information in order to track and regulate high-bandwidth or misbehaving flows. Keeping only partial state information is possible due to the fact that most of the Internet traffic is carried by only a small number of connections, while the remaining large number of connections are low bandwidth flows [29, 38]. Recently, several lightweight fair AQM schemes have been proposed. These have low computational and space complexity, and they don't need extra cooperation from the devices at the edges of the network in order to achieve long-term fairness. These schemes include CHOKe [52], Stochastic Fair Blue (SFB)[19], and CARE [8]. Although the literature suggests that most of these schemes provide good fairness, they still present several problems in different scenarios.

BLACK is the first AQM scheme proposed in this research that provides good fairness performance using a small amount of memory, while overcoming the limitations of the existing schemes. As explained separately by Suter et al. [68] and Laksham and Madhow [42], by controlling the share of buffer space used by the active flows, throughput fairness can be achieved through a FIFO queueing discipline using a memory management mechanism similar to LRU technique [38]. In the BLACK scheme, only large flows such as those that usually cause congestion are tracked. The

fraction of buffer space used by these flows, referred to as $HitFraction$, is monitored. If a flow utilizes a more than a fair share of, the packets of the flow are dropped according to how much extra buffer space is occupied by the flow. A fair proportion of the bandwidth is an inverse value of the number of active flows calculated by the estimation and this is determined by one of the modules in the BLACK scheme. The results from the simulation experiments show that BLACK outperforms the other schemes (i.e. CHOKe, SFB, and CARE) in terms of throughput fairness performance, especially in a scenario in which unresponsive traffic has a high arrival rate compared to the link bandwidth. BLACK also reduces unfairness among TCP connections with different round-trip delays.

Nevertheless, BLACK still has some limitations of achieving fairness in some scenarios, such as unfairness due to an inaccuracy in the estimation of the number of active flows, unfairness when packets are of different sizes, and high variance in the throughput of the flows. These problems, which are not unique to BLACK but also the other schemes, might cause a degradation of their fairness performance. These issues lead to an alternative mechanism to estimate the number of active flows and a development of another fair AQM scheme, AFC.

BLACK's original estimation of the number of active flows is derived from a false assumption that all incoming traffic is of similar intensity. Besides, small buffer size could add up more inaccuracy. Two alternative approaches were recently proposed in the literature: the bitmap approach and the CR-model approach. In extensive experiments that have been conducted, the bitmap approach has been shown to be reliable under a wider range of scenarios than the CR-model approach, which is used by the CARE mechanism. In addition, the bitmap approach is far less complex and uses much less memory space than the CR-model approach.

AFC, Achieve Fairness using a Credit-based mechanism, has been developed to overcome the limitations of BLACK and the other schemes while requiring only a little more overhead than BLACK. AFC includes *Direct Bitmap*, a simple bitmap approach to estimating the number of active flows, and it is enhanced with other new components, such as the *credit-based mechanism*. This mechanism helps AFC provide better fairness in a wide range of scenarios without being overly aggressive to responsive traffic. Simulation results show that AFC not only provide better fairness than other schemes, even when handling multiple high-bandwidth traffic. AFC also provides smoother transfer rates for unresponsive flows that are usually transmitting real-time traffic.

While the other schemes cannot achieve fairness for traffic flows with different packet sizes, AFC approaches the problem collecting $HitFraction$ statistics at byte level rather than at packet level. Another experiment shows that AFC and BLACK can also reduce the unfairness among long-lived TCP connections with different round-trip delays. Other schemes, such as SFB, provide good fairness with unresponsive traffic when there is a well-tuned rate limit, but fail to reach good fairness performance in this case. In a realistic scenario, a variety of web traffic in the background does not interfere with the fairness performance of AFC and BLACK, but it does significantly degrade the performance of CHOKe. In the case of bursty traffic, although these AQM schemes may not be able to provide perfect service since they do not maintain long-term state information, AFC still manages to provide a reasonably fair share of bandwidth, even in this case. Overall, AFC has been shown to be the most effective fair AQM scheme in terms of providing fairness under a wide range of scenarios.

This research is also the first comprehensive performance evaluation of the fairness of different fair AQM schemes in the presence of TCP-compatible protocols and TCP variants. While aggressive TCP-compatible traffic flows obtain substantially higher bandwidth than TCP traffic flows when they co-exist under RED or droptail, the shares of bandwidth are much more equal under BLACK and AFC than under other fair AQM schemes. For TCP-friendly protocols with congestion control mechanisms comparable to TCP, the fair AQM mechanisms provide, at best, only a small improvement over RED in a number of cases. Inferior fairness performance can occur for CHOKe, CARE, and SFB. CHOKe has been found to be too aggresive in dropping packets, since it drops multiple packets at the same time and might create more unfairness for responsive traffic flows that are sensitive to multiple drops. CARE has a similar problem. Without a proper parameter tuning, SFB produces unexpectedly inferior results, even in a scenario with TCP-friendly protocols with congestion control comparable to TCP. In other words, some fair AQM schemes might cause even poorer fairness performance than RED or droptail in the presence of TCP-friendly traffic flows that were originally designed to share bandwidth with TCP traffic in a fair manner. However, with little extra complexity, much better fairness is obtained with BLACK or AFC. In conclusion, an AQM scheme must be chosen not only based on its performance or capability to deal with unresponsive flows, but also based on its performance when other types of flows are included, such as TCP-friendly sources.

In addition, Appendix B includes a brief discussion of an alternative method of controlling traffic in a best-effort IP network. BLACK with a little modification is shown to be capable of providing service with a different policy - such as lower packet drops - to high-bandwidth traffic flows given that they do not utilize a bandwidth over a certain threshold.

6.3 FUTURE RESEARCH

In this dissertation, BLACK and AFC have demonstrated superior performance over existing non-perflow fair AQM schemes in a large number of scenarios. However, it would be interesting to find out if further performance improvement is possible. For example, these fair AQM schemes cannot achieve the same level of fairness with bursty traffic flows as they can with the other types of traffic. An additional mechanism might be needed to better detect bursty traffic flows; alternatively, keeping the flows in a cache memory may be a smarter way to handle the difficult ON-OFF behavior of this type of traffic.

Underlying AQM mechanisms of BLACK and AFC could also be replaced by a scheme better than RED. RED has several advantages over the drop-tail mechanism in its ability to signal congestion to responsive sources at an early stage, thus preventing a problem of global synchronization and keeping the average queue size low. However, the way RED estimates the average queue size can cause instability in the network, as detailed by Arce et al. [5]. These drawbacks could be overcome either by a technique in the authors suggest [5] or by replacing the underlying RED queue with another AQM scheme, such as PI Controller [31], that provides better control of queue lengths. The performance of BLACK and AFC with new underlying AQM mechanism may or may not be better because the characteristics of the traffic after passing through their dropping policies could be much different than the traffic model assumed by control-theoretical based AQM schemes such as PI Controller.

It would also be interesting to apply BLACK or AFC to a best-effort class in a DiffServ network. Even though DiffServ architecture provides guarantee services in EF and AF classes and non-guaranteed service in a best-effort class, a network operator may not want to leave a best-effort service entirely unregulated. As traffic flows in AF class have higher priority for gaining available bandwidth, congestion can easily occur in a best-effort class. In this circumstance, high

bandwidth unresponsive traffic can still gain a high proportion of bandwidth, leaving responsive traffic to suffer with even more severe congestion. Thus, the problem of unfairness could occur in a best effort class. Some side problems are also possible. For example, with the incentive that unresponsive traffic could gain as much bandwidth as possible and the ability of newer streaming media applications to adapt to network congestion, some traffic flows that should be in AF class may class themselves in a best-effort class in order to grasp most of the bandwidth and avoid possible higher billing cost in AF class. These problems of unfairness, and whether or not BLACK and AFC could be good answers to them, should be further investigated.

BIBLIOGRAPHY

[1] "NS Network Simulator," Available from http://www.isi.edu/nsnam/ns/.

[2] "The Network Simulator: Contributed Code," Available from http://www.isi.edu/nsnam/ns/ns-contributed.html.

[3] E. Altman, K. Avrachenkov, and C. Barakat, "A Stochastic Model of TCP/IP with Stationary Random Losses," in *Proc. SIGCOMM*, 2000, pp. 231–242.

[4] F. Anjum and L. Tassiulas, "Fair Bandwidth Sharing Among Adaptive and Non-Adaptive Flows in the Internet," in *Proc. IEEE INFOCOM*, Mar. 1999, pp. 1412–1420.

[5] G. R. Arce, K. E. Barner, and L. Ma, "RED Gateway Congestion Control Using Median Queue Size Estimates," *IEEE Transactions on Signal Processing*, vol. 51, no. 8, pp. 2149–2164, Aug. 2003.

[6] D. Bansal and H. Balakrishnan, "Binomial Congestion Control Algorithms," in *Proc. IEEE INFOCOM*, Apr. 2001, pp. 631–640.

[7] K. P. Burnham and W. S. Overton, "Estimation of the Size of a Closed Population When Capture Probabilities Vary Among Animals," *Biometrika*, vol. 65, pp. 25–633, 1978.

[8] M. Chan and M. Hamdi, "An Active Queue Management Scheme Based on a Capture-Recaptured Model," *IEEE J. Select. Areas Commun.*, vol. 21, no. 4, pp. 352–583, May 2003.

[9] G. Chatranon, M. A. Labrador, and S. Banerjee, "BLACK: Detection and Preferential Dropping of High Bandwidth Unresponsvie Flows," in *Proc. IEEE ICC*, May 2003, pp. 664–668.

[10] G. Chatranon, M. A. Labrador, and S. Banerjee, "A Survey of TCP-Friendly Router-based AQM Schemes," *Computer Communications*, 2004, To appear.

[11] G. Chatranon, M. A. Labrador, and S. Banerjee, "Fairness of AQM Schemes for TCP-friendly Traffic," in *Proc. IEEE Globecom*, Nov. 2004, To appear.

[12] K. Claffy, G. Miller, and K. Thompson, Eds., *The Nature of the Beast: Recent Traffic Measurements from the Internet Backbone*. Proceedings of Inet, 1998, Available from http://www.caida.org/outreach/papers/1998/Inet98/.

[13] D. Clark and Wenjia Fang, "Explicit Allocation of Best-Effort Packet Delivery Service," *IEEE/ACM Trans. Networking*, vol. 6, no. 4, pp. 362–373, August 1998.

[14] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," in *Proceedings of ACM SIGCOMM'89*, August 1989.

[15] C. Estan, G. Varghese, and M. Fisk, "Bitmap Algorithms For Counting Active Flows on High Speed Links," in *Proceedings of Internet Measurement Conference*, Oct. 2003.

[16] C. Estan, G. Varghese, and M. Fisk, "Bitmap Algorithms for Counting Active Flows on High Speed Links," Tech. Rep. CS2003-0738, UCSD, Mar. 2003, Available from http://www.cs.ucsd.edu/users/cestan/papers/countingdistincttechreport.pdf.

[17] A. Feldmann, A. C. Gilbert, P. Huang, and W. Willinger, "Dynamics of IP Traffic: A study of the role of variability and the impact of control," in *Proc. ACM SIGCOMM*, Sept. 1999.

[18] W. Feng, "Blue and Stochastic Fair Blue," Available from http://www.thefengs.com/wuchang/work/blue/.

[19] W. Feng, D. Kandlur, D. Saha, and K. Shin, "Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness," in *Proc. IEEE INFOCOM*, April 2001, pp. 1520–1529.

[20] W. Feng, D. Kandlur, D. Saha, and K. Shin, "Blue An Alternative Approach To Active Queue Management," Aug. 2002, vol. 10, pp. 512–528.

[21] S. Floyd, "TCP and Explicit Congestion Notification," *ACM Computer Communication Review*, vol. 24, no. 5, pp. 10–23, 1994.

[22] S. Floyd, "TCP and Successive Fast Retransmits," Tech. Rep., Feb. 1995, Available from http://www.icir.org/floyd/papers/fastretrans.ps.

[23] S. Floyd, "Recommendation on using the "gentle_" variant of RED," 1999, Available from http://www.icir.org/floyd/red/gentle.html.

[24] S. Floyd and K. Fall, "Router Mechanisms to Support End-to-End Congestion Control," *LBL Technical report*, February 1997.

[25] S. Floyd and K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet," *IEEE/ACM Trans. Networking*, vol. 7, no. 4, pp. 458–472, Aug. 1999.

[26] S. Floyd, M. Handley, and J. Padhye, "A Comparison of Equation-Based and AIMD Congestion Control," Feb. 2000, http://www.aciri.org/tfrc/.

[27] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-Based Congestion Control for Unicast Applications," in *Proc. ACM SIGCOMM*, Stockholm, Sweden, Aug. 2000, pp. 43–56.

[28] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Trans. Networking*, vol. 1, no. 4, pp. 397–413, Aug. 1993.

[29] L. Guo and I. Matta, "The War Between Mice and Elephants," Tech. Rep. 2001-005, Computer Science Department, Boston University, May 2001.

[30] L. Guo and I. Matta, "The War Between Mice and Elephants," in *Proc. 9th IEEE International Conference on Network Protocols (ICNP'01)*, Riverside, CA, Nov. 2001.

[31] C. V. Hollot, V. Misra, D. F. Towsley, and W. Gong, "On Designing Improved Controllers for AQM Routers Supporting TCP Flows," in *Proc. IEEE INFOCOM*, 2001, pp. 1726–1734.

[32] D. P. Hong, C. Albuquerque, C. Oliveira, and T. Suda, "Evaluating the Impact of Emerging Streaming Media Applications on TCP/IP Performance," *IEEE Communications*, vol. 39, no. 4, pp. 76–82, Apr. 2001.

[33] V. Jacobson, "Congestion Avoidance and Control," in *Proc. ACM SIGCOMM*, August 1988, pp. 314–329.

[34] V. Jacobson, "Modified TCP Congestion Avoidance Algorithm," *End-to-End mailing list*, Apr. 1990, Available from ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail.

[35] R. Jain, A. Durresi, and G. Babic, "Throughput Fairness Index: An Explanation," *ATM Forum/99-0045*, Feb. 1999.

[36] K. Keating, C. Schwartz, and D. Moody, "Estimating Numbers of Females with Cubs-of-the-year in the Yellowstone Grizzly Bear Population," *Ursus*, vol. 13, pp. 161–174, 2002.

[37] S. Keshav, *An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network*, Addison-Wesley, Massachusetts, September 1997.

[38] I. Kim, "Analyzing Network Traces to Identify Long-Term High Rate Flows," M.S. thesis, Texas A&M University, May 2001.

[39] S.-W. Kim and W.-K. Whang, "On analyzing errors in a selectivity estimation method based on dynamic maintenance of data distribution," *Information & Software Technology*, vol. 43, no. 4, pp. 265–274, 2001.

[40] E. W. Knightly, "RIAS Fairness Reference Model," Available from http://www.ece.rice.edu/networks/RIAS.

[41] H. T. Kung, T. Blackwell, and A. Chapman, "Credit-Based Flow Control for ATM Networks: Credit Update Protocol, Adaptive Credit Allocation, and Statistical Multiplexing," in *Proc. ACM SIGCOMM*, Aug. 1994, pp. 101–114.

[42] T. Lakshman and U. Madhow, "The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss," *IEEE/ACM Trans. Networking*, vol. 5, no. 3, pp. 336–350, July 1997.

[43] T. V. Lakshman, A. Neidhardt, and T. Ott, "The Drop from Front Strategy in TCP and in TCP over ATM," in *Proceedings of IEEE Infocom*, 1996, pp. 1242–1250.

[44] S. Lee and A. Chao, "Estimating Population Size Via Sample Coverage for Closed Capture-Recapture Models," *Biometrics*, vol. 50, pp. 88–97, 1994.

[45] D. Lin and R. Morris, "Dynamics of Random Early Detection," in *Proc. SIGCOMM*, Sept. 1997, pp. 127–137.

[46] J. Mahdavi and S. Floyd, "TCP-Friendly Unicast Rate-Based Flow Control," *Note Sent to End-to-End Mailing List*, Jan. 1997, Available from http://www.psc.edu/networking/papers/tcp_friendly.html.

[47] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm," *Computer Communication Review*, vol. 27, no. 3, July 1997.

[48] T. Ott, T. Lakshman, and L. Wong, Eds., *SRED: Stabilized RED*. IEEE INFOCOM, 1999.

[49] T. J. Ott, J. H. B. Kemperman, and M. Mathis, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm," *Computer Communication Review*, vol. 27, no. 3, July 1997.

[50] J. Padhye, V. Firoiu, and D. Towsley, "A Stochastic Model of TCP Reno Congestion Avoidance and Control," Tech. Rep. 99-02, Department of Computer Science, Univ. of Massachusetts, Amherst, MA, Jan. 1999.

[51] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," in *Proceedings of ACM SIGCOMM'98*, Vancouver, BC, September 1998.

[52] R. Pan, B. Prabhakar, and K. Psounis, "CHOKe - A Stateless Active Queue Management Scheme For Approximating Fair Bandwidth Allocation," in *Proc. IEEE INFOCOM*, April 2000, pp. 942–951.

[53] K. Papagiannaki, N. Taft, S. Bhattacharyya, P. Thiran, K. Salamatian, and C. Diot, "A Pragmatic Definition of Elephants in Internet Backbone Traffic," in *ACM SIGCOMM Internet Measurement Workshop*, Marseilles, France, Nov. 2002.

[54] K. Psounis, R. Pan, and B. Prabhakar, "Approximate Fair Dropping for Variable-Length Packets," *IEEE Micro*, vol. 21, no. 1, pp. 48–56, January/February 2001.

[55] K. Ramakrishnan and S. Floyd, "A Proposal to Add Explicit Congestion Notification (ECN) to IP," *RFC2481*, January 1999.

[56] K. K. Ramakrishnan and R. Jain, "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks," *ACM Transactions on Computer Systems*, vol. 8, no. 2, pp. 158–181, 1990.

[57] R. Rejaie, M. Handley, and D. Estrin, "RAP: An End-to-End Rate-Based Congestion Control Mechanism for Realtime Streams in the Internet," in *Proc. IEEE INFOCOM*, Mar. 1999, pp. 1337–1345.

[58] I. Rhee, V. Ozdemir, and Y. Yi, "TEAR: TCP Emulation at Receivers – Flow Control for Multimedia Streaming," Tech. Rep., NCSU Technical Report, Apr. 1999.

[59] J. H. Salim and U. Ahmed, "Performance Evaluation of Explicit Congestion Notification (ECN) in IP Networks," December 1999, Available from http://www7.nortel.com:8080/CTL/ecnperf.pdf.

[60] M. Shreedhar and George Varghese, "Efficient Fair Queueing Using Deficit Round Robin," in *Proc. IEEE SIGCOMM*, 1995, pp. 231–242.

[61] B. Sikdar, S. Kalyanaraman, and K. Vastola, "Analytic Models for the Latency and Steady-state Throughput of TCP Tahoe, Reno and SACK," in *Proc. IEEE GLOBECOM*, Nov. 2001.

[62] D. Sisalem, H. Schulzrinne, and F. Emanuel, "The Direct Adjustment Algorithm: A TCP-Friendly Adaptation Scheme," Technical report, GMD-FOKUS, August 1997, Available from http://www.fokus.fraunhofer.de/research/cc/mobis/employees/dorgham.sisalem/.

[63] D. Sisalem and A. Wolisz, "LDA+ TCP-Friendly Adaptation: A Measurement and Comparison Study," in *the 10th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'2000)*, Chapel Hill, NC, USA, June 2000.

[64] K. S. Siyan, *Inside TCP/IP*, New Riders Publishing, Indianapolis, 1997.

[65] W. Stallings, *High-Speed Networks: TCP/IP and ATM Design Principles*, Prentice Hall, Upper Saddle River, NJ, 1998.

[66] Ion Stoica, Scott Shenker, and Hui Zhang, "Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks," in *Proc. SIGCOMM*, 1998, pp. 118–130.

[67] T. Suda, C. Albuquerque, and B. Vickers, "Fair Queuing with Feedback-Based Policing: Promoting Fairness and Preventing Congestion Collapse in the Internet," Tech. Rep., UCI-ICS Technical Report 99-26, University of California, Irvine, USA, September 1998.

[68] B. Suter, T. Lakshman, D. Stiliadis, and A. Choudhury, "Design Considerations for Supporting TCP with Per-Flow Queueing," in *Proc. IEEE INFOCOM*, April 1998, pp. 299–306.

[69] A. S. Tanenbaum, *Computer Networks*, Prentice Hall, Upper Saddle River, NJ, third edition, 1996.

[70] J. Widmer, R. Denda, and M. Mauve, "A Survey on TCP-Friendly Congestion Control," *IEEE Network*, May 2001.

[71] Y. Yang and S. Lam, "General AIMD Congestion Control," Tech. Rep. TR-200009, Department of Computer Science, University of Texas at Austin, May 2000.

[72] I. Yeom and A. L. N. Reddy, "Modeling TCP Behavior in a Differentiated Services Network," *IEEE/ACM Transactions on Networking*, vol. 9, no. 1, pp. 31–46, 2001.