WHITE - Achieving Fair Bandwidth Allocation with Priority Dropping Based on Round Trip Time

by

Choong-Soo Lee

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

by

_____

May 2002

APPROVED:

_____

Professor Mark Claypool, Major Thesis Advisor

_____

Professor Robert Kinicki, Major Thesis Advisor

_____

Professor Craig Wills, Thesis Reader

_____

Professor Micha Hofri, Head of Department

Abstract

Current congestion control approaches that attempt to provide fair bandwidth allocation among competing flows primarily consider only data rate when making decisions on which packets to drop. However, responsive flows with high round trip times (RTTs) can still receive significantly less bandwidth than responsive flows with low round trip times. This paper proposes a congestion control scheme called WHITE that addresses router unfairness in handling flows with significantly different RTTs. Using a best-case estimate of a flow's RTT provided in each packet by the flow source or by an edge router, WHITE computes a stabilized average RTT. The average RTT is then compared with the RTT of each incoming packet, dynamically adjusting the drop probability so as to protect the bandwidth of flows with high RTTs while curtailing the bandwidth of flows with low RTTs. We present simulation results and analysis that demonstrate that WHITE provides better fairness than other rate-based congestion control strategies over a wide-range of traffic conditions. The improved fairness of WHITE comes close to the fairness of Fair Queuing without requiring per flow state information at the router.

Contents

i

ii

List of Figures

iv

Chapter 1

Introduction

The Internet relies upon cooperation between TCP hosts and subnet routers to adjust source data rates in the presence of network congestion along the path of the TCP flow. Currently, drop-tail queue management is the primary queue mechanism used in Internet routers to indicate congestion to edge hosts. While drop-tail schemes are easy to implement and fit within the best-effort nature of the Internet, these routers distribute packet drops arbitrarily among competing flows.

RED [FJ93], the best known Active Queue Management (AQM) approach, uses randomization to distribute packet drops among flows in a manner proportional to their perceived share of the capacity on a bottlenecked link. While RED drops more packets from high bandwidth flows, the same packet drop rate is applied to all flows, regardless of the actual bandwidth used.

Many researchers have shown that end-to-end congestion control can be improved when bandwidth is allocated more fairly among flows [DKS90, She94]. Furthermore, congestion increases due to handling packets that never reach their destination may be the largest unresolved form of congestion collapse in the Internet today [FF99]. In times of heavy congestion, dropping an equal number of packets among all flows

1

regardless of round trip time (RTT) has the potential to increase the network capacity wasted on undelivered packets that have come a long way only to be dropped by a router before reaching their final destination. This inherent bias against flows with high round trip times decreases overall goodput [1].

For TCP-friendly flows, a flow's round trip time (as well as packet size and drop rate) is directly responsible for determining a flow's data rate [Flo91, PFTK98]. With TCP's congestion control algorithms, a flow's throughput varies inversely with RTT. Thus, especially for flows with low round trip times, small RTT differences can dramatically effect the number of packets dropped for a flow. Approaches that are designed for increased fairness over RED [LM97, SSZ98] do not consider round trip time in making packet dropping decisions and are unfair with respect to heterogeneous flows with wide variances in round trip times.
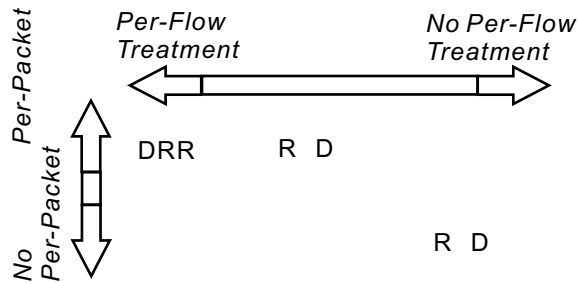
Figure 1.1: Classification of AQMs

There have been numerous approaches to achieving per flow fair bandwidth allocation at congested routers. As noted in [MFW01], there is a continuum of possible per-flow treatments, from complete per flow treatment such as in Fair Queuing, to a complete absence of per-flow treatment such as in drop-tail and RED. Additionally, for queuing mechanisms without per-flow treatment, there is a continuum of possible per-packet treatments, from no per-packet treatments such as in RED to complete

---

[1]As in [FF99], we define the goodput of a flow as the packet delivery rate at the receiver, excluding duplicate packets.

2

per packet treatments from CSFQ core routers. Figure 1.1 depicts the space of possible flow and packet treatment policies, with the approximate placements of policies evaluated in this thesis.

Random Early Detection (RED) [FJ93] keeps no per flow state information. Packets are dropped probabilistically based on the long-term average queue size and fixed indicators of congestion (thresholds). RED uses randomization to drop arriving packets to avoid biases against bursty traffic and roughly drops packets in proportion to the flows data rate at the router. However, flows with high RTTs and small window sizes are bursty, and this burstiness causes high variability in the perceived data rate of these flows as seen by RED routers.

At the other extreme, Deficit Round Robin (DRR) [SV95], a variant of Fair Queuing (FQ) [DKS90], keeps extensive information on every flow. DRR routers send packets approximately in the order a router would send them if packets could be sent one bit at a time. While DRR and other FQ variants achieve good fairness among flows, the per-flow state information required and overhead needed to manage priority queues is expensive. Moreover, these schemes do not scale well with increased number of flows. This study uses DRR as the best case scenario for achieving fairness among heterogeneous flows; namely the goal is to seek fairness comparable to DRR without DRR per-flow costs.

Flow Random Early Drop (FRED) [LM97] uses per-flow preferential dropping to achieve fairer allocation of bandwidth among flows. FRED builds per-flow state at the router by examining those packets that are currently in the queue. The packet drop rate for a flow is determined by the number of packets the flow has in the queue, and is not directly influenced by the flow's data rate or round trip time. We evaluate the effectiveness of FRED as a less expensive means of attempting per-flow fairness.

In Core Stateless Fair Queuing (CSFQ) [SSZ98], edge routers classify flows based on their current sending rate and forward these data rates as labels to core routers. Using these labels, core routers keep a running estimate of the fair share capacity of a flow on an out-going link. The core router drops packets in a manner aimed at giving each flow its fair share of the link throughput. However, such preferential dropping based on data rate alone is not sufficient to achieve fairness. Since the response function for TCP-friendly flows is based on the RTT, dropping packets equally between two flows with the same data rate but different round trip times will result in a higher long-term data rate for the flow with the lower round trip time. While it has been shown that CSFQ achieves fairness for flows with the same RTT, we demonstrate that it is ineffective in achieving fairness among flows with heterogeneous round trip times.

This thesis presents a new approach to fairness that takes into account a flows round trip time in determining a router's responsiveness to congestion avoidance. The primary goal of our work is to achieve fairness [2] among responsive flows with heterogeneous RTTs without negatively impacting overall router performance (i.e., goodput and drop rate). Providing bandwidth fairly to heterogeneous flows partially eliminates the need for Content Distribution Networks (CDNs) because users connecting from both local and remote locations will get the same bandwidth when downloading content. The benefit is more significant for servers who have typically many local clients and few remote clients because the few remote clients may not justify deployment of CDN due to deployment costs.

The framework for our approach, WHITE [3], is the same *edge* and *core* architec-

---

[2] We focus on *min-max fairness*, since it is easy to interpret locally and makes no assumptions about behaviors elsewhere in the network. For completeness, *Jain's fairness index* [Jai91] is also reported for all experiments.

[3] The name WHITE comes from a play on the acronym RED. Considering "red" as a type of wine, there is a family of "white wine" active queue management approaches.

ture presented in CSFQ [SSZ98] wherein core routers drop or mark packets using *hints* sent in packet labels by edge routers. In practice, the edge can be an ingress router or an end host and the hint can consist of the estimated data rate, as in CSFQ, the estimated window size, a delay tolerance, or any other flow attribute. In WHITE, the hint is an edge estimate of the best-case round trip time.

Using round trip time hints, the WHITE core router computes an average round trip time for all packets arriving at the router. When congestion is indicated by the RED thresholds, packets are dropped based on their round trip time in relation to the overall average round trip time. This mechanism preferentially drops packets with lower than average round trip times and favors packets with higher than average round trip times. This protects fragile flows with high RTTs while inhibiting robust flows with low round trip times from grabbing an unfair share of link bandwidth.

Through NS-2 [oCB] simulations, WHITE's effectiveness is demonstrated under a wide range of scenarios. These scenarios include mixes of flows with round trip times varying from 20 ms to 400 ms, scenarios with a disproportionately large numbers of flows in different round trip time clusters, equal clusters of flows in different round trip time clusters, and scenarios with drastic changes in the round trip times of active flows.

Our simulation results show that the WHITE strategy provides far superior fairness among flows than RED for all scenarios tested; achieves fairer link capacity allocation among flows than CSFQ and FRED under all scenarios; provides far better fairness than CSFQ and FRED in many scenarios; yields DRR equivalent fairness for most scenarios; and approximates DRR equivalent fairness for the other scenarios tested. These WHITE improvements in fairness are accomplished while providing performance similar to RED with respect to drop rate, goodput and throughput.

The rest of this thesis is organized as follows: Chapter 2 describes related work

that has been done; Chapter 3 focuses on the details and derivation of the Chardonnay and Chablis[4] mechanisms; Chapter 4 describes the setup and metrics used to evaluate WHITE; Chapter 5 describes the set of simulation experiments run to evaluate WHITE under a wide-range of conditions and includes analysis of the simulation results and detailed comparisons of the performance of WHITE with DRR, CSFQ, RED and FRED. Chapter 6 compares Chablis, the marking version of WHITE with Chardonnay, the dropping version of WHITE. Chapter 7 summarizes our findings and considers further extensions and future work.

---

[4]Chardonnay and Chablis are type of white wine. We named the dropping version of WHITE Chardonnay and the marking version of WHITE Chablis (a better white wine than Chardonnay) because Chablis can improve network goodput.

Chapter 2

Related Work

In this chapter, we present work related to WHITE. In Section 2.1, we discuss other active queue management techniques that have been proposed. In Section 2.2, we describe work related to edge hints and how they are used. In Section 2.3, we present work in modeling TCP behavior.

2.1 Active Queue Management

Active queue management (AQM) is a congestion avoidance mechanism implemented at the router to provide improvements over drop-tail queue management. Drop-tail simply fills up the buffer with incoming packets and drops packets when there is no room left in the buffer. In this section, we present four AQM techniques that have been attempted as improvements over drop-tail queue management.

2.1.1 RED

Drop-tail can cause the problem of global synchronization with TCP flows. Once the buffer is full at a drop-tail router, it drops all incoming packets until there is

7

space in the buffer. These dropped packets can be from many sources and the flows will all reduce their sending window at the same time upon receipt of indication of congestion in the network. Random Early Detection (RED) [FJ93] addresses the shortcoming of drop-tail queue management by removing the global synchronization and lowering queuing delay.

Unlike drop-tail queue management, RED uses average queue size to drop incoming packets probabilistically. RED introduces new parameters for its functionality: max_p, min_{th}, max_{th} and these are used as depicted in Figure 2.1.

```
on receiving packet p
if (q_avg >= max_th) then
    dropPacket(p, 1)
else
    if (avg >= min_th) then
        d = calcDropProbability(q_avg, min_th, max_th)
        dropPacket(p, d)
    else
        enQueuePacket(p)
```
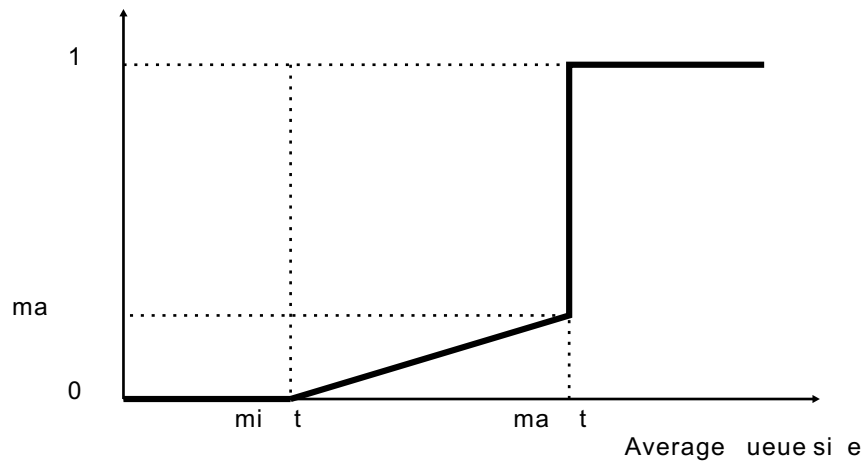

Figure 2.1: RED Algorithm

RED keeps track of the average queue size using an exponentially weighted av-

8

erage and uses it as an indication of congestion. As shown in Figure 2.1, the drop probability is directly related to the average queue size. An average queue size below min_{th} indicates no congestion and RED does not drop any packets. An average queue size between min_{th} and max_{th} indicates some level of congestion and RED drops packets with a linear drop probability between 0 and max_p. An average queue size above max_{th} indicates extreme congestion and RED drops all incoming packets.

RED also has capability to deal with ECN (Explicit Congestion Notification) [Flo94] enabled flows. ECN is an extension of TCP to allow congestion notification without drop of packets. ECN uses 2 bits in the IP header. One indicates if the flow is ECN enabled or not and the other tells if the packet experienced congestion or not. For ECN flows, RED marks the packets instead of dropping them when the average queue size is between min_{th} and max_{th}.

RED is one of the first active queue management techniques proposed and due to its simplicity, it is the base for developing WHITE algorithm.

2.1.2 FRED

As mentioned in Section 2.1.1, RED addresses shortcomings of drop-tail queue management but does not try to solve the problem of bandwidth fairness. Flow RED (FRED) [LM97] is a modification of RED to address the fairness issue. Unlike RED, FRED keeps track of per-flow state information. This means that FRED identifies flows and maintains information about each flow. FRED considers a flow active if there are any packets from the flow present in the queue.

Figure 2.2 is not complete but instead shows the differences from the RED algorithm in Figure 2.1. New parameters: $qlen_i$, min_q and $avgcg$. $qlen_i$ keeps track of how many packets from flow i are in the buffer. min_q is either 2 for small buffers or 4 for large buffers. $avgcg$ is the average per-flow queue size. Instead of just cal-

9

```
on receiving packet p
if (q_avg >= max_th) then
    dropPacket(p, 1)
else
    if (avg >= min_th) then
        if (qlen_i >= MAX(min_q, avgcg)) then
            d = calcDropProbability(q_avg, min_th, max_th)
            dropPacket(p, d)
    else
        enQueuePacket(p)
```

Figure 2.2: FRED Algorithm

culating the probability to drop the packet, FRED measures if the buffer usage of a flow is greater than the average per-flow queue size or min_q. If it is, then FRED calculates the probability to drop the packet. If not, FRED enqueues the packet.

FRED is one modification of RED algorithm to ensure bandwidth fairness. However, maintenance of per-flow state information is expensive and not desired at the router level. In this work, we compare the performance of FRED to WHITE as a comparison against another fairness approach.

2.1.3 CSFQ

FRED is a step towards achieving fairness but maintaining per-flow information is complex and does not scale well. Core-Stateless Fair Queuing (CSFQ) [SSZ98] is an approach to achieve fairness without the complexity of maintaining per-flow information at all rates.

While RED and FRED are driven by average queue size, CSFQ is controlled by rate of the flows. CSFQ distinguishes between edge and core routers. Edge routers are the routers at the boundaries of the network while core routers are internal to the network. Edge routers identify flows and estimate the rates of the flows and label the packets with these estimates. Core routers simply use drop-tail queue

10

```
on receiving packet p
if (edge router) then
    i = classify(p);
    p.label = estimate rate(r_i, p);
prob = max(0, 1 - α / p.label);
if (prob > unifrand(0,1))
    α = estimate α (p, 1);
    drop(p);
else
    α = estimate α (p, 0);
    enqueue(p);
if (prob > 0)
    p.label = α;
```

Figure 2.3: CSFQ Algorithm

management and employ a probabilistic dropping algorithm based on packet labels and each flow's rate estimate.

CSFQ provides a structure to separate edge and core routers so that core routers do not have to maintain per-flow state information by getting hints from edge routers. A similar structure is used for the WHITE architecture where edge routers or sources provide hints and core routers use these hints to treat all flows fairly. Plus, we compare the fairness in bandwidth achieved by WHITE to that achieved by CSFQ.

2.1.4 DRR

Deficit Round Robin (DRR) [SV95] is one implementation of fair queuing. DRR identifies flows and keeps a separate queue for each flow. When the overall queue is full, a packet from the queue of a flow with the most packets is dropped.

Figure 2.4 summarizes the DRR algorithm. Upon receiving a packet, it separates the packet into a flow and sees if it is an active flow at the router. If it is not, then it creates an active state for the flow. It enqueues packets normally until there is

11

```
on receiving packet p
i = Extract Flow (p)
if (ExistsInActiveList(i) == FALSE) then
    InsertActiveList(i)
if no free buffer left then
    FreeBuffer()
Enqueue(i, p);
```

Figure 2.4: DRR Algorithm

no space left and then it drops packets from the queue with the most packets in the buffer.

DRR, being an implementation of fair queuing, provides the best bandwidth fairness possible. Our goal is that WHITE provides bandwidth fairness as close as possible to DRR's.

2.1.5   AECN

Adaptive ECN (AECN) [Zhe01] is an extension of RED using marking. AECN attempts to treat heterogeneous flows fairly by classifying flows into three categories: robust, average and fragile. AECN keeps three virtual queues for each class.

Figure 2.5 summarizes the AECN algorithm. Once a packet comes in with a round trip time hint (described further in Section 2.2), AECN puts the packet in one of the three virtual queues depending on its round trip time. Based on the mark probability, AECN decides to mark the packet and finds the first unmarked packet in the virtual queue it is in.

WHITE's objective is to eliminate these classes and instead have completely dynamic assignment of drop probability to any packet with different round trip time hints.

12

```
on receiving packet p
if (q_avg >= max_th) then
    dropPacket(p, 1)
else
    if (avg >= min_th) then
        class = classify(p.RTT);
        d = calcDropProbability(q_avg, min_th, max_th, class)
        markPacket(d, class)
    else
        enQueuePacket(p, class)
```
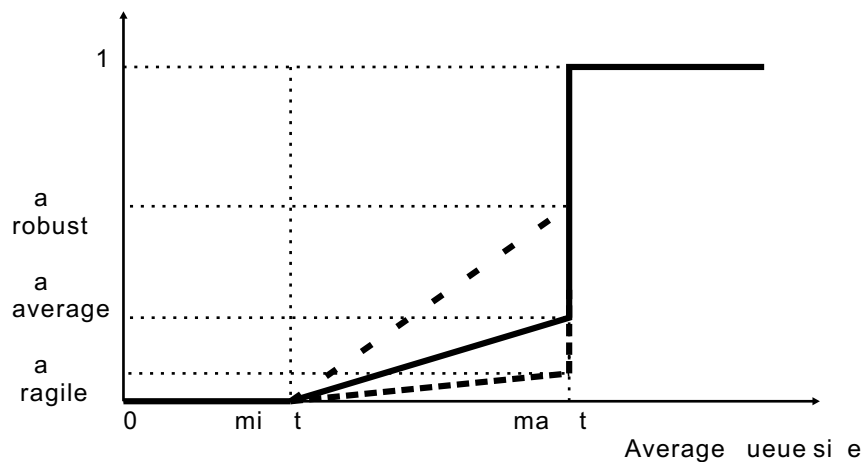

Figure 2.5: AECN Algorithm

2.2 Edge Hints

An edge hint is a packet label that includes information that routers can use to achieve bandwidth fairness or other QoS requirements. CSFQ [SSZ98] uses rate estimate hints. As described in Section 2.1.3, the core routers use them to compute drop probability. Multiprotocol Label Switching (MPLS) [RVC01] networks use labels to accomplish different tasks. MPLS networks rely on labels on the packets for routing, but other labels for MPLS can be implemented. Class Based Threshold (CBT) [PJS99] classifies flows into UDP and TCP flows and uses different thresh-

13

olds for each class. Alternative Best-Effort (ABE) [HKBT01] has two classes of flows: green and blue flows. Green flows require low delay and blue flows do not suffer from delay. ABE applies different drop probability to green and blue packets. AECN [Zhe01], described in Section 2.1.5, uses the round trip time hints from the sources to classify the packets into three different categories: robust, average and fragile. There are other useful hints that sources or edges can provide the routers such as window size of TCP sources, delay requirements and round trip times. In this thesis, round trip time is of particular interest because we deal with heterogeneous flows with different latencies between sources and destinations. We explain more on how this hint was implemented and used at the router in Chapter 3.

2.3   TCP Friendly

"TCP Friendly" is a term used to describe flows whose rate does not exceed any other TCP comformant flows' rate in all circumstances. [PFTK98] models TCP throughput mathematically as shown in the following equation.

$$T = \frac{s}{R\sqrt{\frac{2p}{3}} + t_{RTO}\sqrt{\frac{3p}{8}}p\left(1 + 32p^2\right)} \tag{2.1}$$

T is the throughput, s is the packet size, R is the round trip time, p is the steady state loss rate, t_{RTO} is the retransmission time out. Throughput is directly proportional to the packet size and inversely proportional to round trip time, steady loss rate and retransmission timeout. The larger the packet size, the higher the throughput because there are more bytes sent per packet. Large round trip times reduce the throughput because the TCP flow cannot advance its window as fast. Higher loss rates result in halving of window and this decreases the throughput. Longer retransmission timeouts cause long idle times when packets are dropped,

14

leading to lower throughput. [FF99] discusses ways to detect non TCP-friendly flows based on this TCP modeling. It discusses how measuring throughput, loss rate and others to detect non TCP friendly and unresponsive flows and proposes how to handle them along with mention of limitations of such approaches. TCP Friendly Rate Control (TFRC) [FHPW00] is a transport layer protocol that is rate based unlike window-based TCP. The authors base their rate control mechanism on the formula. This mathematical model is the basis for deriving a fundamental part of WHITE and described in Section 3.3.

Chapter 3

Approach

In this chapter, we present the WHITE algorithm, summarized in Figure 3.1. WHITE is a modification of RED [FJ93]. Each packet contains a round trip time (RTT) label, as described in Section 3.1. For each incoming packet, the RTT label is used to update the RTT average kept by the router, as described in Section 3.2. If there is extreme congestion, indicated by the queue average being above max_{th}, the packet is dropped. If there is no congestion indicated by the queue average being below min_{th}, no packet is dropped. If there is congestion, as indicated by the queue average being between min_{th} and max_{th}, the drop probability for the packet is computed based on the queue parameters, the RTT label, and the RTT average, as described in Section 3.3. If the packet is not dropped, it is enqueued in a normal first-in, first-out fashion. A feature inherited by WHITE from RED is that WHITE can support both dropping and marking of packets. To distinguish the two different versions of WHITE, we call the dropping version of WHITE *Chardonnay* and the marking version *Chablis*.

```
on receiving packet p
if (p.RTT > 0) then
    updateAvg(R_average, R_formula, p.RTT)
if (q_avg >= max_th) then
    dropPacket(p, 1)
else
    if (avg >= min_th) then
        d = calcDropProbability(q_avg, min_th, max_th, p.RTT, R_formula)
        dropPacket(p, d)
    else
        enQueuePacket(p)
```

Figure 3.1: WHITE Algorithm

3.1 Average Round Trip Time at the Edge

As in [SSZ98], we assume an architecture where edge routers on the ingress of a network cloud label a packet with additional information, called an *edge hint.* Core routers on the interior of the network cloud can make more informed packet drop decisions by using the edge hints. In our case, the edge hint is given by the sending host using TCP-Reno and it is the lowest RTT recorded, as computed using the *baseRTT* computation from TCP Vegas code.

Based on discussion in [SZ99], there are from 4 to 17 bits available that can be used to carry edge hint information. We store the RTT in the IP packet using a 16-bit integer, but in practice our range of RTTs would only require about 9 bits. Moreover, 9 bits still allows coverage of up to 80% of RTTs typically observed on the Internet [All00]. At a granularity of 10 ms, 9 bits would be sufficient to cover RTT ranges of up to 5 seconds.

3.2 Round Trip Time at the Router

The average RTT at the router is calculated using an exponential weighted moving average, called $R_{average}$. To adjust quickly to possible RTT changes in the network, the weight w_{RTT} is set to 0.1, which is much higher than a typical w_q of 0.002 used as a weight for a RED queue average. The value of 0.1 is necessary to quickly detect changes in the RTT of incoming packets, caused by the the addition of new flows, the termination of old flows or a change in route of some flows. To prevent excessive variation in RTT under steady state, the algorithm uses a stabilized measure of the RTT, called $R_{formula}$, to compute drop probabilities (see Section 3.3). $R_{formula}$ is only changed when $R_{average}$ has significantly moved (defined to be an interval of 25ms centered around $R_{formula}$ in our implementation) for an entire time interval (defined to be 100ms in our implementation, an approximate RTT (same one used in [FGS01])). If $R_{average}$ has moved from one side of the range to the other, called *crossing over* in Figure 3.2, we reset the time interval. When it is determined that $R_{average}$ has moved, $R_{formula}$ is updated to move towards the new $R_{average}$ with the following formula.

$$R_{formula} = \frac{1}{3}R_{formula} + \frac{2}{3}R_{average} \qquad (3.1)$$

The original approach was to just update $R_{formula}$ with $R_{average}$ but experimentally the above equation worked better at stabilizing $R_{formula}$.

3.3 Drop Probability Based on Round Trip Time

In order to compute a drop probability based on a packet's RTT relative to the average RTT ($R_{average}$), we start with the TCP response function [PFTK98]:

18

```
now = getCurrentTime()
R_average = (1 - w_RTT) R_average + (w_RTT) p.RTT
if (((R_formula-12.5) < R_average AND (R_formula+12.5) > R_average) OR R_average crossed
over) then
        lasttime = now
else
    if ((now - lasttime) > 100ms) then
        lasttime = now
        update R_forumula towards new R_average
```

Figure 3.2: Algorithm for Computing Round Trip Time at the Router
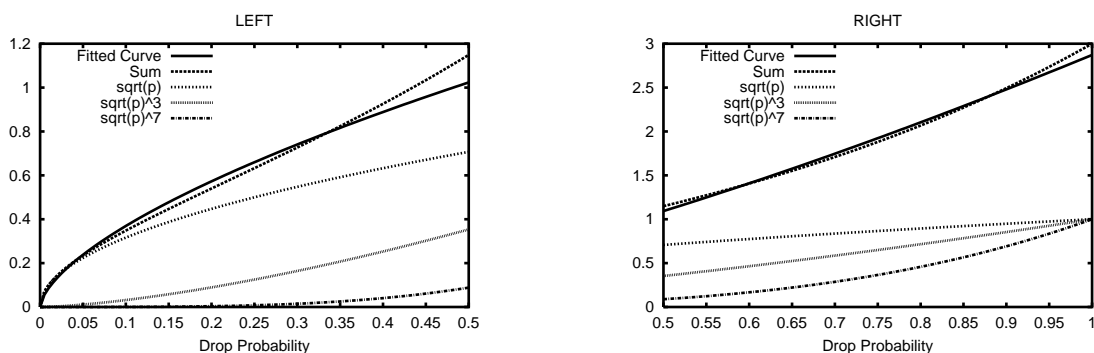
Figure 3.3: Contribution of p-Terms vs. Drop Probability.

$$T = \frac{s}{R\sqrt{\frac{2p}{3}} + t_{RTO}\sqrt{\frac{3p}{8}}\,p\,(1 + 32p^2)} \tag{3.2}$$

This provides the upper bound on the sending rate of T (bandwidth) as a function of the packet size s, steady state loss rate p, round trip time R, retransmission time out t_{RTO}. Although the drop probability in RED is not exactly equivalent to the steady state loss rate p, p will be used to estimate the relationship between the drop probability and the round trip time. Combining the three terms involving p above using a constant c and exponent a, we get the simplified equation:

$$T = \frac{s}{Rcp^a} \tag{3.3}$$

19

Consider two flows with throughputs T_1 and T_2 and round trip times R_1 and R_2, respectively. In order to achieve fair bandwidth allocation, T_1 and T_2 should be equal. For simplicity, we assume that the packet sizes s_1 and s_2 are equal. This leads to the following derivation:

$$T_1 = T_2, s_1 = s_2 = s$$

$$\frac{s}{R_1 c p_1^a} = \frac{s}{R_2 c p_2^a}$$

$$p_2^a = p_1^a \left(\frac{R_1}{R_2}\right)$$

$$p_2 = p_1 \left(\frac{R_1}{R_2}\right)^{\frac{1}{a}} \tag{3.4}$$

The exponent a needs to summarize the behavior of the denominator of the original equation. The three terms involving p in equation (2) have exponents of $1/2$, $3/2$ and $7/2$. When p approaches 0, $p^{1/2}$ dominates, but when p approaches 1.0, each p-term contributes nearly equally. We divide the range of p from 0 to 1 into two regions, 0.0 to 0.5 and 0.5 to 1.0, and calculate a best-fit power curve for the sum of the p terms for each region. Figure 3.3 depicts each p-term's contribution to the sum, and the best fit curve to the sum. When p is small, Figure 3.3-*left* shows that the sum of the p-terms follows closely to square root of p as expected, with the estimated exponent a in the fitted curve function being about 0.63. When p approaches 1, Figure 3.3-*right* shows that each p-term contributes to the sum, with the estimated exponent a in the fitted curve function being about 1.39.

WHITE starts with a base drop probability p computed from min_{th}, max_{th}, max_p and q_{avg}, as in traditional RED [FJ93]. For robust flows whose RTTs are smaller than the average RTT, their drop probability, p_{robust}, should be higher than

20

the average drop probability p_{base}. On the other hand, for fragile flows whose RTTs are higher than the average RTT, their drop probability, $p_{fragile}$, should be lower than p_{base}.

For Chardonnay, the dropping version of WHITE, the robust flow treatment should use the coefficient from the right graph because the drop probably should be in the higher range for the robust flows. The fragile flow treatment should use the coefficient from the left graph because we want to protect the fragile flows by dropping packets less.

$$p_{robust} = p_{base} \left(\frac{R_{formula}}{R_{robust}} \right)^{0.71} \qquad p_{fragile} = p_{base} \left(\frac{R_{formula}}{R_{fragile}} \right)^{1.58} \qquad (3.5)$$

For Chablis, the marking version of WHITE, the same principle applies but for the robust flows, the drop rate is still low because packets get marked instead of being dropped. Therefore, the coefficient from the left graph is also used for the robust flows. The fragile flow treatment remains the same as Chardonnay.

$$p_{robust} = p_{base} \left(\frac{R_{formula}}{R_{robust}} \right)^{1.58} \qquad p_{fragile} = p_{base} \left(\frac{R_{formula}}{R_{fragile}} \right)^{1.58} \qquad (3.6)$$

For the rest of the paper, the exponent used for the drop probability for robust flows is called α and the exponent used for the drop probability for fragile flows is called β. Although the theoretical values are shown as above, simulation results indicate Chardonnay performed the best with α of 0.65 and β of 1.40 and Chablis with α of 1.60 and β of 1.40. We use these latter values in the rest of the work.

WHITE has a basic complexity similar to that of RED. In addition, Chardonnay has four additional variables (α, β $R_{average}$, and $R_{formula}$). For each packet that arrives, Chardonnay must read the round trip time label in each packet, and compute $R_{average}$, adjusting $R_{formula}$ as necessary. No per flow state information is required.

22

Chapter 4

Simulation Setup

In this section, we evaluate Chardonnay and Chablis by comparing their performance with the four additional algorithms described in the introduction: RED, with no explicit attempt at fairness as the current status quo of (un) fairness; DRR, with packet scheduling to achieve bandwidth fairness as the model of fairness; and CSFQ and FRED as less complex ways of achieving the fairness sought by DRR. We used the NS-2 simulator [oCB], which provides packet-level implementation of many TCP protocols and many buffer queue management algorithms including DRR and RED. For CSFQ and FRED, we used the code developed in [SSZ98].

We implemented Chardonnay and Chablis by extending the existing RED code with the drop probability algorithms described in Chapter 3. Then we set up the network topology shown in Figure 4.1. There are 30 sources, N_0 through N_{29} going through a bottleneck router R to destination D. The bottleneck bandwidth is 10 Mbps and the delay is 5 ms. The settings for RED, FRED, CSFQ, Chardonnay, and Chablis are in Figure 4.1. For CSFQ, the averaging constants K (used in estimating the flow rate), K_α (used in estimating the fair rate), and K_c (used in making the decision on whether the link is congested or not) are set as recommended in [SSZ98].
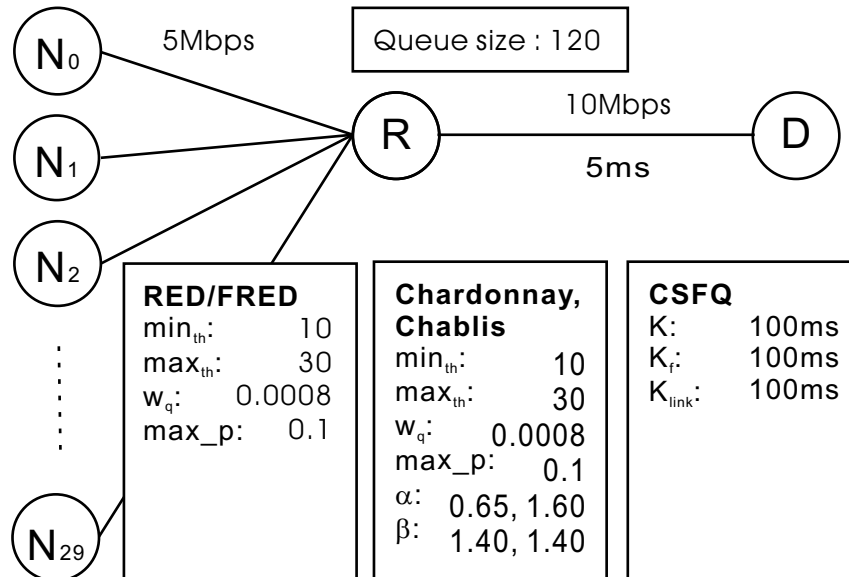
Figure 4.1: Network Topology and Settings

Any setting not listed in the figure is set to the default NS-2 configuration. All flows use TCP-Reno with the RTT modification described in Section 3.2 and a maximum window size of 64 packets.

A set of six experiments were run using this topology. All settings remain the same across experiments, except that the latencies between each node N_i and the router R differ and the active queue management (AQM) technique was one of RED, FRED, CSFQ, DRR, Chardonnay or Chablis. We compare fairness in a series of graphs depicting goodput. To numerically compare the fairness, we computed Jain's fairness index [Jai91] for each experiment (Figure 5.9). Jain's fairness index takes the average throughputs of the flows and computes a normalized number between 0 and 1, where 0 denotes the maximum unfairness and 1 denotes the maximum fairness. We also collected the minimum and maximum goodput values in Figure 5.10.

In general, shown in Chapters 5 and 6, Chardonnay and Chablis achieve reasonable fairness, significantly closer to DRR than RED, and Chardonnay and Chablis

are moderately fairer than CSFQ and FRED.

For our evaluation, we will use a slightly modified Jain's fairness index to use goodput instead of throughput. For convenience, we will still refer to this modified version as Jain's fairness index. Metrics used to make comparison across simulation results are goodput, Jain's fairness index [Jai91] and min-max fairness. Goodput is very similar to throughput except retransmissions are not taken into account in calculation for goodput. Jain's fairness index is a per-flow fairness measure calculated by the following equation.

$$Jain'sFairnessIndex = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2} \tag{4.1}$$

In the above equation, n is the total number of flows and x_i is goodput of flow i. Jain's fairness index is a positive number with maximum value of 1. The closer to 1 it is, the fairer the flows are. In other words, the goodputs of all flows are about the same as fairness index approaches 1.

Min-max fairness is another metric to compare fairness of all flows. As name suggests, it picks out the minimum and maximum goodput among all flows. Smaller difference between the minimum and maximum goodput suggests that all the goodputs are all about the same.

25

Chapter 5

Chardonnay

In this chapter, we evaluate Chardonnay, the dropping version of WHITE by comparing its performance with the four additional algorithms mentioned in the introduction: RED, with no explicit attempt at fairness as the current status quo of (un)fairness; DRR, with packet scheduling to achieve bandwidth fairness as the model of fairness; and CSFQ and FRED as less complex ways of achieving the fairness sought by DRR.

5.1 Uniformly Distributed Latencies

Experiment 1 considers many sources with various round trip latencies. Each source latency is calculated using Equation 5.1, which introduces a linear increase in latency from one source to the next.

$$latency\,(N_i) = 2\,[(i+1)\,5ms + 5ms] \tag{5.1}$$

Therefore, the 30 sources have round trip latencies ranging from 20 ms to 310 ms. The results of experiment 1 were used to tune the α and β parameters for Chardon-

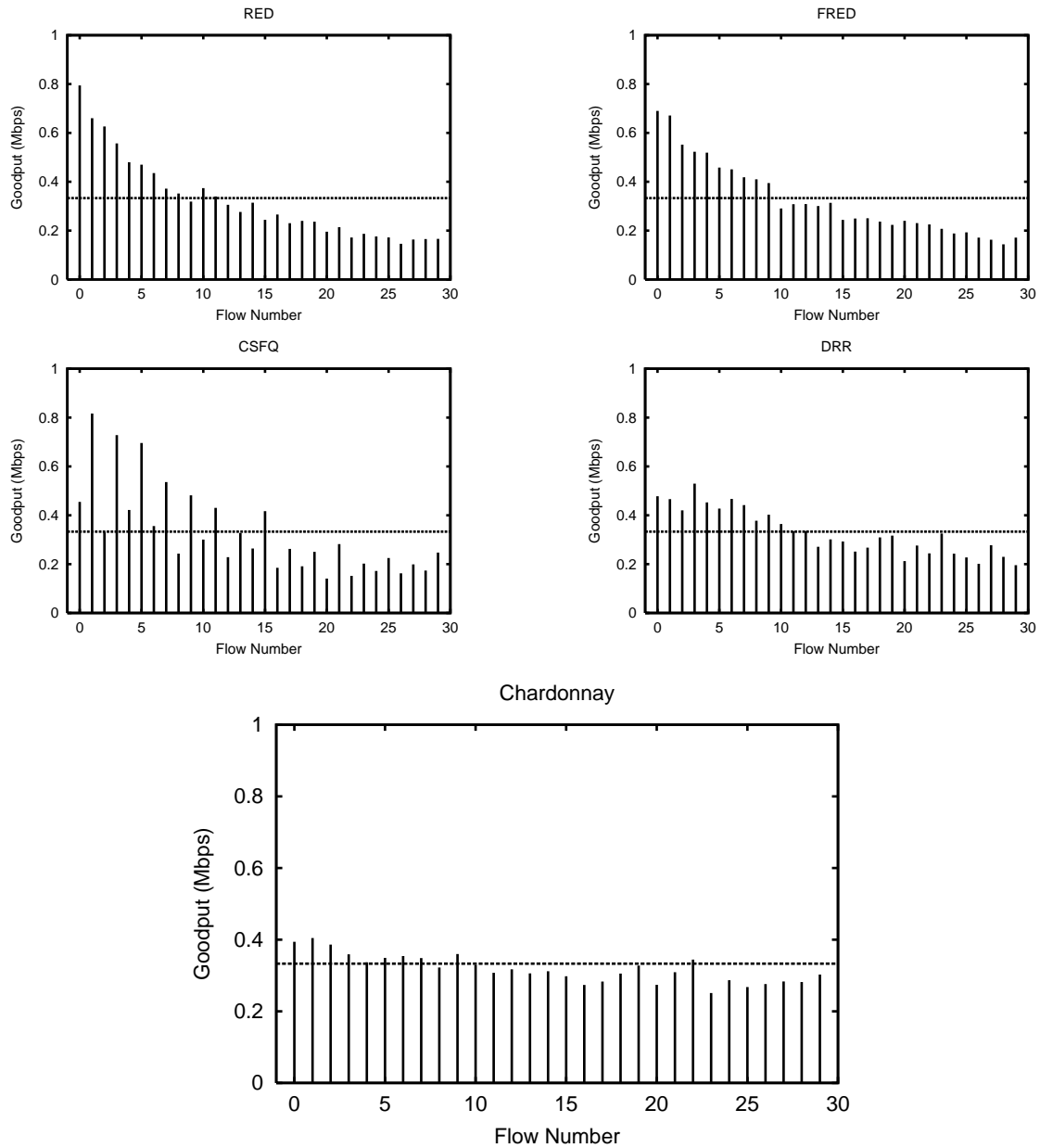Figure 5.1: Experiment 1 (Uniformly Distributed Latencies). The 30 flows have round trip latencies ranging from 20 ms (left) to 310 ms (right).

nay. Although the theoretical values derived in Chapter 3 were 0.72 and 1.58 respectively, the actual values used based on the best results from this experiment, as shown in Figure 4.1, are 0.65 and 1.40.

The experiment simulations were run for 150 seconds and the goodput was av-

27

eraged over a 120 second period starting 30 seconds after the beginning of each simulation. Figure 5.1 depicts the goodput for each flow. With 30 different flows, the fair share of each flow is $\frac{1}{3}$ Mbps, depicted by the horizontal line in each graph. Without any fair treatment, RED provides the most bandwidth (0.79 Mbps) to the most robust source and the least bandwidth (0.15 Mbps) to a fragile source. FRED does not improve the bandwidth for the most fragile flow at all (0.14 Mbps) but reduces the bandwidth of the most robust flow (0.69 Mbps). CSFQ provides high bandwidth to every other robust flow (the largest getting 0.82 Mbps), while the least bandwidth is comparable to that of RED (0.14 Mbps). Visually, DRR and Chardonnay perform much better (more sources are near the horizontal line), with DRR providing reasonable fairness between the most robust flow (0.53 Mbps) to the most fragile flow (0.20 Mbps), and Chardonnay providing the best fairness between the most robust flow (0.40 Mbps) and the most fragile flow (0.25 Mbps).

5.2   Balanced Clustered Latencies

Experiment 2 considers flows uniformly balanced in three clusters of latencies. There are 10 robust flows with 50 ms of round trip latency, 10 average flows with 100 ms of round trip latency, and 10 fragile flows with 200 ms of round trip latency.

The experiment simulations were run for 150 seconds, but only the region between 30 seconds and 60 seconds is depicted to show the network in steady state. Figure 5.2 depicts the goodput averaged over all the flows in each cluster, measured every 250 ms. For the three clusters of flows, the fair share of bandwidth is $\frac{10}{3}$Mbps.

RED provides the fragile flows with only 1.5 Mbps while the robust flows get 6.0 Mbps. FRED improves the bandwidth of the fragile flows to 1.9 Mbps and reduces the bandwidth of the robust flows to 5.0 Mbps. CSFQ yields 2.0 Mbps to the
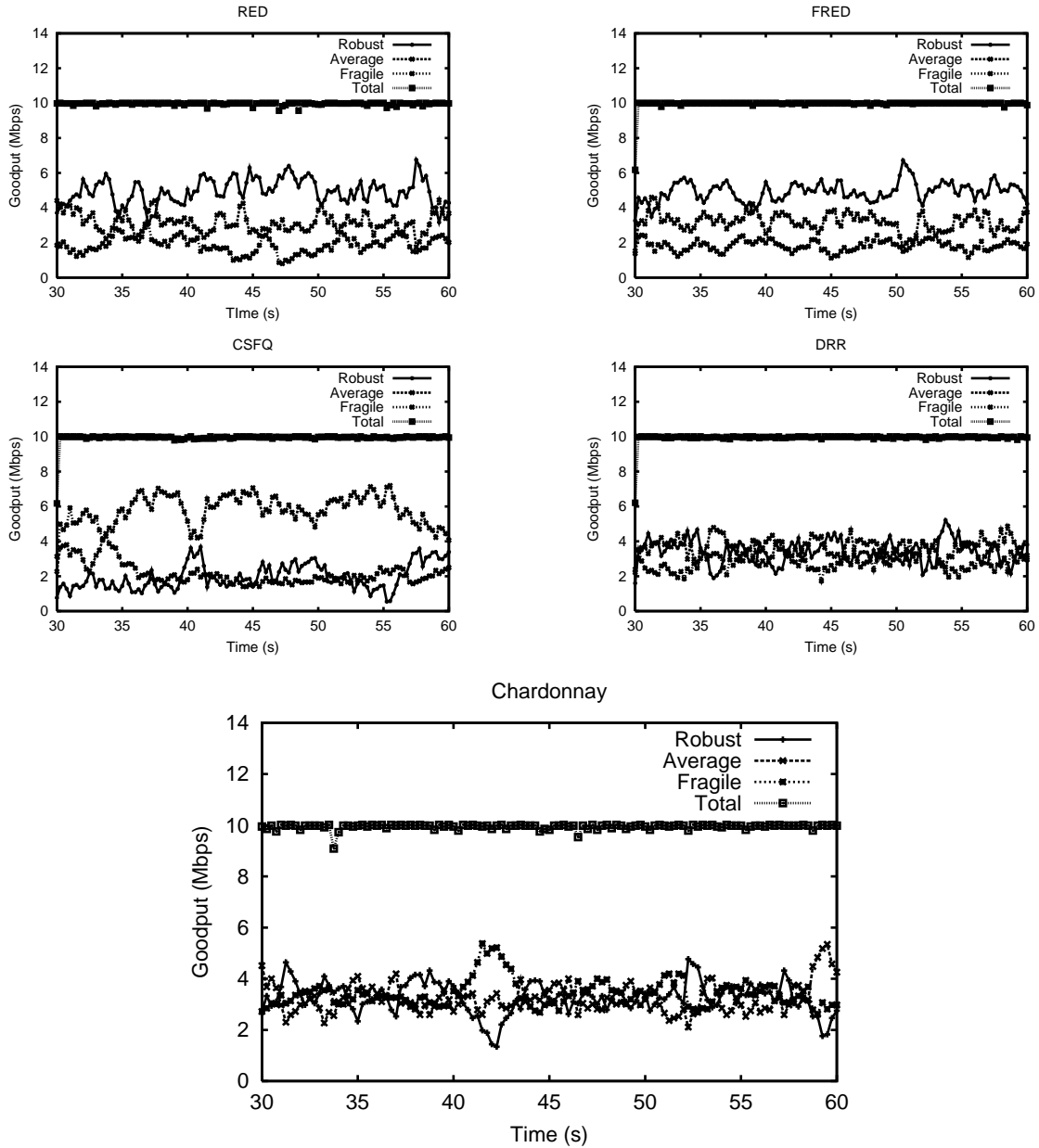
Figure 5.2: Experiment 2 (Balanced Clustered Latencies). 10 robust flows (50 ms round trip latency), 10 average flows (100 ms round trip latency), and 10 fragile flows (200 ms of round trip latency).

clusters of fragile flows and 5.7 Mbps to the cluster of robust flows. DRR provides reasonably fair bandwidth allocation with the robust flows getting 3.5 Mbps and the fragile flows getting 2.8 Mbps. Chardonnay performs the best with the robust

flows getting only 3.6 Mbps and the fragile flows getting 3.1 Mbps.

5.3 Unbalanced Latencies
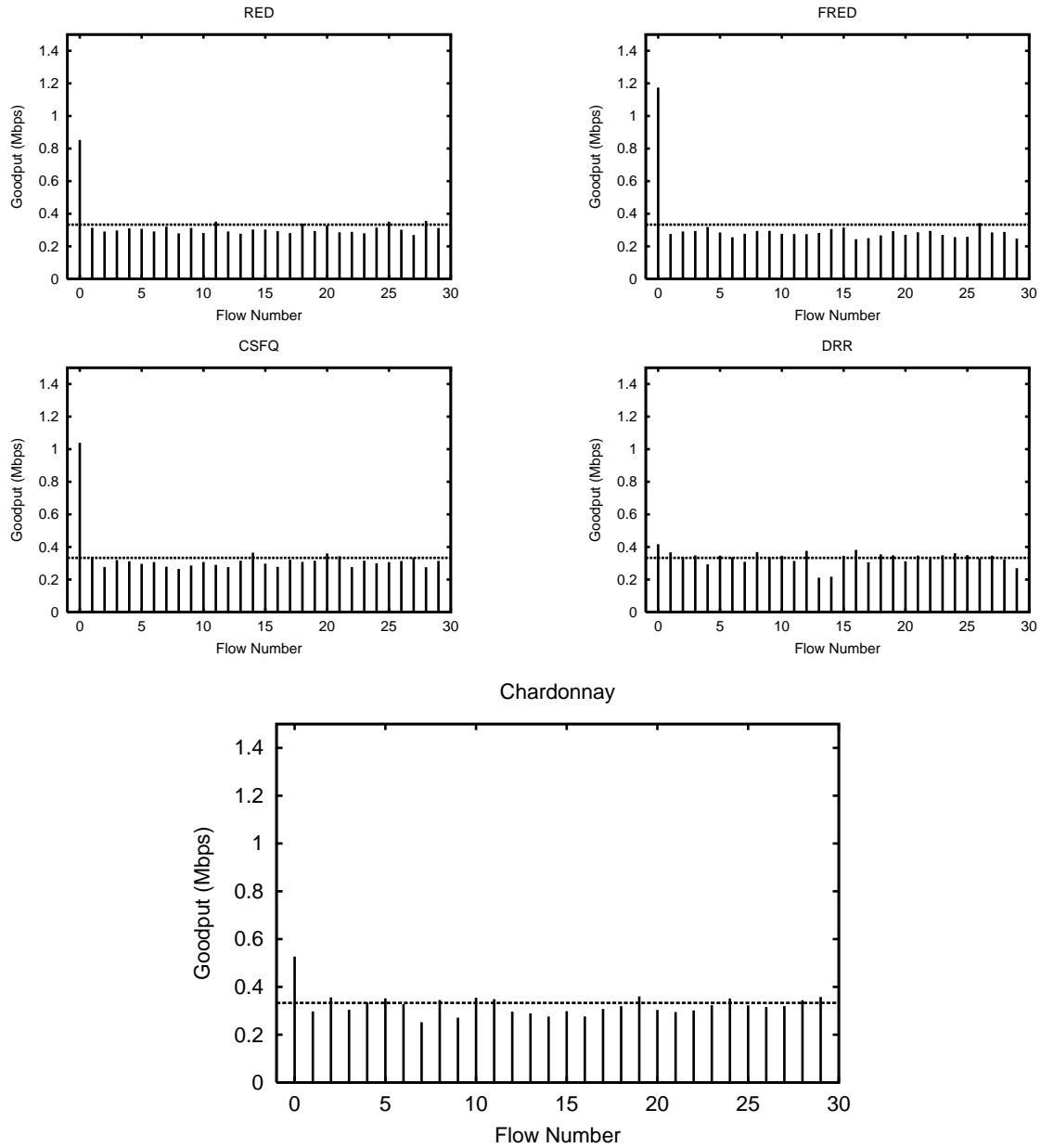
Figure 5.3: Experiment 3 (Unbalanced Latencies). 1 robust (flow 0, 20 ms round trip latency) and 29 fragile (flows 1-29, 200 ms round trip latency).

30
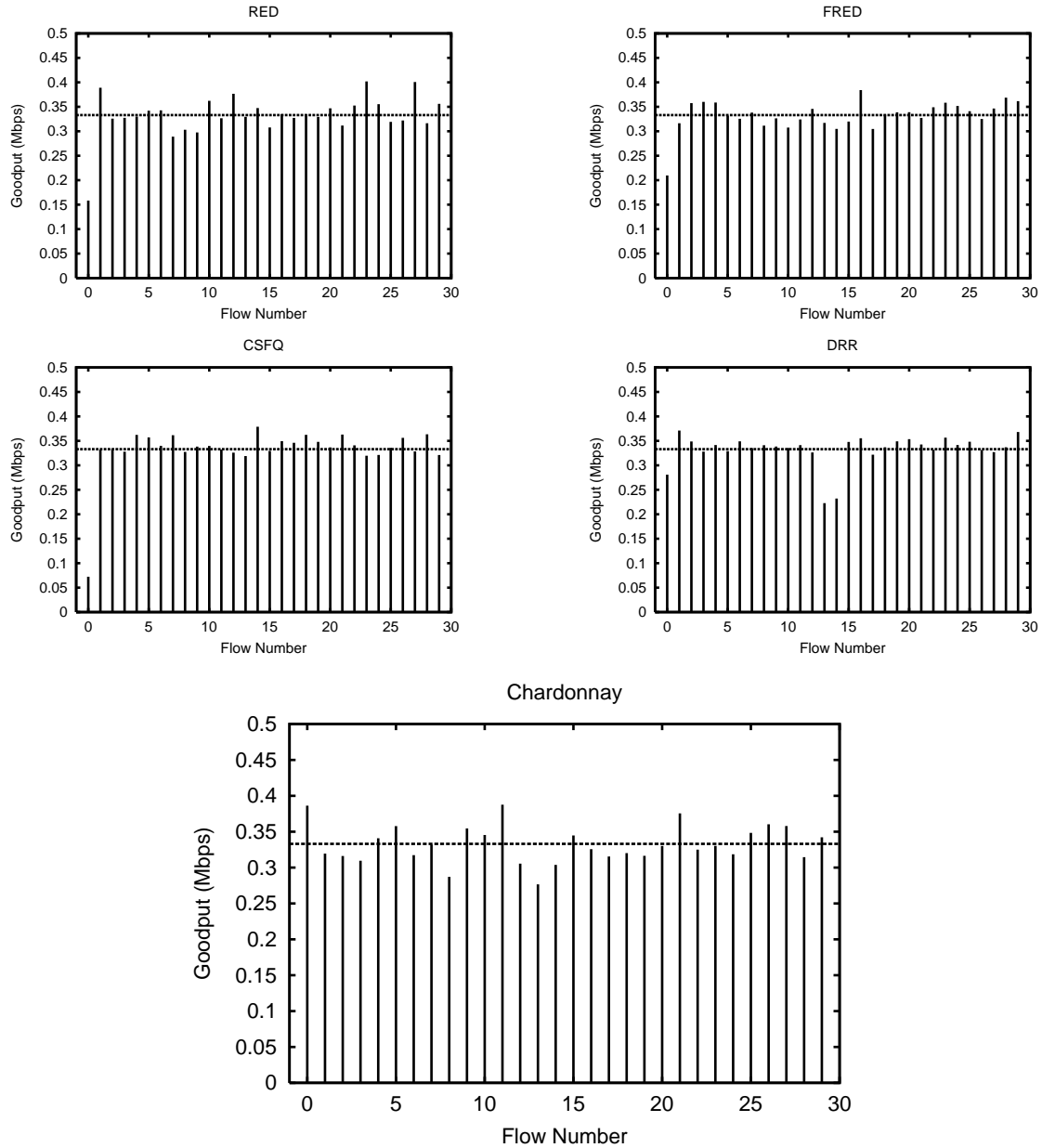
Figure 5.4: Experiment 4 (Unbalanced Latencies). 1 fragile (flow 0, 200 ms round trip latency) and 29 robust (flows 1-29, 20 ms round trip latency).

Experiment 3 and 4 consider cases where most (29) of the flows have the same latency, while one flow has an extremely different latency. Experiment 3 has 1 robust flow (20ms) and 29 fragile flows (200ms). Experiment 4 has 1 fragile flow (200ms) and 29 robust flows (20ms). Both experiments ran simulations for 150 seconds with

31

the first 30 seconds omitted in order to depict a steady state. For both experiments, we compare all 30 flows individually, so the fair share of network bandwidth is $\frac{1}{3}$Mbps.

Figure 5.3 depicts the goodput of each flow for the simulations in experiment 3. RED fails to reduce the goodput of the robust flow (flow 0) and gives it 0.85 Mbps. FRED and CSFQ perform worse than RED, giving the robust flow 1.17 Mbps and 1.04 Mbps, respectively. DRR performs the best, restraining the robust flow to 0.42 Mbps. However, DRR gives as low as 0.21 Mbps to the fragile flows. Chardonnay comes close to DRR by only allowing the robust flow 0.53Mbps, treating the fragile flows better by giving them at least 0.36Mbps.

Figure 5.4 depicts the goodput of each flow for the simulations in experiment 4. RED does not assist the fragile flow, giving it only 0.16 Mbps. FRED is able to help the fragile flow get 0.21 Mbps. CSFQ performs worse than RED, giving the fragile flow only 0.07 Mbps. DRR provides fairness similar to that of FRED, giving the fragile flow 0.22 Mbps. Chardonnay provides the smallest bandwidth gap, giving the fragile flow 0.39 Mbps, slightly more bandwidth than the fair share.

5.4 Dynamic Latencies
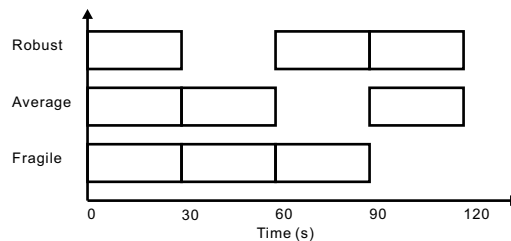
Figure 5.5: Experiment 5 and 6 Timeline

32

Figure 5.6: Experiment 5 (Dynamic Latencies). 10 robust (50 ms round trip latency), 10 average (100 ms round trip latency), and 10 fragile (200 ms of round trip latency).

For experiments 5 and 6, we consider abrupt changes in the latencies of clusters of flows. We set up a scenario where the 3 clusters of flows, 30 flows in each cluster,
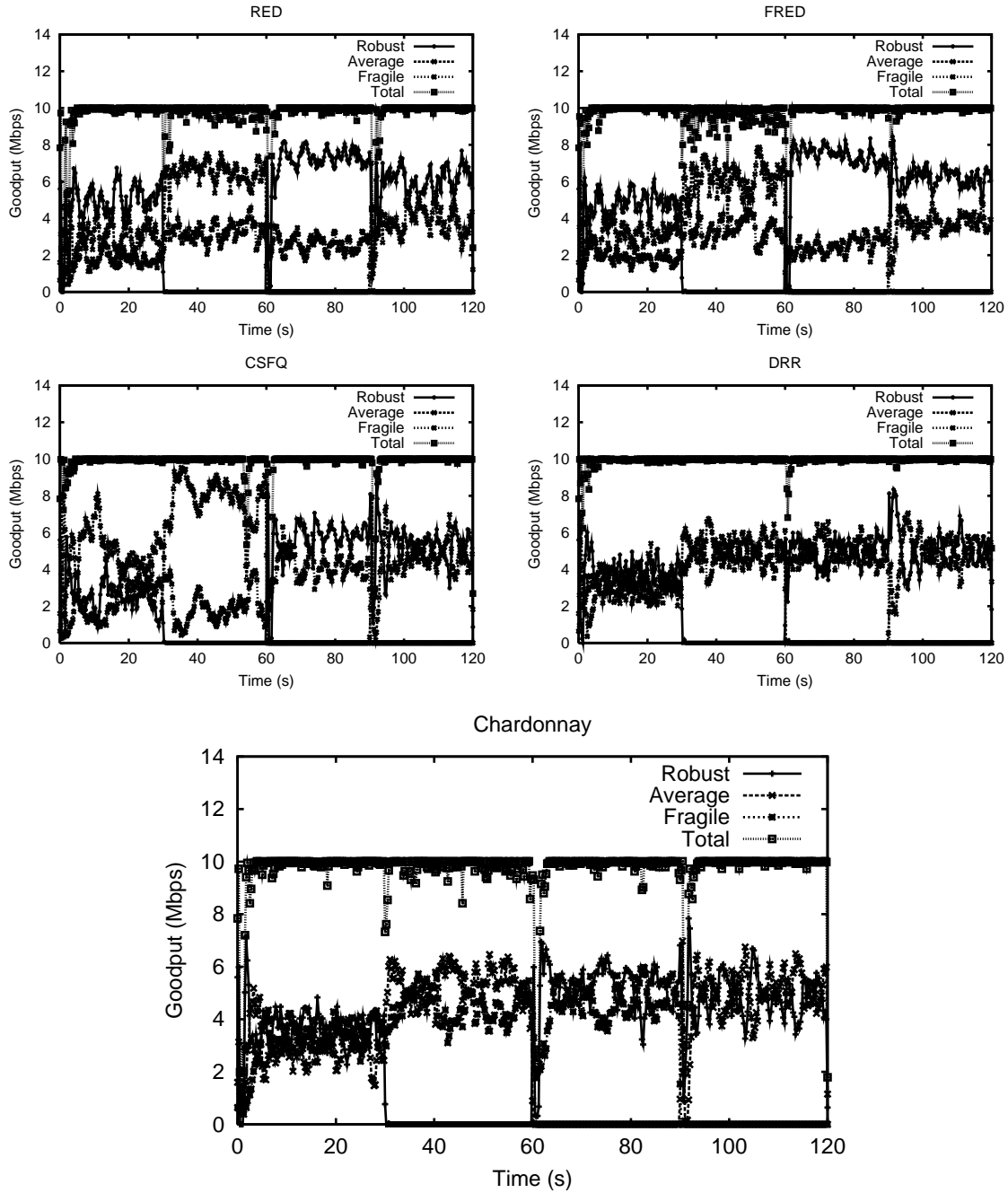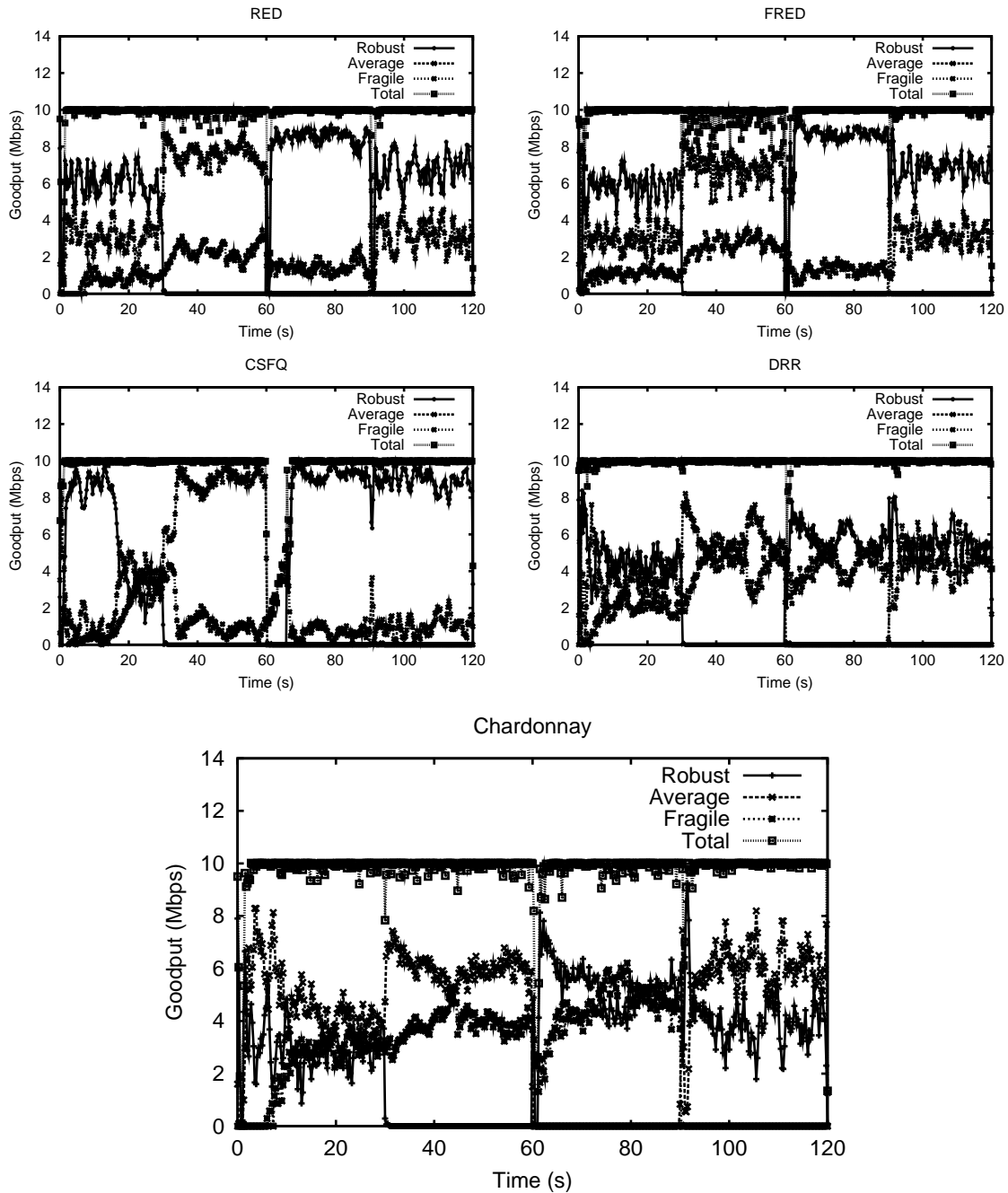
Figure 5.7: Experiment 6 (Dynamic Latencies). 10 robust (20 ms round trip latency), 10 average (80 ms round trip latency), and 10 fragile (320 ms of round trip latency).

where all run for the first 30 seconds (period A). The robust flows stop for the next 30 seconds (period B). The robust flows come back on and the average flows turn

34

off for 30 seconds (period C). Lastly, the average flows turn back on and the fragile flows stop for 30 seconds (period D). Figure 5.5 graphically shows what happens during the 120 seconds of simulation. When there are three clusters of flows, the fair share of bandwidth is $\frac{10}{3}$ Mbps. When there are two clusters of flows, the fair share of bandwidth is $\frac{10}{2}$ Mbps. We run the experiment with two sets of latencies. In Experiment 5, robust flows have 50 ms of round trip time, average flows have 100 ms of round trip time and fragile flows have 200 ms of round trip time. In Experiment 6, we extend the range of round trip time and separate each class latencies farther apart. Robust flows have 20 ms of round trip time, average flows have 80 ms of round trip time and fragile flows have 320 ms of round trip time.

Experiment 5 uses the latency clusters as in Section 5.2, with 10 robust flows having 50 ms of round trip latency, 10 average flows having 100 ms of round trip latency, and 10 fragile flows having 200 ms of round trip latency. In experiment 6, the differences in latencies for each cluster is larger, with 10 robust flows having 20 ms of round trip latency, 10 average flows having 80 ms of round trip latency, and 10 fragile flows having 320 ms of round trip latency.

For experiment 5, Figure 5.6 depicts the goodput averaged over all the flows in each cluster, measured every 250 ms. Visually, RED, FRED and CSFQ are very unfair. Period A is exactly the same as Experiment 2 (Balanced Clustered Latencies) in Section 5.2. In period B, RED, FRED and CSFQ allow the average flows to get around 6-7 Mbps while allowing the fragile flows to get only around 2-3 Mbps. Chardonnay provides fairness very close to that of DRR by giving 4.5 Mbps to the fragile flows and 5.3 Mbps to the average flows. In period C, the difference in bandwidth grows bigger for RED and FRED, with the robust flows getting around 7 Mbps and the fragile flows getting only 2.5 Mbps. CSFQ performs better than RED and FRED, giving 2 Mbps to the fragile flows and 5.6 Mbps to the robust flows.

Chardonnay is again comparable to DRR by giving 4.4 Mbps to the fragile flows and 5.3 Mbps to the robust flows. In period D, RED, FRED and CSFQ provide slightly better fairness by allowing the robust flows to get around 5-6 Mbps and allowing the fragile flows to get 3-4 Mbps. Chardonnay was close to DRR, with the robust flows getting 4.7 Mbps and the fragile flows getting 5.0 Mbps.

For experiment 6, Figure 5.7 depicts the goodput averaged over all the flows in each cluster, measured every 250 ms. Visually, RED, FRED and CSFQ clearly demonstrate unfairness. Both DRR and Chardonnay provide slightly worse fairness than in experiment 5. During period A, DRR gave the fragile flows 1.8 Mbps and the robust flows 4.5 Mbps. Chardonnay gave the fragile flows 2.3 Mbps and the robust flows 4.5 Mbps. In period B, RED, FRED and CSFQ allow the average flows to get around 7-8 Mbps and the fragile flows to get only around 1-2 Mbps. Chardonnay provides fairness similar to that of DRR by giving 4.1 Mbps to the fragile flows and 5.7 Mbps to the average flows. In period C, RED, FRED and CSFQ are even more unfair, with the robust flows getting 7-8 Mbps while the fragile flows get only around 1 Mbps. Once again, Chardonnay is comparable to DRR by giving 4.2 Mbps to the fragile flows and 5.5 Mbps to the robust flows. In period D, RED and FRED provide a bit more fairness by allowing the robust flows to get around 6 Mbps and the fragile flows around 3 Mbps. CSFQ provides the worst fairness by only giving 1 Mbps to the fragile flows and 8.9 Mbps to the robust flows. Chardonnay is close to DRR, with around 4 Mbps for the fragile flows and 6 Mbps for the robust flows.

Comparing the results of Experiment 5 and Experiment 6, we can see that Chardonnay provides better stability and fairness with narrower range of round trip times. Experiment 5 result shows that the flows do not vary too much farther away from the fair share than Experiment 6 result.

36

5.5 Overall Goodput and Drop Rate

It is important that the modifications made to RED for Chardonnay do not degrade the overall performance of RED. In addition to the fairness results for experiments 1-6, we compared the packet drop rate in steady state and the total goodput for the bottleneck link.

Experiment	RED		Chardonnay	
	Drop (%)	Goodput (Mbps)	Drop (%)	Goodput (Mbps)
1	1.85	9.65	1.80	9.59
2	2.85	9.94	2.70	9.91
3	1.45	9.65	1.46	9.67
4	3.61	9.97	3.59	9.96
5	2.90	9.73	2.56	9.76
6	2.60	9.64	2.49	9.69

Figure 5.8: Drop Rate and Total Goodput for RED and Chardonnay

Figure 5.8 shows that Chardonnay's drop rate is about the same as or slightly lower than RED's. The goodput is very comparable between RED and Chardonnay, and in some cases, Chardonnay actually provided more total goodput than RED. In the few cases where Chardonnay's total goodput is lower than RED's, it is only 60 Kbps lower.

37

AQM	Experiment			
	1	2	3	4
RED	0.785	0.872	0.930	0.982
FRED	0.828	0.867	0.788	0.992
CSFQ	0.781	0.796	0.860	0.977
DRR	0.927	0.975	0.984	0.991
Chardonnay	0.982	0.993	0.985	0.991

AQM	Experiment							
AQM	5				6			
	A	B	C	D	A	B	C	D
RED	0.680	0.769	0.654	0.867	0.844	0.885	0.812	0.931
FRED	0.738	0.823	0.642	0.867	0.883	0.900	0.801	0.930
CSFQ	0.740	0.647	0.641	0.610	0.915	0.734	0.965	0.986
DRR	0.880	0.933	0.960	0.970	0.962	0.972	0.985	0.977
Chardonnay	0.888	0.959	0.903	0.915	0.971	0.976	0.970	0.948

Figure 5.9: Jain's Fairness Index for All Experiments

38

AQM	Experiment			
	1	2	3	4
RED	[0.15, 0.79]	[1.47, 5.98]	[0.27, 0.85]	[0.16, 0.40]
FRED	[0.14, 0.69]	[1.85, 4.98]	[0.24, 1.17]	[0.21, 0.38]
CSFQ	[0.14, 0.82]	[1.98, 5.66]	[0.27, 1.04]	[0.07, 0.38]
DRR	[0.20, 0.53]	[2.84, 3.50]	[0.21, 0.42]	[0.22, 0.37]
Chardonnay	[0.25, 0.40]	[3.14, 3.56]	[0.36, 0.53]	[0.28, 0.39]

AQM	Experiment 5			
	A	B	C	D
RED	[1.76, 4.78]	[3.26, 6.48]	[2.61, 7.01]	[3.81, 5.86]
FRED	[1.87, 4.60]	[3.41, 6.15]	[2.50, 7.13]	[3.68, 6.21]
CSFQ	[2.25, 4.41]	[2.02, 7.82]	[4.07, 5.57]	[4.46, 5.26]
DRR	[2.70, 3.68]	[4.66, 5.30]	[4.93, 4.96]	[4.84, 5.10]
Chardonnay	[3.09, 3.28]	[4.51, 5.33]	[4.38, 5.26]	[4.72, 5.08]

AQM	Experiment 6			
	A	B	C	D
RED	[0.68, 5.87]	[2.25, 7.60]	[1.45, 8.26]	[3.13, 6.57]
FRED	[1.01, 5.68]	[2.62, 6.92]	[1.31, 8.28]	[3.14, 6.77]
CSFQ	[1.57, 5.87]	[1.36, 8.61]	[1.20, 7.22]	[1.07, 8.89]
DRR	[1.84, 4.57]	[4.24, 5.71]	[4.24, 5.63]	[4.60, 5.33]
Chardonnay	[2.26, 4.49]	[4.11, 5.77]	[4.16, 5.47]	[3.97, 5.81]

Figure 5.10: Minimum and Maximum Goodput (Mbps) for All Experiments

39

Chapter 6

Chablis

In this section, we evaluate Chablis, the marking version of WHITE by comparing its performance with Chardonnay, the dropping version of WHITE, using the same set of six experiments used in Chapter 5.

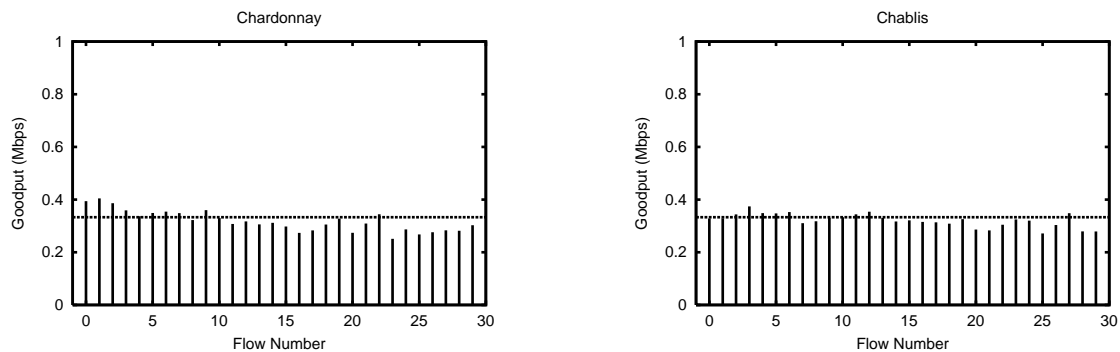## 6.1 Uniformly Distributed Latencies

Figure 6.1: Experiment 1 (Uniformly Distributed Latencies). The 30 flows have round trip latencies ranging from 20 ms (left) to 310 ms (right).

As described in Section 5.1, Experiment 1 has flows with round trip latencies ranging from 20 ms to 310 ms. The results of experiment 1 were also used to

tune α and β parameters for Chablis. Although the theoretical values derived in Chapter 3 were 1.58 for both, the actual values used based on the best results from this experiment, as shown in Figure 4.1, are 1.60 and 1.40.

Figure 6.1 depicts goodput for each flow. With 30 different flows, the fair share of each flow $\frac{1}{3}$ Mbps, depicted by the horizontal line in each graph. Chardonnay provides good fairness between the most robust flow (0.40 Mbps) and the most fragile flow (0.25 Mbps) resulting in Jain's fairness index of 0.982. Chablis performs as well by providing fairness between the most robust flow (0.37 Mbps) and the most fragile flow (0.27 Mbps) with Jain's fairness index of 0.994.

6.2   Balanced Clustered Latencies

As described in Section 5.2, Experiment 2 has clusters of flows: 10 robust flows with 50 ms of round trip latency, 10 average flows with 100 ms of round trip latency, and 10 fragile flows with 200 ms of round trip latency.

Figure 6.2 depicts the goodput averaged over all the flows in each cluster, measured every 250 ms. For the three clusters of flows, the fair share of bandwidth is $\frac{10}{3}$ Mbps. Chardonnay performs fairly with the robust flows getting only 3.6 Mbps and the fragile flows getting 3.1 Mbps with Jain's fairness index of 0.993. Chablis performs about the same with the robust flows getting 3.5 Mbps and the fragile flows getting 3.1 Mbps with Jain's fairness index of 0.994.

6.3   Unbalanced Latencies

As described in Section 5.3, Experiment 3 and 4 have unbalanced latencies: 29 flows with same round trip latency and 1 flow with a very different round trip latency. Experiment 3 has 1 robust flow (20 ms) and 29 fragile flows (200 ms) while

41

Experiment 4 has 1 fragile flow (200 ms) and 29 robust flows (20 ms).

Figure 6.3 depicts the goodput of each flow for the simulations in Experiment 3. Chardonnay allows the robust flow 0.53 Mbps, treating the fragile flows better by giving them at least 0.36 Mbps. Chablis performs slightly better by allowing the robust flow only 0.41 Mbps while treating the fragile flows by giving them at least 0.28 Mbps. Jain's fairness indices for Chardonnay and Chablis are 0.985 and 0.993 respectively.

Figure 6.4 depicts the goodput of each flow for the simulations in Experiment 4. Chardonnay provides a small bandwidth gap, giving the fragile flow 0.39 Mbps, slightly more bandwidth than the fair share. Chablis does not help the fragile flow as much as Chardonnay and allows the fragile flow to get 0.26 Mbps. However, Jain's fairness index is better for Chablis with 0.997 compared to Chardonnay with 0.991. As shown in Figure 6.4, the goodputs for Chablis are closer to each other than those for Chardonnay.

6.4   Dynamic Latencies

As described in Section 5.4, Experiments 5 and 6 involves abrupt changes in the latencies of clusters of flows. In the first period (A) with all three clusters active, the fair share of bandwidth is $\frac{10}{3}$ Mbps. In the rest of the periods (B, C, D) with only two clusters active, the fair share of bandwidth is $\frac{10}{2}$ Mbps.

For experiment 5, Figure 6.5 depicts the goodput averaged over all the flows in each cluster, measured every 250 ms. Period A is exactly the same as Experiment 2 (Balanced Clustered Latencies) in Section 6.2. In Period B, Chardonnay gives 4.5 Mbps to the fragile flows and 5.3 Mbps to the average flows. Chablis performs slightly better by giving 4.7 Mbps to the fragile flows and 5.2 Mbps to the average

42

flows. In Period C, Chardonnay gives 4.4 Mbps to the fragile flows and 5.3 Mbps to the robust flows. Chablis performs a little better by giving 4.8 Mbps to the fragile flows and 5.0 Mbps to the robust flows. In Period D, Chardonnay has the robust flows getting 4.7 Mbps and the fragile flows getting 5.0 Mbps. Chablis performs much fairer with both robust and fragile flows getting 5.0 Mbps.

For experiment 6, Figure 6.6 depicts the goodput averaged over all the flows in each cluster, measured every 250 ms. In Period A, both Chardonnay and Chablis are not as fair as they are in Experiment 5. It takes a long time for both to provide the fair share. Chardonnay gives the fragile flows 2.3 Mbps and the robust flows 4.5 Mbps. Chablis performs better with 3.1 Mbps to the fragile flows and 3.3 Mbps to the robust flows. In Period B, C and D, Chablis performs much fairer than Chardonnay visually. In Period B, Chardonnay gives 4.1 Mbps to the fragile flows and 5.7 Mbps to the average flows. Chablis gives 4.7 Mbps to the fragile flows and 5.1 Mbps to the average flows. In Period C, Chardonnay gives 4.2 Mbps to the fragile flows and 5.5 Mbps to the robust flows. Chablis gives 4.8 Mbps to the fragile flows and 5.0 Mbps to the robust flows. In Period D, Chardonnay gives 4.0 Mbps to the fragile flows and 5.9 Mbps to the robust flows. Chablis gives 4.7 Mbps to the fragile flows and 5.2 Mbps to the robust flows.

6.5   Overall Goodput and Drop Rate

It is important that the movement from Chardonnay, the dropping version of WHITE to Chablis, the marking version does not degrade the overall performance. In addition to the fairness results for experiments 1-6, we compared the packet drop rate in steady state and the total goodput for the bottleneck link.

Figure 5.8 shows that Chablis has rare drops due to marking. The only drops

reported for Chablis are during the stabilization period only. The goodput for Chablis is either same or better than that for Chardonnay because marking does not require a drop of packets which can lead to retransmitted packets.

This chapter shows that Chablis performs as well as or slightly better than Chardonnay in terms of fairness, goodput and drop rate. However, Chapter 5 shows that Chardonnay provides a comparable fairness to DRR, the target of optimum fairness. This indicates that Chardonnay alone is already good enough at providing fairness and also leaves little room for improvement. The significance of Chablis is that it improves on Chardonnay despite the small room for improvement and provides marking of packets which reduces waste of bandwidth considerably.
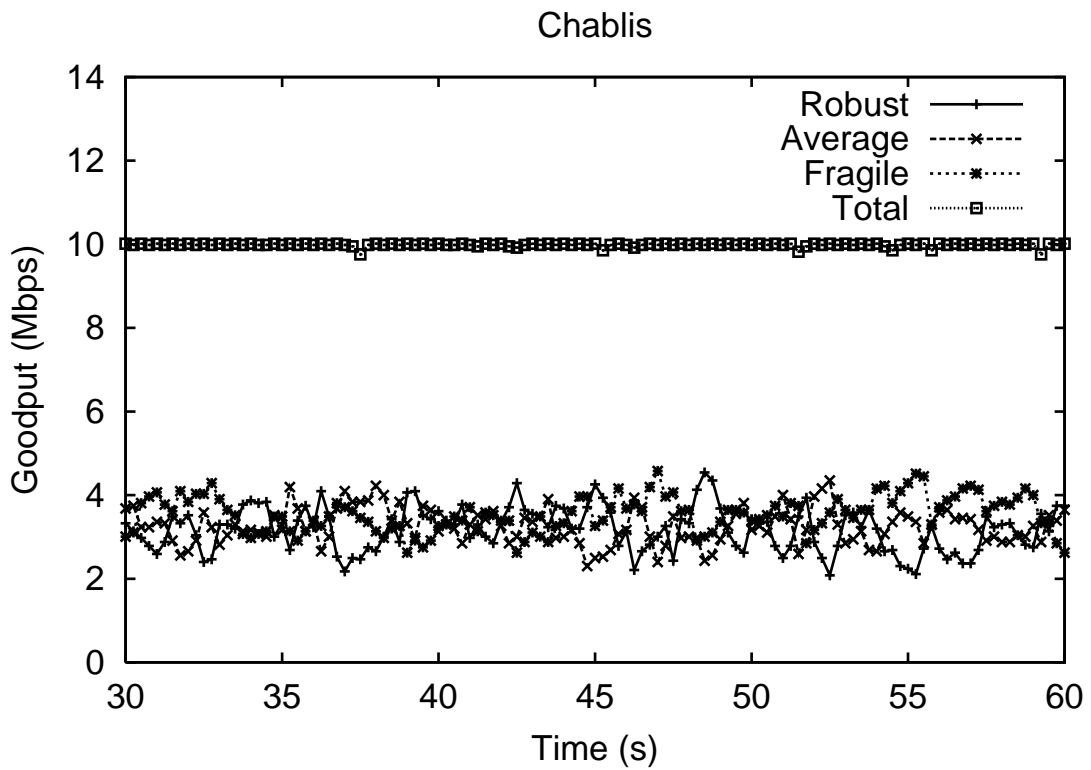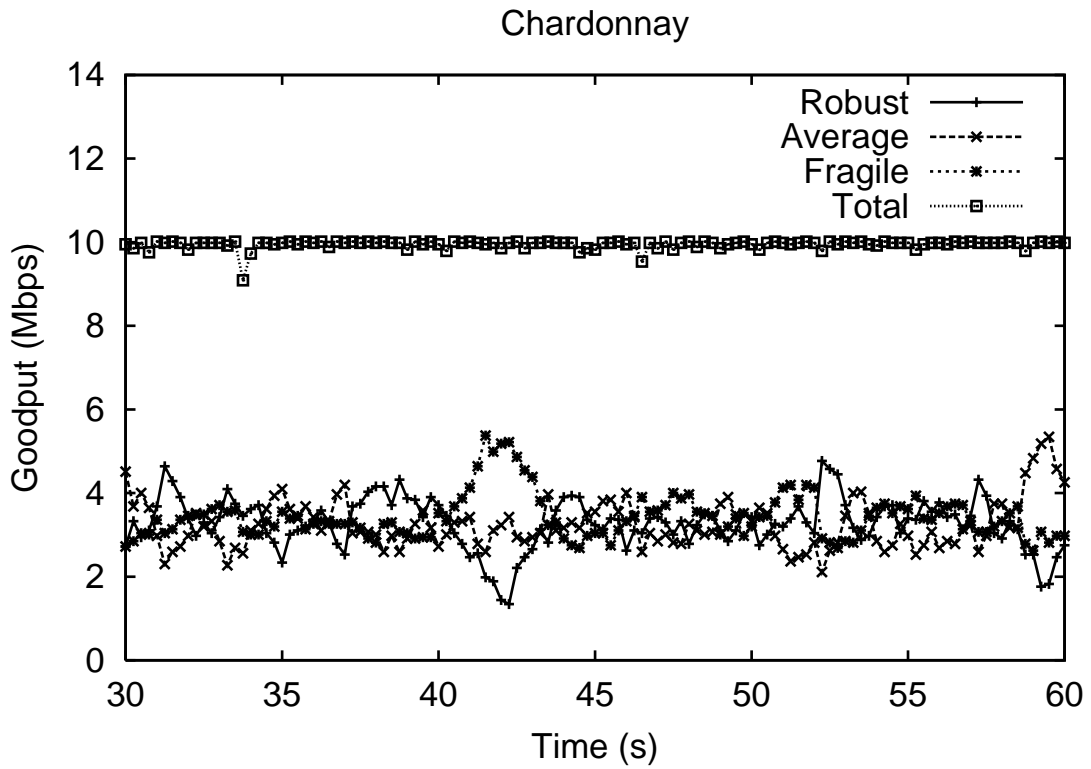
Chardonnay

Chablis

Figure 6.2: Experiment 2 (Balanced Clustered Latencies). 10 robust flows (50 ms round trip latency), 10 average flows (100 ms round trip latency), and 10 fragile flows (200 ms of round trip latency).
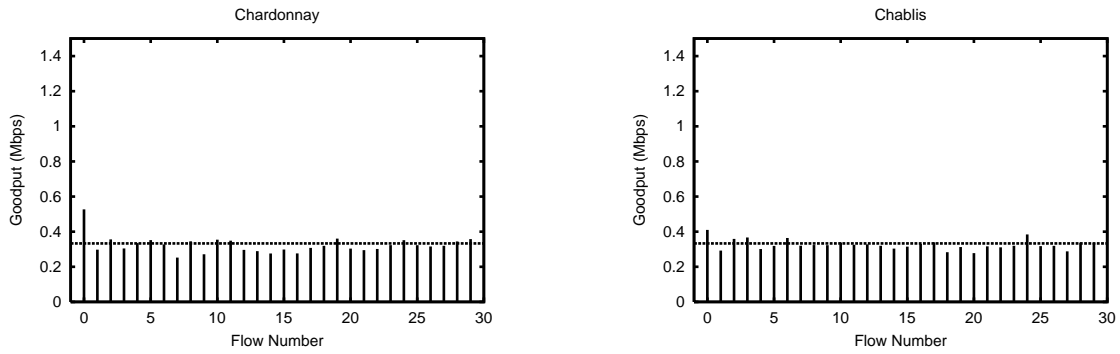
45

Figure 6.3: Experiment 3 (Unbalanced Latencies). 1 robust (flow 0, 20 ms round trip latency) and 29 fragile (flows 1-29, 200 ms round trip latency).
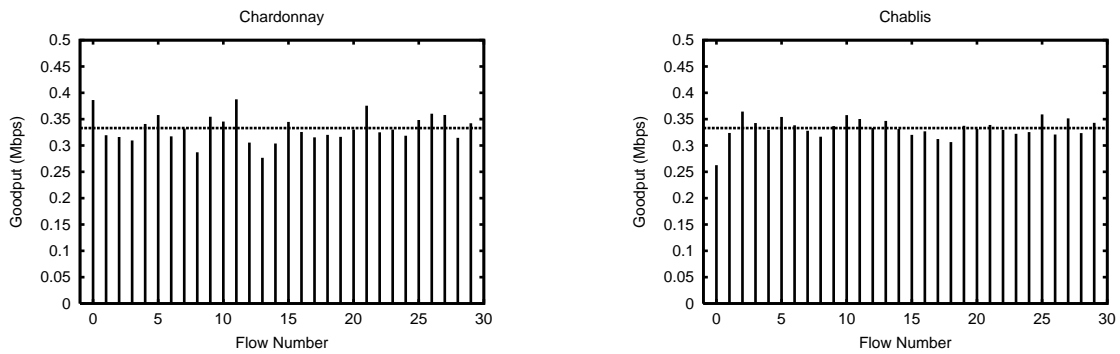

Figure 6.4: Experiment 4 (Unbalanced Latencies). 1 fragile (flow 0, 200 ms round trip latency) and 29 robust (flows 1-29, 20 ms round trip latency).
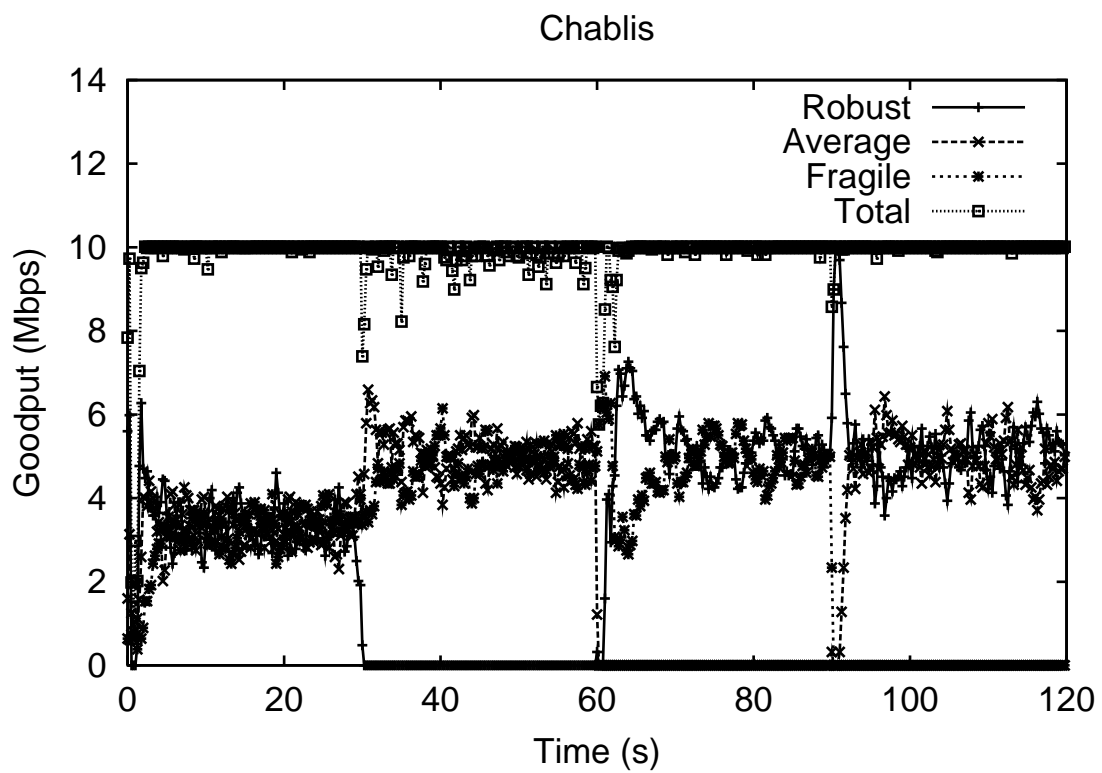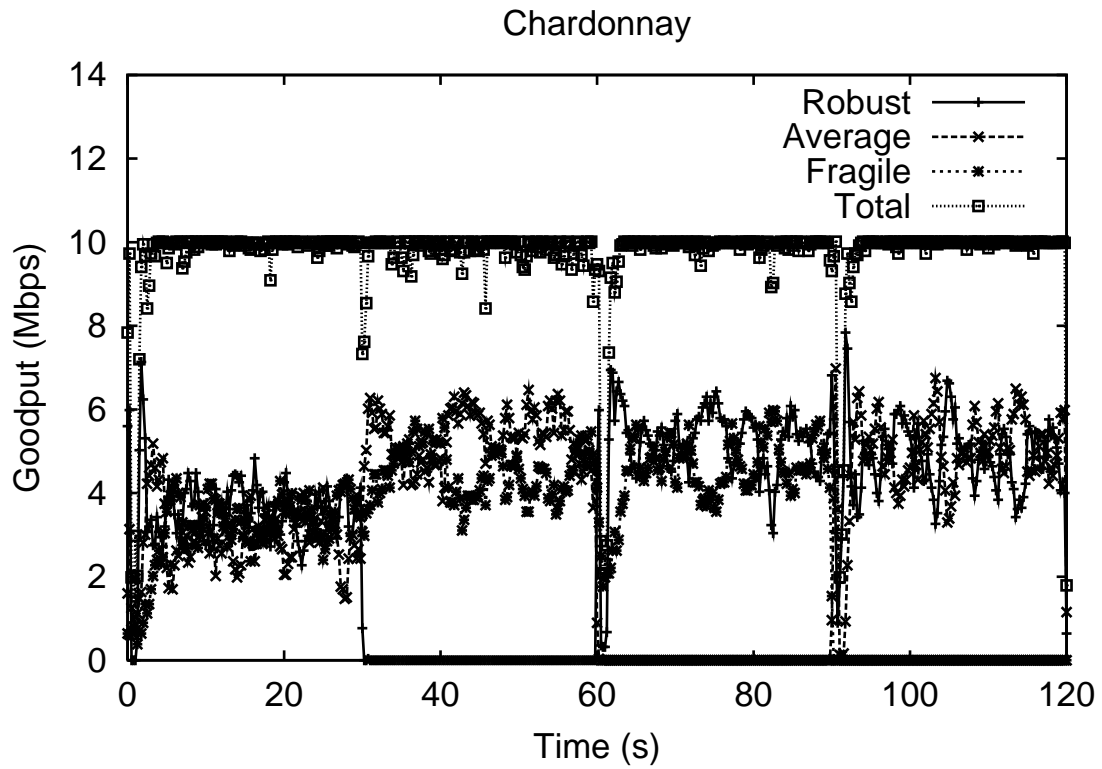
46

Chardonnay

Chablis

Figure 6.5: Experiment 5 (Dynamic Latencies). 10 robust (50 ms round trip latency), 10 average (100 ms round trip latency), and 10 fragile (200 ms of round trip latency).
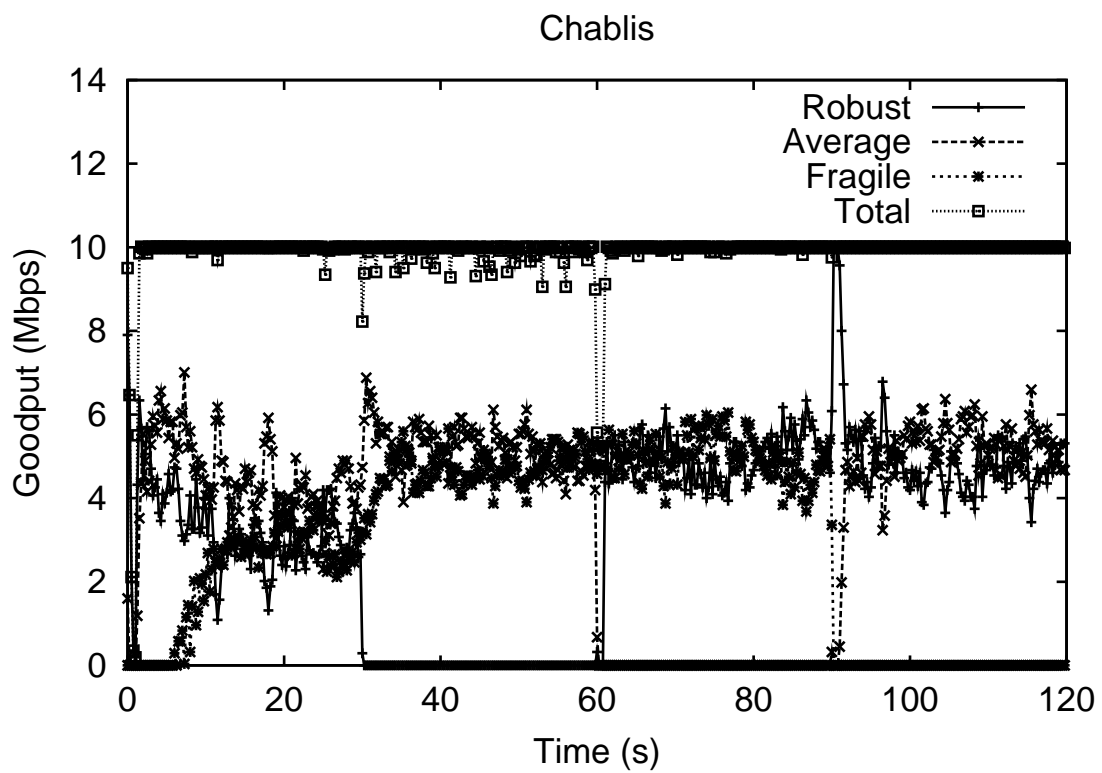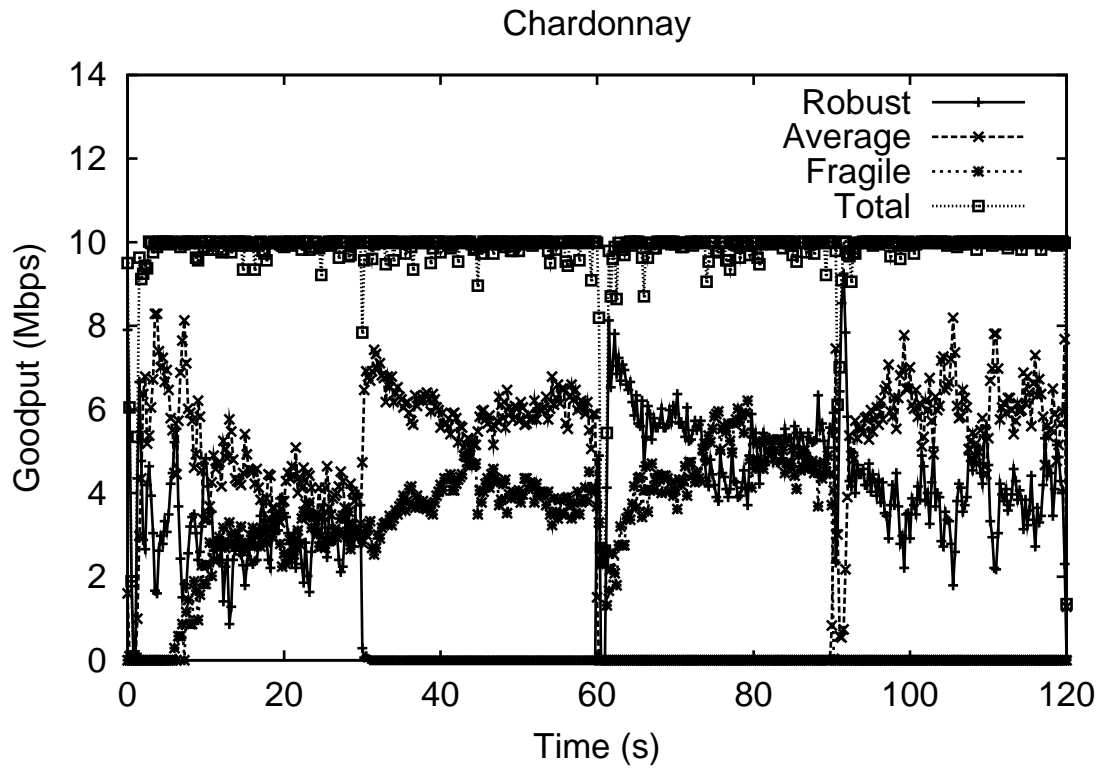
47

Figure 6.6: Experiment 6 (Dynamic Latencies). 10 robust (20 ms round trip latency), 10 average (80 ms round trip latency), and 10 fragile (320 ms of round trip latency).

48

Experiment	Chardonnay		Chablis	
	Drop (%)	Goodput (Mbps)	Drop (%)	Goodput (Mbps)
1	1.80	9.59	0.000	9.65
2	2.70	9.91	0.000	9.98
3	1.46	9.67	0.000	9.78
4	3.59	9.96	0.007	9.96
5	2.56	9.76	0.002	9.85
6	2.49	9.69	0.003	9.82

Figure 6.7: Drop Rate and Total Goodput for RED and Chardonnay

AQM	Experiment			
	1	2	3	4
Chardonnay	0.982	0.993	0.985	0.991
Chablis	0.994	0.994	0.993	0.997

AQM	Experiment							
AQM	5				6			
	A	B	C	D	A	B	C	D
Chardonnay	0.888	0.959	0.903	0.915	0.971	0.976	0.970	0.948
Chablis	0.904	0.987	0.965	0.993	0.990	0.988	0.986	0.992

Figure 6.8: Jain's Fairness Index for All Experiments

AQM	Experiment			
	1	2	3	4
Chardonnay	[0.25, 0.40]	[3.14, 3.56]	[0.36, 0.53]	[0.28, 0.39]
Chablis	[0.27, 0.37]	[3.09, 3.50]	[0.28, 0.41]	[0.26, 0.36]

AQM	Experiment 5			
	A	B	C	D
Chardonnay	[3.09, 3.28]	[4.51, 5.33]	[4.38, 5.26]	[4.72, 5.08]
Chablis	[2.03, 3.32]	[4.69, 5.18]	[4.84, 4.98]	[4.96, 5.01]

AQM	Experiment 6			
	A	B	C	D
Chardonnay	[2.26, 4.49]	[4.11, 5.77]	[4.16, 5.47]	[3.97, 5.81]
Chablis	[3.07, 3.32]	[4.72, 5.10]	[4.76, 5.04]	[4.72, 5.23]

Figure 6.9: Minimum and Maximum Goodput (Mbps) for All Experiments

Chapter 7

Conclusion and Future Work

This thesis presents WHITE, an active queue management approach for achieving fairness among flows with heterogeneous round trip times (RTTs). Using the distributed architecture presented in [SZ99], packets are tagged with their minimum observed RTT, allowing the WHITE router to make dropping decisions based upon the RTT of each flow relative to the average RTT observed at the router. This provides the potential to protect fragile flows with a high RTT from receiving unnecessarily low bandwidth, while curtailing robust flows with a low RTT to their fair bandwidth share. Moreover, the use of the RTT tag at the network edge allows WHITE to avoid keeping per-flow information.

We tested WHITE over a range of flows and over a wide-range of RTT conditions among the flows. We find WHITE achieves a significant degree of fairness under all conditions. We also compared WHITE with CSFQ [SSZ98], FRED [LM97] and DRR [SV95], algorithms specifically designed to achieve fairness, as well as to RED [FJ93]. In all cases, WHITE performs far better than RED, and in many cases, WHITE performs far better than either CSFQ or FRED, often performing nearly as well as does DRR. We are not aware of any other router queue management

techniques that can achieve better fairness than WHITE without using per-flow information.

Currently, WHITE does not curtail unresponsive flows receiving more than their fair share of bandwidth. Unresponsive flows do not react to the congestion control mechanism based on packet drops. Therefore, regardless of RTT hints, unresponsive flows receive more bandwidth if they are sending at a higher rate than the fair share of bandwidth. Most of the multimedia applications use unresponsive flows using UDP and they require high bandwidth, especially for video streams. The natural extension to WHITE is to combine it with a rate-based active queue management technique, such as CSFQ [SSZ98] or RED-PD [MFW01]. High-bandwidth flows could be detected and monitored as in [MFW01], or per packet drop probabilities could be computed based on both the bandwidth used by the flow as well as the RTT.

All experiments in this thesis deals with 30 flows at most. However, WHITE needs to be able to handle flows fairly at all levels of load. As more flows come through WHITE router, the average queue size will exceed max_{th}, forcing a drop of all incoming packets, greatly reducing fairness. Adaptive RED [FGS01] is an extension of RED that handles a larger range of loads by keeping the average queue size at a target between min_{th} and max_{th}. WHITE, being a RED extension, can perhaps be extended using Adaptive RED as well. Preliminary results show that simple combination of ARED and WHITE does not provide the fairness at any loads of network traffic. Further investigation to tune parameters is necessary.

All the evaluation of WHITE has involved a dumbbell topology with one congested router and TCP flows only. We also separate evaluation into cases where the TCP flows use and do not use marking. Practically, the Internet is a more complicated network with many congested routers and flows using variety of protocols. In

51

addition, there are many other protocols such as application-layer protocols using TCP and/or UDP and TCP-friendly protocols like TFRC. WHITE needs to be able to handle any mix of traffic, not just one type. It is also essential to determine the scalability of WHITE. We would like WHITE to provide benefit to fairness even with incremental deployment.

Bibliography

[All00]   Mark Allman. A Web Server's View of the Transport Layer. Oct 2000.

[DKS90]   A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queuing algorithm. pages 3 – 26, Oct 1990.

[FF99]   Sally Floyd and Kevin Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, February 1999.

[FGS01]   Sally Floyd, Ramakrishna Gummadi, and Scott Shenker. Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management. Aug 2001.

[FHPW00] Sally Floyd, Mark Handley, Jitendra Padhye, and Joerg Widmer. Equation-based congestion control for unicast applications. In *ACM SIGCOMM*, Aug 2000.

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

[Flo91]   S. Floyd. Connections with Multiple Congested Gateways in Packet-Switched Networks Part 1: One-way Traffic. pages 30 – 47, Oct 1991.

53

[Flo94]   Sally Floyd. TCP and Explicit Congestion Notification. *Computer Communication Review*, October 1994.

[HKBT01]   P. Hurley, Mourad Kara, J. Y. Le Boudec, and P. Thiran. Abe: Providing a low-delay service within best effort. *IEEE Network Magazine*, (3), May 2001.

[Jai91]   R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley and Sons, Inc., 1991.

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

[MFW01]   R. Mahajan, S. Floyd, and D. Wetherall. Controlling high-bandwidth flows at the congested routers. In *In Proceedings of the 9th International Conference on Network Protocols (ICNP)*, Nov 2001.

[oCB]   Universiy of California Berkeley. The Network Simulator - ns-2. Interent site
             `http://www.isi.edu/nsnam/ns/`.

[PFTK98]   J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *SIGCOMM Symposium on Communications Architectures and Protocols*, August 1998.

[PJS99]   Mark Parris, Kevin Jeffay, and F. Donelson Smith. Lightweight active router-queue management for multimedia networking. *Multimedia Computing and Networking, SPIE Proceedings Series*, Jan 1999.

[RVC01]   E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. Technical Report 3031, Internet RFC, January 2001.

[She94]   S. Shenker. Making greed work in networks: A game theoretical analysis of switch service disciplines. pages 47 – 57, Aug 1994.

[SSZ98]   Ion Stoica, Scott Shenker, and Hui Zhang. *Core*-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks. In *Proceedings of ACM SIGCOMM Conference*, September 1998.

[SV95]   M. Shreedhar and G. Varghese. Efficient Fair Queueing Using Deficit Round Robin. In *Proceedings of ACM SIGCOMM Conference*, pages 231 – 243, September 1995.

[SZ99]   Ion Stoica and Hui Zhang. Providing Guaranteed Service Without Per Flow Management. In *ACM SIGCOMM Computer Communication Review , Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, Aug 1999.

[Zhe01]   Zici Zheng. Adaptive Explicit Congestion Notification (AECN) for Heterogeneous Flows. Master's thesis, Computer Science Department, Worcester Polytechnic Institute, May 2001. Advisor: Bob Kinicki.