AN ACTIVE QUEUE MANAGEMENT SCHEME TO CONTAIN  
HIGH BANDWIDTH FLOWS AT A CONGESTED ROUTER

A Thesis

by

SMITHA

Submitted to the Office of Graduate Studies of  
Texas A&M University  
in partial fulfillment of the requirements for the degree of  
MASTER OF SCIENCE

May 2001

Major Subject: Computer Science
AN ACTIVE QUEUE MANAGEMENT SCHEME TO CONTAIN
HIGH BANDWIDTH FLOWS AT A CONGESTED ROUTER

A Thesis
by
SMITHA

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

__________________________
Riccardo Bettati
(Chair of Committee)

__________________________
A.L. Narasimha Reddy
(Member)

__________________________
Nitin Vaidya
(Member)

__________________________
(Head of Department)

Wei Zhao

Major Subject: Computer Science
ABSTRACT

An Active Queue Management Scheme to Contain High Bandwidth Flows at a Congested Router. (May 2001)
Smitha, B.E., Karnataka Regional Engineering College, India
Co-Chairs of Advisory Committee: Dr. Riccardo Bettati
Dr. A.L.N. Reddy

Incorporating mechanisms in the router to enable end-to-end congestion control is important in order to prevent the network collapse. Routers should be able to recognise misbehaving flows and penalise them.

In this thesis, we propose a queue management scheme that is based on partial state. It empowers the routers to contain high bandwidth flows at the time of congestion. The scheme maintains an LRU cache at the routers to record information about the high-bandwidth flows. This can be incorporated in RED, an active queue management scheme. The proposed scheme possesses all the advantages of RED. In addition, it lowers the drop rates of short-lived flows and also those high bandwidth flows that reduce their sending rate when congestion is indicated, by use of preferential dropping policies.

It is shown by means of simulations that the method is effective in achieving the objective. The overhead involved is low and the operations incur O(1) cost per packet.
my parents, Subbu and Supri
ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Reddy for having guided me through this research. Regular reviews of the research with him were instrumental in keeping this research focused and going, and I'm immensely grateful to him for that. His organized ways during this research were awe-inspiring and made learning easy. I appreciate the freedom he gave me while working on this research and the great amount of interest he has taken in it. I thank him for having given me an opportunity to work with him.

I thank Dr. Bettati and Dr. Vaidya for their interest in this research and also for their invaluable suggestions. I wish to thank Dr. Mahapatra for consenting to review the thesis. Many thanks to the Institute for Scientific Computation for having given me the resources to conduct this research.

I’d like to thank Goli, Rashmi and Sowmya for listening with great patience, at various times during this research, to the details of the thesis.

I'd like to thank my parents for having made this possible and for all the support they have given me. I'd like to thank Subrat for keeping alive the motivational factor and Supri for helping me in many ways.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>INTRODUCTION ......................................... 1</td>
</tr>
<tr>
<td></td>
<td>A. Related work ........................................ 2</td>
</tr>
<tr>
<td></td>
<td>1. Various buffer management schemes ............... 2</td>
</tr>
<tr>
<td></td>
<td>2. Comparison of the schemes ...................... 5</td>
</tr>
<tr>
<td></td>
<td>B. Shortcomings of the existing approaches ......... 7</td>
</tr>
<tr>
<td></td>
<td>C. Organization ....................................... 8</td>
</tr>
<tr>
<td>II</td>
<td>THE PROPOSED SCHEME .................................. 9</td>
</tr>
<tr>
<td></td>
<td>A. Why another scheme? .............................. 9</td>
</tr>
<tr>
<td></td>
<td>B. Overview of the scheme .......................... 10</td>
</tr>
<tr>
<td></td>
<td>1. Identifying high bandwidth flows ............... 11</td>
</tr>
<tr>
<td></td>
<td>2. Cache size ....................................... 15</td>
</tr>
<tr>
<td></td>
<td>3. Penalizing high bandwidth flows ............... 15</td>
</tr>
<tr>
<td></td>
<td>4. LRU coupled with RED ......................... 18</td>
</tr>
<tr>
<td></td>
<td>C. Implementation ................................... 19</td>
</tr>
<tr>
<td></td>
<td>1. LRU operation .................................... 19</td>
</tr>
<tr>
<td></td>
<td>2. Preferential dropping ......................... 19</td>
</tr>
<tr>
<td></td>
<td>3. Cost .............................................. 22</td>
</tr>
<tr>
<td></td>
<td>D. Scheme 1 ......................................... 22</td>
</tr>
<tr>
<td></td>
<td>1. Problems encountered ........................... 24</td>
</tr>
<tr>
<td></td>
<td>E. Scheme 2 ......................................... 25</td>
</tr>
<tr>
<td>III</td>
<td>EVALUATION CRITERIA .................................. 30</td>
</tr>
<tr>
<td></td>
<td>A. Protecting short-lived flows ................... 30</td>
</tr>
<tr>
<td></td>
<td>B. Greater penalty for high bandwidth flows ....... 30</td>
</tr>
<tr>
<td></td>
<td>C. Greater penalty for unresponsive high bandwidth flows .... 31</td>
</tr>
<tr>
<td></td>
<td>D. Performance when compared to other schemes .... 31</td>
</tr>
<tr>
<td>IV</td>
<td>SIMULATION RESULTS .................................... 32</td>
</tr>
<tr>
<td></td>
<td>1. Containing high bandwidth unresponsive flows ... 32</td>
</tr>
<tr>
<td></td>
<td>2. Impact of multiple congested links .............. 36</td>
</tr>
<tr>
<td></td>
<td>3. Reducing RTT bias ................................ 42</td>
</tr>
<tr>
<td></td>
<td>4. Effect of varying load on the bottleneck link .... 45</td>
</tr>
</tbody>
</table>
CHAP\textsc{ter} & \textsc{Page} \\
5. Effect of varying cache size & 47 \\
6. Effect of varying threshold & 52 \\
7. Effect of varying interval & 54 \\
8. Sampling method & 54 \\
V & FUTURE WORK & 57 \\
VI & CONCLUSION & 58 \\
REFERENCES & 59 \\
VITA & 61
# LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE</th>
<th>TCP &amp; UDP data: containing high bandwidth unresponsive flows</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td></td>
<td>34</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LRU operation - 1</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>LRU operation - 2</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>Drop probability for flows in the proposed scheme</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>Operation of the LRU cache</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>Modified drop probabilities</td>
<td>21</td>
</tr>
<tr>
<td>6</td>
<td>Algorithm for scheme 1</td>
<td>23</td>
</tr>
<tr>
<td>7</td>
<td>Too much state information</td>
<td>26</td>
</tr>
<tr>
<td>8</td>
<td>Including 'rate' information in 'count'</td>
<td>27</td>
</tr>
<tr>
<td>9</td>
<td>Modified algorithm for updating 'count' in scheme 2</td>
<td>28</td>
</tr>
<tr>
<td>10</td>
<td>Topology used for simulations</td>
<td>33</td>
</tr>
<tr>
<td>11</td>
<td>Topology used for simulations with multiple congested links</td>
<td>37</td>
</tr>
<tr>
<td>12</td>
<td>UDP throughput and drop rates at the congested routers</td>
<td>39</td>
</tr>
<tr>
<td>13</td>
<td>TCP throughput and drop rates at the congested routers</td>
<td>40</td>
</tr>
<tr>
<td>14</td>
<td>UDP and TCP cache occupancies at the congested routers</td>
<td>41</td>
</tr>
<tr>
<td>15</td>
<td>Comparison of TCP data with other schemes</td>
<td>43</td>
</tr>
<tr>
<td>16</td>
<td>Comparison of UDP data with other schemes</td>
<td>44</td>
</tr>
<tr>
<td>17</td>
<td>Comparison of HTTP data with other schemes</td>
<td>45</td>
</tr>
<tr>
<td>18</td>
<td>Effect of TCP flows with varying RTTs</td>
<td>46</td>
</tr>
<tr>
<td>19</td>
<td>Effect of varying load on the bottleneck link - TCP data</td>
<td>48</td>
</tr>
<tr>
<td>FIGURE</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>20</td>
<td>Effect of varying load on the bottleneck link - UDP data</td>
<td>49</td>
</tr>
<tr>
<td>21</td>
<td>Effect of varying cache size</td>
<td>51</td>
</tr>
<tr>
<td>22</td>
<td>Effect of varying threshold</td>
<td>53</td>
</tr>
<tr>
<td>23</td>
<td>Effect of varying interval - UDP droprate</td>
<td>55</td>
</tr>
<tr>
<td>24</td>
<td>Effect of sampling</td>
<td>56</td>
</tr>
</tbody>
</table>
CHAPTER I

INTRODUCTION

The Internet consists typically of two types of sources, viz., those that respond to congestion by reducing their sending rate and those that do not. Routers, therefore need to implement mechanisms to control congestion, and not merely depend on the end nodes to implement congestion control. The need for end-to-end congestion control has been discussed in [1]. It talks about the negative impacts of non-congestion controlled best effort traffic on the Internet. It emphasizes that relying on end nodes to use end-to-end congestion control as not being a pragmatic approach. This leads to developing mechanisms within the network so that resource allocation and usage can be actively controlled. It talks about various kinds of congestion collapse that might occur and about how unresponsive flows could contribute to this in the Internet. Of these, the paper stresses on congestion collapse due to undelivered packets. This happens when bandwidth is wasted by delivering packets that are dropped in the network even before they reach the destination. This could be a result of increased deployment of open-loop applications that do not use congestion control. Presence of applications that actually increase their sending rate when congestion is indicated could make things worse. Also, there is a need to protect TCP flows that have large RTTs or smaller windows from ones that have smaller RTTs or larger windows from hogging the bandwidth. Thus, routers need to employ mechanisms to allocate their resource (bandwidth here) whether the end nodes implement congestion control or not.

There are two ways by which the router can control its resources, viz., by em-

The journal model is IEEE Transactions on Automatic Control.
ploying different scheduling algorithms and by employing different buffer management schemes. The former decides on which packet to send on the outgoing link, whereas the latter decides on which packet has to be dropped during congestion. Routers need both these to be able to provide different QoS to different flows. In this paper we address the latter issue.

A. Related work

Various buffer management and scheduling schemes have been proposed to bring about end-to-end congestion control. Almost all of the schemes talk about the router dropping packets during congestion. This, of course depends on the buffer management scheme that is built into the router. In order to be able to distinguish between individual flows and give them different QoS by means of a scheduling algorithm and/or buffer management schemes, the routers may need to incorporate per flow state or partial state schemes. Routers in the core of the Internet are typically handling millions of flows, and maintaining per flow state will not scale well. To be able to provide some level of QoS and not maintain per flow state would mean designing schemes that do not incur much overhead and maintain little state. This has led to schemes that derive their buffer management mechanisms by maintaining partial state information. This would maintain information only about those flows that are consuming resources heavily.

1. Various buffer management schemes

The Drop/Tail buffer management scheme drops packets when the buffer at the router is full. It is simple to implement and does not require per flow state information. The advantages are offset by the inherent weakness in that it is unable to distinguish
between flows, it therefore drops packets indiscriminately. Other associated problems discussed in [2] are those of (a) lock out and (b) full queue. The former may result in one or many flows monopolising the entire queue. The latter results in an increase in the end-to-end delay and does not accommodate bursts of data.

RED (Random Early Detection)[3], is an active queue management scheme that claims to solve the problems that were described earlier. This does not require any per flow state information and it helps detect congestion early on. Its ability to mark or drop a packet on detecting congestion ensures that it does not merely rely on the sources to reduce their rates on indication of congestion. RED drops packets that arrive at the router in a probabilistic manner and this probability grows with the estimated average size of the queue (this does not represent the instantaneous queue size). RED algorithm consists mainly of two parts, one involves the estimation of average queue size using a simple exponentially weighted moving average and second, the decision to drop the packet. Two parameters, minth and maxth (min threshold and max threshold), play an important role. No packets will be dropped when the calculated average queue size is below minth. When the queue size exceeds maxth, all the packets are dropped and as it varies between minth and maxth, packets are dropped with a probability that varies linearly from 0 to maxp.

By means of analytical models, the performance of RED has been studied [4], these studies aim at bringing to light the benefits of an active queue management technique. Some of the major findings include the fact that RED is accommodating to bursty traffic, and it does so by increasing the drop rate of non-bursty traffic. Consecutive packet drops occur in RED also, as opposed to the claim that this was eliminated [3]. It does reduce queueing delay but increases jitter of non-bursty traffic.

LQD (Longest Queue Drop)[5], another buffer management scheme requires per flow state information. It stores the number of buffers assigned to each individual
flow and on congestion drops a packet from the flow that has the longest buffer length/queue. Each flow is assigned a minimum buffer size \( b_i \) from the buffer pool of size 'B'. If a flow requires more buffers than \( b_i \), it is allocated the required buffers provided the total current buffer size is less than 'B'. If the total current buffer size is equal to 'B', and the flow under consideration has its allocated buffer space less than \( b_i \), a packet from another flow is dropped. The flow is chosen such that the difference \((q_i - b_i)\) is the maximum, where \( q_i \) is the current buffer size of flow \( i \). Another way to choose a flow would be to pick one randomly such that \( q_i \) is greater than \( b_i \). LQD is based on the fact that if a flow uses more buffer space, then it is the one that uses more than its fair share of bandwidth during congestion.

CHOKe [6], an active buffer management scheme based on RED attempts to contain aggressive flows without maintaining any state information. It is based on the assumption that the contents of the FIFO queue give an indication of which flows are consuming resources heavily. It picks up a packet randomly from the queue when a packet arrives at the router and compares the two. If both the packets belong to the same flow, it drops both of them. If the packets do not match, the randomly drawn packet is left intact and the arriving packet is admitted with the same probability as in RED. CHOKe assumes that aggressive flows have more packets arriving at the router, therefore they have more packets than other flows in the queue. The former triggers more comparisons and the latter results in more matches than the other flows, therefore the scheme would penalise the misbehaving flows.

SRED [7] is a buffer management scheme based on a mechanism that estimates the number of active flows without collecting information on individual flows. This is used to identify misbehaving flows (meaning those taking more than their fair share of bandwidth). It stores information about misbehaving flows in a 'zombie' list. It compares the arriving packet with a randomly picked entry from the zombie list. If
they belong to the same flow, then a 'hit' is declared and the 'count' is incremented in the corresponding entry in the zombie list. If a 'miss' happens, the entry in the zombie list is replaced with a probability 'p'. The assumption is that misbehaving flows are more likely to get 'hits' and therefore can be identified easily. Also, it estimates the number of active flows in the system in order to do the above.

RQRD, Rate and Queue Controlled Random Drop[8] is another scheme developed that has taken its distributed architecture based on core-stateless fair queueing [9]. In addition, it uses two drop precedences, one based on the queue size and the other based on the rate information, to make decisions about dropping packets. RQRD claims that it performs well in the presence of UDP and TCP traffic.

Aggregate Traffic Performance with Active Queue Management and Drop From Tail [10] studies the performance of different active queue management schemes, 'RED' and 'gentle RED' on aggregate traffic at a congested router. The most important observation of the paper is that varying AQM parameters has little effect on the metrics that were used. It brought to light the sensitivity of the same on different traffic characteristics and has compared how droptail fared against any of these schemes.

2. Comparison of the schemes

Mechanisms like DropTail and LQD do not detect congestion early on, they wait for the buffers to get full and when they can't accommodate any more, they indicate congestion to the sources by dropping packets. RED and CHOKe detect congestion early on and indicate the same to the sources. This helps reduce global synchronization and also accommodates sudden bursts.

RED and CHOKe do not require maintainence of any state information. They pick packets randomly for dropping/marking and this is proportional to the flow's
arrival rate at the router. DropTail does not require any per flow state information too and it drops packets indiscriminately. LQD, on the other hand requires per flow state information to make decisions about which packet it can drop during congestion.

DropTail does not contain aggressive flows. It has been shown [2] that aggressive flows can starve the other flows. RED, however claims that its random dropping and marking policies are based on the flows arrival rate, so it may contain sources that are aggressive. CHOKe follows a similar scheme as RED and CHOKe can contain aggressive flows as long as there are more packets from those flows in the buffer at the time of congestion. LQD can contain aggressive flows provided the number of flows in the system does not exceed the buffer available at the router. Also, if the traffic is bursty, it may not be able to penalise the aggressive flows unless it is backed by an appropriate scheduling algorithm.

Most of these schemes evaluate the performance by measuring the 'fairness' criterion. Fairness is defined as follows:

If \( N = \) Number of flows and \( C = \) Capacity of the bottleneck link, the fair share of each of the 'N' flows, \( \alpha \) is given by:

\[
\alpha = \frac{C}{N}. \tag{1.1}
\]

It is evident that for the above definition of fairness to be met, the system must be able to identify the number of flows (instantaneous or otherwise) in the network at any point in time. Based on the above, DropTail does not provide any guarantees for fairness. LQD does provide fair sharing of bandwidth provided there are enough buffers to hold all the flows in the system, and also if it incorporates a scheme to alleviate the effects of multiple packet drops in a single window for TCP flows. However, aggressive UDP flows can reduce the fairness provided by LQD[5]. RED and CHOKe are able to provide fair allocation of bandwidth among the TCP
flows and are not able to contain aggressive flows completely. Even among the TCP flows, the achieved fairness of RED is not close to 1.1 or that of LQD[5].

The inherent RTT bias against TCP flows that have long RTTs is an important issue that most buffer management schemes attempt at addressing. RED and CHOKe do not reduce the effects of this bias. LQD does eliminate this to a certain extent. DropTail does not eliminate this bias.

B. Shortcomings of the existing approaches

All of the schemes mentioned in the previous section fail to take into account the heterogeneity of the Internet to the truest sense. The Internet typically consists of two types of flows, the long-lived high rate flows and the short-lived flows that do not pump in as much data as the long-lived flows. It has been shown that [11], most of the traffic in bytes in the Internet is generated by flows that are long lived and that are high bandwidth in nature. But most of the flows in the Internet are the short-lived type that do not generate as much traffic as the long-lived flows. These short-lived flows may be the ON-OFF type that send in some data, stay idle for a period and then send some more data.

Most schemes try to achieve fairness by estimating the number of flows that are present in the system to be able to satisfy the definition of fairness given in reffair. This is a difficult thing to do. With a great number of short lived flows in the system that are idle for longer periods than active, it is hard to estimate the number of flows accurately. This being the case, the fair share of bandwidth defined for each flow does not hold. This is because there could be 'n' short-lived flows that would require only a few KB of the bandwidth, but a few long lived flows that would need more bandwidth than their fair share. It would be inappropriate in this context to base
the evaluation criterion on fairness. Trying to achieve max-min fairness would not be possible since the demand of the flows is not known apriori.

Ideally, one would want to keep track of the long-lived flows and drop packets from these flows in situations of congestion, rather than drop packets from the short-lived flows. The rationale behind this is that dropping packets from short-lived flows may not reduce congestion that is being experienced as they are typically not the cause for the congestion. As a result even if they reduce their rate, it would not make a significant difference to the network status. More importantly these flows may belong to HTTP traffic or Telnet like traffic that are sensitive to delays because of their interactive nature, so dropping a packet from these flows would not be a wise thing, unless absolutely necessary. Whereas, dropping packets from flows that are causing the congestion to happen, viz., the high bandwidth flows as seen by the router, would make a great deal of difference. These high bandwidth flows may reduce their rate when they notice congestion and therefore experience less drops or they may not respond and continue experiencing high drops.

C. Organization

In chapter II, the necessity for a new scheme and also the proposed scheme are discussed. It also gives the implementation details with the cost analysis. Chapter III discusses the evaluation criteria and chapter IV evaluates the proposed scheme by simulation results. Chapter V discusses future work and chapter VI concludes the research.
CHAPTER II

THE PROPOSED SCHEME

A. Why another scheme?

The previous chapter discussed various merits and drawbacks of some of the existing buffer management schemes. We propose a new scheme that aims at addressing some of those issues. The new approach is not based on 'fairness' like the traditional schemes. The reasons for this have been discussed in the previous chapter.

There is an increasing deployment of video and other applications that do not employ congestion control. Even though these applications constitute a small fraction of the current network load, it is expected that these applications will continue to increase their share of the network traffic. In this scheme, we propose able to protect low bandwidth flows and also flows that respond to congestion.

To motivate our work, we will first show the ineffectiveness of some schemes discussed in chapter I by means of simulations and show how the proposed scheme fairs better. We will show that most of the schemes are ineffective when a large number of non-responsive applications constitute the majority of the traffic.

We also propose a way to identify high bandwidth flows, which can be totally decoupled from the underlying buffer management scheme in the routers. Meaning, it can be employed irrespective of the kind of buffer management schemes used.

In addition, we also propose a method to couple the above with RED. Despite the fact that [12] gives reasons why RED should not be deployed. We claim that an appropriate mechanism for identifying high bandwidth flows will help RED make wise decisions about dropping packets during congestion. This would improve the performance of RED.
Also, by means of the simulation results, we show that Droptail performs better than RED in a heterogenous network. This is in accordance with the findings in [13].

We claim that the proposed scheme (a) recognises high bandwidth flows accurately (b) penalises these by giving them more drop rates than low bandwidth flows and (c) protects short-lived flows. We will show through simulations that the claims are true.

B. Overview of the scheme

We consider various types of flows in this scheme viz., long-term high bandwidth flows (referred to as high bandwidth flows), short-lived flows, and low bandwidth flows. Flows that pump data at a rate that is greater than acceptable to the network (this is typically decided by the ISP) over a period of time are long-term high bandwidth flows. Those that pump bursts of data over a short period and stay idle for some period and continue this process are short-lived flows. The others are classified as low bandwidth flows simply because they do not violate the rate limit. Among the long-term high bandwidth flows we identify two classes, one that reduces its rate and starts sending data at a lower rate when congestion is indicated. The second class of applications are non-responsive to congestion. TCP flows are typical examples of the long-term high bandwidth flows that respond to congestion. UDP sources pumping data at high rates with no congestion control mechanism built into them can be classified as long-term high bandwidth flows that do not respond to congestion. HTTP transfers over the Internet can be classified as short-lived flows. UDP sources that send at a low rate and telnet type interactive applications can be classified as low bandwidth flows.

We propose a scheme that can be used by a router to distinguish high bandwidth
flows, short-lived flows, those that respond to congestion and those that do not. We exploit this to recognise long-term high bandwidth flows and provide higher droprates for them when compared to short-lived flows and low bandwidth flows. Also, our scheme can distinguish between high bandwidth flows that respond to congestion and those that do not, to be able to give them different drop rates. By doing so, we propose to be able to give short-lived flows and responsive flows, higher throughput.

There are two components involved in this process (a) identifying high bandwidth flows accurately (b) penalising the identified high bandwidth non-responsive flows aggressively.

1. Identifying high bandwidth flows

Packets from high bandwidth flows will be seen at the router more often than those from short-lived flows or even other low bandwidth flows. For eg., a flow that is pumping data at the rate of 4Mbps with 1000 byte packet size, would roughly pump 1000 packets per second. This means that the router will see 1000 packets of this flow every second. While, a flow that is pumping at the rate of 1Mbps will send 125 packets per second. Short-lived flows, that are characterized by HTTP transfers are typically the ON-OFF type, send data intermittently. Thus, packets from such flows are not seen at a constant rate at the router. When they are seen, the data is much less than that of high bandwidth flows. So, by observing the arrival rates of flows for a period of time, the router can distinguish between high bandwidth and low bandwidth flows.

In order to identify high bandwidth flows at the router, an LRU (Least Recently Used) cache is employed. This cache is of a fixed pre-determined size, 'S'. In an LRU cache every new entry is placed at the topmost (front) position in the cache. The entry that was least recently used is at the bottom. This is chosen to be replaced
when a new entry has to be added and there is not enough space in the cache. This mechanism ensures that the recently used entries are present in the cache and in a position closer to the top. The objective is to store state information for only long-term high bandwidth flows in the LRU cache.

The router keeps track of the flow and the amount of packets it has pumped, in its LRU cache. Every time the router sees a packet, it searches the cache to check if an entry for that flow has been made in the cache. If there is an entry for that flow, we say that a hit for that flow has occurred. If there is no entry for that flow we say that a miss for that flow has occurred. If a miss occurs, it adds this entry (consisting of the flow id and the packet count) into the cache if there is space in the cache. If there is no space in the cache, it replaces the entry for the flow whose packet the router least recently saw (the bottom most in the cache) with a probability 'p'. It adds this entry in the topmost position in the cache, making it the most recently seen flow, and also initialises its packet count. If a hit occurs, the router updates the position of the entry in the cache (brings it to the topmost position) and updates the packet count.

Everytime a hit occurs, or everytime a miss occurs and a flow is replaced, the flow under consideration is brought to the topmost position in the cache so that it is replaced with the least probability when compared to the other entries in the cache. The reason being, this was the most recently seen flow, and the others in the cache were seen some period of time earlier than this one. Therefore, if we need space in the cache, the element to be replaced should be one of the older flows that are lower in the cache. So, in this way the cache automatically marks the age of the entries in the cache. Ultimately, this topmost flow would be pushed to the bottom of the cache if the router does not see its packets often enough. This works well with our scheme wherein we only want to maintain state for high bandwidth flows.
When there is no space in the LRU cache, the oldest flow is replaced only with a certain probability to make room for the new flow. This is because, we only want to keep track of the actual high bandwidth flows in the LRU. If an entry were to be replaced every time a new flow arrived at the router, some of the high bandwidth flows could end up being replaced by short-term or low bandwidth flows. This would lead to entries in the cache not being the ones corresponding to the high bandwidth flows. For a flow to have an entry in the cache, it has to be seen by the router at least a certain number of times before it can be classified as a high bandwidth flow and put in the cache. This eliminates recording short-lived, low bandiwdth flows in the cache and saves space for flows that are actually high bandwidth in nature. With a cache of limited size, a flow has to arrive at the router frequently enough to remain in the cache. Short-term flows or low bandwidth flows are likely to be replaced by other flows fairly soon. These flows do not pump packets fast enough to keep their cache entries at the top of the LRU list and hence become candidates for replacement. High bandwidth flows are expected to retain their entries in the LRU cache for long periods of time.

However, one can’t classify a flow as high bandwidth by merely checking if it is present in the cache. For instance, at a particular router, one may categorize flows that pump data greater than 3Mb as high bandwidth flows. But a flow that sends data at a constant rate of 1Mb may also have an entry in the cache. Therefore some mechanism must be incorporated by means of which the router can discern the difference between a high bandwidth flow and a flow that is pumping in data at a constant low rate. A parameter called 'threshold' is employed for doing this. At any instant, flows that have sent data greater than the 'threshold' are considered high bandwidth flows. Flows that have not sent as much data are considered low bandwidth flows despite the fact that they are present in the cache. Short-term
flows and low bandwidth flows are likely to be replaced from the cache before they accumulate a count of 'threshold'. The parameter 'threshold' allows us to account for the burstiness of flows.

The following experiment shows that the scheme works and does indeed identify the high bandwidth flows.

A setup with 300 short-lived HTTP flows, 20 long-lived TCP flows (which depict at times, high bandwidth responsive flows) and 20 long-lived UDP flows (of which some depict high bandwidth unresponsive flows) was used. A cache size of 30 and probability of 1/40 was used, and the flows were allowed to start and stop at random times. Figures 1 and 2 show that the (1) LRU cache is indeed able to capture flows that are high bandwidth in nature and those that pump data at a constant rate, (2) that the cache is able to take the dynamic nature of flows starting and stopping at various periods into account and record only data about the appropriate flows and (3) responsive flows stay for shorter periods in the cache.

These graphs represent the state of the cache at various intervals between 0 and 500 seconds during which UDP flows pumped data for random periods. The x-axis for the UDP graphs represents UDP rate in Mb and for the TCP graphs, the flow number. The y-axis in represents the time the flow stayed in the cache (in seconds). It is clear from these that when a UDP flow stops sending data, it gets out of the cache making room for other flows, and when it starts sending data, it gets captured in the cache again. For example, in the time period between 250 to 300 seconds, there were no UDP flows sending data. The graphs at this time show that the cache has state for only TCP flows. Once the UDP flows start sending data again, after 300th second, they start occupying space in the cache. This results in most of the TCP flows being replaced. Also, it is evident from these graphs that a UDP flow that pumps data at a high rate occupies the cache for a longer period than a low
bandwidth flow, or even a TCP flow.

2. Cache size

The size of the LRU cache plays a significant role in being able to identify high bandwidth flows. If the cache size is smaller than the number of high bandwidth flows, the router would not be able to identify all the high bandwidth flows. As a result, some may escape punishment. These, would appear as high bandwidth flows at routers downstream, and would eventually be penalized. If the cache size is large enough to identify all the high bandwidth flows, they would be punished during congestion. If the cache size is larger than the number of high and low bandwidth flows together, all of them would remain in the cache. By choosing a proper function to update the 'count' parameter, high bandwidth flows could be identified accurately.

3. Penalizing high bandwidth flows

After having identified the high bandwidth flows at the router, their drop probabilities are increased. The parameter 'threshold' is used to increase the drop probability of these flows. 'threshold' is a parameter that gives a measure of how many packets a flow can send before it is considered a high bandwidth flow. For instance, a flow may get into the cache on account of sending data at a high rate and/or at a constant rate. If we want to start penalising flows that are greater than 1Mb, we can assume 'threshold' to be 125, based on the fact that each packet is 1000 bytes and the 'count' is incremented each time the router sees a packet of the flow. Thus, once the flows accumulate a count greater than this and remain in the cache, they will be dropped at a higher rate until they drop the rate sufficiently enough to be thrown out of the cache. By doing this, we are able to increase the drop probability of high bandwidth flows compared to the other flows. This 'policy enforcement' mechanism can be coupled
Fig. 1. LRU operation - 1
Fig. 2. LRU operation - 2
with any buffer management scheme. Below, we explain how this can be done with RED.

4. LRU coupled with RED

The scheme is incorporated in RED, so it preserves all the properties of RED. The scheme modifies RED’s drop probabilities using the information in the LRU cache.

![Diagram showing drop probability for flows in the proposed scheme](image)

**Fig. 3. Drop probability for flows in the proposed scheme**

When the queue length builds up, and it is in the region between minth and maxth, RED calculates the drop probability of a packet to be dropped. It is here that we bring the distinction of the high bandwidth and the low bandwidth flows.
In this region, we increase the drop probability of flows that are high bandwidth in nature. This is explained in figure 3.

C. Implementation

1. LRU operation

Every router has an LRU list that records the packet counts and flow ids of the high bandwidth flows and flows that pump data at a constant rate. The algorithm for the LRU operations is given in figure 4. The LRU is implemented as a doubly linked list. Each node contains an entry for the flow id and the packet count. In order to make the search into the linked list easy, it is indexed by a hash table.

2. Preferential dropping

The scheme is incorporated in RED, so flows that are not high bandwidth in nature experience a drop probability that is similar to that in ordinary RED. But flows that are classified as high bandwidth flows, have their drop probability scaled by a factor that is proportional to the observed 'packet count' and 'threshold’. Meaning, for flows that are not high bandwidth flows:

\[ P_{drop} = P_{red} \]

(2.1)

and for flows that are high bandwidth flows:

\[ P_{drop} = P_{red} \ast f(\text{packetcount}, \text{threshold}) \]

(2.2)

Since packet count gives an indication of the arrival rate, the router is able to give different drop probabilities to high bandwidth flows.

If a flow is not a high bandwidth flow, it would not have an entry in the cache and therefore it would observe drop rates similar to what it would in ordinary RED.
A new packet arrives at the router

Is the flow in cache?
- Yes
  1. Update 'count'
  2. Update position of flow in cache

- No

Is there space in cache?
- Yes
  1. Record the flowid of this flow in the cache, in the topmost position.
  2. Initialise 'count' to 1.

- No
  1. Admit this flow into the cache with a probability 'p', removing the least recently seen flow.
  2. If flow is admitted into the cache initialise 'count' to 1.

enqueue packet

Fig. 4. Operation of the LRU cache
But if the flow is a high bandwidth flow, its entry would be found in the cache and it would observe drop rates that are scaled and much higher than those of the low bandwidth flows. By giving high bandwidth flows greater drop rates, we propose to be able to keep the drop rates low for the low bandwidth flows and give them higher throughput.

![Flowchart]

Fig. 5. Modified drop probabilities

Also, an increase in the drop probability of the high bandwidth flow may cause a packet of that flow to be dropped. If this is a responsive high bandwidth flow, (like some TCP flows), then on a packet drop, which it discerns as an indication of congestion, it would drop the rate at which it is sending. If this happens, the router would see fewer packets from this flow. So, 'count' for that entry in the cache will
be updated slowly when compared to what it was earlier. Also, because the router sees packets from this flow less often, it would eventually get to the bottom of the cache and might even get thrown out of the cache while making space for another high bandwidth flow. Once this happens, it would no longer be classified as a high bandwidth flow and would observe drop rates similar to what it would in ordinary RED. Whereas, if a high bandwidth flow does not respond to congestion (like many UDP flows) and does not reduce its sending rate on a packet drop, the router would continue to see many more packets from this flow. It would continuously figure in the top positions in the cache and its 'count' would be updated often. This would result in a greater drop probability for these flows.

The algorithm for this is given in figure 5. Depending on what functions we choose to update the 'count' field and for calculating \( p_{ru} \), we can realize a family of controls. In section D, we highlight two such schemes.

3. Cost

The LRU being implemented as a doubly linked list, insertion and deletion of a flow takes \( O(1) \) time. Searching for a flow in the linked list would take linear time if it were a simple doubly linked list. A hash table is used to make the search \( O(1) \). Every time a new flow is added to the LRU, a hash table entry is made corresponding to this so that a search would take \( O(1) \) time. The memory cost is proportional to the size 'S' of the cache.

D. Scheme 1

This method proposes a way to record data about the flow in the LRU cache and also gives a function to scale the drop probability in RED.
Every time the arrival of a packet results in a hit, the 'count' is incremented by one. This increment by one indicates that the router has seen one more packet of that flow.

When scaling the drop probability of cached flows that have the count greater than threshold, we employ the following function:

\[
p_{ru} = \frac{\text{count}}{\text{threshold}}
\]  

(2.3)

\[
P_{\text{drop}} = p_{\text{red}} \times p_{ru}
\]  

(2.4)

The algorithm for this scheme is given in figure 6.

When the queue length is between \text{minth} and \text{maxth} do the following

if ( flow is in the cache )

begin

if ( flow's count \geq 'threshold' )

\[
p_{ru} = \frac{\text{count}}{\text{threshold}}
\]

\[P_{\text{drop}} = p_{\text{red}} \times p_{ru}\]

end

else

\[P_{\text{drop}} = p_{\text{red}}\]

end

drop the packet under consideration with a probability \(p_{\text{drop}}\)

use the following function to update 'count' for each flow

if ( it is a new entry )

\[\text{count} = 1\]

else

\[\text{count} = \text{count} + 1\]

Fig. 6. Algorithm for scheme 1

In this scheme, 'threshold' gives the upper limit on the number of packets a flow can send and be considered a low bandwidth flow. If the 'threshold' is set to 125 packets, a flow is considered a low bandwidth flow if it pumps at a rate such that its count never increases beyond this threshold of 125. All other flows that exceed this
threshold are considered high bandwidth flows and are subject to punishment.

The quantity 'count/threshold' is always greater than one for flows that have count greater than threshold. This amounts to an increase in the drop probability for the high bandwidth flow by a factor that corresponds to count/threshold. The more aggressively the flow sends packets, the greater will the value of 'count' be and the greater is its drop probability.

Also, when there is no congestion in the network, or when $p_{red}$ is zero, the factor $p_{ru}$ does not contribute anything to the drop probability, so the scheme behaves exactly like ordinary RED.

1. Problems encountered

The simple way of incrementing the count by one on a packet arrival seemed to work in most cases except when there were fewer flows and the cache size was large. In such a scenario, all the long-lived flows remained in the cache. After a certain period of time, the 'count' for each one of the flows would reach a state where $p_{ru}$ would always be greater than one. At times of congestion, the probability of dropping packets from all the flows therefore was almost the same. The reason this happened was because none of the flows got swapped out of the cache and the counts for none of them was reset to one. This was due to the reason that there was too much state information maintained and the information recorded was not a true indication of the 'rate' at which the flow pumped its packets into the network. Even after the TCP flows responded to congestion by reducing their rate, they were not thrown out of the cache since there was no contention for the cache space. The scheme proposed in the next section eliminates this flaw.

To illustrate, an experiment was conducted with 20 TCP, 20 UDP flows and a cache size of 50. Figure 7 shows the results. TCP flows were penalised less than the
UDP flows. Since all the long-lived flows were cached and TCP flows reduced their sending rate on a packet drop, UDP flows were able to achieve a greater percentage of the bandwidth. The scheme was unable to discern the difference between flows that responded to congestion and those that did not.

E. Scheme 2

In scheme 2, we overcome some of the problems that were present in scheme 1. The update function was modified to include the 'rate' information of every flow that was cached.

Basically since the high bandwidth flows were all getting cached, a mechanism that would differentiate the penalty given to these was required. An obvious solution is to include the rate information in the 'count'. In order to incorporate the 'rate' information, we define a new parameter 'δ'. 'δ' is used to control the amount of the penalty imposed on the high bandwidth flows. It is a configurable parameter that is used to decide the rate above which flows that are high bandwidth would have their 'count' incremented. For instance 'δ' could be 8ms, meaning all flows that send at this rate or higher would experience a constant increase in their 'count' value. The flows sending packets slower than 8ms would have their counts decremented. This gives an additional mechanism for controlling the 'threshold' rate over which packets will experience higher drop rates.

Figure 8 explains the principle behind this mechanism. On the arrival of a packet, the router looks up the cache. If this results in a miss and an entry for that flow is made in the cache, the time of arrival is recorded in the cache along with the flow id as the 'timestamp'. If the lookup results in a hit, the current time is compared with the existent 'timestamp'. If it is less than 'δ', meaning the packet has arrived at a rate
Fig. 7. Too much state information
faster than that acceptable, its 'count' is incremented accordingly. If the difference is
greater than 'δ', the flow is sending packets at a rate slower than the limit, so this flow
needs to be protected and the 'count' should not be incremented. Also, a scenario
may arise wherein a flow sends data over a period of time and therefore has an entry
in the cache. After a while, it may stop sending data and may resume later. At this
point, we need to incorporate this information into 'count', this function helps record
all this.

Let 't' = rate at which the flow is pumping data.
'delta' = rate above which scheme 2 might start punishing flows.

In this case, t < delta, meaning the flow is pumping data at a higher rate than acceptable.
The scheme therefore has to accumulate 'count' for such flows more aggressively than the others
in order that they be punished more aggressively.

In this case, t > delta, meaning the flow is pumping at a rate that is acceptable and is a low
bandwidth flow. The scheme should protect this flow and not give it drop rates greater than
what it would observe in ordinary RED.

Fig. 8. Including 'rate' information in 'count'

The algorithm in figure 9 describes this. The scheme is able to distinguish be-
tween flows that pump at different rate because we increment 'count' based on the rate at which the flow sends packets. Previously, the following method was followed in case of an update.

On arrival of a packet,

if (cache lookup = 'hit')
begin
    count = count + 2 - (currenttime - flow->timestamp)/δ 
    flow->timestamp = currenttime
end
else
    count = 1
    flow->timestamp = currenttime
end

Fig. 9. Modified algorithm for updating 'count' in scheme 2

\[ \text{count} = \text{count} + 1 \quad (2.5) \]

The new method to increment count is according to:

\[ \text{count} = \text{count} + 2 - (\text{currenttime} - \text{timestamp})/\delta \quad (2.6) \]

The RHS of the above is derived based on the following:

\[ (\delta - \text{currenttime} - \text{timestamp})/\delta \quad (2.7) \]

Equation 2.7 gives how much faster the flow is sending its packets compared to our defined rate. If the flow is sending at a slower rate, then (currenttime - timestamp) would be greater than δ and the above term would be negative. This would amount to a decrease or in some cases a slight increase of 'count'(depending on the way one implements the update of the 'count'). If the (currenttime - timestamp) is less than δ, the flow is sending at a rate faster than acceptable, so the above term would be
positive, resulting in an increase of 'count'. Also, the above takes care of the varying rates of flows. A flow that sends at a higher rate has its 'count' incremented at a higher rate.
CHAPTER III

EVALUATION CRITERIA

In this approach, we have moved from the usual metric of 'fairness'. We define other criteria based on which we evaluate the scheme.

A. Protecting short-lived flows

The fact that bulk of the traffic on the Internet is generated by long-lived flows and that the short-lived flows like the HTTP transfers and telnet sessions etc generate very little traffic has been established [11]. These short-lived flows are typically not responsible for congestion. Therefore, if we truly need to reduce the congestion in the network, we need to drop packets from flows that are responsible for causing the network to be congested. These typically are the long-lived flows. Also, some of these short-lived flows are delay sensitive. In order to be able to serve these better, few or no packets from these flows should be dropped. Existing schemes do not take this into account, whereas the proposed scheme does.

B. Greater penalty for high bandwidth flows

We have defined a certain rate above which flows are branded as 'high bandwidth flows'. These could be both responsive and unresponsive flows. Given that these flows consume a greater bandwidth than others, they ought to be penalised more than the others. Meaning, these flows should have more packets dropped during congestion than others.
C. Greater penalty for unresponsive high bandwidth flows

Among the high bandwidth flows, some respond to a packet drop during congestion and drop their sending rates. This results in a period when they pump data at a rate that is lower than 'threshold'. The mechanism takes into account this and gives these flows lower drop rates when compared to flows that do not respond to congestion and do not lower their sending rate.

D. Performance when compared to other schemes

In general, we compare the performance of this scheme with other schemes viz., Droptail, RED, LQD and CHOKe and show by means of simulations that the scheme fairs better than any of these schemes.
CHAPTER IV

SIMULATION RESULTS

NS-2 [14] was used to simulate the network conditions required to prove the results. The topology used for these shown in figure 10. The bottleneck bandwidth between routers R1 and R2 is 40Mb with a link delay of 2ms. The TCP and UDP sources are connected to R1 by means of links that have 10Mb and a delay of 32ms (unless otherwise mentioned). The TCP and UDP sinks are connected to R2 by means of links with 10Mb and 32ms delay. HTTP traffic was generated randomly between the link R1 and R2 using TCP flows on both sides. RED parameters that were used in the simulation were, minthresh = 1/4 * buffer size at R1, maxthresh = 3/4 * buffer size at R1, maxp = 0.1 and queue weight = 0.002. The flows pumped packets of size 1000 bytes. The TCP and UDP throughputs are always the average throughput, unless otherwise mentioned.

1. Containing high bandwidth unresponsive flows

In this experiment, we have 20 UDP flows that are pumping traffic at the rate equivalent to the full capacity of the link, meaning 40Mbps. Also, we have 20 TCP flows. The former represent unresponsive high bandwidth flows and the latter responsive flows that may be high bandwidth flows at times. 300 HTTP flows were present in the system at different periods. Buffer size was 160 packets. Flows were started at random times and the simulation was run for a period of 500 seconds. LRU cache size was 30, the threshold was set to 125 and the interval to 8ms. Table I shows the results of this simulation.

All the UDP flows are sending data at a constant rate, therefore they end up replacing the TCP flows more often and occupy the cache for a period of time equal
Fig. 10. Topology used for simulations
<table>
<thead>
<tr>
<th>UDP Rate</th>
<th>Time in sec</th>
<th>UDP droprate</th>
<th>UDP Thru in Mb</th>
<th>Time in sec</th>
<th>TCP droprate</th>
<th>TCP Thru in Mb</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.10</td>
<td>1</td>
<td>0.0</td>
<td>0.100</td>
<td>0.77</td>
<td>5.25</td>
<td>0.829</td>
</tr>
<tr>
<td>0.40</td>
<td>500</td>
<td>0.0</td>
<td>0.400</td>
<td>0.80</td>
<td>5.32</td>
<td>0.812</td>
</tr>
<tr>
<td>0.50</td>
<td>500</td>
<td>0.0</td>
<td>0.500</td>
<td>0.79</td>
<td>5.39</td>
<td>0.816</td>
</tr>
<tr>
<td>0.60</td>
<td>499</td>
<td>43.0</td>
<td>0.341</td>
<td>0.78</td>
<td>5.54</td>
<td>0.804</td>
</tr>
<tr>
<td>1.00</td>
<td>500</td>
<td>46.6</td>
<td>0.533</td>
<td>0.78</td>
<td>5.40</td>
<td>0.822</td>
</tr>
<tr>
<td>1.00</td>
<td>499</td>
<td>44.0</td>
<td>0.559</td>
<td>0.77</td>
<td>5.61</td>
<td>0.821</td>
</tr>
<tr>
<td>1.00</td>
<td>499</td>
<td>43.7</td>
<td>0.563</td>
<td>0.78</td>
<td>5.71</td>
<td>0.813</td>
</tr>
<tr>
<td>1.25</td>
<td>500</td>
<td>43.6</td>
<td>0.705</td>
<td>0.78</td>
<td>5.26</td>
<td>0.819</td>
</tr>
<tr>
<td>2.00</td>
<td>499</td>
<td>45.8</td>
<td>1.083</td>
<td>0.76</td>
<td>5.23</td>
<td>0.835</td>
</tr>
<tr>
<td>2.10</td>
<td>499</td>
<td>43.5</td>
<td>1.186</td>
<td>0.76</td>
<td>5.44</td>
<td>0.811</td>
</tr>
<tr>
<td>2.50</td>
<td>499</td>
<td>42.3</td>
<td>1.440</td>
<td>0.76</td>
<td>5.37</td>
<td>0.823</td>
</tr>
<tr>
<td>2.70</td>
<td>499</td>
<td>43.0</td>
<td>1.538</td>
<td>0.78</td>
<td>5.70</td>
<td>0.806</td>
</tr>
<tr>
<td>2.75</td>
<td>499</td>
<td>43.1</td>
<td>1.564</td>
<td>0.81</td>
<td>5.79</td>
<td>0.798</td>
</tr>
<tr>
<td>3.00</td>
<td>499</td>
<td>43.1</td>
<td>1.703</td>
<td>0.74</td>
<td>5.52</td>
<td>0.821</td>
</tr>
<tr>
<td>3.30</td>
<td>499</td>
<td>43.0</td>
<td>1.882</td>
<td>0.79</td>
<td>5.63</td>
<td>0.815</td>
</tr>
<tr>
<td>3.50</td>
<td>500</td>
<td>42.8</td>
<td>2.005</td>
<td>0.77</td>
<td>5.83</td>
<td>0.815</td>
</tr>
<tr>
<td>3.90</td>
<td>499</td>
<td>42.8</td>
<td>2.229</td>
<td>0.79</td>
<td>5.37</td>
<td>0.812</td>
</tr>
<tr>
<td>4.00</td>
<td>499</td>
<td>44.9</td>
<td>2.200</td>
<td>0.77</td>
<td>5.41</td>
<td>0.809</td>
</tr>
<tr>
<td>4.00</td>
<td>500</td>
<td>46.7</td>
<td>2.132</td>
<td>0.79</td>
<td>5.38</td>
<td>0.817</td>
</tr>
</tbody>
</table>
to the length of the simulation, 499s approximately. In this context, because the interval is 8ms, all flows pumping data at a rate greater than 0.5Mb are considered high-bandwidth flows. It is evident that the cache is holding them for a longer period when compared to a 0.1Mb flow. The low-rate UDP flows, pump data at a constant rate and therefore end up replacing the TCP flows more often than being replaced by the TCP flows. But since the "rate" factor has been taken into account in updating 'count', these flows do not experience any added packet drops by virtue of being in the cache. The TCP flows, since they do not pump data at a constant rate, figure for very short periods in the cache. Also, during congestion, they reduce their rates, so ultimately even if a TCP flow gets into the cache, it would be replaced by another flow.

Most of the UDP flows that were sending data at a rate greater than 1Mb, with a few sending below that rate. With an interval of 8ms, the goal was to protect flows that were pumping data at a rate 0.5Mb or lower. As is evident from the results, these flows experience no packet drops.

Since the UDP flows stayed in the cache, their 'count' increased as they continued to send data during congestion. This resulted in more of their packets being dropped, approximately 48%. The TCP flows on the other hand observed only 5% drop rate as they responded to congestion and dropped the rate at which they sent data. This resulted in better throughput for TCP flows as the UDP flows were unable to starve them.

The UDP flows that are high bandwidth get into the cache, and remain in the cache for the duration of the simulation. Most of the flows hardly get replaced by TCP flows. This means, that the 'count' keeps increasing. After a certain point, if it has increased beyond the 'threshold', the ability to distinguish between a 3Mb flow from a 1Mb flow is lost despite including the 'rate' factor in 'count'. This is typical
of the cases when the UDP flows alone dominate the cache and are never replaced. If they do get replaced, their 'counts' are set to zero. This problem remains to be fixed.

The HTTP flows observed zero packet drops in this simulation.

2. Impact of multiple congested links

This experiment was conducted to observe the effect of multiple congested links. The topology for this is shown in figure 11. The buffer sizes at the routers was 160 and 120 at R1 and R2 respectively. The bottleneck bandwidths and the delays were 40Mb, 2ms and 30Mb, 2ms. The LRU cache size was 30, threshold was 135, probability was 1/40 and the interval was 8ms at both the routers. This meant that flows that were pumping data at or lower than 0.5Mb are considered low-bandwidth flows. There were 40 UDP flows, pumping at the link capacity of 40Mb, 20 TCP flows, and about 300 HTTP flows.

When the number of flows exceeds the amount of state we have in the cache, (in this case, the total number of long-lived flows was 60 and the LRU size was 30) the cache is unable to capture all the long-lived flows. As a result, some of them escape punishment. These flows could eventually be captured and punished at routers further downstream. If a high bandwidth flow escapes one router, it is recognised at another router and is penalised there. Eventually the other flows do not suffer as much.

Thus, if there are many UDP flows that are high bandwidth flows, and the cache at the first router does not capture all of them, the mechanism depends on the downstream routers to penalise them. In this simulation therefore, the UDP flows observe a higher bandwidth as few of their packets are dropped at the first congested router R2. But they are penalised at the second congested router R3, and obtain lower throughput. This is evident from figure 12. Also, figure 14 clearly shows that if R2 holds a high bandwidth flow at the cache and penalises it, it appears as non
Fig. 11. Topology used for simulations with multiple congested links
high bandwidth flow for R3, and hence not hold it in R3’s cache for a long time. This is logical, because the high bandwidth flow is being penalised and therefore fewer packets of that flow are seen at R3. It therefore appears as a low bandwidth flow to R3. This results in increasing the ability of R3 to capture other high bandwidth flows that might have escaped punishment at R2. There are some flows that are at the extreme of being high bandwidth in nature, so both R2 and R3 capture them in their cache. Figure 12 shows the drop rates for the UDP flows at both the routers. Flows that experience a high drop rate at R2 experience low drop rates at R3. This depends on which of them were cached at which routers. Flows that were cached at the routers for a longer period experience more drop rate than the others. Flows that stay in both the routers for long periods (flows with rate greater than 1.7Mb), experience high drop rates at both routers.

Figure 13 shows the throughput that the TCP flows obtain at both the routers. When congestion happens, R2 could start dropping packets from a TCP flow if it was a high bandwidth flow. The TCP flow would then drop its rate, and therefore would be seen less often at both R2 and R3. Also, figure 13 shows that no packets from the TCP flows are dropped at router R2, therefore these flows may appear as possible high bandwidth flows at R3. Figure 14 supports this argument and shows that TCP flows that escape the cache at R2 get held at the cache in R3. Due to this, they experience a non zero drop rate at R3. But because these flows respond to congestion, they drop their rates and get out of the cache soon, therefore their drop rates are low and about 1.5%, when compared to UDP drop rates that are as high as 50%.

Figure 15 shows the effects of having an increasing number of UDP applications in the system. RED, LQD, CHOKe, Droptail and the LRU scheme are analyzed here. With 20, 40, 60 and 80 UDP flows pumping at the bottleneck link capacity of 40Mb,
Fig. 12. UDP throughput and drop rates at the congested routers
Fig. 13. TCP throughput and drop rates at the congested routers
Fig. 14. UDP and TCP cache occupancies at the congested routers
the LRU scheme does consistently better than the other schemes considered here. As
the number of UDP flows increased from 20 to 80, the aggregate rate at which they
were pumping was kept the same, viz., 40Mb. This resulted in an increasing number
of UDP flows that pumped data at smaller rates per UDP flow as we move from 20 to
80. The interval being 80ms, the goal was to protect all the flows that were pumping
at the rate of 0.5Mb. When the number of UDP flows was 20, most UDP flows were
sending at a rate greater than 1Mb. So most of the UDP flows were penalised which
resulted in TCP obtaining high throughput. As the number of UDP flows increased,
the number of possible candidates for high bandwidth flows decreased and fewer of
them were punished. This resulted in a lower throughput for TCP flows.

Figure 16 shows the UDP data from the same simulation. There are many
potential high bandwidth flows when the number of flows is less, therefore the UDP
flows end up being penalised. As this number decreases, they are penalised less
and therefore observe higher throughput. The throughput depicted in the figure
is average throughput, so the decreasing curve does not necessarily mean a lower
throughput for the UDP flows on the whole. The scheme does as well as the others
under consideration in those extreme cases and does better than all of them in the
other cases.

Figure 17 shows the effect on short-lived HTTP flows. We only consider drop
rates here, because talking about throughput for these flows does not exactly describe
the effectiveness of the scheme. As is evident from the figure, HTTP flows observe
low drop rates when compared to the other schemes discussed here.

3. Reducing RTT bias

In this experiment, the RTTs of various TCP flows was varied to study the new
scheme when flows have different RTTs. A setup consisting of mainly TCP flows
Fig. 15. Comparison of TCP data with other schemes
Fig. 16. Comparison of UDP data with other schemes
Fig. 17. Comparison of HTTP data with other schemes

was used. There were 18 TCP sources with 6 different RTTs. The bottleneck link capacity was reduced to 20Mb, the rest of the parameters remained the same. There were no UDP flows. The cache size was 30.

Figure 18 shows the results of the experiment. The LRU scheme was able to give TCP flows with short RTTs, greater drop rates compared to TCP flows with longer RTTs. RED and CHOKe, give similar drop rates to all the flows. LQD, though it is supposed to reduce the RTT bias, the difference in the drop rates is not significant. Droptail does not reduce the RTT bias because the dropping is not based on any criteria and is indiscriminate.

4. Effect of varying load on the bottleneck link

In these set of experiments, we address the cases of lightly loaded and heavily loaded bottleneck link. The experiments consisted of varying the number of UDP flows and
Fig. 18. Effect of TCP flows with varying RTTs
the amount of data they pumped into the network. For the cases where the load was
less than 100% of the bottleneck link bandwidth, viz., for the 25%, 50% and the 75%
cases, there were 10 UDP sources each pumping data at the rate of 10Mb, 20Mb and
30Mb together. There were 20 UDP flows pumping data at 40Mb for the 100% case,
and 60 UDP flows pumping data at 60Mb for the 150% case. The number of TCP
sources was 20, and HTTP flows was 300. All other parameters remained the same.

Figure 19 shows the results of the simulations for TCP flows. It is evident that
the proposed scheme does better than the rest in all the cases. We are able to give
smaller drop rates to TCP flows and penalise the high bandwidth unresponsive flows.

Figure 20 shows the results for UDP flows. The drop rates for the UDP flows in
cases when the link is underutilized is zero, showing that the scheme does not affect
the utilization adversely. Also this shows that the dropping functionality is employed
only when there is congestion and not otherwise. Once the load goes above 100%,
the drop rates increase. As is evident from this, we perform better than the schemes
considered.

5. Effect of varying cache size

Internet is dynamic in addition to being heterogeneous, while there is a need to
capture the misbehaving flows all the time, this is not possible or necessary. The
mechanism used to identify the high bandwidth flows in the proposed scheme is by
maintaining an LRU cache. The size of the cache reflects the amount of information
that the router can possess to know approximately the various high bandwidth flows.
This information may be insufficient. The LRU list size may be smaller than the
number of high bandwidth flows. It is also possible that the LRU size is larger than
required to capture the high bandwidth flows.

The following experiments study the effectiveness of the proposed scheme with
Fig. 19. Effect of varying load on the bottleneck link - TCP data
Fig. 20. Effect of varying load on the bottleneck link - UDP data
different LRU cache sizes. The number of TCP and UDP flows were 20 each. UDP sources were pumping data at full link capacity (40Mb). The probability was set to 1/40, threshold to 135 and interval to 8ms, meaning flows that were pumping data at the rate greater than 0.5Mb were branded high bandwidth flows, and were penalised.

Figure 21 shows the results of the experiment. With an LRU size of 20, with 20 UDP flows pumping data at a constant rate, the probability that a TCP flow replacing a UDP flow is low. Even if it does, the TCP flows get replaced soon by the UDP flows. This being the case, the UDP flows, all of them accumulate a large value for the 'count' and tend to have more drop rates. In this scenario, most of the UDP flows pumped at a rate greater than 1Mb, making them all candidates for being penalised. This was to the advantage of TCP flows that got a huge amount of bandwidth. When the LRU size is 30, a few of the TCP flows were also in the cache and therefore they got penalised too. When the size is 50, all the flows were cached and they were all penalised. Here, there is a problem of excess state information. Since none of them were ever replaced, they accumulated 'count' and this created problems in finding the right drop probability. But the effect was the same for all the flows and we were able to give different drop rates to responsive high bandwidth and unresponsive high bandwidth flows even here. This is because the count would accumulate at a lower rate for the TCP flows that respond to congestion since we include the 'rate' factor in this, we are able to realise the drop in their rate and give them lower drop rates.

Another experiment with 'interval' set to 4ms was conducted. In this case, as is evident from figure 21, the throughput and drop rates remained almost the same with the three cache sizes. This shows that the scheme is robust, and that in the first scheme, the parameters were chosen such that TCP flows would get 90% of the bandwidth.
Fig. 21. Effect of varying cache size
6. Effect of varying threshold

The parameter 'threshold' decides the limit beyond which we start penalising a cached flow. We can contain and punish high bandwidth flows that accumulate 'count' beyond a certain 'threshold', in the proposed scheme. This would mean that a flow can send data and accumulate count to 'threshold - $\theta$', where $\theta$ is a very small number, and still get away unpunished if it is able to get out of the cache. It is therefore necessary to know what effect 'threshold' has on different flows. The following experiment explores this. The larger the value of 'threshold', the burstier the flows can be without getting penalized. We expect larger values of 'threshold' to benefit TCP flows.

The setup for the experiment had 20 UDP and TCP flows each, with a probability of $1/40$, LRU cache size of 30 and an interval of 8ms. The rest of the parameters were unchanged.

Figure 22 shows the results of the experiment, when the 'threshold' was varied from 50 to 175. With a smaller 'threshold', there is a greater chance of the TCP flows being penalised before they get replaced. The reason is that the LRU size being 30, there are at least 10 TCP flows that are present in the cache at any point in time. They are punished during their stay in the cache based on the 'count' they accumulate and also on the 'threshold'. It is obvious that the chances that their 'counts' reach the smaller value for 'threshold' is greater than at higher threshold values. Therefore, TCP flows get punished more when 'thresholds' are smaller. Also, it is worthy to mention that despite the fact that they do get penalised, their drop rate is much less compared to those of the UDP flows. HTTP flows obtain zero drop rate in all the cases.
Fig. 22. Effect of varying threshold
7. Effect of varying interval

The ability to classify flows as high bandwidth or not is based on the rate at which they pump data. We decide a particular rate beyond which a flow, it is branded a high bandwidth flow. This is configurable using the parameter 'interval'.

The following set of experiments clearly show that we are able to control the drop rates of flows that are not high bandwidth by using 'interval'.

The experiment had 20 UDP and TCP flows, probability 1/40, LRU cache size of 30 and 'threshold' 135.

Figure 23 shows that with an interval of 4ms, when our target was to protect all flows that were pumping data at less than or equal to 1Mb, we achieved our goal.

8. Sampling method

The present scheme does work for every packet that arrives at the router. To make the scheme more efficient, a new scheme was designed. This scheme samples every other packet at the router. Since the router now looks at only 50% of the packets, 'threshold' and 'probability' are reduced to half their previous values. 'δ' is increased to twice its previous value. The work done is 50% of the previous value and the results obtained are almost similar to the previous scheme. This is shown in figure 24.
Fig. 23. Effect of varying interval - UDP droprate
Fig. 24. Effect of sampling
CHAPTER V

FUTURE WORK

It is evident that the setting of the various parameters in the proposed scheme is not easy. But they certainly are well defined and clearly understood. Further analysis into these by developing analytical models would help arrive at the right values for different networks.

It is possible to tailor the parameters to the current workload at the router. For example, when the available cache space is much smaller than the number of flows exceeding the defined rate, we could adaptively increase the \( \delta \) parameter or decrease the probability \( p \) of a flow’s acceptance into the cache.

Incorporating ways to make the scheme more efficient, viz., handle fewer packets to identify high bandwidth flows.
CHAPTER VI

CONCLUSION

We have proposed a mechanism that could be incorporated in routers to (a) identify high bandwidth flows (b) penalize the high bandwidth flows (c) protect responsive flows (d) protect short-lived flows. The proposed approach maintains partial state at the router to accomplish these objectives. Simulation results show that the scheme is successful in achieving all of these. In addition, it is able to give higher drop rates for TCP flows with small RTT compared to those with larger RTTs.

The identification of high bandwidth flows uses a simple LRU cache and this is independent of the underlying queue management scheme. RED, an active queue management scheme was chosen to prove the effectiveness of the scheme. Also, that it provides more control on the how the network could allocate its resources. This is accomplished by manipulating the various parameters, viz., 'threshold', 'interval', 'lru cache size', and 'probability'.

It is also shown that the packet handling cost remains $O(1)$ and that the memory cost is proportional to the size of the LRU cache.
REFERENCES


[12] May, M., Bolot, J., Diot, C., and Lyles, B “Reasons not to deploy RED”


[20] Ulf Bodin, Olov Schelen and Stephen Pink. “Load-tolerant Differentiation with Active Queue Management.” in Computer Science and Electrical Engineering Lulea University of Technology
VITA

Smitha was born on February 15, 1976 in Mysore, India. She received her Bachelor of Engineering degree in Computer Science from Karnataka Regional Engineering College, Suratkal, India in July 1997. She worked for 2 years in Wipro Infotech Global R&D, Bangalore, India and joined the master’s program in Computer Science at Texas A&M University in August 1999. At Texas A&M her research has been focussed on issues related to congestion management in networks. Her address is Department of Computer Science, Texas A&M University, College Station, Texas 77843-3112.

The typist for this thesis was Smitha.