

Enhancing Internet Robustness against Malicious Flows Using RECHOKe

Visvasuresh Victor Govindaswamy^{1,*}, Gergely Záruba², G. Balasekaran³

¹College of Art and Sciences, Concordia University Chicago, Illinois, 60639, United States

²College of Computer Science and Engineering, The University of Texas at Arlington, Texas, 76019, United States

³Human Bioenergetics Laboratory, Nanyang Technology University, Singapore, 639798, Singapore

Abstract xCHOKe, CHOKe and RED-PD are some of the schemes that have been proposed to control malicious flows in IP networks. Both xCHOKe and CHOKe use CHOKe hits while RED-PD uses RED drops in identifying these flows. In this paper, we are proposing a new scheme called RECHOKe (REpeatedly CHOOse and Keep for responsive flows, REpeatedly CHOOse and Kill for unresponsive flows) that can be used for detection, control and punishment of these malicious flows. It does this by combining the techniques used by CHOKe, xCHOKe and RED-PD. The difference is, unlike in xCHOKe and CHOKe, RECHOKe does not drop packets during CHOKe hits, thereby eliminating the complexity of dropping or marking randomly selected packets already queued. By using both the CHOKe hit and CHOKe-RED drop histories, we show using ns2 1) that RECHOKe is able to detect, control and punish these flows more accurately, when compared to RED, CHOKe and xCHOKe while providing more protection for TCP-friendly flows, 2) how accurate these histories are, when used in conjunction, in identifying malicious flows and 3) the unreliability of RED, CHOKe and xCHOKe in affecting TCP-friendly flows. Other similar proposed schemes either did not perform as well or incur too much overhead.

Keywords TCP Congestion Control, Congestion Avoidance, Active Queue Management (AQM), Buffer Management, Random Early Detection (RED), TCP/IP, Malicious Flows, Denial of Service (DoS)

1. Introduction

TCP (Transmission Control Protocol) is the connection oriented transport layer protocol of the Internet. One of the services TCP provides is the adjustment of flow transmission rates whenever congestion is encountered. Although TCP, together with active queue management (AQM) [1] schemes at the routers, works in controlling the rate thus reducing congestion in the network, it works well only with TCP-friendly or responsive flows. These flows use end-to-end congestion control and reduce their rates in response to packet drops within the network. Their packets are dropped whenever the queue of the router along the path of the flow is already full. However, TCP fails in the presence of malicious or non-responsive flows, allowing these flows to flood the network, robbing valuable bandwidth from TCP-friendly flows. To control TCP-friendly flows from non-adaptive sources such as UDP and non TCP-friendly sources, we are proposing a new preferential dropping scheme called RECHOKe. It aims to improve on existing schemes such as CHOKe [2] and its variants such as Back CHOKe [2], Front CHOKe [2],

Multi-drop CHOKe (M-CHOKe) [2], Adaptive CHOKe (A-CHOKe) [2], CHOKe+ [3] and xCHOKe [4].

Despite its simplicity, CHOKe has been shown [4] to be slow in recognizing malicious flows, and too often punishes responsive flows. CHOKe's implementation advantage of being stateless becomes a disadvantage; if it had kept some state information, it would be able to make better decisions. Moreover, CHOKe is not likely to perform well when the number of flows is large and/or when the high-bandwidth flows have only a few packets in the queue as they have been cut back. Hence, xCHOKe was introduced to keep partial state of flows. However, it has been shown that xCHOKe punishes responsive flows as well but not to the extent of that of CHOKe. Moreover, xCHOKe, like CHOKe, does not control malicious flows with bandwidths smaller or slightly greater than the link capacity. It behaves like CHOKe when the number and/or bandwidth of malicious flows are small.

There are other approaches to punish unresponsive flows but most of the time they incur too much overhead. Schemes like RED-PD [5] require proper values for their parameters such as the target RTT, K, and M. The overhead of using schemes like RED-NB [6] in today's networks lies in that they are complex for the router to constantly calculate the RTT of a flow since RTT is heavily dependent on the ever-changing state of the network. Schemes like RED-NB and Self Adjustable CHOKe [7] also need to access the *protocol* field in the IP packet header to differentiate

* Corresponding author:

ovebat814@yahoo.com (Visvasuresh Victor Govindaswamy)

Published online at <http://journal.sapub.org/ijn>

Copyright © 2015 Scientific & Academic Publishing. All Rights Reserved

between TCP and UDP flows and hence, are not safe as these fields can be disguised in denial-of-service attacks. Schemes like Core-Stateless Fair Queuing (CSFQ) scheme [8] requires modifications to the entire network and an additional field in the IP header; making them impractical for today's networks. Schemes like Flow Random Early Detection (FRED) [9], which incur too much overhead as they seek to insure that all flows receive a fair share of the link's capacity by logically managing the queue on a per-flow basis and maintaining statistics for every flow that has a packet queued in the router. Another scheme, which uses approximate fair queuing, is Stochastic Fair Queuing (SFQ) [7]. SFQ classifies packets into a smaller number of queues than Fair Queuing using a hash function, requiring around 1000 to 2000 queues in a typical router to achieve its full potential [11], thereby making it too costly.

Our scheme combines the ideas behind xCHOKe and RED-PD to detect, control and punish malicious flows which, we believe, are faster than xCHOKe and RED-PD. Since xCHOKe, uses the CHOKe hit history and RED-PD the RED drop/mark history, our scheme uses both these histories to isolate, control and punish malicious flows. We extend xCHOKe, by using RED drop history; this makes intuitive sense since the probability that a gateway chooses a particular flow to mark or drop during congestion is roughly proportional to that flow's share of the bandwidth at the router. We believe that using both histories helps to lower the number of "false alarms" and detect malicious flows faster, especially when the numbers of flows using the queue are large. The simulations in this paper show that RED, CHOKe and xCHOKe are limited in what they can achieve since malicious flows still get much more than their fair share and responsive flows get mistakenly penalized. We will also conduct experiments to verify how accurate the technique of using CHOKe hit and RED drop histories is in detecting malicious flows. RECHOKe helps to isolate, control and punish malicious flows when used with active queue management schemes such as Random Early Detection (RED) [12]. This is an extended version of [13].

The rest of the paper is organized as follows: In sections II introduced the RECHOKe scheme while the simulation details, results as well as analysis are covered in section III. The paper is concluded with future work in section IV.

2. RECHOKe

This section presents the functional description of RECHOKe. In RECHOKe, if the average queue length (avq) is smaller than a minimum threshold Th_{min} , then arriving packets are accepted. If avq is greater than the Th_{min} but smaller than a maximum threshold Th_{max} , a lookup table is queried to see whether the arriving packet's flow label is present there. If it indeed is (called a table hit), the arriving packet is marked for dropping and the flow's associated hit counter is incremented. A packet is then selected at random from the queue and its flow label is compared with that of the

arriving packet. If the flow labels are the same (called a CHOKe hit), the flow label is added to the lookup table with an initial value of one for the associated hit counter. If the flow is already in the table, the associated hit counter is incremented. Unlike xCHOKe or CHOKe, the packets are not dropped or marked for dropping as a result of a CHOKe hit. They are allowed to enter the FIFO buffer. This removes some of the complexity of dropping the random packet after it has been admitted into the buffer and the unreliability of CHOKe hits. To remove this complexity from CHOKe, the authors in [2] proposed to add one extra bit to the packet header so that the bit is set to one if the random packet is to be dropped. When this packet advances to the head of the RED buffer, the status of this bit determines whether it is to be immediately discarded or transmitted on the outgoing line. We feel that this idea leads to a wasteful use of valuable buffer space and the addition of complexities due to the bit modification.

In the RED FIFO, if a packet is dropped or marked (in the presence of ECN [14]) by RED (called a RED hit), then its label is added to a lookup table with an initial value of one for an associated hit counter. If the flow is already in the table, the associated hit counter is incremented. If avq is greater than Th_{max} , the packet is dropped and its label is added to a lookup table with an initial hit counter value of 1. If the flow is already in the table, the associated hit counter is incremented.

The hit counter n of an entry in the lookup table is used to compute the probability p^* and an indicator value c^* which are used to drop packets; these values are computed as follows:

$$p^* = \min(1, p_a \times 2^n) \quad (1)$$

$$\text{and } c^* = f \left(n \geq \frac{\sum_{i=1}^m n_i}{(2 \times m)} \right) \quad (2)$$

where p_a is RED's dropping probability and m is the number of entries in the table. If c^* evaluates to true then the packet will be marked or dropped with probability p^* . The purpose of c^* is that we do not want to punish a TCP-friendly flow, whose previous packets just "accidentally" experienced a CHOKe hit leading to one of its recent packets experiencing a table hit. This scenario is common in CHOKe and xCHOKe, which results in TCP-friendly or responsive flows getting punished unnecessarily.

Hence, in RECHOKe, we update the table when an incoming packet has: i) table hits, ii) CHOKe hits, or iii) RED hits. The first two cases are similar to xCHOKe, while the third is similar to RED and RED-PD behavior. We can use RED drop/mark history in the presence of ECN.

In [16] detailed simulation results can be found on CHOKe hit, CHOKe miss and RED drop behavior. To show problems with CHOKe buffer management we have looked

at two different slightly modified versions of CHOKe referred to as Half1 and Half2. In Half1 CHOKe, packets are randomly chosen from the first half of the buffer while in Half2 CHOKe from the bottom half. We have seen that depending on bandwidth required by malicious flows one of these approaches outperforms the other (and vice versa). RECHOKe starts off with the original CHOKe algorithm, sampling at certain intervals, say at time $t=250\text{ms}$, the first quarter of the buffer with the entries in the table. It computes the occupancy percentage at the first quarter of the occupied queue by flows with entries in the table. If this value is greater than 30%, then RECHOKe reverts to Half1 CHOKe, and to Half2 CHOKe otherwise.

In RECHOKe, we associate a time-to-live (TTL) with each entry in the lookup table. However, TTLs are not refreshed when an incoming packet has: i) table hits, ii) CHOKe hits, and iii) RED Hits. The first two cases again are similar to xCHOKe while the third case is unique to RECHOKe.

2.1. Complexity Cost

The computational complexity for RECHOKe is limited to: i) a hash lookup in the table for each packet which takes a constant time, ii) the computation of p^* and c^* if there is a table hit for that packet, and iii) the sampling of the first quarter of the occupied queue. The space complexity of RECHOKe is the size of the lookup table which can be either implemented as a variable table which grows as needed to approximately the order of magnitude of malicious flows or based on a Least Recently Used or Least Frequently Used scheme.

2.2. Theoretical Analysis

The approximate analysis for RECHOKe is an extension of the analysis for xCHOKe [4] where the system is modeled as a queue with a FIFO queuing discipline and assumes that each source is independent and identically distributed, and the arrival process is Poisson. Likewise, we denote the arrival rate of the i^{th} source as λ_i and the mean service time for each packet as $1/\mu$.

An incoming packet may suffer a hit in three phases of the RECHOKe algorithm: i) table hit, ii) CHOKe hit and iii) RED hit. We will use a similar notation to [4], where P_{TABLE} denotes the probability that an incoming packet will suffer a table hit, P_{CHOKe} denotes the probability that the packet will suffer a CHOKe hit, and P_{RED} denotes the RED drop probability. Let us denote the number of table, CHOKe and RED hits for a table entry by n . If the incoming packet's flow label is in the lookup table, then the probability of dropping an incoming packet on a table hit can be given by

$$p^* = \min(1, P_{\text{RED}} \times 2^n) \times c^* \quad (3)$$

where n is a function of P_{TABLE} , P_{CHOKe} and P_{RED} and c^* is Boolean function represented by

$$c^* = f \left(n \geq \frac{\sum_{i=1}^m n_i}{(2 \times m)} \right) \quad (4)$$

where m is the number of entries in the table.

Assuming that no TTLs are refreshed, the maximum value of n can be approximated by:

$$\lambda_i * \text{TTL} * (P_{\text{TABLE}} + P_{\text{CHOKe}} + P_{\text{RED}}) \quad (5)$$

where TTL is the initial time to live value for an entry in the lookup table. This is greater than xCHOKe's

$$k = \lambda_i * \text{TTL} * (P_{\text{CHOKe}}) \quad (6)$$

This is reflected in Fig. 18 implying the identification process is more rigorous in RECHOKe than in xCHOKe. Hence, the chances of the malicious flow's packet getting dropped are greater in RECHOKe than in xCHOKe. The probability that a packet is not dropped by the RECHOKe algorithm is

$$P_{\text{NoDrop}} = (1 - P_{\text{TABLE}} \times P_{\text{RED}} \times 2^n \times c^*) \times (1 - P_{\text{RED}}), \quad (7)$$

thus:

$$P_{\text{NoDrop}} = (1 - P_{\text{TABLE}} \times P_{\text{RED}} \times 2^n \times f \left(n \geq \frac{\sum_{i=1}^m n_i}{(2 \times m)} \right)) \times (1 - P_{\text{RED}}) \quad (8)$$

xCHOKe's P_{NoDrop} can be given by:

$$P_{\text{NoDrop}} = (1 - P_{\text{TABLE}} \times P_{\text{RED}} \times 2^k) \times (1 - P_{\text{CHOKe}}) \times (1 - P_{\text{RED}}) \quad (9)$$

The value of P_{NoDrop} in (8) is smaller than in (9), since at any instant of time, $n > k$ and if c^* is one, then it implies that the chances of dropping packets from high-bandwidth flows is greater for RECHOKe than for xCHOKe. This is reflected in Figs. 19 and 22. If c^* is zero, it implies that the chances of dropping packets from low-bandwidth flows is greater for xCHOKe than for RECHOKe. This is reflected in Figs. 21 and 23.

3. Simulation Results and Analysis

Using NS-2 [15], we simulate a bar-bell topology to analyze the effects on the RED, CHOKe, xCHOKe and RECHOKe buffers in terms of buffer occupancy by a malicious flow, a constant-bit-rate UDP flow, and competing with 10 TCP flows over a 1Mbps link. All TCP flows have the same round trip propagation delay of 20ms with each

output link having a latency of 1ms and capacity of 10Mbps. The parameters for the RED, CHOKe, xCHOKe and RECHOKe buffers are: $Th_{min} = 10$, $Th_{max} = 50$, $P_{max} = 0.1$.

We ran 11 simulations, starting with a 0.5Mbps and 1Mbps UDP flow and then increasing the rate of the flow in 1Mbps increments. Each simulation was long enough for initial transients to settle and be insignificant.

3.1. Experiments on CHOKe

To analyze the CHOKe buffer, we define the following terms. *CHOKe victim*: the packet that is randomly selected from the FIFO buffer to be compared with each arriving packet. A *CHOKe hit* occurs when the CHOKe victim's flow matches that of the arriving packet; otherwise a *CHOKe miss* has occurred. A CHOKe hit can be either a good or bad; it is good if the arriving packet belongs to a malicious flow (and hence, the CHOKe victim is also from that malicious flow); it is bad otherwise. A CHOKe miss can also be either good or bad. *Good CHOKe miss* occurs when the arriving packet belongs to a TCP-friendly flow and the CHOKe victim is from any other but that particular flow; otherwise we talk about a *bad CHOKe miss*.

Fig. 1 shows the percentages of CHOKe hits and misses. We can observe that when the UDP flow is less than three times the link capacity, the percentages of misses is about 5-40% greater than the percentages of hits. However, they drop rapidly from 68% to 53% when the UDP flow rate is increased from 0.5Mbps to 2Mbps. Increasing the rate of the UDP flow further only decreases the percentages of misses to about 3-7%.

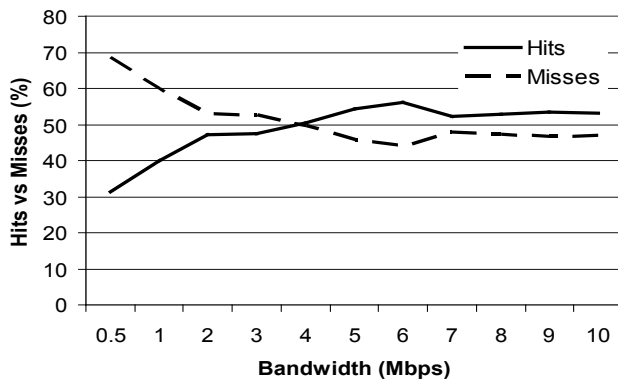


Figure 1. Hits vs Misses

As for hits, its percentages increase rapidly from 31% at 0.5Mbps to about 47% at 2Mbps. After 2Mbps, the increase hovers around 1-8% and improves only about 1-6% over that of the misses. This increase is small compared to that of the UDP rate. The increase in its rate should imply that more UDP packets should occupy the buffer enabling more hits and less misses. However, from Fig. 1, this was not the case.

Fig. 1 presents an interesting phenomenon. To investigate this, we divide the packets within the buffer into 4 quarters, where Qtr1 is at the head of the queue. We call these quarters as Qtr1, Qtr2, Qtr3 and Qtr4. We measure the percentages of average buffer occupancy of the UDP flow at each quarter

during CHOKe hits for every 10s of the 30s simulation. These are shown in Figs. 2 and 3 respectively. The third 10s period is similar to that of the second and hence, is left out of this discussion. High UDP buffer occupancy percentages at the tail of the queue are expected since the UDP packets arrive at the tail section of the buffer. However, this results in choking the TCP flows. The UDP flows advance in the queue and are tapered off slowly due to the effects of CHOKe hits. The head of the queue has the least UDP buffer occupancy percentages showing that CHOKe has been successful in reducing the number of packets in the buffer. Although there are some reductions before the first 10 seconds, the average UDP occupancy in Qtr1 is between 22 and 58% as the UDP rate is varied. After the first 10 seconds, increasing the duration of the flows has no big effect on the outcome. The reduction in Qtr1 reflects the reduction in the bandwidth of the UDP flow but the unresponsiveness in the reduction is due to bad hits (see Fig. 4) and bad misses (see Fig. 5).

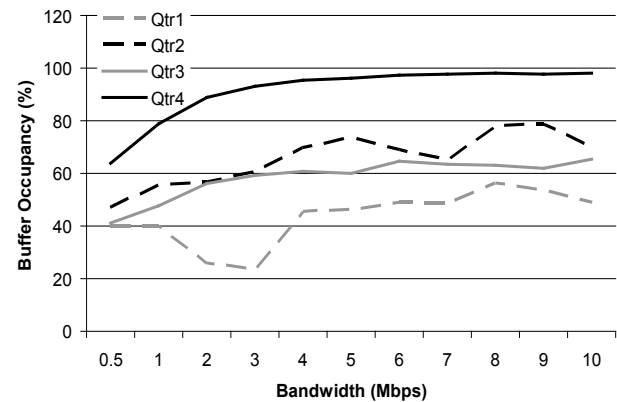


Figure 2. Percentages of average buffer occupancy by the UDP flow during CHOKe hits (0-10s)

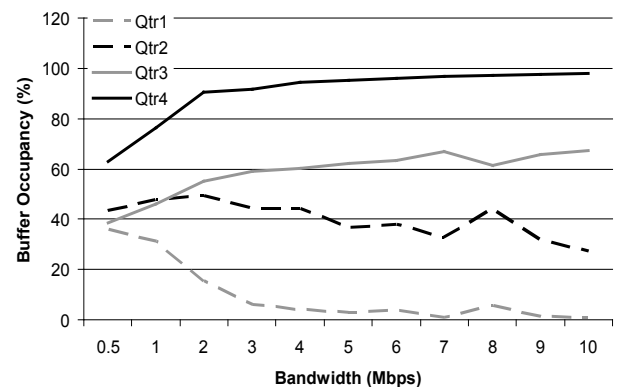


Figure 3. The percentages of average buffer occupancy by the UDP flow during CHOKe hits (10-20s)

Although the number of bad hits, compared to those of good hits, is small, it has an adverse effect on the throughput of the TCP flows since it forces the TCP sources into reducing their rates. At 0.5Mbps, the number of good hits is 14 times greater than those of bad hits. As the UDP rate is increased, the number of UDP packets arriving and

occupying the buffer increases. This leads to an increased number of good hits. Meanwhile, the number of bad misses, in comparison to those of good misses is large. This leads to a greater number of UDP packets occupying and leaving the buffer in comparison to instances when there are successful good hits. Bad misses lead to increases in the throughput of the UDP flow. Hence, to improve the performance of CHOKe we need to *reduce drastically the number of bad hits and misses*. This would lead to more reduction in the throughput of the UDP flow.

Fig. 5 shows CHOKe's ratio of bad misses over good misses. At 0.5Mbps UDP rate the ratio is at 1.4 implying that the number of bad misses is only slightly higher than those of good misses. However this ratio increases rapidly with a growing UDP transmission rate, implying that arriving UDP packets are increasingly compared with queued TCP packets.

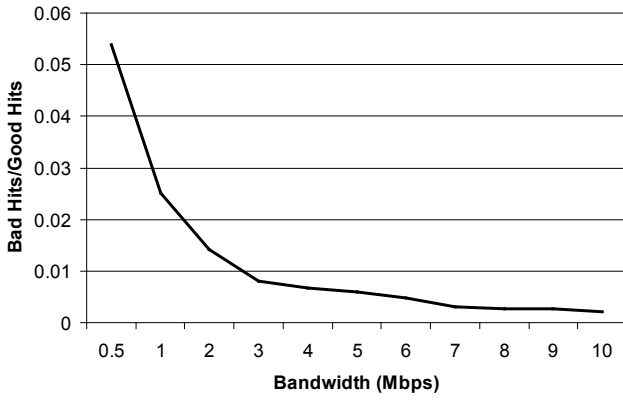


Figure 4. Ratio of Bad and Good Hits against UDP's bandwidth

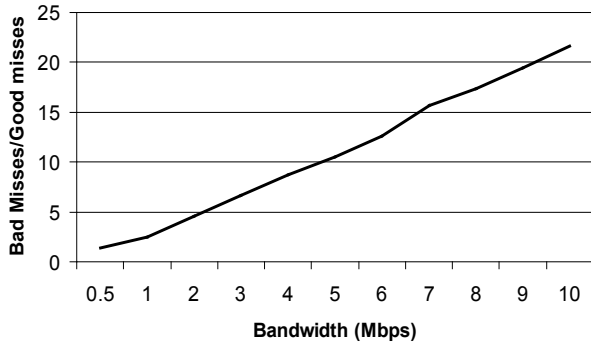


Figure 5. Ratio of Bad Misses over Good Misses

Fig. 6 shows the average UDP buffer occupancy percentages during CHOKe misses. Qtr4's average UDP occupancy percentages increase from 46% to 82% (compare to the 63% to 98% during CHOKe hits in Fig. 2). Instead of the average UDP occupancy percentages increasing in Qtr3 from 41% to 70% as is in the case with CHOKe hits, it decreases after it has initially increased. In fact, as expected, all 4 quarters show reduced average UDP buffer occupancy during CHOKe misses when compare to during CHOKe hits.

Of all the quarters, Qtr4 reduces the least. This is mainly because most of the random packets are chosen from the first 3 quarters as seen from Figs. 6 and 8, leading to the large number of misses. As a result, UDP packets are increasingly allowed to enter the buffer.

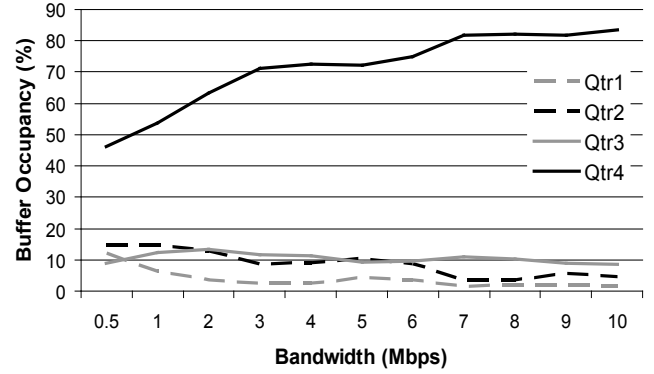


Figure 6. Percentages of average buffer occupancy by the UDP at each quarter during CHOKe Misses for the entire simulation

From Figs. 2, 3 and 6, the chances of achieving good hits improves tremendously if victims are chosen from Qtr4. The reason is that Qtr4 has the most number of UDP packets during hits and misses.

Figs. 7 and 8 show the percentages of location of victims within each quarter; when the UDP flow's rate is changed, during CHOKe hits and misses respectively. Fig. 7 clearly shows that victims chosen from Qtr3 and Qtr4 increasingly matched the arriving packets as the rate of the UDP flow increased. The rate of increase, however, was more in Qtr4 than Qtr3 since more UDP packets occupied the region (see Fig. 3). Next we analyze these CHOKe hits to distinguish between good and bad hits (see Figs. 9 and 10 respectively). Fig. 9 shows more good CHOKe hits for Qtr3 and Qtr4 as the rate is increased due to increasing occupancy of Qtr3 and Qtr4 by UDP packets. On the other hand, good CHOKe hits decline for Qtr1 and Qtr2 since there is a decreasing occupancy of Qtr1 and Qtr2 by UDP packets as the UDP rate increases.

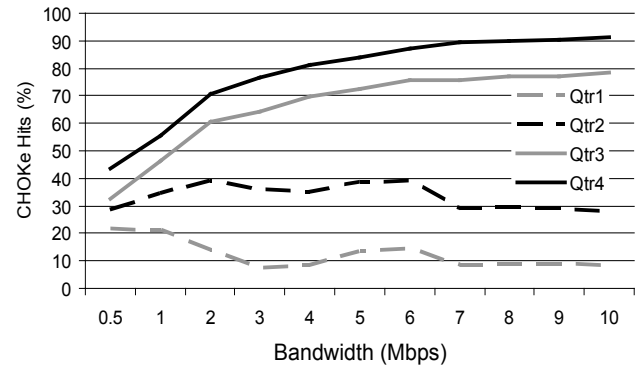


Figure 7. Percentages of CHOKe Hit locations

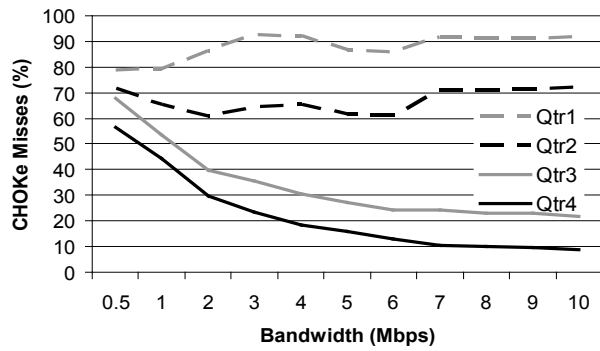


Figure 8. Percentages of CHOKe Miss locations

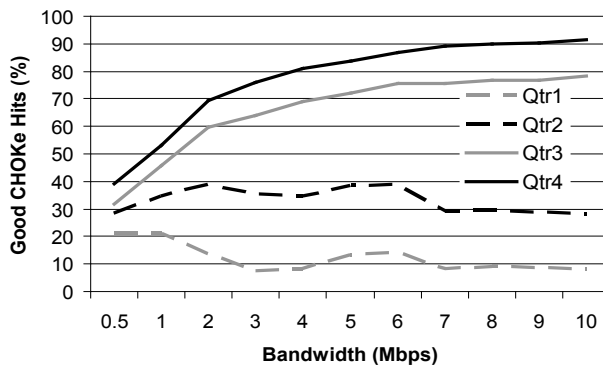


Figure 9. Percentages of Good CHOKe Hit locations

In Fig. 10, Qtr4 suffers more bad CHOKe hits when the UDP flow is at or around the link capacity. As a result, the TCP flows suffered losses leading to low throughput resulting in a reduction of the number of bad CHOKe hits at the other quarters. As the UDP rate increased, the number of bad CHOKe hits in Qtr4 has reduced due to the large increase in UDP packets.

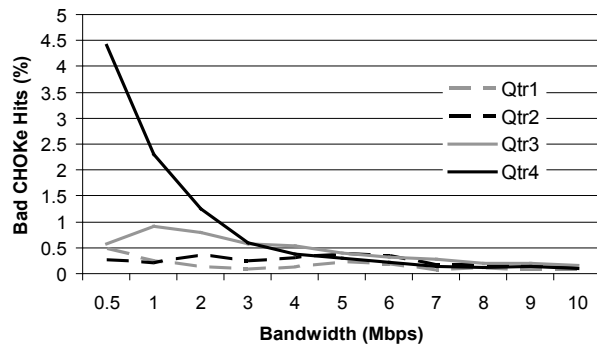


Figure 10. Percentages of Bad CHOKe Hit locations

In Fig. 7, the victim hits are very small for Qtr1 since this quarter has the lowest number of UDP packets among the 4 regions (see Figs. 2 and 3). Conversely, in Fig. 8, the victim misses are very high for Qtr1 since this quarter has the lowest UDP packets among the 4 regions. However, we need to analyze these victim misses again to check whether they are good or bad CHOKe misses (see Figs. 11 and 12 respectively).

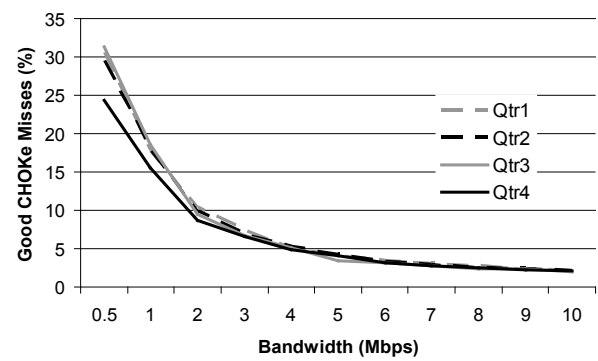


Figure 11. Percentages of Good CHOKe Miss locations

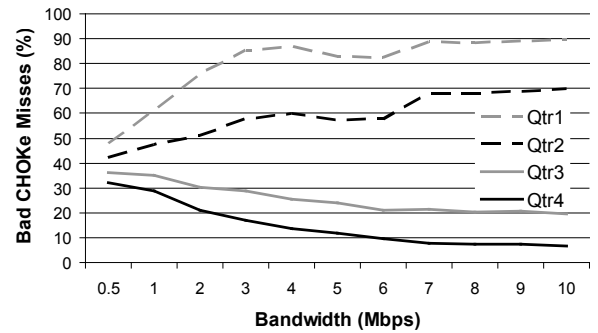


Figure 12. Percentage of Bad CHOKe Miss locations

In Fig. 12, the percentages of bad CHOKe misses were directly proportional to the UDP rate for Qtr1 and Qtr2 while inversely proportional to the UDP rate for Qtr3 and Qtr4. This was mainly because of the concentration of UDP packets in Qtr3 and Qtr4 (see Figs. 2 and 3).

Thus, the number of bad CHOKe hits increases i) when the number of flows using the buffer decreases, ii) when the rates of the TCP-friendly flows using the buffer increases, and iii) when the rates of the malicious flows using the buffer decreases. On the other hand, the number of good CHOKe hits increases i) when the number of flows using the buffer decreases, ii) when the rates of the TCP friendly flows using the buffer decreases and iii) the rates of the malicious flows using the buffer increases.

In Fig. 11, the percentages of good CHOKe misses are inversely proportional to the UDP rate. As the UDP rate is increased, the percentages of good CHOKe misses decrease. The percentages of good CHOKe misses for all the quarters are large at and around the link capacity since the number of packets from different flows using the buffer are greater at low UDP rates.

The number of bad CHOKe misses increases i) when the number of flows using the buffer increases, ii) when the rates of the TCP-friendly flows using the buffer increases, and iii) when the rates of the malicious flows using the buffer decreases. On the other hand, the number of good CHOKe misses increases i) when the number of flows using the buffer increases, ii) when the rates of the TCP friendly flows using the buffer decreases, and iii) when the rates of the malicious flows using the buffer increases.

3.2. Experiments on CHOKe-RED

CHOKe victims that escape being dropped (see Fig. 1 - those involved in CHOKe misses) are received by the RED queue which either accepts or drops (marks in the case of ECN) them. To analyze CHOKe with RED, we define the following terms for RED drops. A *CHOKe miss victim* is a packet which after experiencing a CHOKe miss enters the RED queue. *CHOKe miss RED hit* occurs when the flow id of the CHOKe miss victim matches the flow whose packets dominate the buffer at the time of selection of the CHOKe miss victim; otherwise a *CHOKe miss RED miss* has occurred. A CHOKe miss RED hit could be either good or bad. *Good CHOKe miss RED hit*: during a CHOKe miss RED hit, the CHOKe miss victim belongs to a malicious flow; it is bad otherwise. *CHOKe miss RED miss* again could be either good or bad; it is good if during a CHOKe miss RED miss, the CHOKe miss victim belongs to a non-malicious flow; it is bad otherwise.

Until now, our simulations made use of a single UDP flow. However, we have carried out many simulations with more UDP flows as well. We have found that when the number of UDP flows is increased, the amount of buffer space occupied by each one of them is dependent on that flow's rate. The greater the rate of a UDP flow, the greater the buffer space it will occupy at Qtr4. For example, if there are two UDP flows (at 2 and 10Mbps) the packets from the UDP flow of 10Mbps will occupy more buffer space in Qtr4 than those from the 2Mbps UDP flow. However, the rate of reduction in buffer space for the UDP flow in Qtr1 is greater for 10Mbps than for 2Mbps.

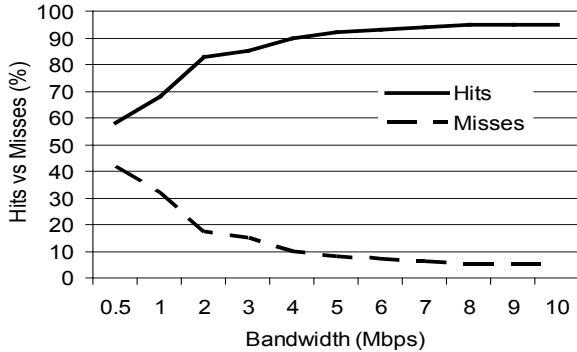


Figure 13. Percentages of CHOKe-RED Hits and Misses after CHOKe Misses

Fig. 13 shows that of the packets that are dropped by the RED buffer, a large majority experience RED hits. Almost all RED hits are good CHOKe miss RED hits, thus *RED was accurate in identifying malicious flows from CHOKe misses*. It seems that at lower rates, there is a large number of CHOKe misses. However, these are very few since at these rates, the queue drops or marks few packets. We also find that all CHOKe miss RED misses, although very few in numbers were all bad. Hence, RED also acted as another filter, after CHOKe, to weed out the packets belonging to these malicious flows. Although CHOKe-RED drops/marks

are more accurate than CHOKe hits and misses, it occurred less often since in RED, a packet is either accepted or dropped whenever avq is between Th_{min} and Th_{max} . Moreover, only one packet is dropped while in CHOKe, two packets are dropped for each hit. Hence, in the next section, we aim to improve on the accuracy of CHOKe hits and misses.

3.3. Half1 and Half2 CHOKes

From the experiments above, we see that if UDP flow rates are high, and if a victim packet is chosen from Qtr3 and Qtr4 (or Half-2), it will improve the chances of having i) a good CHOKe hit if the arriving packet belongs to a malicious flow, or ii) a good CHOKe miss if the arriving packet belongs to a TCP-friendly flow, while decreasing the chances of iii) bad CHOKe hits if the arriving packet belongs to a malicious flow, or iv) bad CHOKe misses if the arriving packet belonged to a TCP friendly flow. However, for UDP rates at or below the link capacity, the victim packet should be chosen from Qtr1 and Qtr2 (or Half-1).

Although at low rates, Half-2 has slightly greater percentages in good CHOKe hits than at Half-1, the latter gives better protection to non-malicious flows (see Fig. 10). Here we show by experiments that at lower rates, Good CHOKe hits are more effective in Half-1 than at Half-2. Hence, we propose two versions of CHOKe. The first, Half2 CHOKe, chooses victim packets from Half-2 whereas the second, Half1 CHOKe, chooses from Half-1. Initially, we keep these two variations apart to demonstrate their effectiveness. In RECHOKe, they are used as mentioned in Section II. From Figs. 2 and 3, UDP packets occupancy of Qtr1 is consistently greater than 30% during CHOKe hits when the UDP rate is at and less than the link capacity. This held true whenever the summation of the UDP rates was at and less than the link capacity. Hence, at $t=250ms$, if the occupancy percentage at the first quarter of the occupied queue by flows with entries in the table during CHOKe hits is greater than 30%, we use Half1 CHOKe, otherwise use Half2 CHOKe.

Using the previous network topology, simulations are carried out to compare Half1, Half2 and CHOKe. Figs. 14 and 15 illustrated two examples when the UDP rate is set to a rate greater than the link capacity. Here, Half2 CHOKe performs better than both Half1 CHOKe and the normal CHOKe algorithm. Even when the number of UDP flows is increased, as long as the sum of their bandwidth is greater than to the link capacity, the Half2 CHOKe version outperforms the other two. Many other experiments (that we have performed but are omitted here) confirm this result.

Fig. 16 illustrates an example when the sum of UDP rates is greater than the link capacity. Half2 CHOKe performs better than both Half1 CHOKe and the normal CHOKe algorithm. To illustrate an example when the sum of UDP rates is less than the link capacity we use three UDP flows with a rate of 0.3 Mbps each. Fig. 17 shows that Half1 CHOKe performs better than both Half2 CHOKe and the normal CHOKe algorithm in this case. When the number of

UDP flows increases, as long as the sum of their rates is less than or equal to the link capacity, the Half1 CHOKe outperforms the other two. The results show the effects of CHOKe hits are far greater at Half1 than at Half2.

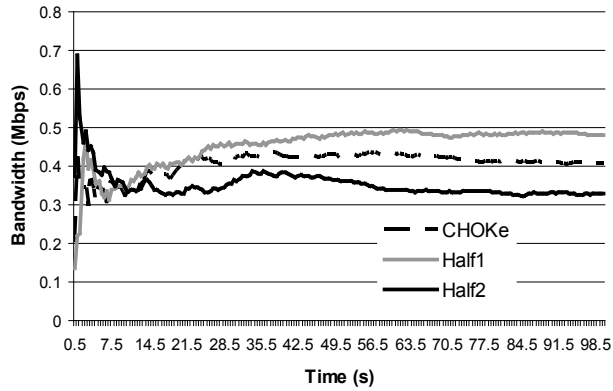


Figure 14. Link Utilization of UDP flow at 2 Mbps

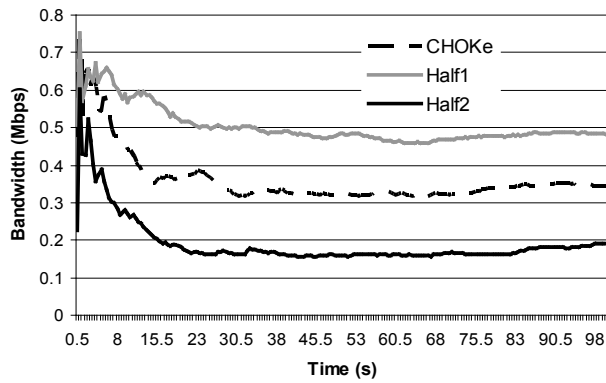


Figure 15. Link utilization of UDP flow at 10 Mbps

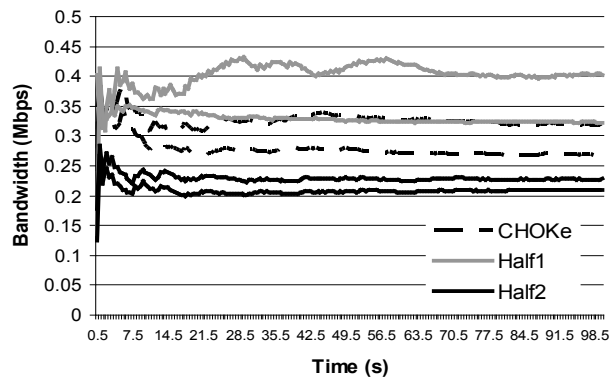


Figure 16. Link utilization of 2 UDP flows at 1 Mbps

3.4. Evaluation of xCHOKe and RECHOKe

In this section we analyze xCHOKe and RECHOKe buffers; to do this, we define the following terms. A *table hit* occurs when the flow id of the arriving packet matches a flow id entry already in the table. A table hit can be either good or bad. The table hit is good if the flow that the hit packet belongs to is malicious; it is a bad hit otherwise. *Table drops* are table hits that were selected for dropping by

xCHOKe (after drawing a random with probability p^*); for RECHOKe, these are table hits that were selected after evaluating p^* and c^* . *Good table drops* occur when the dropped packet belongs to a malicious flow; *bad table drops* occur otherwise. A *CHOKe miss victim* is a packet which after experiencing a CHOKe miss enters the RED queue. *CHOKe miss RED hit* occurs when the flow id of the CHOKe miss victim matches the flow whose packets dominate the buffer at the time of selection of the CHOKe miss victim; *CHOKe miss RED miss* occur otherwise. A CHOKe miss RED hit could be either a good or bad: it is good if the victim packet belongs to a malicious flow; it is bad otherwise. A *CHOKe miss RED miss* again could be either a good or a bad CHOKe miss RED miss. *Good CHOKe miss RED miss* is when the victim packet belongs to a non-malicious flow; it is bad otherwise.

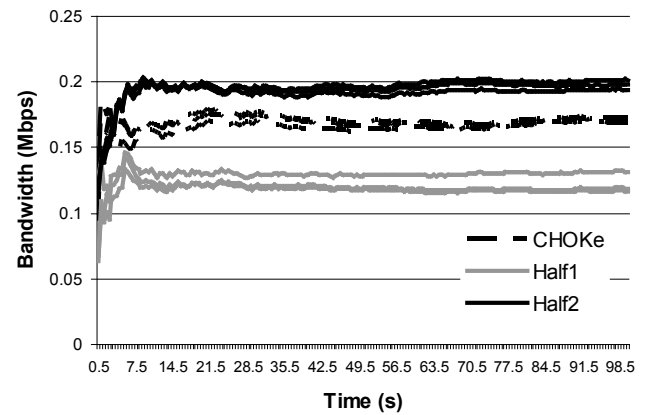


Figure 17. Link utilization of 3 UDP flows at 0.3 Mbps

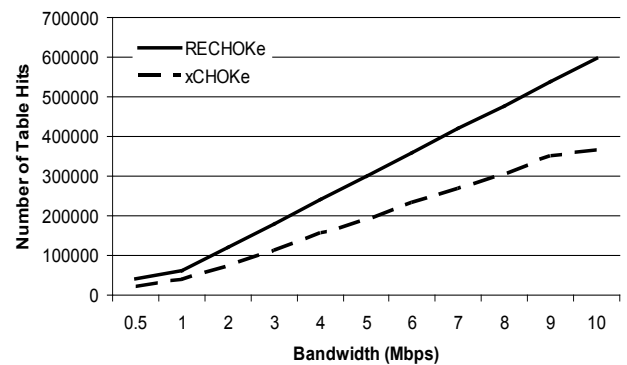


Figure 18. The number of table hits vs. UDP rate

Using the same network topology as before but this time using 3Mbps link, we analyze the xCHOKe and RECHOKe buffers in terms of buffer occupancy by the malicious flow. Fig. 18 shows the number of table hits for both the xCHOKe and RECHOKe algorithms. The number of hits with RECHOKe is 50-65% greater than with xCHOKe. The combination of CHOKe and CHOKe-RED histories allow RECHOKe to register a large number of hits and using these hits, RECHOKe can isolate malicious flows faster and more accurately. RECHOKe does punish malicious flows more rigorously while not penalizing TCP-friendly flows as it can

be observed by comparing Figs. 35 and 36. The majority of the hits shown in Fig. 18 are good table hits as witnessed by Figs. 19 and 20, pointing to RECHOKe identifying malicious flows better than xCHOKe. Fig. 19 also shows that the number of good table hits for RECHOKe is 50-65% greater than that of xCHOKe; Fig. 20 shows that the number of bad table hits for RECHOKe is 2-16 times less as well and constant no matter what the rate of the UDP flow is.

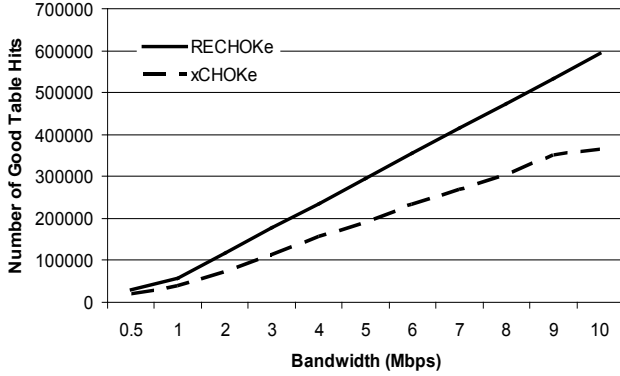


Figure 19. The number of good table hits vs. UDP rate

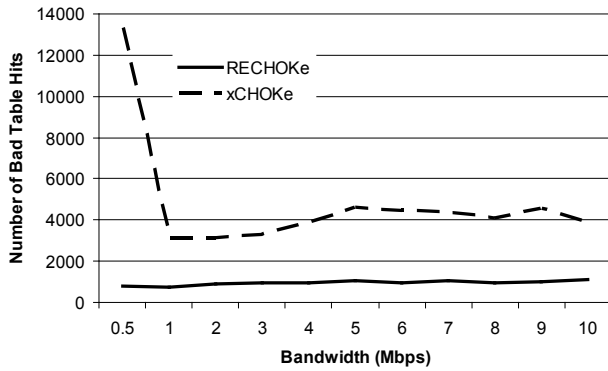


Figure 20. The number of bad table hits vs. UDP rate

Next, we analyze the number of table drops for both algorithms. Ideally drops should only consist of good table drops. Fig. 21 shows the number of table drops for both algorithms. Fig. 22 shows the number of good table drops; we can observe that RECHOKe outperforms xCHOKe significantly. In Fig. 23 we can also observe that the same is true in terms of bad table drops. Nearly all the table misses (not shown here) for both algorithms are good table misses. We can conclude that RECHOKe drops fewer packets from TCP-friendly flows than xCHOKe. Due to the large number of good table drops and the almost insignificant number of bad table drops, RECHOKe is faster and more accurate in the identification of flows. Unlike xCHOKe, RECHOKe updates the table after table hits. This is necessary as with the lifetime of flows, CHOKe hits become more unreliable since the number of packets belonging to the malicious flow in the buffer are reduced drastically leading to a greater number of bad CHOKe hits (see Figs. 25 and 26). The same phenomenon can be observed in xCHOKe and CHOKe,

which leads to unnecessary punishment of TCP-friendly flows.

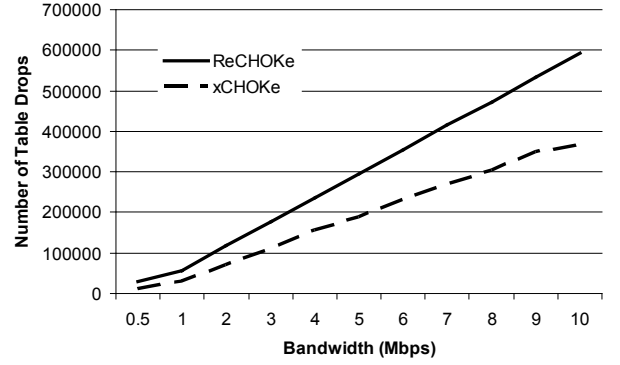


Figure 21. The number of table drops vs. UDP rate

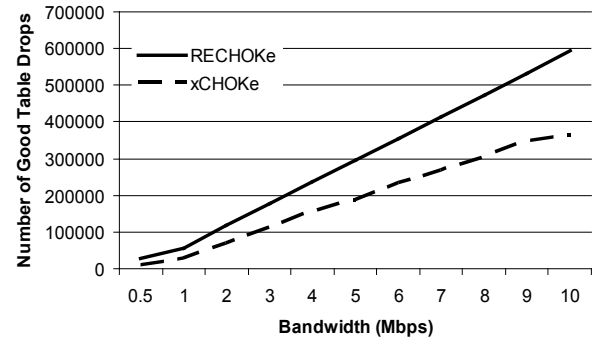


Figure 22. The number of good table drops vs. UDP rate

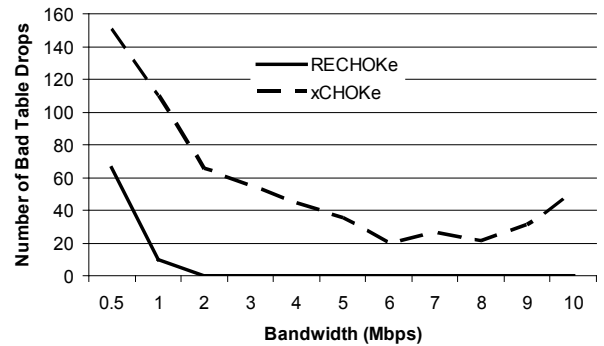


Figure 23. The number of bad table drops vs. UDP rate

Figs. 24 and 25 show that the number of CHOKe hits and misses for both algorithms, reflect this trend. CHOKe hits did not affect RECHOKe as the packets responsible for these hits were not dropped but passed on to the RED buffer. Although these hits were updated in the table, the c^* condition prevents unnecessary drops of TCP-friendly packets.

Upon analyzing the CHOKe misses in Figs. 28 and 29, we have found that RECHOKe significantly outperforms xCHOKe in bad CHOKe misses and good CHOKe misses despite having the larger number of CHOKe misses. This shows that RECHOKe allows packets from TCP-friendly flows to enter the RED buffer more frequently than xCHOKe.

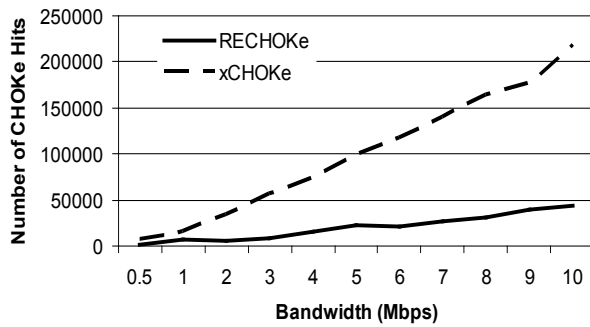


Figure 24. The number of CHOKe hits vs. UDP rate

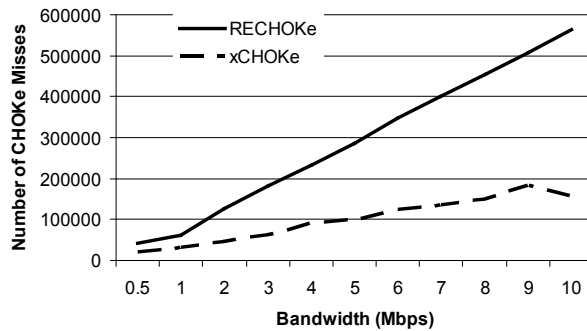


Figure 25. The number of CHOKe misses vs. UDP rate

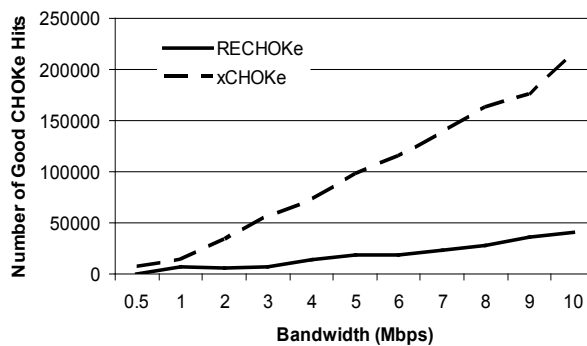


Figure 26. The number of good CHOKe hits vs. UDP rate

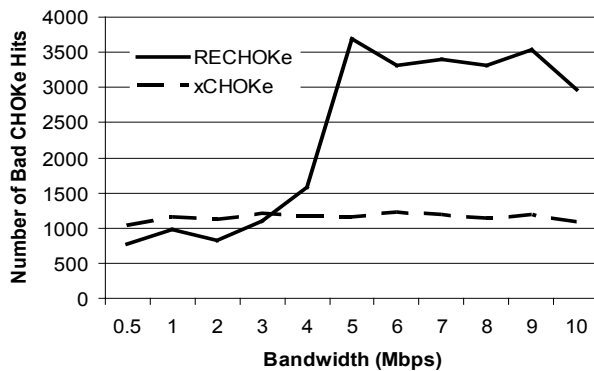


Figure 27. The number of bad CHOKe hits vs. UDP rate

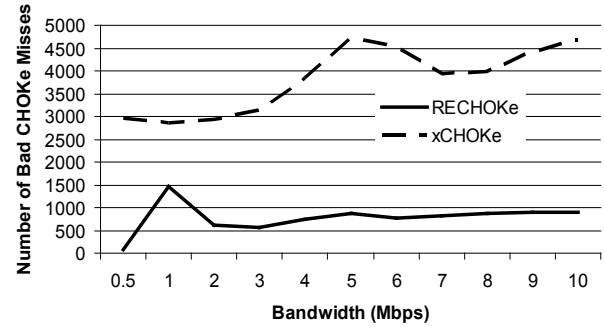


Figure 28. The number of bad CHOKe misses vs. UDP rate

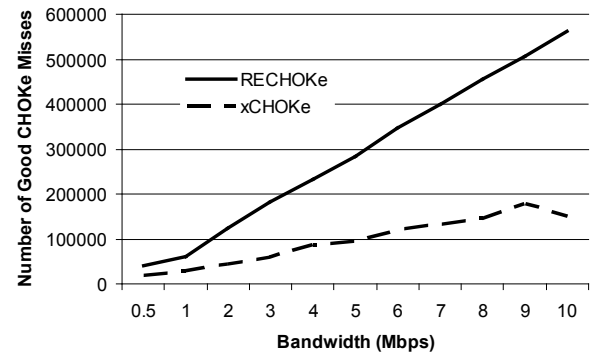


Figure 29. The number of good CHOKe misses vs. UDP rate

Next, we analyze RED hits and misses for both RECHOKe and xCHOKe. Fig. 30 shows that the number of dropped packets is greater when using RECHOKe. This is true because during CHOKe hits, packets are not dropped but admitted into the buffer; as a result, more packets need to be dropped by the RED buffer. Analyzing Figs 31 and 32 and we can see that most of the RED hits with RECHOKe are good compared to those of xCHOKe, i.e., most dropped packets belonged to malicious flows. This happens in spite of most of the admitted packets being from the ten TCP friendly flows. After the good RED hits, the malicious flows became no longer dominant. Hence, RED took out more UDP packets from (the already the few) that had been admitted into the buffer.

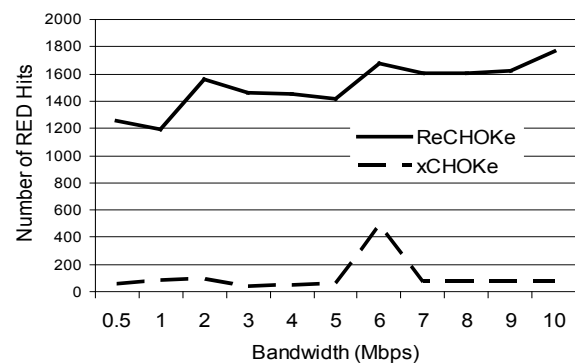


Figure 30. The number of RED hits vs. UDP rate

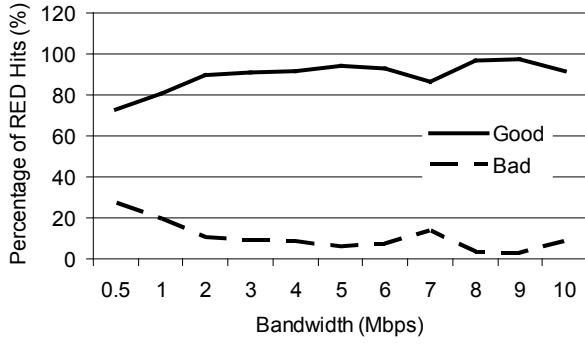


Figure 31. Percentages of good and bad RED hits vs. UDP rate (RECHOKe)

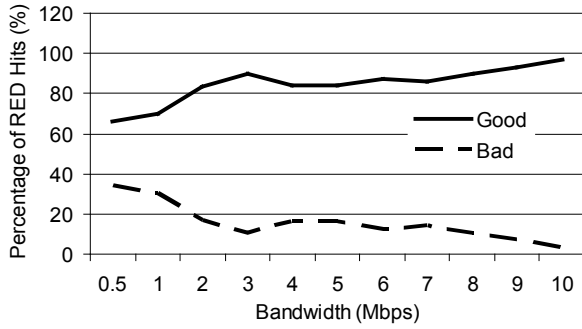


Figure 32. Percentages of good and bad RED hits vs. UDP rate (xCHOKe)

Next, we introduce two UDP flows: UDP-1 is set at 1Mbps and UDP-2 at 3Mbps with the 10 TCP flows varied at 2ms delay increments starting at 20ms (TCP-1). We use the same bar-bell topology with the same RED parameters as before.

Figs. 33, 34, 35 and 36 show the bandwidth allocation for the congested link for each flow for RED, CHOKe, xCHOKe and RECHOKe respectively. RED does not control the malicious flows (UDP-1 and UDP-2) against gaining control of almost the entire link bandwidth. The bandwidths of the TCP-friendly flows (TCP-1 to TCP-10) were almost zero. CHOKe and xCHOKe did control them more than RED did and during the process of controlling these malicious flows, (due to the fact that TCP flows suffering CHOKe hits [16]) they suffered bandwidth losses which in turn, were used by the very flows that these algorithms were trying to control. By minimizing bad CHOKe hits, RECHOKe controls malicious flows better than CHOKe and xCHOKe. RECHOKe also performed best in protecting TCP-friendly flows. Fig. 37 shows the UDP flows at RECHOKe, xCHOKe, CHOKe and RED buffers.

Figs. 38, 39, 40 and 41 compares each scheme at rates of 1.2 and 2Mbps. In these sets of experiments we again use the same bar-bell topology as before, however with two malicious flows. Each of these flows is a constant-bit-rate UDP flow set at a) 1.2Mbps and b) 2Mbps, competing with 10 TCP flows over a 3Mbps link. (UDP rates of 1.2Mbps result in flows with a capacity less than the link capacity, while using 2Mbps flows put us beyond this capacity.) From figs. 38, 39 and 40, as the rate of the malicious flows

increased from 1.2Mbps to 2Mbps, TCP flows loses more of its bandwidth to the malicious flows (UDP1 and UDP2) using the RED, CHOKe and xCHOKe buffers as these malicious flows gain greater control of the congested link bandwidth. This is evident from the increases in the UDP flows as their rates are increased from 1.2Mbps to 2Mbps. However, from fig. 41, RECHOKe increases its identification, control and punishment of the two malicious flows as evident from the decrease in their utilization of the link bandwidth as the rate of the two flows are varied from 1.2Mbps to 2Mbps. RECHOKe, unlike xCHOKe, CHOKe and RED, does control malicious flows with bandwidths smaller or slightly greater than the link capacity.

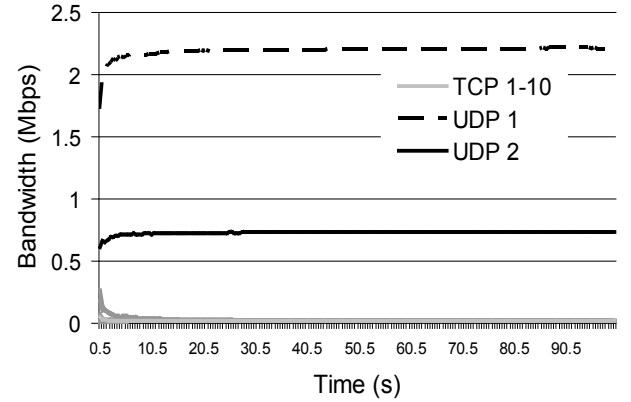


Figure 33. Link utilization at the RED buffer

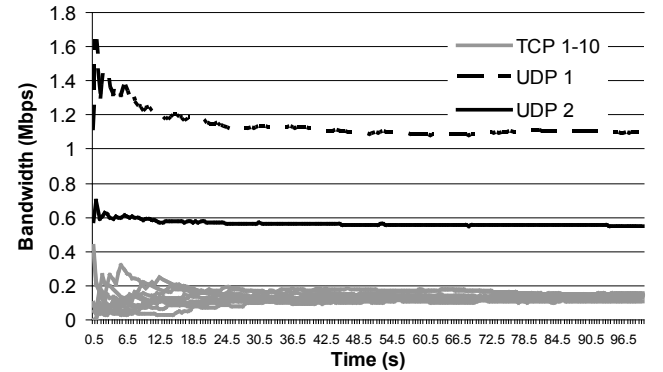


Figure 34. Link utilization at the CHOKe buffer

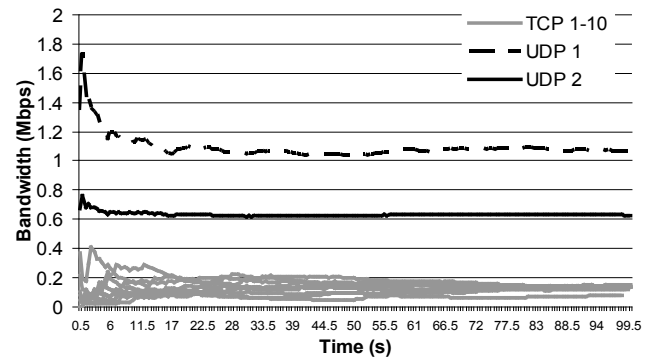


Figure 35. Link utilization at the xCHOKe buffer

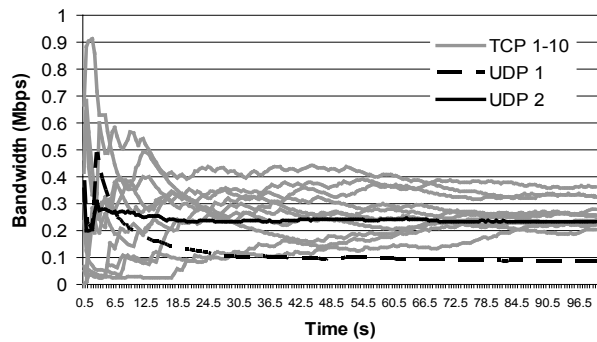


Figure 36. Link utilization at the RECHOKe buffer

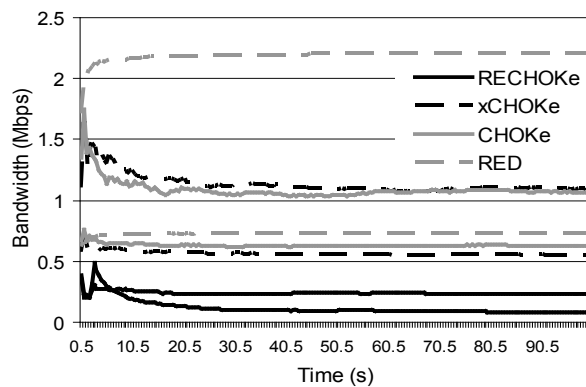


Figure 37. UDP flows at RECHOKe, xCHOKe, CHOKe and RED buffers

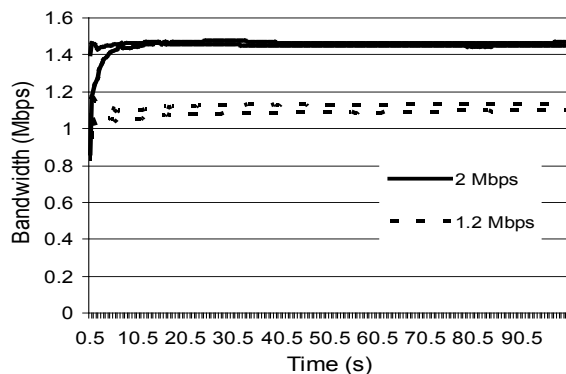


Figure 38. Link utilization of UDP flows at 1.2 and 2Mbps at the RED buffer

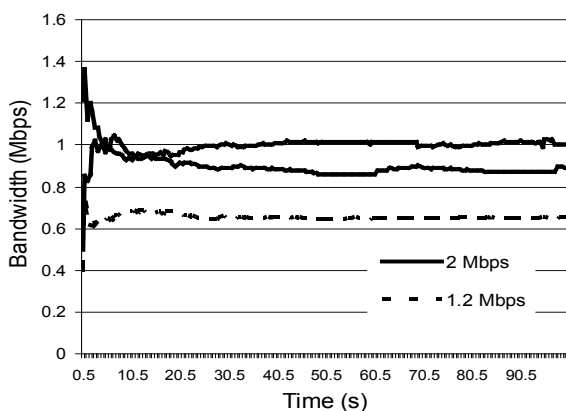


Figure 39. Link utilization of UDP flows at 1.2 and 2Mbps at the CHOKe buffer

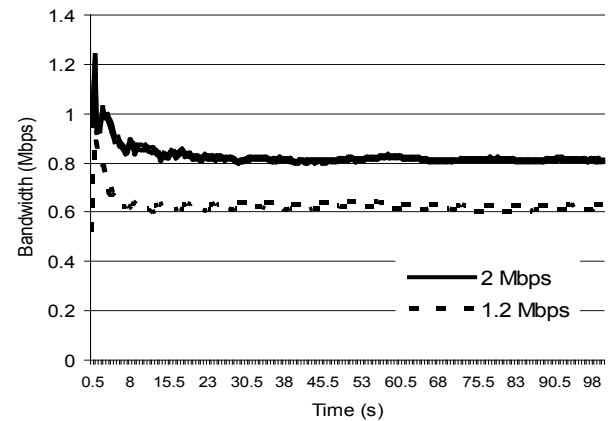


Figure 40. Link utilization of UDP flows at 1.2 and 2Mbps at the xCHOKe buffer

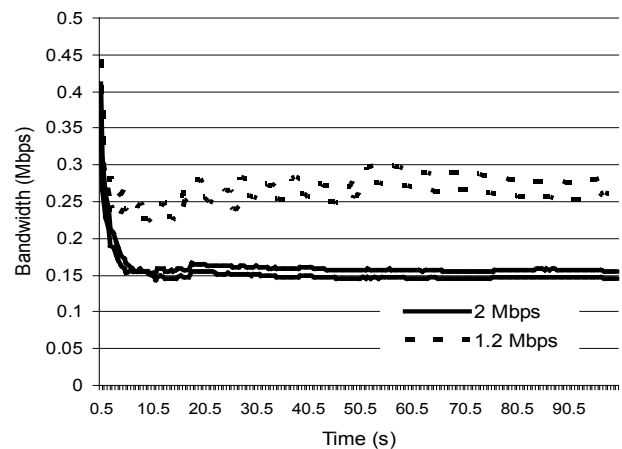


Figure 41. Link utilization of UDP flows at 1.2 and 2Mbps at the RECHOKe buffer

4. Conclusions and Future Work

In this paper, we have presented RECHOKe, a scheme for detecting, controlling and punishing malicious flows in IP networks. We showed that RECHOKe outperforms RED, CHOKe and xCHOKe by combining the techniques used by xCHOKe and RED-PD in identifying and punishing malicious flows while eliminating the complexity of dropping or marking randomly selected packets already queued (a method used by both CHOKe and xCHOKe) and the unreliability of CHOKe hits. We are currently working on implementing RECHOKe in Linux based packet routers and by using various flavors of TCP with multi-bottleneck topologies.

REFERENCES

- [1] V. Firoiu and M. Borden, A study of active queue management for congestion control, Proceedings of INFOCOM 2000, pp. 1435 - 1444, vol. 3, March, 2000.

- [2] R. Pan, B. Prabhakar, and K. Psounis, "CHOKe: a stateless active queue management scheme for approximating fair bandwidth allocation," *Proc. of IEEE INFOCOM*, Tel Aviv, Israel, March 2000, vol. 2, pp. 942–951.
- [3] A. Aditya, K. Richard, P. Christos, S. Srinivasan and S. Scott, Selfish Behavior and Stability of the Internet: A Game-Theoretic Analysis of TCP, ACM SIGCOMM 2002, Pittsburgh PA.
- [4] P. Chhabra, A. John, H. Saran, and R. Shorey. "Controlling Malicious Sources at Internet Gateways," *IEEE Int. Conf. Communication*, Anchorage, Alaska, May 2003, vol. 3, pp. 1636-1640.
- [5] R. Mahajan, S. Floyd and D. Wetherall, "Controlling high-bandwidth flows at the congested router," 9th Int. Conf. Network Protocols, Nov. 2001, pp. 192-201.
- [6] T. Wang, I. Matta, A. Bestavros, "Efficiently and Fairly Allocating Bandwidth at a Highly Congested Link," Technical Report, Boston University, December 2, 2003.
- [7] Y. Jiang, M. Hamdi, J. Liu, "Self Adjustable CHOKe: An Active Queue Management Algorithm For Congestion Control and Fair Bandwidth Allocation," 8th IEEE Symposium on Computers and Communications 2003.
- [8] I. Stoica. Stateless Core: A Scalable Approach for Quality of Service in the Internet. PhD thesis, Carnegie Mellon University, Department of Electrical and Computer Engineering, December 15, 2000.
- [9] D. Lin and R. Morris, "Dynamics of random early detection," Proceedings of SIGCOMM 1997, pp. 127–136, vol. 27, September, 1997.
- [10] P. McKenny, "Stochastic Fairness Queueing", Proceedings of INFOCOM '90, pp 733-740.
- [11] A. Manin, K. Ramakrishnan, "Gateway Congestion Control Survey", IETF RFC (Informational) 1254, August 1991.
- [12] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Networking*, August, 1993, vol. 1, no. 4, pp. 397-413.
- [13] V. V. Govindaswamy, G. Záruha and G. Balasekaran, "RECHOKe: A Scheme for Detection, Control and Punishment of Malicious Flows in IP Networks", Global Telecommunications Conference, 2007. GLOBECOM '07. IEEE, 26-30 Nov. 2007, Page(s):16 – 21.
- [14] K. Ramakrishnan, S. Floyd and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," *RFC 3168*, September, 2001.
- [15] NS2 simulator. Available: <http://www.isi.edu/nsnam/ns/>.
- [16] V. V. Govindaswamy, G. Záruha and G. Balasekaran, "RECHOKe and RCUBE," UTA Technical Report 2006-7, September, 2006.