

XCHOKe: Malicious Source Control for Congestion Avoidance at Internet Gateways

Parminder Chhabra¹, Shobhit Chuig, Anurag Goel, Ajita John²,
Abhishek Kumar³, Huzur Saran⁴, Rajeev Shorey⁵

¹parminder_chhabra@hp.com, ²ajita@dset.com, ³akumar@cc.gatech.edu, ⁴saran@iitd.ernet.in, ⁵srajeev@in.ibm.com

Abstract

This paper describes an algorithm to control flows from non-adaptive sources in the Internet in a simple and lightweight fashion. The paper presents experimental results from simulations that demonstrate that the algorithm outperforms current well-known algorithms for buffer management in providing better protection for TCP flows under varying degrees of attack from non-adaptive flows.

1 Introduction

RFC 2309 divides flows into three classes: (a) TCP-compliant flows, (b) unresponsive flows like UDP flows, and (c) flows that are responsive but are not TCP-compliant (aggressive in nature by not implementing congestion avoidance) [1]. We refer to sources for the latter two classes of flows as Malicious Sources (MS).

Stateless buffer management algorithms such as RED [2] address adaptive flows but have not been able to address MS. Stateless or partial-state algorithms such as CHOKe [3] and RED-PD [4] attempt to address MS and perform well with TCP-compliant flows in the presence of a few MS. However, they fall short of offering an easily implementable and deployable mechanism that works for a wide range of attack scenarios from MS. We propose XCHOKe as an extension to CHOKe and RED. The XCHOKe algorithm requires memory that is of the order of the number of MS. The per-packet computation is small with no floating-point operations. We show, through simulations, that XCHOKe outperforms CHOKe and RED in offering a significantly enhanced level of protection to adaptive flows.

2 XCHOKe

The motivation for XCHOKe is to design a RED-like algorithm that uses a lightweight mechanism to identify candidates for MS and to control the flows from these candidates. The approach should be scalable with increasing traffic flows in the Internet. XCHOKe is based on the intuition that a hit for a flow in the CHOKe algorithm [3] is an indication of a high packet arrival rate for that flow. XCHOKe qualifies a CHOKe hit to be the probable detection of an MS and stores its flow label in a lookup table. (The SRED mechanism may also be used for this [5].) The more the hits on a flow label, the higher the probability that the flow is an MS and the greater should be

the control on it. We use the number of hits to determine the drop probability for the flow.

XCHOKe proposes a pre-filter to the RED gateway that implements CHOKe, as shown in Figure 1. Incoming packets are pre-filtered against a lookup table that stores probable candidates for MS. They are then handed over to the RED gateway for the CHOKe algorithm. If the packet qualifies for a CHOKe hit, its flow label is added to the lookup table. CHOKe hits act as the identification mechanism and the pre-filter acts as the controlling mechanism for MS.

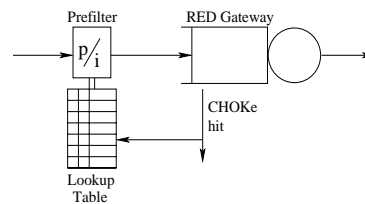


Figure 1: XCHOKe: pre-filter to CHOKe and RED, p : drop probability for flow i

The details of the XCHOKe algorithm are as follows. For any arriving packet, if the average queue size is less than the minimum threshold of the buffer, $\text{Min}(\text{th})$, we admit the packet. If the average queue size is greater than the maximum threshold, $\text{Max}(\text{th})$, we mark the packet for dropping. Otherwise, the lookup table is checked to see if the arriving packet's flow label is present in it. If it is (table hit), the arriving packet is marked for dropping with probability p^* whose computation is discussed below.

A packet is then selected at random from the FIFO buffer and its flow label is compared with that of the arriving packet. If the flow labels are the same (CHOKe hit), both packets are marked for dropping and the flow label is added to the lookup table with an initial value of 1 for an associated hit counter. 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^* with which an incoming packet with the same flow label as the entry will be dropped. In our simulations, we set $p^* = \min(1, p(\text{RED}) * 2^n)$. If the arriving packet is not marked for dropping through either a table hit or a CHOKe hit, it is dropped with a probability computed by RED. XCHOKe may penalize adaptive TCP sources. Since we can only statistically say that the flow

label in the lookup table is a malicious source, we associate a time-to-live (TTL) with each entry in the lookup table.

The computational complexity of XCHOKe for each packet is limited to a hash lookup in the table and the computation of p^* , which is a simple integer operation. The space complexity of XCHOKe is the size of the lookup table. The table size is expected to be of the order of the number of malicious flows arriving at the buffer. Simulations results not reported in this paper support this.

3 Simulation results for XCHOKe

All simulations were run using Network Simulator (ns) [6]. As in other work [3] [5], we use a dumbbell network topology with a 45Mbps bottleneck link with a 1ms delay. Each packet size is 576 bytes. For MS, we use UDP sources with an end-to-end delay of 21ms. The RED gateway uses the following buffer parameters: $Min(th)=200$ packets, $Max(th)=400$ packets. The maximum buffer size is 600 and the maximum RED drop probability is $1/30$. TTL is set to 100ms. Three source setups are used: Setup 1: 60 TCP sources (50 have $RTT=42ms$, 5 have $RTT=10ms$, 5 have $RTT=202ms$) and 2 UDP sources, Setup 2: 60 TCP sources (30 have $RTT=10ms$, 30 have $RTT=42ms$) and 2 UDP sources, Setup 3: 500 TCP sources (all have $RTT=42ms$) and 5-100 UDP sources.

The stabilized TCP throughputs in XCHOKe, RED, and CHOKe are compared in Tables 1 and 2 in Setups 1 and 2, respectively. The rate for each UDP source is varied as 1%-100% of the bottleneck link. It can be seen that XCHOKe outperforms RED and CHOKe as the UDP rate increases. TCP throughput in Setup 2 is better than that in Setup 1. This can be explained by the presence of the TCP flows with a high value of RTT (202ms) in Setup 1, which respond slowly in increasing window sizes as the effects of congestion are mitigated.

UDP Sending Rate →	1	5	10	20	40	75	100
TCP in XCHOKe	99	91	82	80	67	67	67
TCP in CHOKe	99	88	81	66	60	45	39
TCP in RED	99	88	80	40	22	0-1	-0

UDP Sending Rate →	1	5	10	20	40	75	100
TCP in XCHOKe	98	91	85	86	78	81	88
TCP in CHOKe	98	91	83	72	58	46	41
TCP in RED	98	88	80	61	24	0-1	-0

Figures 2a and 2b show the TCP throughput as a function of time in CHOKe and XCHOKe, respectively, in Setup 3. Each UDP source is sending packets at a rate of 10% of the bottleneck link. The curves for CHOKe have been obtained by choosing (k-1) packets at random from the buffer where k is the number of UDP sources (k-CHOKe) [3]. It can be seen that k-CHOKe fails to scale with increasing number of UDP flows whereas XCHOKe

achieves very high TCP throughput even with very large number of UDP flows.

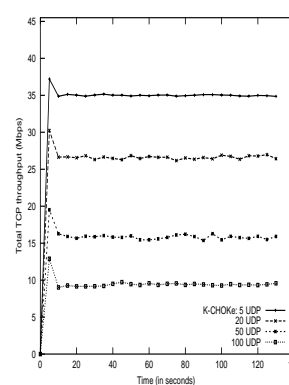


Figure 2a: k-CHOKe TCP throughput, 5-100 UDP sources

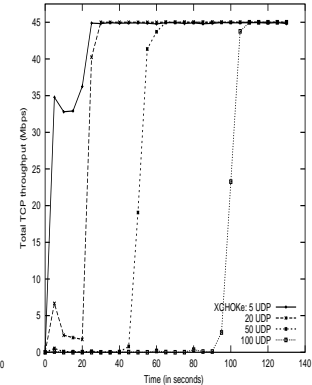


Figure 2b: XCHOKe TCP throughput, 5-100 UDP sources

4 Conclusions

In this paper we present an algorithm (XCHOKe) to control flows from malicious sources. The algorithm requires state that is of the order of the number of malicious sources. The per-packet computation is simple and no floating-point operations or bookkeeping operations are involved. The strength of XCHOKe lies in the enhanced level of protection provided to adaptive flows under attack from flows that are non-adaptive through a simple and lightweight extension to the widely deployed RED algorithm. The goal of our work is to neither prevent malicious sources from entering the buffer nor deny them their fair share of bandwidth. Rather, the attempt is to prevent a sustained attack of non-adaptive flows at Internet Gateways at the cost of denying the adaptive flows their fair share of bandwidth. This should serve as a motivation to be TCP-compliant as defined in [7].

5 References

- [1] B. Braden et al, Recommendations on Queue Management and Congestion Avoidance in the Internet. IEEE RFC(Informational) 2309, April 1998.
- [2] S. Floyd, and V. Jacobson, "Random Early Detection gateways for Congestion Avoidance", IEEE/ACM Transactions on Networking, V.1 N.4, August 1993, pp. 397-413.
- [3] R. Pan, B. Prabhakar, and K. Psounis, "CHOKe, A stateless active queue management scheme for approximating fair bandwidth allocation", IEEE Infocom 2000, Tel Aviv, Israel, March 26-30, 2000.
- [4] R. Mahajan, S. Floyd, D. Wetheral. Controlling High-Bandwidth Flows at Congested Router. International Conference on Network Protocols, 2001.
- [5] J. T. Ott, T. Lakshman and L. Wong, "SRED: Stabilized RED". IEEE Infocom 1999, New York, USA, March 1999.
- [6] S. McCanne and S.Floyd, "ns - Network Simulator", <http://www-nrg.ee.lbl.gov/ns/>.
- [7] Braden, R., Ed., "Requirements for Internet Hosts -- Communication Layers", STD 3, RFC 1122, October 1989.