

Protecting Bursty Applications Against Traffic Aggressiveness

Anat Bremler-Barr
Efi Arazi School of Computer Science
Interdisciplinary Center Herzliya
Email: bremler@idc.ac.il

Nir Halachmi
Efi Arazi School of Computer Science
Interdisciplinary Center Herzliya
Email: halachmi.nir@idc.ac.il

Hanoch Levy
School of Computer Science
Tel-Aviv University
Email: hanoch@cs.tau.ac.il

I. INTRODUCTION

In recent years the welfare of the Internet is threatened by malicious attacks of distributed denial of service (DDoS). DDoS attacker consumes the resources of the victim, a server or a network, causing a degradation in performance or even total failure of the victim. One of the common remedies suggested in the literature [1] and practically used [2] is to use traffic control mechanisms to bound the amount of traffic that can reach the victim from a source address ¹.

Weighted Fair Queueing is one of the common control mechanisms that have been thoroughly studied and widely used in network devices such as network routers and Web servers. WFQ aims at granting an application an "equal share" of the network resource (link, server); in a nutshell - a WFQ mechanisms provides that the instantaneous fraction of the resource is fairly (typically equally) divided among all applications competing for that resource at that instance.

Applying WFQ on the sources that send traffic to a victim which is under DDoS attack, will slow down the traffic rate of an attacker to be as low as that of a regular user. However, as shown in this work, in an environment of bursty applications this is not enough. Bursty applications are applications whose traffic demand varies heavily over time. Attackers can use this traffic characteristic to conduct a new type of attack by running "aggressive applications". These are applications that continuously pose traffic demands over long periods of time.

We propose a new mechanism called *Aggressiveness Protective Queueing* (APQ), which is based on the principles of the WFQ scheme and which uses *dynamic weights* for its operation. APQ uses a simple mechanism for tracking the resource usage by the flows and uses this accounting to properly and dynamically control the weights of WFQ. We provide analytic results and simulation results to demonstrate the effectiveness of APQ in controlling bursty traffic.

II. WFQ: UNFAIR SERVICE TO BURSTY FLOWS IN THE PRESENCE OF AGGRESSIVE USERS

Web traffic is characterized as a bursty application. This burstiness results from the nature of the user activity, in which

¹Using traffic control mechanisms is applicable to attacks that use non spoof packets, and where the number of attackers is moderate in comparison to the number of legitimate users. Other types of attacks are out of the scope of this paper.

the user requests a Web page (Active time) stops for a while to read the document (Idle time) and then requests another Web page and so on. *Aggressive users* are users that constantly pose traffic demands on the networks. Their motivation can be either to exploit the network in a selfish manner (such as using a WEB pre-fetcher) or even to maliciously harm the network (DDoS attackers). An *aggressive user* takes advantage of the idle time in order to send data (i.e. sending data constantly). WFQ cannot handle polite users in a fair way in the presence of aggressive users. The reason for it is that a polite user will receive its share of the bandwidth only when it is not idle (which is a small fraction of the time), while the aggressive user "requests" and receives its share of the bandwidth "all of the time". Thus, over-all, the aggressive user receives much more than its "fair share".

III. AGGRESSIVENESS PROTECTIVE QUEUEING (APQ)

Our solution is a new mechanism, *Aggressiveness Protective Queueing* (APQ), which is a variation on WFQ. APQ satisfies the following requirements:

- 1) It supports bursty applications.
- 2) It supports legitimate heavy traffic applications.
- 3) It negatively discriminates bursty users that use more than the average rate for a long period of time.
- 4) The limitation imposed by the system on the users is a function of the system load.

For every user APQ counts the amount of traffic that the user has generated in the near history and uses this amount to affect the WFQ weight given to it. The counters are calculated over a sliding window, where the window size covers the short history relevant to the dynamic weight function. Specifically, the window size Δ (measured in time units) is set to roughly the expected inter-burst time, namely about BS/APR , where BS is the average burst size (measured in packets) and APR is the average packet rate.

In order to describe APQ, we define the following parameters. Let $R(t)$ be the rate (measured in packets/sec) of the user offered traffic at time t . Let $D_{in}(x, y) = \int_x^y R(t)dt$ be the total traffic (measured in packets) offered by the user to the queueing system in the interval $[x, y]$. Let $SM(t) = D_{in}(\min(0, t - \Delta), t)$ be the amount of offered traffic during the last sliding window.

Aggressive Type	WFQ	APQ ($\alpha = 1$)	APQ ($\alpha = 2$)
Naive	f	$1 + \log f$	2
Sophisticated	f	$\sqrt{2 \cdot (f - 1) + 2}$	$\sqrt[3]{3 \cdot (f - 1) + 2}$

TABLE I
THE VULNERABILITY FACTOR FOR APQ AND WFQ.

Let w_o be the original fixed weight assigned to a user and let $w(t)$ be the dynamic weight assigned to the user at time t .

We study the following dynamic weight function of APQ:

$$w(t) = \begin{cases} w_o & SM(t) \leq BS \\ w_o \cdot \left(\frac{BS}{SM(t)}\right)^\alpha & SM(t) > BS, \end{cases}$$

where α is a penalty factor which is configurable in the system. We require that $\alpha \geq 1$. The greater α is, the smaller is the weight that is assigned to an aggressive user.

The function adaptively changes the weight assigned to the user in a way that for a user that offers to the queueing mechanism more than BS traffic in a period of duration Δ , the weight is reduced proportionally to the deviation from BS . Legitimate applications that require high constant packet rate will be configured with a suitable high BS .

IV. ANALYSIS OF APQ

We analyze the effect of APQ against two types of aggressive users: **1) Naive Aggressive** - A user that is not aware of APQ and its dynamic weight modification. Its strategy is to *continually* transmit at peak rate (as opposed to a polite user who goes idle between bursts), and **2) Sophisticated Aggressive** - A user that is aware of the APQ algorithm, namely the actual dynamic weight function. Its aim is to maximize the bandwidth APQ will grant it and its strategy is to offer traffic in a sophisticated way as to achieve this objective.

Our analysis provides an upper bound on the amount of traffic an aggressive application can gain. We also provide a lower bound proof and show that the upper bound is tight. Due to space limitation the details of the algorithm and the proof exist in the full version of the paper[3]. Let f be the burst factor $= \frac{T_{on} + T_{off}}{T_{on}}$, where T_{on} and T_{off} are the active time and idle time of a bursty user, respectively (note that in this case we have $\Delta = T_{on} + T_{off}$). This factor plays a major role in the analysis results. Let $D_{user}^{schedule}$ be the data transmitted by the queueing schedule (WFQ or APQ) for the particular type of user (*polite*, *naive-aggressive* or *sophisticated-aggressive*) during an interval of size Δ . To evaluate the quality of a scheduling algorithm we introduce the *Vulnerability Factor* of a scheduling policy with respect to the "aggressiveness" of the aggressive users. The factor indicates the number of denied-service polite-users per aggressive user. Specifically, we will evaluate it via $V_{user}^{schedule} = \frac{D_{user}^{schedule}}{D_{polite}^{schedule}}$ where *user* is either of *naive-aggressive* or *sophisticated-aggressive*, and *schedule* is either of WFQ or APQ. Note that the higher the vulnerability factor the less protection the scheduler provides

to polite users. In Table I we present a summary of the results, and show the vulnerability factor under the various scheduling policies and for the various users. To evaluate these results for Web applications, one can roughly estimate $T_{on} = 2$ and $\Delta = 30$ and hence $f = 15$. If $\alpha = 1$ then the factor is $\log f + 1$ (less than 5) for a polite user and less than 5.5 for a sophisticated user.

V. STOCHASTIC ENVIRONMENT: SIMULATION RESULTS

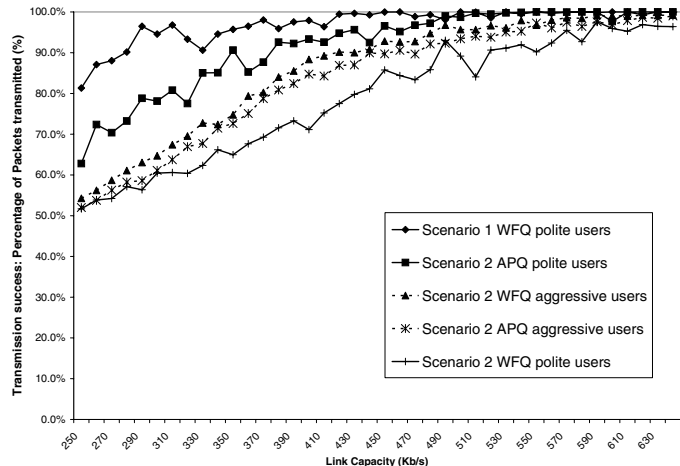


Fig. 1. Experiment: Percentage of transmitted packets as a function of Link Capacity

We conduct the simulation using the NS2 network simulator [4] and use the WFQ implementation contributed by Paolo Losi [5] as a base line for developing the algorithm (APQ). In the experiment we examine the percentage of packets transmitted per user as a function of the link capacity (varying between 250 Kb/sec and 650 Kb/sec). Under this experiment we consider two scenarios. In *Scenario 1* there are 12 users in the system, all polite. In *Scenario 2* there are 12 users, two of which are aggressive and the rest are polite. As Figure 1 shows, the transmission success of polite users using WFQ under the presence of aggressive users (Scenario 2) was significantly lower (up to 65% of what they get under Scenario 1), and the aggressive users get up to 20% higher transmission success than the polite users. In contrast, using APQ in the same scenario the transmission success of the polite users approaches that of Scenario 1, and that of the aggressive users is significantly lower than that of the polite users.

REFERENCES

- [1] R. Thomas, B. Mark, T. Johnson, and J. Croall, "Netbouncer: Client legitimacy-based high-performance ddosfiltering," *discex*, p. 14, 2003.
- [2] "Fair queuing combats ddos," *NANOG mailing list 2005*, <http://www.cctec.com/maillists/nanog/historical/0002/msg00298.html>.
- [3] "Aggressiveness protective fair queueing for bursty applications," <http://www.cs.tau.ac.il/~hanoch/Papers/BremnerHalachmiLevy2006.pdf>.
- [4] "Ns2 network simulator," <http://www.isi.edu/nsnam/ns/>.
- [5] "Wfq implementation code donated by paolo losi," <http://www.cclinf.polito.it/s77950/>.