Botz-4-Sale: Surviving Organized DDoS Attacks That Mimic Flash Crowds

Srikanth Kandula Dina Katabi
MIT

Matthias Jacob
Princeton

Arthur Berger
MIT/Akamai

ABSTRACT

Recent denial of service attacks are mounted by professionals using Botnets of tens of thousands of compromised machines. To circumvent detection, attackers are increasingly moving away from pure bandwidth floods to attacks that mimic the Web browsing behavior of a large number of clients, and target expensive higher-layer resources such as CPU, database and disk bandwidth. The resulting attacks are hard to defend against using standard techniques as the malicious requests differ from the legitimate ones in intent but not in content.

We present the design and implementation of Kill-Bots, a kernel extension to protect Web servers against DDoS attacks that masquerade as flash crowds. Kill-Bots provides authentication using graphical tests but is different from other systems that use graphical tests. First, instead of authenticating clients based on whether they solve the graphical test, Kill-Bots uses the test to quickly identify the IP addresses of the attack machines. This allows it to block the malicious requests while allowing access to legitimate users who are unable or unwilling to solve graphical tests. Second, Kill-Bots sends a test and checks the client's answer without allowing unauthenticated clients access to sockets, TCBs, worker processes, etc. This protects the authentication mechanism from being DDoSed. Third, Kill-Bots combines authentication with admission control. As a result, it improves performance, regardless of whether the server overload is caused by DDoS or a true Flash Crowd. This makes Kill-Bots the first system to address both DDoS and Flash Crowds within a single framework. We have implemented Kill-Bots in the Linux kernel and evaluated it in the wide-area Internet using PlanetLab.

1. INTRODUCTION

Denial of service attacks are increasingly mounted by professionals in exchange for money or material benefits [44]. Botnets of thousands of compromised machines are rented by the hour on IRC and used to DDoS online businesses to extort money or obtain commercial advantages [53, 35, 23]. The DDoS business is thriving; increasingly aggressive worms infect about 30,000 new machines per day, which are used for DDoS and other attacks [51, 23]. Recently, a Massachusetts businessman paid members of the computer underground to launch organized, crippling DDoS attacks against three of his competitors [44]. The attackers used Botnets of more than ten thousand machines. When the simple SYN flood failed, they launched an HTTP flood; pulling large image files from the victim server in overwhelming numbers. At its peak the onslaught allegedly kept the victim company offline for two weeks. In another instance, attackers ran a massive numbers of queries through the victim's search engine, bringing the server down [44].

To circumvent detection, attackers are increasingly moving away from pure bandwidth floods to stealthy DDoS attacks that masquerade as flash crowds. They profile the victim server and mimic legitimate Web browsing behavior of a large number of clients. These attacks target higher layer server resources like sockets, disk bandwidth, database bandwidth and worker processes [44, 18, 32]. We call such DDoS attacks CyberSlam, after the first FBI case involving DDoS-for-hire [44]. The MyDoom worm [18], many DDoS extortion attacks [32], and recent DDoS-for-hire attacks are all instances of CyberSlam [44, 32, 17].

Countering CyberSlam is a challenge because the requests originating from the zombies are indistinguishable from the requests generated by legitimate users. The malicious requests differ from the legitimate ones in intent but not in content. The malicious requests arrive from a large number of geographically distributed machines; thus they cannot be filtered on the IP prefix. Also, many sites do not use passwords or login information, and even when they do, passwords could be easily stolen off the hard disk of a compromised machine. Further, checking the site specific password requires establishing a connection and allowing unauthenticated clients to access socket buffers, TCBs, and worker processes, making it easy to mount an attack on the authentication mechanism itself. Defending against CyberSlam using computational puzzles, which require the client to perform heavy computation before accessing the site, is not effective because computing power is usually abundant in a Botnet. Furthermore, in contrast to bandwidth attacks [48, 36], it is difficult to detect big resource consumers when the attack targets higher-layer bottlenecks such as CPU, database, and disk because commodity operating systems do not support fine-grained resource monitoring [13, 12, 57]. Further, detecting big resource consumers becomes particularly hard if the attacker resorts to mutating attacks which cycle between different bottlenecks [34, 33].

This paper proposes Kill-Bots, a kernel extension that protects Web servers against CyberSlam attacks. It is targeted towards small and medium online businesses, as well as non-commercial Web sites. Kill-Bots combines two functionalities: authentication and admission control.

(a) Authentication: The authentication mechanism is activated during periods of severe overload. It has two stages.

- In $Stage_1$, Kill-Bots requires each new session to solve a reverse Turing test to obtain access to the server. Humans can easily solve a reverse Turing test, but zombies cannot. We focus on graphical tests, though Kill-Bots works with other types of Turing tests. Legitimate clients either solve the graphical test, or try to reload a few times and, if they still cannot access the server, decide to come back later. In contrast, the zombies which want to congest the server continue sending new requests without solving the test. Kill-Bots uses this difference in behavior between legitimate users and zombies to identify the IP addresses that belong to zombies and drop their requests. Kill-Bots uses SYN cookies to prevent spoofing of IP addresses and a Bloom filter to count how often an IP address failed to solve a puzzle. It discards requests from a client if the number of its unsolved tests exceeds a given threshold (e.g., 32 unsolved puz-

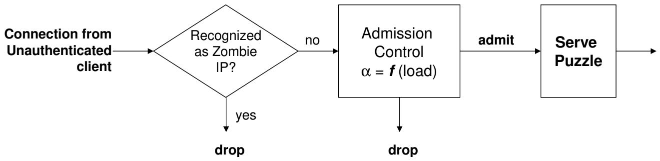


Figure 1: Kill-Bots Overview. Note that graphical puzzles are only served during $Stage_1$.

zles).

- Kill-Bots switches to $Stage_2$ after the set of detected zombie IP addresses stabilizes (i.e., filter does not learn any new bad IP addresses). In this stage, puzzles are no longer served. Instead, Kill-Bots relies solely on the Bloom filter to drop requests from malicious clients. This allows legitimate users who cannot, or don't want to, solve graphical puzzles access to the server despite the ongoing attack.

(b) Admission Control: Kill-Bots combines authentication with admission control. A Web site that performs authentication to protect itself from DDoS encounters a general problem: It has a certain pool of resources, which it needs to divide between authenticating new arrivals and servicing clients that are already authenticated. There is an optimal balance between these two tasks. Spending a large amount of resources on authentication might leave the server unable to fully serve the authenticated clients, and hence, wastes server's resources on authenticating new clients that it cannot serve. On the other hand, spending too many resources on serving the clients reduces the rate at which new clients are authenticated and admitted into the server, which might result in idle periods with no clients in service.

Kill-Bots computes the admission probability α that maximizes the server's goodput (i.e., the optimal probability with which new clients should be authenticated). It also provides a controller that allows the server to converge to the desired admission probability using simple measurements of server's utilization. Admission control is a standard mechanism for combating server overload [20, 24, 56, 19, 57, 54], but Kill-Bots examines admission control in the context of malicious clients and connects it with client authentication.

Fig. 1 shows a block diagram of Kill-Bots. When a new connection arrives, it is first checked against the list of detected zombie addresses. If the IP address is not recognized as a zombie, Kill-Bots admits the connection with probability $\alpha = f(load)$. In $Stage_1$, admitted connections are served a graphical puzzle. If the client solves the puzzle, it is given a Kill-Bots HTTP cookie which allows its future connections, for a short period, to access the server without being subject to admission control and without having to solve new puzzles. In $Stage_2$, Kill-Bots no longer issues puzzles; admitted connections are immediately given a Kill-Bots HTTP cookie.

Kill-Bots has a few important characteristics.

- **Kill-Bots addresses graphical tests? bias against users who are unable or unwilling to solve them.** Prior work that employs graphical tests ignores the resulting user inconvenience as well as their bias against blind and inexperienced humans [41, 6]. Kill-Bots is the first system to employ graphical tests to identify humans from automated zombies, while limiting their negative impact on legitimate users who cannot or do not want to solve them.
- **Kill-Bots sends a puzzle without giving access to TCBs or socket buffers.** Typically sending the client a puzzle requires establishing a connection and allowing unauthenticated clients to access socket buffers, TCB's, and worker processes, making it easy to DoS the authentication mechanism itself. Ideally, a DDoS protection mechanism should minimize the resources consumed by unauthenticated clients. Kill-Bots introduces a modification to the server's TCP stack that can send a 1-2 packet puzzle at the end of the TCP handshake without maintaining any connection state, and while preserving the semantics of TCP congestion control.
- Kill-Bots improves performance, regardless of whether server overload is caused by DDoS attacks or true Flash Crowds, making it the **first system to address both DDoS and Flash Crowds within a single framework**. This is an important side effect of using admission control, which allows the server to admit new connections only if it can serve them.
- The paper presents **a general model of resource consumption** in a server that implements an authentication procedure in the interrupt handler, a standard location for packet filters and kernel firewalls [38, 47]. We use the model to devise an admission control scheme that maximizes the server's goodput by finding the optimal probability with which new clients should be authenticated. Our model is fairly general and is independent of how the authentication is performed; the server may be authenticating the clients by checking their login information, verifying their passwords, or asking them to solve a puzzle.
- In addition, Kill-Bots requires no modifications to client software, is transparent to Web caches, and is robust to attacks in which the human attacker solves a few graphical tests and distributes the answer to a large number of zombies.

We implement Kill-Bots in the Linux kernel and evaluate it in the wide-area network using PlanetLab. Additionally, we conduct an experiment on human users to quantify user willingness to solve graphical puzzles to access a Web server. On a standard 2GHz Pentium IV Linux machine with 1GB of memory and 512kB L2 cache running a mathopd [14] server on top of a modified Linux 2.4.10, Kill-Bots serves graphical tests in 31μs, blocks malicious clients using the Bloom filter in less than 1μs, and can survive DDoS attacks of up to 6000 HTTP requests per second without affecting response times.[1] Compared to a server that does not use Kill-Bots, our system survives attack rates 2 orders of magnitude higher, while maintaining response times around their values with no attack. Furthermore, in our Flash Crowds experiments, Kill-Bots delivers almost twice as much goodput as the baseline server and improves response times by 2 orders of magnitude.

2. THE DESIGN OF KILL-BOTS

Kill-Bots is a kernel extension to Web servers. It combines authentication with admission control.

2.1 Threat Model

Kill-Bots aims to improve server performance under CyberSlam attacks, which mimic legitimate Web browsing behavior and consume higher layer server resources such as CPU, memory, database and disk bandwidth. Prior work proposes various filters for bandwidth floods [36, 11, 22, 28]; Kill-Bots does not address these attacks. Attacks on the server's DNS entry or on the routing entries to prevent

[1] These results are for the traditional event driven system that relies on interrupts. The per-packet cost of taking an interrupt is fairly large $\approx 10\mu$s [30]. We expect an even better performance with polling drivers [39].

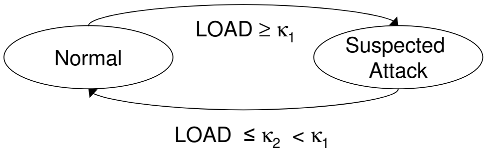


Figure 2: A Kill-Bots server transitions between NORMAL and SUSPECTED_ATTACK modes based on server load.

clients from accessing the server are also outside the scope of this paper.

We assume the attacker may have full control over an arbitrary number of machines that can be widely distributed across the Internet. The attacker may also have arbitrarily large CPU power and memory resources. However, we assume that the server's link bandwidth and the device driver are NOT congested by the volume of attack traffic. An attacker cannot sniff packets on a major link which might carry traffic for a large number of legitimate users. Further, the attacker does not have access to the server's local network or physical access to the server itself. Finally, we assume the zombies cannot solve the graphical test and the attacker is not able to concentrate a large number of humans to continuously solve reverse Turing tests.

2.2 Authentication

During periods of severe overload, Kill-Bots authenticate clients before granting them service. The authentication has two stages. First, Kill-Bots authenticates clients using graphical tests, as shown in Fig. 4. The objective of this stage is to improve the service experienced by human users who solve the graphical tests, and to learn the IP addresses of the automated attack machines. The first stage lasts until Kill-Bots concludes it has learned the IP addresses of all the zombies participating in the attack. In the second stage, Kill-Bots no longer issues graphical tests; instead clients are authenticated by checking that their IP addresses do not match any of the zombie IP addresses that Kill-Bots has learned in the first stage. Below, we explain the authentication mechanism in detail.

2.2.1 *Activating the Authentication Mechanism*

A Kill-Bots Web-server is in either of two modes, `NORMAL` or `SUSPECTED_ATTACK`, as shown in Fig. 2. When the Web server perceives resource depletion beyond an acceptable limit, κ_1, it shifts to the `SUSPECTED_ATTACK` mode. In this mode, every new connection has to solve a graphical test before allocation of any state on the server takes place. When the user correctly solves the test, the server grants the client access to the server for the duration of an HTTP session. Connections that begin before the server switched to the `SUSPECTED_ATTACK` mode continue to be served normally until they terminate or timeout. However, the server will time out these connections if they last beyond a certain interval (our implementation uses 5 minutes). The server continues to operate in the `SUSPECTED_ATTACK` mode until the load goes down to its normal range and crosses a particular threshold $\kappa_2 < \kappa_1$. The load is estimated using an exponential weighted average. The values of κ_1 and κ_2 will vary depending on the normal server load. For example, if the server is provisioned to work with 40% utilization, then one may choose $\kappa_1 = 70\%$ and $\kappa_2 = 50\%$.

Several points are worth noting. First, the server behavior is unchanged in the `NORMAL` mode, and thus the system has no overhead in the common case of no attack. Second, the cost for switching back and forth between the two modes is minimal. The only potential switching cost is the need to timeout very long connections that started in the `NORMAL` mode. Long connections that started in a prior `SUSPECTED_ATTACK` mode need not be timed out because their users have already been authenticated.

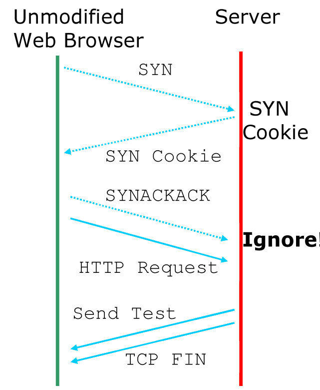


Figure 3: A Kill-Bots server sends a test to a new client without allocating a socket or any other connection resources.

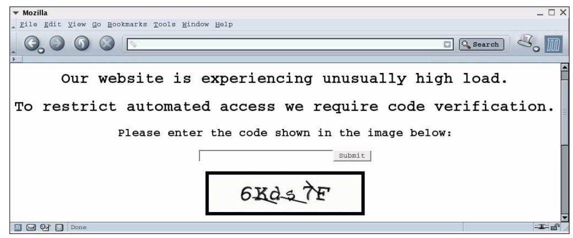


Figure 4: Screenshot of a graphical puzzle.

```
<html>
  <form method = "GET" action="/validate">
    <img src="PUZZLE.gif">
    <input type="password" name="ANSWER">
    <input type="hidden" name="PUZZLE_ID" value="[]">
  </form>
</html>
```

Figure 5: HTML source for the puzzle

2.2.2 *Stage 1: CAPTCHA-Based Authentication*

After switching to the `SUSPECTED_ATTACK` mode, the server enters $Stage_1$, in which it authenticates clients using graphical tests, i.e., CAPTCHAs.

(a) Modifications to Server's TCP Stack: Upon the arrival of a new HTTP request, Kill-Bots sends a graphical test and validates the corresponding answer sent by the client without allocating any TCBs, socket buffers, or worker processes at the server. We achieve this by a minor modification to the server TCP stack. As shown in Figure 3, a Kill-Bots server responds to a SYN packet with a SYN cookie. The client receives the SYN cookie, increases its congestion window to two packets, transmits a SYNACKACK[2] and the first data packet that usually contains the HTTP request. The server's kernel does not create a new socket upon completion of the TCP handshake. Instead the SYNACKACK packet is discarded because the first data packet from

[2] Just a plain ACK that finishes the handshake.

Puzzle ID (P)	Random (R)	Creation Time (C)	Hash (P, R, C, secret)
32	96	32	32

Figure 6: Kill-Bots Token

the client repeats the same acknowledgment sequence number as the SYNACKACK. When the server receives the client's data packet, it first checks whether is a puzzle answer.[3] If the packet does not contain an answer, the server replies with a new graphical test, embedded in an HTML form (Fig. 5), and immediately closes the connection by sending a FIN packet. Our implementation uses CAPTCHA images that fit in 1-2 packets. The server kernel does not wait for the FIN ack. On the other hand, if the packet is an answer, the kernel checks the cryptographic validity of the ANSWER (see (c) below). If the check succeeds, a socket is established and the request is delivered to the application. Note that the above scheme preserves TCP congestion control semantics, does not require any modifications to client software, and prevents attacks that hog TCBs and sockets by establishing connections that exchange no data. When a human answers the graphical test, the HTML form (Fig. 5) generates an HTTP request `GET /validate?answer=` $\texttt{ANSWER}_i$ that reports the answer to the server.

(b) One Test Per Session: It would be inconvenient if legitimate users had to solve a puzzle for every new HTTP request or every new TCP connection. The Kill-Bots server gives an HTTP cookie to a user when he solves the test correctly. This cookie allows the user to re-enter the system for a specific period of time. (In our implementation, this period is set to half an hour). If a new HTTP request is accompanied by a cryptographically valid HTTP cookie, the Kill-Bots server creates a socket and hands the request to the application without serving a new graphical test.

(c) Cryptographic Support: When the Kill-Bots server issues a puzzle, it generates a Kill-Bots Token as shown in Fig. 6. The token consists of a 32-bit puzzle ID P, a 96-bit random number R, the 32-bit creation time C of the token, and a 32-bit collision-resistant hash of P, R, and C along with the server secret. The token is embedded in the same HTML form as the puzzle (Fig. 6) and sent to the client.

When a user solves the puzzle, the browser reports the answer to the server along with the Kill-Bots token. The server first verifies the token by recomputing the hash. Second, the server checks the Kill-Bots token to ensure the token was created no longer than 4min ago. Next, the server checks if the answer to the puzzle is correct. If these checks are successful, the server creates a Kill-Bots cookie and gives it to the user. The Kill-Bots cookie is created from the token by updating the token creation time and recording the token in the table of valid Kill-Bots cookies. Subsequently, when a user issues a new TCP connection with an existing Kill-Bots HTTP cookie, the server validates the cookie by recomputing the hash and ensuring that the cookie has not expired, i.e. no more than 30min have passed since cookie creation. The Kill-Bots server also keeps track of the number of simultaneous HTTP requests that belong to each cookie.

(d) Protecting Against Copy Attacks: What if the attacker solves a single graphical test and distributes the HTTP cookie to a large number of bots? Kill-Bots introduces a notion of per-cookie fairness to address this issue. Each correctly answered graphical test allows the client to execute a maximum of 8 simultaneous HTTP requests. Distributing the cookie to multiple zombies makes them compete among themselves for these 8 connections. Most legitimate web browsers open no more than 8 simultaneous connections to a single server [25].

2.2.3 *Stage 2: Authenticating Users Who Do Not Answer CAPTCHAs*

An authentication mechanism that relies solely on CAPTCHAs has two disadvantages. First, the attacker can force the server to continuously send graphical tests, imposing an unnecessary overhead on the server. Second, and more important, humans who are unable or unwilling to solve CAPTCHAs may be denied service.

To deal with this issue, Kill-Bots distinguishes legitimate users from zombies by their reaction to the graphical test rather than their ability to solve it. Once the zombies are identified, they are blocked from using the server. When presented with a graphical test, legitimate users may react as follows: (1) they solve the test, immediately or after a few reloads; (2) they do not solve the test and give up on accessing the server for some period, which might happen immediately after receiving the test or after a few attempts to reload. The zombies have two options; (1) either imitate human users who cannot solve the test and leave the system after a few trials, in which case the attack has been subverted, or (2) keep sending requests though they cannot solve the test. However, by continuing to send requests without solving the test, the zombies become distinguishable from legitimate users, both human and machine.

In $Stage_1$, Kill-Bots tracks how often a particular IP address has failed to solve a puzzle. It maintains a Bloom filter whose entries are 8-bit counters.[4] Whenever a client is given a graphical puzzle, its IP address is hashed and the corresponding entries in the Bloom filter are incremented. In contrast, whenever a client comes back with a correct answer, the corresponding entries in the Bloom filter are decremented. Once all the counters corresponding to an IP address reach a particular threshold (in our implementation $\xi = 32$), the server drops all packets from that IP address and gives no further tests to that client.

When the attack starts, the Bloom filter has no impact and users are authenticated using the graphical puzzles. Yet, as the zombies receive more puzzles and do not answer them, their counters pile up. Once a client has ξ unanswered puzzles, it will be blocked. As more zombies get blocked, the server's load will decrease and approach its normal level. Once this happens the server does not need to use the graphical tests any more. The server no longer issues puzzles; instead it relies solely on the Bloom filter to block requests from the zombie clients. We call this mode of operation $Stage_2$. Sometimes the attack rate is so high that even though the Bloom filter catches all the attack packets, the overhead of receiving the packets by the device driver becomes noticeable. If the server notices that both the load is stable and the Bloom filter is not catching any new zombie IP addresses, then the server concludes that the Bloom filter has caught all attack IP addresses and switches off issuing puzzles, i.e., the server switches to $Stage_2$. If subsequently the load increases, then the server resumes the issuing of puzzles.

In our experiments, the Bloom filter takes only a few minutes to detect and block all offending clients. In general, the higher the attack rate, the faster the Bloom filter will detect the zombies and block their requests. A full description of the Bloom filter is in § 4.

2.3 Resource Allocation & Admission Control

[3] A puzzle answer has an HTTP request of the form `GET /validate?answer=`$\texttt{ANSWER}_i$, where i is the puzzle ID.

[4] Please refer to [15] for a general description of the Bloom filter.

Symbol	Description
α	Admission Probability; Unauthenticated clients are dropped with prob. $1-\alpha$.
λ_a	Arrival rate of attacking HTTP requests
λ_l	Arrival rate of legitimate HTTP requests
λ_s	Arrival rate of legitimate sessions
$\frac{1}{\mu_p}$	Mean time to serve a puzzle
$\frac{1}{\mu_h}$	Mean time to serve an HTTP request
ρ_p	Fraction of time the server spends authenticating clients
ρ_h	Fraction of time the server spends serving authenticated clients
ρ_i	Fraction of time the server is idle
$\frac{1}{q}$	Mean # of requests per legitimate session

Table 1: Variables used in the analysis

A Web site that performs authentication to protect itself from DDoS attacks is faced by a general problem. It has a certain pool of resources, which it needs to divide between authenticating the clients and servicing the ones already authenticated. There is an optimal balance between these two functionalities. Spending a large amount of resources on the authentication might leave the server unable to fully service the authenticated clients. Hence, the server wastes resources on authenticating new clients that it cannot serve. On the other hand, spending too many resources on serving authenticated clients reduces the rate at which new clients are authenticated and admitted into the server, which might result in idle periods with no clients in service.

In this section, we model a Web server that implements an authentication procedure in the interrupt handler. This is a standard location for packet filters and kernel firewalls [47, 38, 4]. It allows dropping unwanted packets as early as possible. We use the model to devise an admission control scheme that maximizes the server's goodput by finding the optimal probability with which new clients should be authenticated. Our model is fairly general and independent of how the authentication is performed. The server may be authenticating the clients by checking their login information, verifying their passwords, or asking them to solve a puzzle. Furthermore, we do not make any assumption on the distribution or independence of the interarrival times of legitimate sessions, or of attacker requests or of service times. Table 1 shows the variables used in the analysis.

2.3.1 *Results of the Analysis*

This section summarizes the results of our analysis and discusses their implications. The detailed derivations are in § 2.3.2.

The admission probability that maximizes the server's goodput (the time spent on serving HTTP requests) is:

$$\alpha^* = \min\left(\frac{q\mu_p}{(B+q)\lambda_s + q\lambda_a}, 1\right) \quad \text{and} \quad B = \frac{\mu_p}{\mu_h}, \qquad (1)$$

where λ_a is the attack request rate, λ_s is the legitimate users' session rate, $\frac{1}{\mu_p}$ is the average time taken to serve a puzzle, $\frac{1}{\mu_h}$ is the average time to serve an HTTP request, and $\frac{1}{q}$ is the average number of requests in a session. When a request from an unauthenticated client arrives, the server should drop it with probability $1-\alpha^*$, and attempt to authenticate it with probability α^*. This yields an optimal server goodput, which is given by:

$$\rho_g^* = \min\left(\frac{\lambda_s}{q\mu_h}, \frac{\lambda_s}{(1+\frac{q}{B})\lambda_s + q\frac{\lambda_a}{B}}\right). \qquad (2)$$

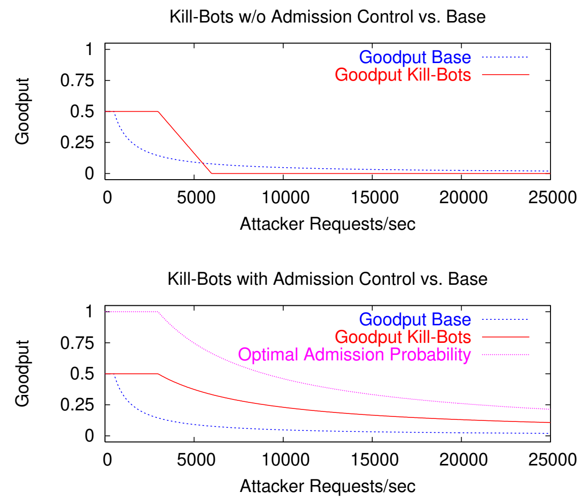


Figure 7: Comparison of the goodput of a base/unmodified server with a Kill-Bots server. Server has a legitimate load of 50%. (TOP) Kill-Bots without admission control. (BOTTOM) Kill-Bots with admission control. The graphs show that Kill-Bots improves server goodput, is even better with admission control, particularly at high attack rates.

In comparison, a server that does not use authentication has goodput:

$$\rho_g^b = \min\left(\frac{\lambda_s}{q\mu_h}, \frac{\lambda_s}{\lambda_s + q\lambda_a}\right). \qquad (3)$$

But authentication is effective in combating DDoS only when authentication consumes less resources than service, i.e., $\mu_p >> \mu_h$. Hence, $B >> 1$, and the server with authentication can survive attack rates that are up to B times larger without any loss in goodput.

Also, compare ρ_g^* with the goodput of a server which implements authentication without admission control (i.e., $\alpha = 1$) given by:

$$\rho_g^a = \min\left(\frac{\lambda_s}{q\mu_h}, \max\left(0, 1 - \frac{\lambda_a + \lambda_s}{\mu_p}\right)\right). \qquad (4)$$

For attack rates, $\lambda_a > \mu_p$, the goodput of the server with no admission goes to zero, whereas the goodput of the server that uses admission control decreases more gracefully.

Fig. 7 illustrates the above results; A Pentium-IV, 2.0GHz 1GB RAM, machine can serve 1-2 pkt puzzles at a peak rate of 6000/sec ($\mu_p = 6000$). Assume, conservatively, that each HTTP request fetches a file of size 15KB($\mu_h = 1000$), that a user makes about 20 requests in a session($q = 1/20$) and the normal server load is about 50%. Fig. 7 qualitatively compares the goodput of a server which does not use admission control (a base server) with the goodput of a Kill-Bots server for both the case of $\alpha = 1$ and α^*. These are computed using equations 2, 4, and 3 respectively, for the above parameter values. The top graph in Fig 7 shows that authentication improves server's goodput. The bottom graph shows the additional improvement gained from adapting the admission probability α.

2.3.2 *Analysis*

(a) Server with no authentication (base server): Let us first analyze the performance of an attacked Web server that does not use any authentication mechanism. Fig. 8 shows a model of such server. The server serves HTTP requests with an average rate μ_h. Attacking HTTP requests arrive at a rate λ_a. Legitimate users/sessions, on the other

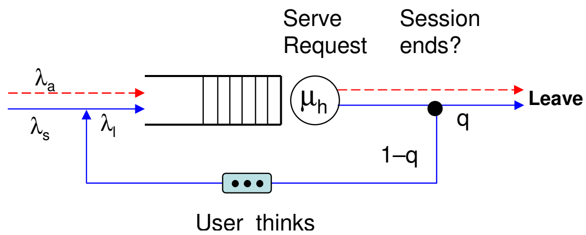


Figure 8: Model of a server that does not use authentication.

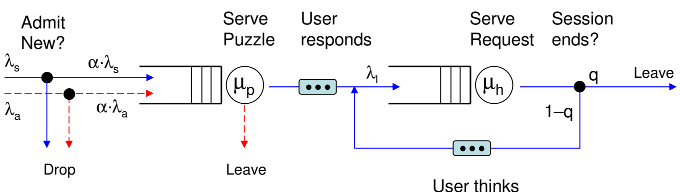


Figure 9: A server that implements some authentication mechanism.

hand, arrive at an average rate λ_s, where a given session consists of some random number of HTTP requests. When their HTTP request is served, legitimate users either leave the web site with probability q or send another HTTP request with probability $1-q$, (potentially after some thinking time). At the input to the queue in Fig. 8, the average rate of legitimate HTTP requests, denoted λ_l, equals the sum of λ_s plus the rate from the feedback loop of subsequent requests, where the latter is $1-q$ times the departure rate from the server of legitimate HTTP requests, denoted λ_d:

$$\lambda_l = \lambda_s \ + \ (1-q)\lambda_d \tag{5}$$

Given that the server occupancy is less than one, i.e. given that the offered load can be handled by the server (though possibly with significant delay), then $\lambda_d = \lambda_l$ and solving equation 5 for λ_l yields:

$$\lambda_l = \frac{\lambda_s}{q}. \tag{6}$$

One can view $1/q$ as the mean number of requests per session.

We make some simplifying assumptions. First, the system of Fig. 8 is in steady state, i.e. for practical purposes we assume that a time interval where the parameters are essentially constant exists. Second, we assume that the server will process requests if any are present(work-conserving). However, we do NOT make any assumptions on the distribution or independence of the interarrival times of legitimate sessions, or of attacker requests or of service times. Under these conditions, the occupancy of the server, ρ, i.e. the fraction of time the server is busy, will be the offered load, whenever this load is less than 1. Otherwise, ρ will be capped at 1. The offered load is the arrival rate of requests $\lambda_a + \lambda_l$ times the mean service time $\frac{1}{\mu_h}$, thus

$$\rho = \min\left(\frac{1}{\mu_h}(\lambda_a + \frac{\lambda_s}{q}), 1\right). \tag{7}$$

The goodput of the server is the fraction of the occupancy due to processing legitimate requests:

$$\rho_g^b \ = \ \frac{\lambda_s/q}{\lambda_s/q \ + \ \lambda_a}\rho \ = \ \frac{\lambda_s}{\lambda_s + q\lambda_a}\rho. \tag{8}$$

When there are no attackers, the server's goodput is simply its occupancy, which is $\frac{\lambda_s}{q\mu_h}$. However for large attack rates, the server's goodput decreases proportionally to the attack rate. Moreover, for offered loads greater than one, the goodput could degrade further depending on how the real system handles congestion.

(b) Server provides authentication: Next, we present a general model of the performance of a server that implements some authentication mechanism. Fig. 9 illustrates the model. The server divides its time between authenticating new clients and serving the ones already authenticated. A new client is admitted to the authentication phase with a probability α that depends on the occupancy of the server, i.e., how busy the server is. Authentication costs $\frac{1}{\mu_p}$ cpu time. (When applied to Kill-Bots, $\frac{1}{\mu_p}$, is the average time to send a graphical test). Other parameters are same as before. The server spends a fraction of time, ρ_p, on authenticating clients, and a fraction of time, ρ_h, serving HTTP requests from authenticated clients. Since the server serves HTTP requests only from authenticated clients, the goodput, ρ_g equals ρ_h.

Using the same general assumptions as for Fig. 8, we wish to determine the value of the admit probability, α^*, that maximizes the goodput, ρ_h, given the physical constraint that the server can not be busy more than 100% of the time. That is:

$$\begin{aligned} \max_{0\le\alpha\le1} & \qquad \rho_h & (9)\\ \text{such that } \ \rho_p + \rho_h & \ \le \ 1, & (10)\\ \text{and given constraint 10: } \rho_p & \ = \ \alpha\frac{\lambda_a + \lambda_s}{\mu_p} & (11)\\ \rho_h & \ = \ \alpha\frac{\lambda_s}{q\ \mu_h} & (12) \end{aligned}$$

Since the goodput ρ_h is increasing in α, (Eq. 12), we would want to make α as big as possible, subject to the constraint (10). Consider first the simple case where $\rho_p + \rho_h$ is strictly less than 1 even when all clients are admitted to the authentication step, i.e., $\alpha = 1$. Then the optimal choice for α, denoted α^*, is 1, and the maximal goodput ρ_g^* is $\frac{\lambda_s}{q\ \mu_h}$. For the more interesting case, now suppose that the constraint (10) would be binding at some value of α less than or equal to one. The value of α at which the constraint first becomes binding is the largest feasible value for α, and thus is the value that maximizes the goodput. Substituting Eq. 11 and 12 into $\rho_p + \rho_h = 1$ and solving for α yields the maximizing value. Summarizing the two cases, the optimal value for the admission probability is:

$$\alpha^* = \min\left(\frac{q\mu_p}{(B+q)\lambda_s + q\lambda_a}, 1\right) \quad \text{and} \quad B = \frac{\mu_p}{\mu_h}. \tag{13}$$

Substituting α^* into (12) yields the maximal goodput:

$$\rho_g^* = \rho_h^* = \min\left(\frac{B\lambda_s}{(B+q)\lambda_s + q\lambda_a}, \frac{\lambda_s}{q\mu_h}\right). \tag{14}$$

Note that since the authentication is performed in the interrupt handler, it preempts serving HTTP requests. The expressions for occupancy Eq. 11 and 12 can incorporate the constraint Eq. 10 as:

$$\begin{aligned} \rho_p & \ = \ \min(\alpha\frac{\lambda_a + \lambda_s}{\mu_p}, 1), & (15)\\ \rho_h & \ = \ \min(\alpha\frac{\lambda_s}{q\ \mu_h}, 1 - \rho_p). & (16) \end{aligned}$$

Setting α to 1 in Eq. 15 and 16 yields the goodput, $\rho_g^a = \rho_h$, obtained when the web server tries to authenticate all new clients regardless of the offered load.

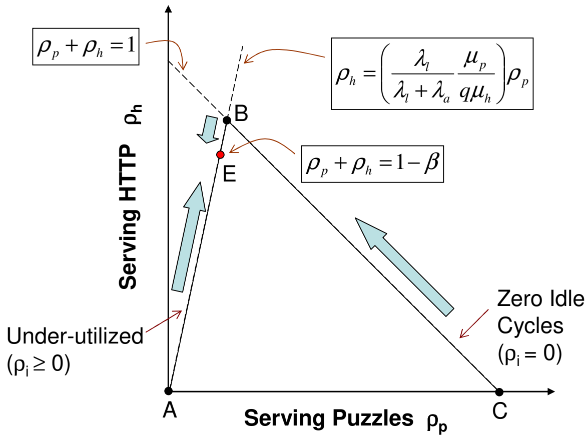


Figure 10: Phase plot showing how Kill-Bots adapts the admission probability to operate at a high goodput

2.4 Adaptive Admission Control

How to run the server at the optimal admission probability? To compute α^* from Eq. 1 requires values for parameters that are typically unknown at the server and change over time, such as the attack rate, λ_a, and the legitimate session rate, λ_s. We devise an adaptive scheme that gradually changes the admission probability α based on measurements of the server's idle cycles. The model of Fig. 9 assumes the CPU has only two tasks: authentication and serving requests. However, in practice where there are other tasks it is important that the adaptive control leave some idle cycles and not attempt for 100% occupancy. Let ρ_i denote the fraction of time the server is idle. And still considering only the task of serving puzzles and requests, we have:

$$\rho_h + \rho_p + \rho_i = 1. \quad (17)$$

If the current admission probability $\alpha > \alpha^*$, the server spends more resources than necessary authenticating new clients. Legitimate users already in the system starve, and the server runs out of idle cycles. On the other hand, if $\alpha < \alpha^*$, the server issues fewer puzzles than necessary, admits fewer legitimate users and goes idle. Thus, if the server is experiencing idle times above some threshold, it should increase its value of α, otherwise it should decrease it. To determine how much the server should increase/decrease α, we note that given $\alpha < \alpha^*$, there will be some idle cycles and by substituting Eq. 11 and 12 in Eq. 17:

$$\forall \alpha < \alpha^* : \quad \alpha \left(\frac{\lambda_a + \lambda_l}{\mu_p} + \frac{\lambda_l}{q\mu_h} \right) = 1 - \rho_i.$$

Hence,

$$\forall \alpha^1, \alpha^2 < \alpha^* : \quad \frac{\alpha^1}{\alpha^2} = \frac{1-\rho_i^1}{1-\rho_i^2}. \quad (18)$$

Thus, we can increase α proportionally to the non-idle cycles (i.e. the occupancy).

We use Fig. 10 to argue the rationale underlying the design of our adaptive admission controller. The figure shows the *relation* between the fraction of time spent on authenticating clients ρ_p and that spent serving HTTP requests ρ_h. The line labeled "Zero Idle Cycles" refers to the states in which the system is highly congested $\rho_i = 0 \rightarrow \rho_p + \rho_h = 1$. The line labeled "Underutilized" refers to the case in which the system has some idle cycles, in which case, taking the ratio of Eq. 11 and 12 leads to $\rho_h = \left(\frac{\lambda_s}{\lambda_s+\lambda_a} \frac{\mu_p}{q\mu_h}\right)\rho_p$. As the fraction of time the system is idle ρ_i changes, the system state moves along the solid line segments A→B→C. Ideally, one would like to operate the system at point B which maximizes the system's goodput, $\rho_g = \rho_h$, and corresponds to $\alpha = \alpha^*$. However, it is difficult to operate at point B because the system cannot tell whether it is at B or not; all points on the segment B-C exhibit $\rho_i = 0$. It is easier to stabilize the system at point E where the system is slightly underutilized because small deviations from E exhibit a change in the value of ρ_i, which we can measure. We pick E such that the fraction of idle time at E is $\beta = \frac{1}{8}$. Thus, every T=10s, we adapt the admission probability according to the following rules:

$$\Delta\alpha = \begin{cases} \gamma_1 \alpha \frac{\rho_i - \beta}{1-\rho_i}, & \rho_i \geq \beta \\ -\gamma_2 \alpha \frac{\beta - \rho_i}{1-\rho_i}, & 0 < \rho_i < \beta \\ -\gamma_3 \alpha. & \rho_i = 0 \end{cases} \quad (19)$$

where γ_1, γ_2, and γ_3 are constant parameters, which Kill-Bots set to $\frac{1}{8}$, $\frac{1}{4}$, and $\frac{1}{4}$ respectively. The above rules allow us to move α proportionally to how far we are from the chosen equilibrium point E, unless the system has no idle cycles in which case we decrease the admission probability aggressively to go back to the stable regime around point E.

3. SECURITY ANALYSIS

In this section, we discuss Kill-Bots's ability to handle a variety of attacks from a determined adversary.

(a) Socially-engineered attack: In a socially-engineered attack, the adversary tricks a large number of humans to solve graphical puzzles on her behalf. Recently, spammers employed this tactic to bypass graphical tests used by Yahoo and Hotmail to prevent automated creation of email accounts [7]. The spammers opened and advertised a Web site containing pornography. Visitors to the porn site were asked to enter the word contained in a CAPTCHA graphic before they were granted access. The porn site downloaded its CAPTCHAs from Yahoo or Hotmail email creation Web page, presented them to the porn site visitors, and used the answers to create new email accounts.

We argue that Kill-Bots is much more resilient against socially engineered attacks than the CAPTCHA system used by email creation sites. In contrast to email account creation where the client is given an ample amount of time to fill in the registration form and solve the puzzle, puzzles in Kill-Bots expire 4 minutes after they have been served. Thus, the attacker cannot accumulate a sufficient amount of answers from human users to mount an attack with a rate high enough for Denial-of-Service. The attacker needs a continuous stream of visitors to his porn site. Indeed, a stream of visitors to the porn site is a necessary but not sufficient condition. The attacker needs to control a porn server at least as popular as the victim Web server. Recall, that Kill-Bots employs a loose form of fairness among authenticated clients; it allows each of them a maximum of 8 parallel connections. Because of this relatively fair treatment of authenticated clients, the attacker needs to maintain the number of authenticated malicious clients larger than that of legitimate users, so that she can grab most of the server's resources. Such a popular porn site is an asset. It is unlikely that the attacker will be willing to jeopardize her popular porn site (or other popular servers) to DDoS an equally or less popular Web site. Furthermore, one should keep in mind that security is a moving target; by forcing the attacker to resort to socially engineered attacks, we made the attack harder and the probability of being arrested higher.

(b) Polluting the Bloom filter: The attacker may try to spoof the IP address and pollute the entries in the Bloom filter, causing Kill-Bots to mistake legitimate users as a malicious. This attack however is not possible because the Bloom filter entries are incremented when the client is sent a puzzle, and decremented when Kill-Bots receives an answer from the client. Both events require a SYN cookie check first.

(c) Copy attacks: In a copy attack, the adversary solves one graphical puzzle, obtains the corresponding HTTP cookie, and distributes it to many zombie machines to give them access to the Web site. It might seem that the best solution to this problem is to include a secure one-way hash of the IP address of the client in the cookie. Unfortunately, this approach does not deal well with proxies or mobile users. Kill-Bots protects against copy attacks by limiting the number of in-progress requests per puzzle answer (our implementation sets this limit to 8).

(d) Replay attack: A session cookie includes a secure hash of the time it was issued, is only valid during a certain time interval, and only covers up to eight simultaneous accesses. If an adversary tries to replay a session cookie outside its time interval it gets rejected. An attacker may solve one puzzle and attempt to replay the "answer" packet to obtain many more Kill-Bots cookies. Recall that Kill-Bots issues a cookie for a valid answer only if it is accompanied with a valid Token (Fig 6). Further, the cookie is an updated version of the token. Hence, replaying the "answer" packet yields the same cookie.

(e) Collecting a database of all graphical puzzles and their answers: The adversary might try to collect all possible puzzles and the corresponding answers. When a zombie receives a puzzle, it searches its database, find the corresponding answer, and send it back to the server. To protect from this attack, Kill-Bots assumes the presence of a large number of puzzles, that are periodically replaced with a new set. Generation of the graphical puzzles is relatively easy [55], and can either be done on the web server itself in periods of inactivity (at night) or on a different machine dedicated for this purpose. Also puzzles may be purchased from a trusted third party. Since the space of all possible graphical puzzles is huge, building a database of these puzzles and their answers is a daunting task. Making this database available to the zombies, and ensuring they can search it and obtain answers within the 4 minute lifetime window of a puzzle is very difficult.

(f) DoS attack on the authentication mechanism: Kill-Bots is highly robust against DDoS attacks on the authentication code. As stated earlier, Kill-Bots does not allow unauthenticated clients to access any connection state such as TCBs or sockets. The computational cost of authenticating a client is dominated by the cost of interrupts. Serving a puzzle incurs a total computational overhead of only $\approx 40\mu$s.

(g) Concerns regarding in-kernel HTTP header processing: Kill-Bots does not parse HTTP headers; it pattern matches the argument to the `GET` and the `Cookie:` fields against the fixed string *validate* and against a 192-bit Kill-Bots cookie respectively. The pattern-matching is done in-place, i.e. without copying the data and is inexpensive; less than 8μs (§ 5.1.2) per request.

(h) Breaking the CAPTCHA: Prior work on automatically solving simple CAPTCHAs exists [42], but such programs are not widely available to the public for security reasons [42]. However, when CAPTCHAs can be broken, Kill-Bots can switch to another kind as long as they fit in a few packets.

4. KILL-BOTS SYSTEM ARCHITECTURE

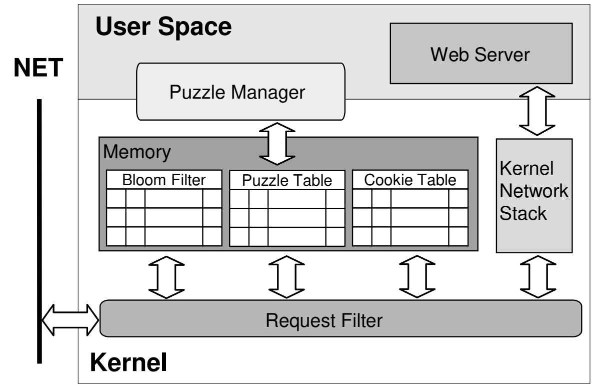


Figure 11: **A Modular representation of the Kill-Bots code.**

Kill-Bots is implemented as a kernel update with key components illustrated in Fig. 11. We provide a high level description of these components and omit the details for lack of space.

(a) The Puzzle Manager consists of two components. First, a user-space stub that asynchronously generates new puzzles and notifies the kernel-space portion of the Puzzle Manager of their locations. Generation of the graphical puzzles is relatively easy [2], and can either be done on the web server itself in periods of inactivity (at night) or on a different machine dedicated for this purpose. Also puzzles may be purchased from a trusted third party. The second component is a kernel-thread that periodically loads new puzzles from disk into the Puzzle Table in memory that the Request Filter then distributes to new sessions.

(b) The Request Filter (RF) processes every incoming TCP packet addressed to port 80. It is implemented in the bottom half of the interrupt handler to ensure that unwanted packets are dropped as early as possible.

Fig. 12 provides a flowchart representation of the RF code. When a TCP packet arrives for port 80, the RF first checks whether it belongs to an established connection in which case the packet is immediately queued in the socket's receive buffer and left to standard kernel processing. Otherwise the filter checks whether the packet starts a new connection (i.e., is it a SYN?), in which case, the RF replies with a SYNACK that contains a standard SYN cookie. If the packet is not a SYN, we examine whether it contains any data; if not, the packet is dropped without further processing. Next, the RF performs two inexpensive tests in an attempt to drop unwanted packets quickly; it hashes the packet's source IP address and checks whether the corresponding entries in the Bloom filter have all exceeded ξ unsolved puzzles, in which case the packet is dropped. Otherwise, the packet goes through admission control and is dropped with probability $1 - \alpha$. If the packet passes all of the above checks, we need to look for 4 different possibilities: (1) this might be the first data packet from an unauthenticated client, and thus we should send it a puzzle and terminate the connection immediately; (2) this might be a packet from a client which has already received a puzzle and is coming back with an answer. In this case, we need to verify the answer and assign the client an HTTP cookie, which allows it access to the server for a prolonged period of time; (3) Or it is an authenticated client which has a Kill-Bots HTTP cookie and is coming back to retrieve more objects; (4) If none of the above is true then the packet should be dropped. These checks are ordered according to their increasing cost to allow the system to shed away attack clients with as little cost as possible.

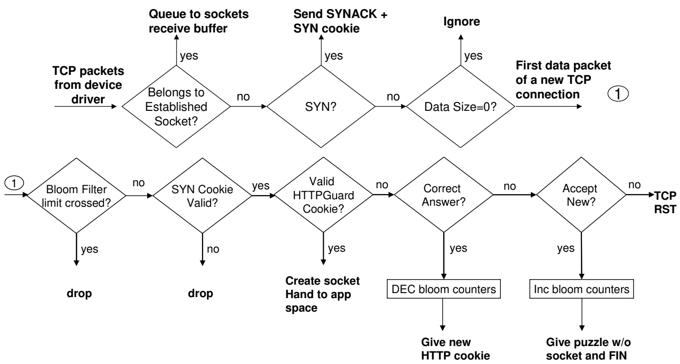


Figure 12: **The path traversed by new sessions in Kill-Bots. This code is performed by the Request Filter module.**

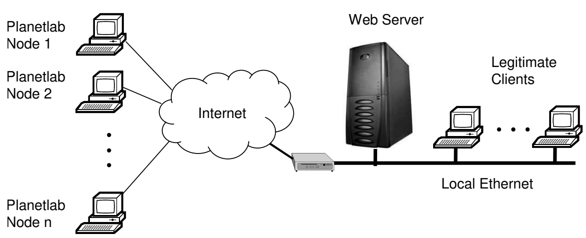


Figure 13: **Our experimental setup.**

(c) The Puzzle Table maintains the puzzles available to be served to users. We implement a simple mechanism to avoid races between writes and reads to the puzzle table by dividing the Puzzle Table into two memory regions, a write window and a read window. The Request Filter fetches puzzles from the read window, while the Puzzle Manager loads new puzzles into the write window. Once the Puzzle Manager completes loading all puzzles, the read and write windows are swapped atomically.

(d) The Cookie Table maintains the number of connections that are in progress for each Kill-Bots cookie. When this number reaches 8, no further connections are allowed for that cookie.

(e) The Bloom Filter counts unanswered puzzles for each IP address, allowing the Request Filter to block requests from IPs with more than ξ unsolved puzzles. (Our implementation sets $\xi = 32$). Usually, Bloom filters are characterized by two parameters; the number of entries N in the array and the number of hash functions k that map keys onto elements of the array. Our implementation uses $N = 2^{20}$ and $k = 2$. Since a potentially large set of keys, in our case 32 bit IP addresses, are mapped onto much smaller storage, Bloom filters are essentially lossy. This means that there is a non-zero probability that all k counters corresponding to a legitimate user pile up to ξ due to collisions with attackers (false positives). Assuming a distinct attacker zombies and uniformly random hash functions, the probability a legitimate client is classified as an attacker is approximately $(1 - e^{-ka/N})^k \approx (\frac{ka}{N})^k$. Given our chosen values for N and k, this probability for 75,000 attack machines is 0.023.

5. EVALUATION

We have a Linux-based kernel implementation of Kill-Bots, which we evaluate in the wide-area network using PlanetLab.

5.1 Experimental Environment

Function	CPU Latency
Bloom Filter Access	.7 μs
Processing HTTP Header	8 μs
SYN Cookie Check	11 μs
Serving puzzle	31 μs

Table 2: **Kill-Bots Microbenchmarks**

(a) Web Server: The web server is a standard 2GHz Pentium IV Linux machine with 1GB of memory and 512kB L2 cache running an unmodified mathopd [14] server on top of a modified Linux 2.4.10 kernel.[5] In our implementation of Kill-Bots, we modified about 300 lines of kernel code, mostly in the TCP/IP protocol stack. The puzzle manager, the bloom filter and the adaptive controller are implemented in an additional 500 lines. To obtain realistic server workload, we replicate both static and dynamic content served by two web-sites, our lab's Web server and a Debian mirror server.

(b) Modeling Request Arrivals: Legitimate clients generate requests by replaying HTTP traces collected at our Lab's Web server and a Debian mirror server. Multiple segments of the same long trace are played simultaneously to control the load generated by legitimate clients. An attacker issues requests at a desired rate by randomly picking a URI (static/dynamic) from a list of content available on the server. The results presented herein are for the case when zombie request arrivals are CBR (constant bit rate). Attack requests with Poisson arrivals exhibit qualitatively similar results.

(c) Experiment Setup: We evaluate Kill-Bots in the wide-area network using the setup in Fig. 13. The Web server is connected to a 100Mbps local Ethernet. We launch CyberSlam attacks from 100 different nodes on PlanetLab using different port ranges to simulate multiple attackers per node. Each PlanetLab node simulates up to 256 zombies, which results in a total of 25,600 attack clients. We emulate legitimate clients on multiple machines connected to our local network to ensure that any difference in their performance is due to the service they receive from the Web server, rather than wide-area path variability.

(d) Emulating Clients: We use WebStone2.5 [3] to emulate both legitimate Web clients and attackers. WebStone is a benchmarking tool that issues HTTP requests to a web-server given a specific distribution over the requests, and reports statistics. We extended WebStone in two ways. First, we added support for HTTP sessions, cookies, and for replaying requests from traces. Second, we need the clients to issue requests at specific rates independent of how the web-server responds to the load. For this, we rewrote WebStone's networking code using libasync [37], an asynchronous socket library. We increased the kernel limit on file descriptors per process `fdlimit` to 40960 on each of our local machines, to emulate large numbers of legitimate users using few machines. Since, PlanetLab nodes have a rigid `fdlimit` of 4096, we scaled the number of zombies by using many more nodes.

5.1.1 Metrics

We evaluate Kill-Bots by comparing the performance of a base server (i.e., a server with no authentication) with its Kill-Bots mirror operating under the same conditions. Server performance is measured using these metrics:

(a) Goodput of legitimate clients: This is the amount of bytes per second delivered to *all* legitimate client applications. Goodput ignores

[5]We picked mathopd because of its simplicity.

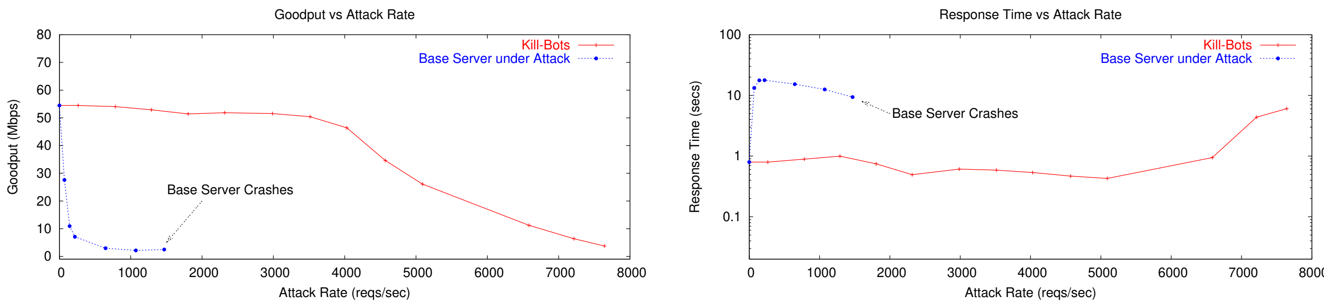


Figure 14: Kill-Bots under CyberSlam: Goodput and average response time of legitimate users at different attack rates for both a base server and its Kill-Bots version. Graphs show that Kill-Bots substantially improves server's performance under high attack rates.

TCP retransmissions and is averaged over 30s windows.

(b) Response times of legitimate clients: Response time is the elapsed time before a request is completed or timed out. We timeout incomplete requests after 1 minute.

(c) Cumulative number of legitimate requests dropped: This metric measures the total number of legitimate requests dropped since the beginning of the experiment until the current time.

5.1.2 *Microbenchmarks*

We run microbenchmarks on the Kill-Bots kernel to measure the time taken by the various modules. We use the x86 `rdtsc` instruction to obtain fine-grained timing information; `rdtsc` reads a hardware timestamp counter that is incremented once every CPU cycle. On our 2GHz web-server, this yields a resolution of 0.5 nanoseconds. The measurements are for CAPTCHAs of 1100 bytes.

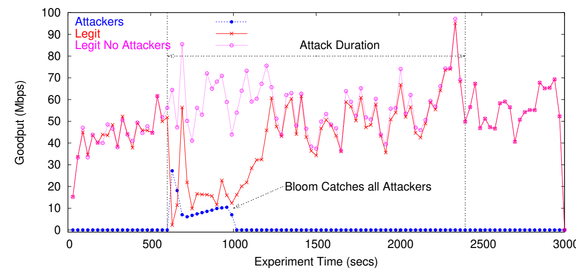
Table 2 shows our microbenchmarks. The overhead resulting from sending a graphical puzzle is $\approx 40\mu s$ (processing http header +serve puzzle), which means that the CPU can process puzzles faster than the time to transmit a 1100B puzzle on our 100Mb/s Ethernet. However, the authentication cost is dominated by standard kernel code for processing incoming TCP packets, mainly the interrupts ($\approx 10\mu s$ per packet [30], about 10 packets per TCP connection). Thus, the CPU is the bottleneck for authentication and as shown in § 5.4, performing admission control based on CPU utilization is beneficial.

Note also that checking the Bloom filter is much cheaper than other operations including the SYN cookie check. Furthermore, neither the SYN cookie check nor the HTTP header processing can be done at line-speed (i.e., for 100Mb/s requests). Hence, for incoming requests, we perform the Bloom filter check before the SYN cookie check (Fig. 16). In $Stage_2$, the Bloom filter drops all zombie requests; hence the performance of Kill-Bots is limited by the cost for interrupt processing and device driver access. We conjecture that using polling-based drivers [30, 39] will improve performance at high attack rates.

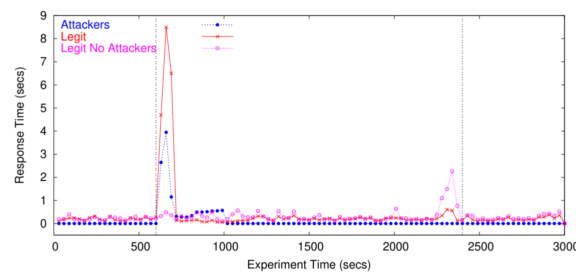
5.2 Performance of Kill-Bots under CyberSlam

We evaluate the performance of Kill-Bots under CyberSlam attacks, using the setting described in § 5.1. We assume that zombies cannot solve CAPTCHAs. We also assume only 60% of the legitimate clients solve the CAPTCHAs; the others are either unable or unwilling to solve them. This is supported by the results in § 5.6.

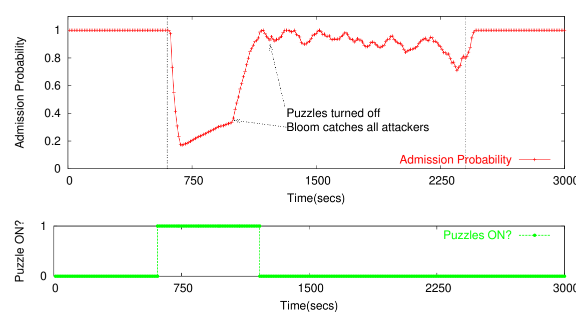
Fig. 14 compares the performance of Kill-Bots with a base (i.e., unmodified) server, as a function of increased attack request rate. Fig. 14a shows the goodput of both servers. Each point on the graph is the average goodput of the server in the first twelve minutes after the beginning of the attack. A server protected by Kill-Bots endures attack rates multiple orders of magnitude higher than the base server. At very high attack rates, the goodput of the Kill-Bots server decreases as the cost of processing the interrupts becomes excessive. Fig. 14b shows the response time of both web servers. The average response time experienced by legitimate users increases dramatically when the base server



(a) Goodput



(b) Average response time of legitimate users



(c) Admission probability

Figure 15: Comparison of Kill-Bots' performance to server with no attack when only 60% of the legitimate users solve puzzles. Attack lasts from 600s to 2400s. (a) Goodput quickly improves once bloom catches all attackers. (b) Response times improve as soon as the admission control reacts to the beginning of attack. (c) Admission control is useful both in $Stage_1$ and in $Stage_2$, after bloom catches all zombies. Puzzles are turned off as soon as Kill-Bots enters $Stage_2$ improving goodput.

is under attack. In contrast, the average response time of users accessing the server using Kill-Bots is unaffected by the ongoing attack.

Fig. 15 shows the dynamics of Kill-Bots during a CyberSlam attack, with $\lambda_a = 4000$ req/s. The figure also shows the goodput and mean response time with no attack, as a reference. The attack begins at $t =$ 600s and ends at $t = 2400$s. At the beginning of the attack, the goodput decreases (Fig. 15a) and the mean response time increases (Fig. 15b). Yet, quickly the admission probability decreases (Fig. 15c), causing the mean response time to go back to its value when there is no attack. The goodput however stays low because of the relatively high attack rate, and because many legitimate users do not answer puzzles. After a few minutes, the Bloom filter catches all zombie IPs, causing puzzles to no longer be issued (Fig. 15c). Kill-Bots now moves to $Stage_2$ and performs authentication based on just the Bloom filter. This causes a large increase in goodput (Fig. 15a) due to both the admission of users who were earlier unwilling or unable to solve CAPTCHAs and the reduction in authentication cost. In this experiment, despite the ongoing CyberSlam attack, Kill-Bots' performance in $Stage_2$ ($t = 1200s$ onwards), is close to that of a server not under attack. Note that the normal load significantly varies with time and the adaptive controller (Fig. 15c) reacts to this load between $t = 1200s$ and $t = 2400s$ keeping response times low, yet providing reasonable goodput.

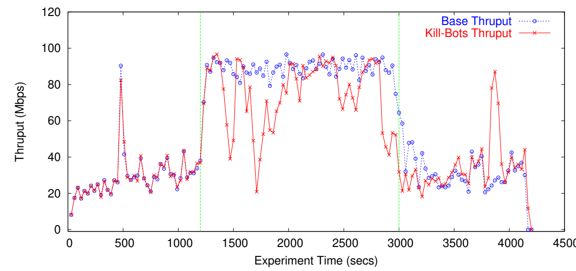
5.3 Kill-Bots under Flash Crowds

We evaluate the behavior of Kill-Bots under a Flash Crowd. We emulate a Flash Crowd by playing our Web logs at a high speed to generate an average request rate of 2000 req/s. The request rate when there is no flash crowd is 300 req/s. This matches Flash Crowd request rates reported in prior work [25]. In our experiment, a Flash Crowd starts at $t = 1200s$ and ends at $t = 3000s$.

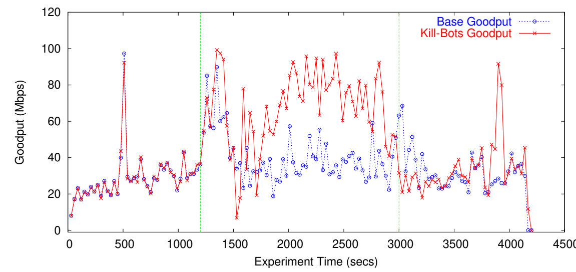
Fig. 16 compares the performance of the base server against its Kill-Bots mirror during the Flash Crowd event. The figure shows the dynamics as functions of time. Each point in each graph is an average measurement over a 30s interval. We first show the total throughput of both servers in Fig. 16a. Kill-Bots has slightly lower throughput for two reasons. First, Kill-Bots attempts to operate at $\beta = 12\%$ idle cycles rather than at zero idle cycles. Second, Kill-Bots uses some of the bandwidth to serve puzzles. Fig. 16b reveals that the throughput figures are misleading; though Kill-Bots has a slightly lower throughput than the base server, its goodput is substantially higher (almost 100% more). This indicates that the base server wasted its throughput on retransmissions and incomplete transfers, whereas Kill-Bots used its admission control policy to prevent server's overload. This is further supported by the results in Fig. 16c and 16d, which show that Kill-Bots drastically reduces the average response time and the average time for connection establishment.

That Kill-Bots improves server performance during Flash Crowds might look surprising. Although all clients in a Flash Crowd can answer the graphical puzzles, Kill-Bots computes an admission probability α such that the system only admits users it can serve. In contrast, a base server with no admission control accepts additional requests into the system even when it is overloaded, which decreases throughput and response time. Fig. 16e supports this argument by showing how the admission probability α changes during the Flash Crowd event to allow the server to shed away the extra load.

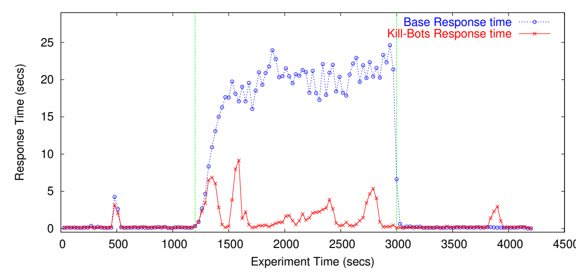
Finally, Fig. 17 shows the cumulative number of dropped requests and dropped sessions during the Flash Crowd event for both the base server



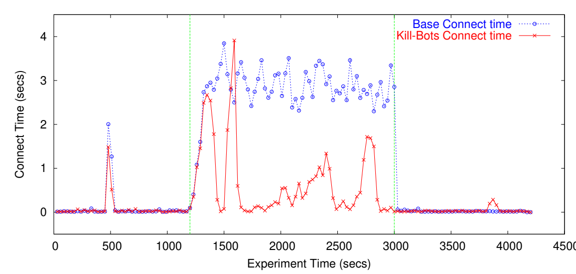
(a) Throughput of the server



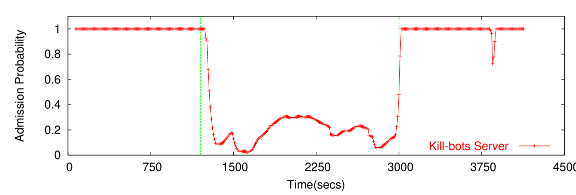
(b) Average goodput of users



(c) Average response time perceived by users



(d) Average time for connection establishment



(e) Admission probability at the server

Figure 16: Kill-Bots under Flash Crowds: The Flash Crowd event begins at $t = 1200s$ and ends at $t = 3000s$. Though the throughput of Kill-Bots is slightly lower than that of the base server, its Goodput is much higher and its average reponse time is much lower.

and the Kill-Bots server. Interestingly, the figure shows that Kill-Bots drops more sessions but fewer requests than the base server. The base server accepts new sessions more often than Kill-Bots but keeps dropping their requests. Kill-Bots drops sessions early during the authentication phase, but once a session is admitted it is given a Kill-Bots

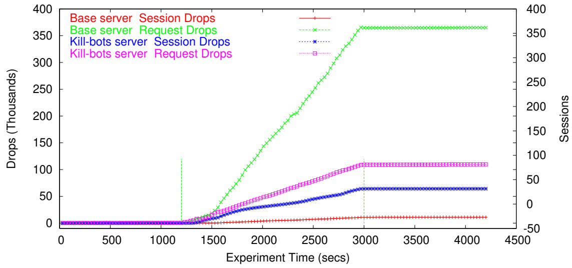


Figure 17: Cumulative numbers of dropped requests and dropped sessions under a Flash Crowd event lasting from $t = 1200s$ to $t = 3000s$. Kill-Bots adaptively drops sessions upon arrival, ensuring that accepted sessions obtain full service, i.e. have fewer requests dropped.

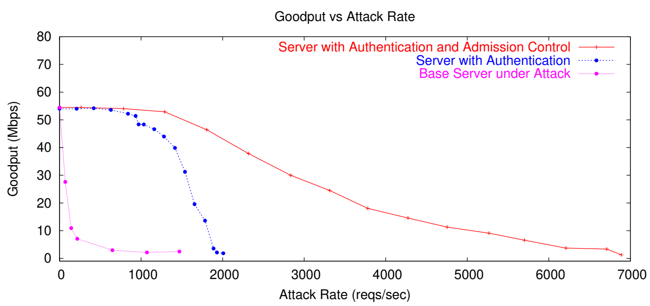


Figure 18: Server goodput substantially improves with adaptive admission control. Figure is similar to Fig. 7 but is based on wide-area experiments rather than analysis. (For clarity, the Bloom filter is turned off in this experiment.)

cookie which allows it access to the server for the next half an hour.

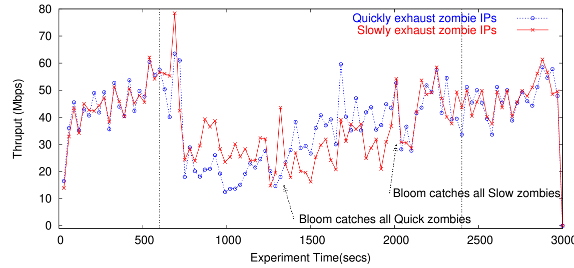
5.4 Importance of Admission Control

In § 2.3.1, using a simple model, we showed that authentication is not enough, and good performance requires admission control. Fig. 18 provides experimental evidence that supports adapting the admission probability and our analysis in § 2.3. The figure compares the goodput of a version of Kill-Bots that uses only puzzle-based authentication, with a version that uses both puzzle-based authenticaiton and admission control. We turn off the Bloom filter in these experiments because we are interested in measuring the goodput gain obtained only from admission control. The results in this figure are fairly similar to those in Fig. 7. It shows that admission control dramatically increases server goodput. Further, the goodput of the server that uses admission control decreases more gracefully with increased attack rates.

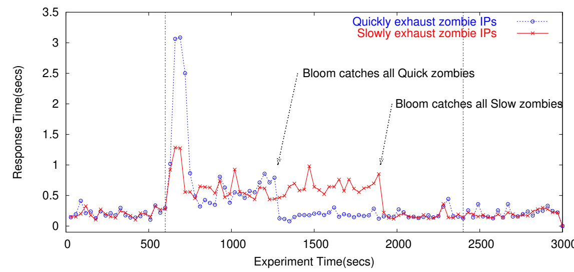
5.5 Impact of Different Attack Strategies

The attacker might try to increase the severity of the attack by prolonging the time until the Bloom filter has discovered all attack IPs and blocked them, i.e., prolonging the time until the system transitions from $Stage_1$ to $Stage_2$. To do so, the attacker needs to slowly use the set of IP addresses in her Botnets, keeping fresh IPs for as long as possible. We show however, that the attacker does not gain much by using her IP zombie addresses slowly. Indeed, there is a tradeoff between using all zombie IPs quickly to create a severe attack for a short period, and using them slowly to prolong a milder attack.

Fig. 19 shows the performance of Kill-Bots under two attack strategies; A fast strategy in which the attacker introduces a fresh zombie IP every 2.5 seconds, and a slow strategy in which the attacker introduces fresh



(a) Goodput



(b) Average response time of legitimate users

Figure 19: Comparison between 2 attack strategies; A fast strategy that uses all fresh zombie IPs in a short time, and a slow strategy that consumes fresh zombie IPs slowly. Graphs show a tradeoff; the slower the attacker consumes the IPs, the longer it takes the Bloom filter to detect all zombies. But the attack caused by the slower strategy though lasts longer has a milder impact on the goodput and response time.

Case	Fraction of Users
Answered puzzle	55%
Did not answer puzzle	45%
Interested surfers who answered puzzle	74%

Table 3: The percentage of users who answered a graphical puzzle to access the Web server. We define interested surfers as those who access two or more pages on the Web site.

a zombie IP every 5 seconds. In this experiment, the total number of zombies in the Botnet is 25000 machines, and the aggregate attack rate is constant and fixed at $\lambda_a = 4000$ req/s. The figure shows that the fast attack strategy causes a short but high spike in mean response time, and a substantial reduction in goodput that lasts for a short interval (about 13 minutes), until the Bloom filter catches the zombies. On the other hand, the slow strategy affects the performance for a longer interval (about 25 minutes) but has a milder impact on goodput and mean response time.

5.6 User Willingness to Answer CAPTACHAs

We conducted a user study to evaluate the willingness of users to solve CAPTCHAs. We instrumented our research group's Web server to present puzzles to 50% of all external accesses to the *index.html* page. Clients that answer the puzzle correctly are given an HTTP cookie that allows them access to the server for an hour. The brief experiment lasted from Oct. 3 until Oct. 7. During that period, we registered a total of 973 accesses to the page, from 477 distinct IP addresses.

We want to compute the fraction of human users that solve graphical puzzles to access the server. When a client fails to solve the puzzle, we need to determine whether the client is a frustrated human or an automated script. Some automated requests are easy to identify because the User Agent Field in the HTTP request states it is a robot(e.g.

Googlebot); others are not so easily identified.

We compute two types of results. First, we filter out requests from known robots and compute the fraction of clients who answered our puzzles. We find that 55% of all clients answered the puzzles. As explained above, this number underestimates the fraction of humans that answered the puzzles. Second, we distinguish between clients who check only the group's main page and leave the server, and those who after accessing the main page follow one of the links therein. We call the latter *interested surfers*. We would like to check how many of the interested surfers answered the graphical puzzle because these users probably bring more value to the Web site. To answer this question we apply the standard Bayes' rule:

$$Pr[A/B] = \frac{Pr[A, B]}{Pr[B]},$$

where, event A is "user solves CAPTCHA" and event B is "user is an interested surfer". Using the information in the logs and replacing probabilities by the fraction of users, we find that $Pr[B] = 0.507$, $Pr[A, B] = 0.381$ and thus, the probability that an interested surfer answers the puzzle is $0.381/0.507$ which is 74%. Tab. 3 summarizes our results. These results may not be representative of users in the Internet, as the behavior of user populations may differ from one server to another.

6. RELATED WORK

Related work falls into the following areas.

(a) Denial of Service: Much prior work on DDoS exists; It describes specific attacks (e.g., SYN flood [45], the Smurf attack [16], reflector attacks [43]), and presents detection techniques, or proposes specific solutions and countermeasures. In particular, Moore et al. [40] propose the backscatter technique, which detects DDoS instances by monitoring traffic sent to unused segments of the IP address space. Savage et al. [48] propose a traceback mechanism that allows the victim of a DoS attack to trace the offending packets to their source. Other researchers propose variations on the traceback idea to make it applicable to attacks with a small number of packets [9, 49, 58]. Gil et al [22] detect a bandwidth flood attack against a Web server by comparing the number of packets from client to server with those from server to client. Anderson et al. propose a modified Internet architecture that protects a destination from unwanted traffic. Only packets with the right capabilities are delivered to the destination [11]. The pushback [36] proposal modifies the routers to detect big bandwidth consumers and propagate this information toward upstream routers to throttle the traffic closer its the source. Juels and Brainard propose to use computational client puzzles to counter SYN flood attacks [26]. In addition, a few commercial filters exist such as Webscreen which uses heuristics to detect abnormal traffic and attack patterns [31, 8, 5].

Recently, researchers have proposed to use overlays as distributed firewalls [10, 28]. The server IP address is known only to the overlay. Clients who want to access the server have to go through the overlay nodes, which check the incoming packets and apply any necessary filtering. The authors of [41] extend the overlay approach to use graphical Turing tests.

(b) CAPTCHAs: Our authentication mechanism uses graphical tests or CAPTCHAs. Von Ahn et. al [55] and several others [29, 46, 21] proposed to use CAPTCHAs to identify humans from machines. CAPTCHAs are currently a popular user authentication mechanism used by many online businesses and free Web mail providers (e.g. [6, 1]). Morein et. al [41] proposed to use CAPTCHAs for protecting from DDoS attacks. Our work differs from theirs as we use CAPTCHAs only as an intermediate step to detect the offending IP addresses and discard their packets. Furthermore, we combine authentication with admission control and focus on efficient kernel implementation.

(c) Flash Crowds and Server Overload: The authors of [20, 24, 56, 19, 57] discuss the importance of admission control in improving the performance of servers under overload and propose various admission control schemes. Also, much prior work has looked at extensions to the operating system that allow better resource management and improved server performance during periods of overload [12, 54, 13]. In addition, Jamjoom et. al [25] propose persistent dropping of TCP SYN packets in routers to tackle Flash Crowds. Finally, A number of paper propose to use overlays and peer-to-peer networks to shed load off servers during Flash Crowds [27, 50, 52].

7. LIMITATIONS & OPEN ISSUES

A few limitations and open issues are worth discussing. First, Kill-Bots interacts in a complex way with Web proxies. If all clients behind the proxy are legitimate users, then the existence of the proxy has no impact on the clients' surfing experience. In contrast, if a zombie shares the proxy with legitimate clients and uses the proxy to mount an attack on the Web server, Kill-Bots will learn the proxy's IP address and block all requests from that proxy, including the ones from legitimate users. Thus, Kill-Bots imposes fate sharing on clients that use the same proxy. Similarly, it imposes fate sharing on clients that use a single NATed IP address.

Second, the system has a few parameters which we have assigned values based on intuition and our experience with the operational environment. For example, we example, we set the Bloom filter threshold $\xi = 32$ because we want to allow the legitimate users to drop a number of puzzles because of congestion or indecisiveness without being punished. There is nothing special about 32, but we need a value that is neither too big nor too small. Similarly, we allow a client that answers a CAPTCHA a maximum of 8 parallel connections because this number seems to provide a good tradeoff between the improved performance gained from parallel connections and the desire to limit the resources that might be lost because of a compromised cookie. Other system parameters are similarly chosen based on intuition or experimentation.

Third, Kill-Bots assumes that the first data packet of the TCP connection will contain the `GET` and `Cookie` lines of the HTTP request. In general the request may span multiple packets, but this happens rarely. Further many application-level firewalls and HTTP proxies make a similar assumption [59].

Fourth, eventually, the Bloom filter needs to be flushed since compromised zombies may turn into legitimate clients. The Bloom filter can be cleaned either by resetting all entries simultaneously or by decrementing the various entries at a particular rate. In the future, we will examine which of these two strategies is more suitable.

We have experimented with a few attack strategies. In the future, we would like to model and analyze the performance of Kill-Bots under *any* attack strategy. We will use the formalization to try to obtain performance guarantees for Kill-Bots that are independent of attacker strategy.

8. CONCLUSION

The Internet literature contains a large body of important research on denial of service solutions and countermeasures. The vast majority of it assumes that the destination can distinguish between malicious and legitimate traffic by performing simple checks on the content of the packets, their headers, or their arrival rates. Yet, attackers are increasingly disguising their traffic by mimicking legitimate users access patterns, which allows them to defy traditional filters. This paper focuses on protecting Web servers from DDoS attacks that masquerade as Flash Crowds. Underlying our solution is the assumption that most online services value human surfers much more than automated accesses. We present a novel design which uses CAPTCHAs to distinguish the IP addresses of the attack machines from those of legitimate clients. In contrast to prior work on CAPTCHAs, our system allows legitimate users access to the attacked server even if they are unable or unwilling to solve graphical tests. We have implemented our design in the Linux kernel and evaluated it in the Internet. We intend to make our implementation publicly available under an open source license.

9. REFERENCES

[1] Hotmail. http://www.hotmail.com.
[2] Jcaptcha. jcaptcha.sourceforge.net/.
[3] Mindcraft Inc. Webstone - The Benchmark for Web Servers. http://www.mindcraft.com/webstone/.
[4] Netfilter/Iptables. http://www.netfilter.org.
[5] Web Screen Technology. http://www.webscreen-technology.com/.
[6] Yahoo! EMail. http://mail.yahoo.com.
[7] Porn gets spammers past Hotmail, Yahoo barriers. CNet News, May 2004. http://news.com.com/2100-1023_3-5207290.html.
[8] A. Akella, A. Bharambe, M. Reiter, and S. Seshan. Detecting DDoS attack on ISP networks. In *ACM MPDS Workshop*, 2003.
[9] Alex Snoeren et al. Hash-Based IP Traceback. In *SIGCOMM*, 2001.
[10] D. Andersen. Mayday: Distributed filtering for Internet services. In *USITS*, 2003.
[11] T. Anderson, T. Roscoe, and D. Wetherall. Preventing Internet Denial-of-Service with capabilities. In *HotNets*, 2003.
[12] G. Banga, P. Druschel, and J. C. Mogul. Resource containers: A new facility for resource management in server systems. In *OSDI*, 1999.
[13] G. Banga, J. C. Mogul, and P. Druschel. Better operating system features for faster network servers. In *WISP*, 1998.
[14] M. Boland. Mathopd. http://www.mathopd.org.
[15] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. In *Allerton*, 2002.
[16] CERT. CERT Advisory CA-1998-01 Smurf IP Denial-of-Service Attacks, 1998. http://www.cert.org/advisories/CA-1998-01.html.
[17] CERT. CERT Advisory CA-2003-20 W32/Blaster worm, 2003.
[18] CERT. CERT Incident Note IN-2004-01 W32/Novarg.A Virus, 2004.
[19] A. Chandra and P. Shenoy. Effectiveness of dynamic resource allocation for handling Internet. University of Massachussets, TR03-37, 2003.
[20] L. Cherkasova and P. Phaal. Session based admission control: A mechanism for improving the performance of an overloaded web server. HP-Labs, HPL-98-119, 1998.
[21] A. Coates, H. Baird, and R. Fateman. Pessimal print: A Reverse Turing Test. In *IAPR*, 1999.
[22] T. M. Gil and M. Poletto. MULTOPS: A Data-Structure for bandwidth attack detection. In *Usenix Security*, pages 23–38, 2001.
[23] E. Hellweg. When Bot Nets Attack. *MIT Technology Review*, September 2004. http://www.technologyreview.com/articles/04/09/wo_hellweg092404.asp.
[24] R. Iyer, V. Tewari, and K. Kant. Overload control mechanisms for Web servers. In *Workshop on Perf. and QoS of Next Gen. Networks*, 2000.
[25] H. Jamjoom and K. G. Shin. Persistent dropping: An efficient control of traffic. In *ACM SIGCOMM*, 2003.
[26] A. Juels and J. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *NDSS*, 1999.
[27] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites. In *WWW Conf.*, 2002.
[28] A. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure Overlay Services. In *ACM SIGCOMM*, 2002.
[29] G. Kochanski, D. Lopresti, and C. Shih. A Reverse Turing Test using speech. In *ICSLP*, 2002.
[30] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM TOCS*, 2000.
[31] C. Kruegel and G. Vigna. Anomaly detection of web-based attacks. In *ACM CCS*, 2003.
[32] J. Leyden. East European gangs in online protection racket, 2003. http://www.theregister.co.uk/2003/11/12/east_european_gangs_in_online/.
[33] J. Leyden. WorldPay recovers from massive attack, Nov. 2003. http://www.theregister.co.uk/2003/11/11/worldpay_recovers_from_massive_attack/.
[34] J. Leyden. DDoSers attack DoubleClick, 2004. http://www.theregister.co.uk/2004/07/28/ddosers_attack_doubleclick/.
[35] J. Leyden. The illicit trade in compromised PCs, 2004. http://www.theregister.co.uk/2004/04/30/spam_biz/.
[36] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *Computer Communications Review*, 32(3):62–73, July 2002.
[37] D. Mazieres. A toolkit for user-level file systems. In *USENIX Tech. Conf.*, 2001.
[38] S. McCanne. The Berkeley Packet Filter Man page, May 1991. BPF distribution available at ftp://ftp.ee.lbl.gov.
[39] J. Mogul and K. K. Ramakrishnan. Eliminating receive livelock in an interrupt-driven kernel. In *USENIX Tech. Conf.*, 1996.
[40] D. Moore, G. Voelker, and S. Savage. Inferring Internet Denial-of-Service activity. In *USENIX Security*, 2001.
[41] W. G. Morein, A. Stavrou, C. Cook, A. D. Keromytis, V. Misra, and R. Rubenstein. Using graphic Turing Tests to counter automated DDoS attacks against web servers. In *ACM CCS*, 2003.
[42] G. Mori and J. Malik. Recognizing objects in adversarial clutter: Breaking a visual captcha. In *CVPR*, 2003.
[43] V. Paxson. An analysis of using reflectors for distributed Denial-of-Service attacks. *CCR*, 2001.
[44] K. Poulsen. FBI busts alleged DDoS mafia, 2004. http://www.securityfocus.com/news/9411.
[45] L. Ricciulli, P. Lincoln, and P. Kakkar. TCP SYN flooding defense.
[46] Y. Rui and Z. Liu. ARTiFACIAL: Automated Reverse Turing Test using FACIAL features. In *ACM Multimedia*, 2003.
[47] R. Russell. Linux IP Firewall Chains.
[48] S. Savage, D. Wetherall, A. R. Karlin, and T. Anderson. Practical network support for IP traceback. In *ACM SIGCOMM*, 2000.
[49] D. X. Song and A. Perrig. Advanced and authenticated marking schemes for IP traceback. In *IEEE INFOCOM*, 2001.
[50] T. Stading, P. Maniatis, and M. Baker. Peer-to-peer caching schemes to address flash crowds. In *IPTPS*, 2002.
[51] S. Staniford, V. Paxson, and N. Weaver. How to 0wn the Internet in your spare time. In *USENIX Security*, 2002.
[52] A. Stavrou, D. Rubenstein, and S. Sahu. A lightweight, robust, P2P system to handle Flash Crowds. *IEEE JSAC*, 2004.
[53] L. Taylor. Botnets and Botherds.
[54] T. Voigt and P. Gunningberg. Handling multiple bottlenecks in web servers using adaptive inbound controls. In *Prot. f. High-Speed Networks*, pages 50–68, 2002.
[55] L. von Ahn, M. Blum, N. Hopper, and J. Langford. Captcha: Using hard ai problems for security. In *EUROCRYPT*, 2003.
[56] M. Welsh and D. Culler. Adaptive overload control for busy internet servers. In *USITS*, 2003.
[57] M. Welsh, D. Culler, and E. Brewer. SEDA: an architecture for well-conditioned, scalable internet services. In *ACM SOSP*, 2001.
[58] A. Yaar, A. Perrig, and D. Song. Pi: A path identification mechanism to defend against DDoS attacks. In *IEEE Security and Privacy*, 2003.
[59] Checkpoint. www.checkpoint.com/products/downloads/applicationintelligence_whitepaper.pdf.