# Guaranteeing Access in Spite of Distributed Service-Flooding Attacks

Virgil D. Gligor

VDG Inc., 6009 Brookside Drive, Chevy Chase, Maryland 20815*

**Abstract.** We argue that open networks designed using end-to-end arguments are particularly vulnerable to flooding, and that this vulnerability persists as hardware and operating systems technologies advance. An effective end-to-end approach to counter distributed flooding attacks against public services and provide access guarantees to their clients is to establish and enforce *"user agreements"* among clients outside the public services they access. Among the user agreements designed to protect servers from flooding attacks, those requiring client proofs of work (e.g., client puzzles using hash functions) are both ineffective and unnecessary whenever strong access guarantees are desired. In contrast, simple rate-control agreements can be defined to provide strong guarantees based on waiting-time limits. These agreements are established by special-purpose servers and verified before request processing at network-line rate, and hence cannot be flooded.

## 1 Introduction

Imagine that all distributed denial of service (DDoS) attacks at and below the transport layer of an open network, such as the Internet, are handled, and that all protocol and server flaws/features that enable DDoS attacks at the application layers are removed. In this ideal setting, flooding attacks against the publicly accessible services remain a significant and persistent concern. Vulnerability to flooding arises because request rates of network access points (e.g., the peak network "line" rate) exceed the throughput (e.g., the peak request rate) of publicly accessible application servers to which they are attached, often by more than an order of magnitude. For example, analysis of Internet flooding events shows that peak line rates can exceed 600K packets per second [19], yet specialized firewalls designed to protect servers from TCP SYN flood attacks can be disabled by line rates as low as 14K packets per second [5]. Simulations show that even a highly replicated content-distribution service (i.e., 64 servers) designed to withstand flooding attacks reaches less than 40K requests per second, with the average request of 6KB [22] fitting in a single packet. Hence, clients' aggregate request rates to application servers could legitimately reach much higher levels than application servers could tolerate and yet *not* trigger any network-layer alarms. For

* Permanent address: Department of Electrical and Computer Engineering, University of Maryland, College Park, Maryland, 20742. Email: gligor@umd.edu

example, during the much-publicized flooding attacks of February 2000 against public services of Yahoo!, Ebay, and E*trade, and against Microsoft's name servers in January 2001, Internet service providers (ISPs) did not notice any unusual network traffic in most attack instances. This clearly shows that flooding attacks are an *end-to-end problem* that requires *end-to-end solutions* no matter how effective IP-layer flooding countermeasures might be.

In this paper, we argue that open networks designed using end-to-end arguments [21] are particularly vulnerable to flooding, and that this vulnerability persists as hardware and operating systems technologies advance. We present an end-to-end approach aimed at providing access guarantees to clients, in spite of an adversary's attempts to flood public servers with requests from rogue programs running on adversary-controlled client computers. We review several types of access guarantees based on waiting-time limits (e.g., maximum [8, 9], finite [25], and probabilistic [18] waiting-time limits), explain their relationships, and show how to achieve some of the strongest of these based on the notion of *"user agreements"* [25]. Simple agreements are generated and enforced on dedicated special-purpose servers that cannot be flooded as they are designed to operate at the peak line rates reached at their network-access points. We strengthen these guarantees by controlling client proliferation on adversary-captured machines with either reverse Turing tests (e.g., CAPTCHAs) [24, 1] or with remote verification of human-activated, hardware-implemented, trusted path on TCPA-equipped client machines [20].

Although user agreements are necessary for preventing end-to-end flooding attacks, some agreements cannot offer strong client guarantees. We argue that agreements requiring client proofs of work (i.e., client puzzles using hash functions) are both ineffective and unnecessary. They impose a high overhead on legitimate client requests and only offer very weak guarantees, yet other user agreements exist that provide same guarantees, or stronger, at the same or lower request overhead. We illustrate this observation with simple examples.

Our approach to countering distributed flooding attacks does not require user registration or state maintenance. However, it does rely on routers to perform the typical *ingress filtering* at the network edges already required for other reasons (e.g., prevent IP address spoofing by virus propagation mechanisms). We do not aim to handle flooding attacks within network layers and assume that these are handled by separate protocols [14].

## 2 End-to-End Flooding and Countermeasures

> *We therefore can't just jump from the observation that a resource is held "in common" to the conclusion that "freedom in a commons brings ruin to all."*
>
> Lawrence Lessig [17]

Open networks offer public services for various applications and, at the same time, rely on public services themselves for infrastructure functions, such as security (e.g., user registration, authentication), naming (e.g., DNS), and user-level communications (e.g., mail servers). Thus, in principle, all open networks

are vulnerable to flooding. However, open networks that are designed using the end-to-end argument, such as the Internet, are particularly vulnerable. The persistence, if not the existence, of the gap between network-line and application-server rates is a consequence of applying the end-to-end argument to network design and, if anything, the gap will grow larger in the future.

The end-to-end argument suggests that network computers (e.g., routers) perform very simple functions that are common to all applications, while complex functions required by fewer applications be implemented in end-servers. This is arguably the single most important technical idea for assuring network performance on large scale since it makes the "common case" fast (viz., Amdahl's law [13]). It is also the single most important reason for the gap between the network-line rates and application-server rates: common, simple network functions are made fast whereas less common, complex functions that would slow down the network are pushed to end-servers. The intended consequence is that network-line rates are at least as high as the request rates of the vast majority of the end-servers. The unintended consequence is that the rate gap becomes pervasive and enables flooding attacks that cannot be detected and handled within the network.

*Persistent Vulnerability.* Simplicity of network functions translates ever-increasing hardware performance into higher network-line rates almost directly. In contrast, similar performance increases are quickly offset by increased complexity of new end-server applications and operating system features.[1] Thus, there is no reason to anticipate that future technologies will close the rate gap. On the contrary, there is reason to believe that the rate gap will grow larger in the future. The typical rate gap of the early ARPA network was smaller than that witnessed in the Internet today.

End-to-end arguments also rule out the closing of the rate gap artificially by slowing network-access points to match application-server rates. This would only shift the flooding vulnerability from servers to network-access points where there is less freedom, and thus harder, to remove it. Attempts to push back traffic along the multiple routes terminating at an access point of the Internet would likely encounter ISPs that could justifiably refuse to allow external pushbacks into their networks for security or policy reasons (e.g., the external pushback request may appear to be a disguised attempt to sabotage their local network traffic by a competitor, insufficient selectivity of, or justification for, the pushback request) [14]. Further, the alternative of closing the rate gap globally in an adaptive manner would be ill-advised, as it would require solving a complex optimization problem continuously on a global scale – thereby penalizing the performance of

[1] Complexity of applications is not the only reason why cycles of improved hardware performance vanish rapidly in end-servers. Lampson's explanation [16] of where orders of magnitude in improved hardware performance went in PCs also applies here; i.e., hardware resources traded for quick time to market by software developers, lots of features delivered quickly lacking first-class design, integration of different features, compatibility with old hardware and systems, and faster response to user's actions.

common functions – for the purpose of eliminating a relatively infrequent event [19] that could be handled, whenever required, end-to-end.

*Attacks.* The rate gap is a cause of undesirable dependencies among clients of shared services. Since all shared services have finite resources (e.g., finite request queue), whether a client's request receives a (timely) response depends on other clients' behavior in using that service. However, the rate gap allows a group of otherwise-legitimate client programs running on a large number of adversary-controlled hosts to prevent other legitimate clients from accessing an application server for a long time, and thus service is being denied [8, 9]. The detrimental nature of undesirable dependencies to service availability has a direct analog in economics, and thus is not a mere artifact of technology. A public service is a "rivalrous" resource since its use by a client competes with that of others; and, if use of a rivalrous resource is open to all, the resource may be depleted by the consumption of all.[2][17]. This is known as *the tragedy of commons* (i.e., "freedom in a commons brings ruin to all" [12]).

*End-to-End Countermeasures.* In computing, the remedy proposed to counter undesirable service-access dependencies, and hence to avert the tragedy of commons, has been the establishment and enforcement of *"user agreements"* among clients outside services they share [25, 18]. User agreements are constraints placed on the behavior of service clients to counter undesirable dependencies and help guarantee client access. For example, constraints could require that clients solve a cryptographic puzzle before server access, which would act as a pricing mechanism for server access and could prevent flooding; e.g., "junk mail" flooding [7]. This idea has been used to counter connection-depletion attacks [15, 23] and flooding of authentication servers [2], such as that of TLS [6]. The role of user agreements in public service access seems to have also been observed by economists in many different contexts and for much the same reason; i.e., communities regulate over-consumption of rivalrous resources by establishing *"norms" of user behavior* [4]. Hence, independent motivation for user agreements indicates that they can lead to sound end-to-end solutions to flooding problems.

## 3 Client Guarantees and Attempts to Provide Them

*We cannot enter into alliance with neighboring princes unless we are acquainted with their designs.*
Sun Tzu, *The Art of War*

General questions regarding user agreements include these: (1) given a specification of request scheduling in a public service, what is the weakest user agreement that supports a desired guarantee of client access?, and conversely (2) if the scheduling specification could be changed, what change would assure a desired

[2] The fact that a service or resource is publicly accessible, or open to all, does *not* imply that it is incorrectly priced. It only means that the pricing model is different from the access model.

guarantee for a given user agreement? Answers to both these questions require that client guarantees be defined.

Access guarantees that specify upper bounds on waiting times are intuitively appealing because they help establish performance bounds for network applications. A wide variety of such guarantees can be defined according to their (1) scope (e.g., per request, per service), (2) type of upper bounds (e.g., maximum, finite, and probabilistic waiting time), and (3) quality of these upper bounds (e.g., constant, variable, variable-dependent or -independent of the attack). Guarantees that do not necessarily specify upper bounds of waiting times, such as those based on average waiting times, are also helpful particularly when they are used to refine the upper-bound guarantees. For example, a strong waiting-time guarantee may have a constant upper bound and yet have an average waiting time that is higher than that of a weaker guarantee that has only a variable upper bound.

Guarantees that specify bounded waiting time on a per-request basis are of practical interest for open networks. Per-service guarantees of similar type and bound quality are stronger, but some may be impractical for open networks. For example, per-service maximum waiting-time guarantees with tight constant bounds, which are useful in real-time systems, typically require special-purpose networks with limited client populations. In contrast, guarantees that do not specify bounded waiting times can be very weak. For example, a guarantee which specifies that a "server will not crash in a flooding attack" is implied by all waiting-time guarantees, and is extremely weak. Clients' requests may be denied access to a service in the absence of server crashes, and the majority of requests that are accepted may be those of adversaries' clients.

### 3.1 Definitions

The principal parameters used in the per-request, waiting-time guarantees are: (1) $S$ = a server's maximum request rate; (2) $L$ = maximum number of entries in a server's request queue; (3) $\tau$ = length of time a queued request has to wait until the server processes it, and thus $\tau \leq L/S$; (4) $\Delta$ = propagation delay incurred by a client request to reach a server, and is bounded by our assumption that the network layer is denial-of-service free; (5) $T$ = waiting-time bound specified at the time of the request, and thus $T \geq \Delta + \tau$; (6) $N$ = the maximum network-line rate, and $N - S > 0$ is the rate gap. Using these parameters, we specify the following types of *per-request*, waiting-time guarantees, of which the first three have been defined in prior work [8, 25, 18]:

*1. Maximum Waiting Time (MWT).* A client's request is accepted for service in time $T$.

*2. Finite Waiting Time (FWT).* A client's request is accepted for service *eventually*.

In practice, this specification means that there is a finite number of clients whose request can be accepted for before any other client's request. Thus, if a client has to retry its request, then the number of retries is finite.

*3. Probabilistic Waiting Time (PWT).* The probability that a client's request is accepted for service in time $T$ is not less than $\rho$, where $\rho \neq 0$ is independent of the flooding attack.

The attack-independence of the lower bound $\rho$ means that the expression for $\rho$ does not include any parameters of, or dependent on, the adversary's attack; e.g., the number of adversary's clients launching the attack, their individual and aggregate request rates, the number of request retries forced by the adversary on a legitimate client's request. Nevertheless, $\rho$ may be a variable that depends on network parameters, such as those of the rate gap, $S$ and $N$. PWT guarantees can be strengthened by asking that $\rho$ be a constant, or even one [18]. It can also be weakened, by allowing a different non-zero lower bound $p$ that may depend on flooding attack parameters.

*4. Weak Probabilistic Waiting Time (wPWT).* The probability that a client's request is accepted for service *eventually* is not less than $p$, where $p \neq 0$.

Although $wPWT$ is a very weak guarantee, it is not the weakest. The weakest non-trivial guarantee, denoted by $WPWT$ below, is one in which the lower bound $p$ of $wPWT$ could be zero. This guarantee is meaningful and not unusual. It refers to cases when an adversary can deny service to some class of client requests but not others. It means that "some client requests will be accepted" by the server, and assures that during the "up periods" of a server, at least one client request completes service.

With appropriate experiments defined, one can obtain slightly weaker v (i.e., probabilistic) variants of the $MWT$ and $FWT$ guarantees that help establish the following relationships between the above guarantees:
$MWT \Rightarrow PWT \Rightarrow wPWT \Rightarrow WPWT$,
$MWT \Rightarrow FWT \Rightarrow wPWT \Rightarrow WPWT$, and
$FWT \not\Leftrightarrow PWT$,
where the relationship "$A \Rightarrow B$" means that if guarantee A is satisfied so is B. Using the relationship " $\Rightarrow$ " one can extend the simple lattice above with the new top being the per-service $MWT$ with the smallest practical $T$.

A variety of other guarantees beyond those considered in this section may be (more) useful. As in the case with *most* results based on bounds, care must be exercised in determining the "strength of guarantees." The intent of the partial order above is to illustrate one such notion of strength.

### 3.2 Guarantees of Client Puzzles Based on Hash Functions

Client puzzles require that each client solves a challenge puzzle as proof of work along with each request to a server. Only after receiving the proof of work the server decides whether to consider a clients' request for processing. The level of strength $k$ of a challenge puzzle is either determined by the server or chosen by the clients themselves. Servers that schedule requests on the basis of the strength of the puzzles solved – by preempting queued requests with lower puzzle levels when a new request arrives with a solved higher-level puzzle – give clients and incentive to select increasingly stronger puzzles in a bid to obtain service ahead of other clients (viz., the "puzzle auctions" [23]). The server scheduler checks

all puzzle solutions supplied by clients' requests, at the network-line rate. All client's requests that solved the server-specified puzzle incorrectly, or not at all, are dropped. If the clients' aggregate request rate is still high at puzzle level $k$, the server drops the extra requests and either expects the clients whose requests are dropped to bid higher-strength puzzles or specifies a higher-strength challenge puzzle, for example $k+1$, itself.

Typical puzzles use cryptographic hash functions, which assures that the cost of a puzzle solution to a client is exponential in $k$, where $k$ is much smaller than the length of the hash function output; i.e., the range of $k$ is between 1 and 64 bits whereas the hash function output is between 128 and 160 bits. Finding a hash function output that has $k$ consecutive zeros in the high-order bits is a typical client puzzle [2], and attempts to solve such puzzles can be viewed as Bernoulli experiments that have probability $2^{-k}$ of success [23].

An advantage of user agreements based on hash-function puzzles is that they can be *stateless* [2]. Additional properties of client puzzles using hash functions are defined below in terms of the guarantee parameters defined in the previous section, and (1) $N_Z^{k_r}$ = the aggregate request rate at which $Z$ clients solve puzzles of strength $k_r$; (2) $s$ = the rate at which clients can execute hash function operations, (3) $t_L^r$ ($t_Z^r$) = the time necessary for $Z$ clients to solve at least $L$ ($Z$) puzzle at level $k_r$; and (4) $c_r$ = additional time buffer or, equivalently, space $Sc_r$, beyond the server's processing time, $\tau$, and queuing, $S\tau$, needs.

*Time Buffer.* The additional time buffer, $c_r$, includes three components. The first is $t_Z^r - t_L^r$. The second, denoted by $c^N$, accounts for any possible randomization of request routes from clients and servers in the network, and for other random delays that might be introduced by the network layer to help slow the coordination of an adversary's clients. The third, $L/S - \tau$, represents the buffer provided by additional storage for requests explicitly provided by the server. Thus the total additional time buffer $c_r = t_Z^r - t_L^r + c^N + L/S - \tau$. The component $c^N + L/S - \tau > 0$ enables the use of lower puzzle levels than otherwise might be necessary to control the aggregate request rate and avoid flooding.

*Proposition 1 (Solution Latency for Level-$k_r$ Puzzles):*
*With high probability,*
*a) $Z \geq 2L + 2\sqrt{6L+9} + 6$ clients solve at least $L$ puzzles at level $k_r$ in $2^{k_r-1}Z$ steps and time $t_L^r = 2^{k_r-1}/s$; and*
*b) $Z$ clients solve at least $Z$ puzzles with level $k_r$ in $2^{k_r+1}Z$ steps and time $t_Z^r = 2^{k_r+1}/s$.*
*Proof:*

a) We find the bound of the desired probability as follows:
$Pr[$ Z clients solve at least $L$ puzzles at level $k_r$ in $2^{k_r-1}Z$ steps]
$= 1 - Pr[$ Z clients solve fewer than $L$ puzzles at level $k_r$ in $2^{k_r-1}Z$ steps]
$\geq 1 - Pr[$ Z clients solve at most $L$ puzzles at level $k_r$ in $2^{k_r-1}Z$ steps]
However, $Pr[$ Z clients solve at most $L$ puzzles at level $k_r$ in $2^{k_r-1}Z$ steps]
$\leq exp(-x)$, where $x = \frac{(Z/2-L)^2}{Z}$. The proof of this fact is omitted as it is similar to that of Proposition 2 in [23]. It is obtained using the variant of the *Chernoff/Hoeffding Bound* [11], which states that $Pr[E[X] - X \geq \epsilon E[X]] \leq$

$exp(-\epsilon^2 E[X]/2)$, where $0 < \epsilon < 1$, by requiring that $0 < \epsilon \leq 1 - 2L/Z$. Thus, $Pr[$ Z clients solve at least $L$ puzzles at level $k_r$ in $2^{k_r-1}Z$ steps$] \geq 1 - exp(-x)$. $Z \geq 2L + 2\sqrt{6L+9} + 6$ implies that $x \geq 3$, which means that the lower bound of this probability is at least 0.95.

b) This follows along the same lines as part (a).
$Pr[$ Z clients solve at least $Z$ puzzles at level $k_r$ in $2^{k_r+1}Z$ steps$]$
$= 1 - Pr[$ Z clients solve fewer than $Z$ puzzles at level $k_r$ in $2^{k_r+1}Z$ steps$]$
$= 1 - Pr[$ Z clients solve at most $Z-1$ puzzles at level $k_r$ in $2^{k_r+1}Z$ steps$]$
However, $Pr[$ Z clients solve at most $Z-1$ puzzles at level $k_r$ in $2^{k_r+1}Z$ steps$]$
$\leq exp(-y)$, where $y = \frac{(Z+1)^2}{4Z}$. Thus,
$Pr[$ Z clients solve at least $Z$ puzzles at level $k_r$ in $2^{k_r+1}Z$ steps$] \geq 1 - exp(-y)$.

*Proposition 2 (Request-Rate Control): Let the time buffer $c_r$ be such that $3\frac{2^{k_r-1}}{s} \leq c_r < Z/S$. Then with high probability, $N_Z^{k_r} \leq S \Leftrightarrow k_r \geq 1 + \lceil \log(\frac{Z}{S} - c_r)s \rceil$ in a time interval of length $t_L^r + c_r$.*
*Proof:*
By Proposition 1, with high probability, $Z$ clients solve at least $Z$ puzzles in time $t_L^r + c_r$, where $c_r \geq t_Z^r - t_L^r \geq 3\frac{2^{k_r-1}}{s}$. Also, by definition, $N_Z^{k_r} \geq Z/(t_L^r + c_r) = Z/(2^{k_r-1}/s + c_r)$. Thus, $S \geq N_Z^{k_r} \geq Z/(2^{k_r-1}/s + c_r) \Leftrightarrow k_r \geq 1 + \lceil \log(\frac{Z}{S} - c_r)s \rceil$.

Proposition 1 gives a lower bound for the additional buffer time $c_r$ whereas Proposition 2 gives an upper bound for $c_r$ and a lower bound for $k_r$. With high probability, $c_r < Z/S$ for *any* puzzle scheme based on hash functions. This bound holds for all puzzle levels up to and including $k_r$, at which point $N_Z^{k_r} \leq S$. Were $c_r \geq Z/S$ for some $i \leq r$, then client puzzles would become unnecessary at level $k_r$ since the aggregate request rate would already be at or below the server rate; i.e., $N_Z^{k_r} \leq S$.

*$WPWT$ Guarantees.* Proposition 2 shows that user agreements based on client puzzles using cryptographic hash functions can provide the weakest guarantees, even when extra time buffer or storage is unavailable (i.e., when $c_r$ reaches its lower bound). If a server that keeps dropping and re-challenging clients' requests with higher-difficulty puzzles, or if the clients' bid for access with increasingly higher-difficulty puzzles, the server will not crash and thus *some* clients' requests will be serviced. This seems to have been the goal of most prior work in this area [15, 2, 6].

*$wPWT$ Guarantees.* Wang and Reiter [23] provide an interesting model for analyzing "puzzle auctions" and show that an upper bound on the probability of denied access at the $r$-th retry, denoted by $Pr[\xi_r]$ below, can be found if one makes an approximation assumption on the size of $L$. The bound is:
$Pr[\xi_r] < (1 - 2^{-k_0})^{\frac{2^{k_0-1}L}{Z} - s\tau} \prod_{i=1}^{r} (1 - 2^{-k_i})^{\frac{(2^{k_i-1} - 2^{k_{i-1}-1})L}{Z}} \neq 0$,
where $k_0, \cdots, k_r$ is a sequence of puzzle levels a client would have to solve and bid to get access to a server in $r+1$ rounds (i.e., $r$ retries after the first attempt). A $wPWT$ guarantee is provided, but not a $PWT$ guarantee. That is, for any number of retries $R$ that is fixed at the time of the request,

$Pr[$ any client C's request is accepted in time T $] \geq$
$1 - Pr[$ any client C's request is denied after $R$ retries $]$
is dependent on the attack (i.e., on the number of clients $Z$ controlled by the adversary).

The approximation assumption says that if $L$ *is long enough*, the adversary's probability of solving $L$ puzzles at level $k$ within $2^{k-1}L$ steps can be ignored. How long is long enough ? The variant of the Chernoff/Hoeffding bound used in Wang and Reiter's Proposition 2, requires that $L$ has a strict lower bound in terms of the number of adversary's clients, $Z$, namely $L > Z/2$.

### 3.3 Limitations of Client Puzzles based on Hash Functions

In this section we argue that user agreements based on client puzzles implemented with hash functions are ineffective in the role of user agreements for preventing DDoS attacks, as they combine weak service-access guarantees with high request overhead.

*Attack Coordination.* Proposition 2 provides a lower bound for the puzzle level $k_r$ such that $N_Z^{k_i} \leq S$. Hence, for all $k_i \geq k_r, i \geq r$, an adversary has little or no chance to deny access to *all* legitimate clients. However, Proposition 1 implies that, after solving at least $L$ puzzles (i.e., after $t_L^i$), the adversary can coordinate its clients to issue their requests at rate $N_Z^{k_i}$ over an interval of length $\delta$ at every puzzle level, where $L/N_Z^{k_i} < \delta < Z/S$, such that $N_Z^{k_i} > S$, even when $L > \frac{Z}{2}$. The goal of such an attack would be to ensure that in these intervals *at least* one client's request and retries will be denied access at successive puzzle levels $k_i$ repeatedly with non-zero probability. These attacks would deny strong access guarantees, and yet would be undetectable at the network layer, since $N_Z^{k_i} \leq N$.

*Weak Guarantees.* When a coordinated attack is in progress, (1) the ability to distinguish among client requests based on puzzle levels is lost, since all requests have the same level $k_i$, and (2) a legitimate client's best chance of access is at the highest puzzle level $k_m \geq k_r$ in an interval of length $\frac{2^{k_m+1}}{s} - \delta$. Let $p_m = \max(p_i), i = r, \cdots, m$ denote the probability of client access in this interval, for any request scheduling discipline based on puzzle levels. By assuming that a client has this chance at all puzzle levels $k_i$, $k_r \leq k_i < k_m$, we obtain an upper bound on the probability of a client's access within $m$ retries, denoted by $Pr[\gamma_m]$, as follows.
$Pr[\gamma_m] < p_m + (1-p_m)p_m + (1-p_m)^2 p_m + \cdots + (1-p_m)^{m-r}$
$= p_m \sum_{i=0}^{m-r}(1-p_m)^i = 1 - (1-p_m)^{m-r+1} < 1$, for any finite $m \geq r > 0$.
This indicates that a $FWT$ guarantee cannot be supported. In a similar manner, one can obtain a non-zero lower bound for the probability of client access that would also depend on attack parameters – just as in the example of puzzle auctions above – indicating that a $PWT$ guarantee cannot be supported.

Attempts to counter coordinated attacks by adding more constraints on the number of adversary's clients, $Z$, and $N_Z^{k_i}$ would be impractical. Both parameters may change dynamically and may not have a useful upper bound. In fact, any

bound on $Z$, and hence on $N_Z^{k_i} < N$, would be impractical in open networks. Since end-to-end solutions cannot distinguish legitimate requests from those of an adversary's clients, a bound on the *total number* of clients would have to be imposed. This would be incongruent with the very notions of open networks and public services. Further, if different scheduling policies, independent of puzzle levels, would prove effective for providing $PWT$ and $FWT$ guarantees, they would be effective stand-alone, and would not need client puzzles, as illustrated in the next section.

*High Request Overhead.* As an example of request overhead, we use the five-bid sequence $k_1 = 15, \cdots, k_5 = 19$ of a puzzle auction assuming that that the rate of hash function operations is $s = 10^6$/second [23]. Assuming a round-trip message delay of 140 milliseconds, which is typical in the Internet, this five-bid sequence takes 1.72 seconds. However, fairly typical Web servers have a response time to client requests ranging from 1 second down to to 100 ms. Thus, the overhead of this sequence would range from over 170% to over $1,000$%, for every client request, legitimate or not. This overhead would only be cut in half if clients start bidding at the top of the sequence, namely at $k_5 = 19$. The above sequence would also impose much higher overhead on specialized public servers of the network infrastructure that are designed to operate a higher speeds. For instance, the response time of a TLS/SSL server under load is only about 16 ms [6]. The request overhead of the five-bid puzzle sequence would exceed $1,000$% even if we could increase the rate of hash function operations five fold. Some clients, such as those running on low-power devices, would not be able to tolerate such overhead.

### 3.4 Simple User-Agreement and Service-Scheduling Specifications

In this section we give *very simple* user agreements and service scheduling specifications that can provide $wPWT$ guarantees at the same request overhead as, or lower than, that of puzzle auctions. We also give a very simple user agreement that provides a $PWT$ guarantee, although its lower bound is very low. In both examples, the probability bounds are independent of the number of adversary's clients, $Z$.

*User Agreement Specification.* Suppose that instead of solving any puzzle, a client simply agrees to resubmit a dropped request or retry. This is the weakest possible agreement in the sense that it does not place any constraint on clients' behavior. However, the adversary's desire to keep the aggregate rate of all its clients be under $N$ – to avoid tripping network-layer alarms – imposes implicit self-constraints for both the number of clients and their request/retry rates.

*1. Service Scheduling Specification: Random $L$ requests without preemption.* Suppose that the service scheduler buffers up to $N\tau > L$ requests, picks up to $L = S\tau$ of them at random, places them in a $L$-entry queue, and drops the rest. This is the weakest scheduling strategy without preemption in the sense that is does not depend on any request parameter. For these specifications, we obtain the following bound:

$Pr[$ any client C's request is accepted *eventually* $] \geq$
$Pr[$ any client C's request is accepted after $r$ retries $]$
$= 1 - Pr[$ any client C's request is denied after $r$ retries $] \geq 1 - [1 - \frac{S}{N}]^{1+r} \neq 0$

This bound depends only on gap rate parameters $S$ and $N$ and on the number of retries $r$. Just as with the client-puzzle bound when $p_m = S/N$, it approaches 1, but only asymptotically, as an attack forces $r \to \infty$. Hence, the guarantee provided is still only $wPWT$, instead of $FWT$.

*Relative Cost.* The cost of this user-agreement scheme relative to that of puzzle auctions depends both on the round-trip message delays and on the size of the rate gap, $S/N$. For instance, for a round-trip message delay of 140 milliseconds, and for a $S/N$ ratio that can be as low as 1/8, this type of user agreement offers about the same guarantees as the five-bid puzzle auctions above for about the same cost. That is, for $r + 1 = 12$ the probability of *denied* access is 0.2248, and the total cost is 1.68 seconds. For the same probability of denied access and $L = Z = 1024$, $\tau = 250$ microseconds, the cost of the five-bid puzzle sequence $k_1 = 15, \cdots, k_5 = 19$, is 1.72 seconds. On the other hand, if the rate-gap ratio $S/N$ is somewhat higher but still a very low 1/5, then for the same probability of denied access, the number of round trips drops to $r+1 = 7$, and the total cost would be only 0.98 seconds. In contrast, even if we double the clients' speed, $s$, the cost of the five-bid sequence would be higher, namely 1.2 seconds. Finally, if a message round-trip time decreases to 15 milliseconds, which would not be unusual in a 50 Mbps network, the cost per request for $S/N = 1/8$ and $S/N = 1/5$ would be 0.18 and 0.10 seconds, respectively, yet the cost of the five-bid sequence would only drop to 0.5 seconds even when $s$ doubles.

*2. Service Scheduling Specification: Random L requests with preemption.*[3] If request buffering is unavailable, the server schedules any new request as follows. If the server's $L$-entry queue is not full, the server queues the new client's request. Otherwise, it picks a uniformly distributed random number in the interval $[0, L]$. If the picked number is zero, the server drops the new request and asks the client to retry. If not, it frees a queue slot by dropping the client's request already placed in the scheduler queue's at the entry indexed by the (non-zero) random number (i.e., it preempts a request), and asks that client to retry. Then, it places the new client's request in the queue slot just freed. For this type of user agreements we have a $PWT$ guarantee. That is

$Pr[$ any client C's request is accepted for service in time T $\geq \Delta + \tau]$
$\geq Pr[$ any client C's request is queued in time $\Delta]$
$\times Pr[$ client C's request is not preempted by $n$ new requests in time $\tau]$
$= [1 - \frac{1}{L+1}][\frac{1}{L+1} + \frac{L-1}{L+1}]^n = [\frac{L}{L+1}]^{1+n} \geq [\frac{S\tau}{S\tau+1}]^{1+N\tau} \neq 0$

which depends only on the rate-gap parameters $S, N$ and thus is independent of any attack parameter. Hence, a PWT guarantee can be satisfied, although the non-zero lower bound is very small; e.g., much smaller than the bound of scheduling random $L$ requests without preemption above.

We note that the scheduling of random $L$ requests with preemption could be added to puzzle auctions (1) to remove the approximation assumption, and (2)

[3] courtesy of Vijay G. Bharadwaj

to support a $PWT$ guarantee. The high per-request overhead of client puzzles would still remain a significant concern.

## 4 Simple Rate-Control Agreements for MWT Guarantees: An Example

In our approach, a *rate-control service (RCS)* is associated with each application service. RCS ensures that the aggregate rate of all clients, both legitimate and adversary's, does not exceed the maximum rate of the application service $L/\tau$ in any time interval $\tau$ or larger. To accomplish this, RCS uses an access reservation policy whereby each client is allowed one or more server accesses in a reserved time window. (Of course, other rate-control policies can be used to accomplished this goal [3].) The user agreement requires that each client obtains an access ticket from RCS and places its server request within the time window specified by RCS in the ticket. For services that require multiple accesses within a well-defined protocol, such as those for authentication, naming, and mail, RCS authorizes multiple accesses with a single time window. However, the number of accesses may be lower than that required by specific protocols to prevent degradation of service performance by the adversary's clients obtaining but not using request tickets – a potential hazard with all time-window reservation policies. For services that are accessed at the client's discretion, RCS's default is a single access within a time window. In either case, per-request MWT guarantees are provided since each client request can count on accessing the server within the upper limit of the time window.

Each client request to a server must include the ticket authorizing access to the server. Ticket validity is checked by a *Verifier* located between the network access point and the application server. The Verifier checks that a client's request is authorized in the current time window, and if not, it drops the request. The Verifier and RCS are time synchronized within tolerances comparable to network delays, and share a symmetric key. RCS uses the key to generate a Message Authentication Code (MAC) for each ticket issued to a client, and the Verifier uses the key to check the authenticity of each client's request ticket. MAC generation and verification are the most time consuming operations RCS and the Verifier perform. However, since request tickets are short (e.g., under 1 KB) and these operations can be executed fully in parallel [10], RCS and the Verifier can execute their operations at rates that far exceed the network line rate, and hence cannot be flooded.

*Request Tickets.* A client request for an access ticket includes the number of accesses desired, the source IP address from which the requests will be issued, the start time of the window in which the request will be issued, $t_s$, and, if a multiple-access ticket is desired, the maximum interval between two client requests for that application, $\delta_r$. RCS verifies that the requested IP address is the source address of the requester's message, that the number of requested accesses and $\delta_r$ are consistent with the server-access protocol, and that $t_s$ is within the ticket postdating limit allowed. If these checks pass, RCS issues a ticket and its MAC.

The ticket contains (1) a start time, $t_i$; (2) an end time, $t_{i+1}$; (3) a count of the maximum number of accesses, $w$, where $1 \leq w \leq L/n$ and $n \geq 1$ is the number of clients that can access the server within the same window; (4) the source IP address for the request; and (5) the time of ticket issue, $t_{RCS}$. The window start time is set to $t_i = t_w + \Delta$, where $t_w > t_s$ is the first available time window when this client can issue its first request. The RCS time, $t_{RCS}$, allows the client to synchronize its request-issue time with the Verifier, $\Delta$ ensures that the ticket is valid upon client receipt, and $t_s$ that the client has time to issue the request. The window end time is set to $t_{i+1} = t_i + w(\tau + 2\Delta + \delta_r)$. This allows for the network delay $\Delta$ before the request reaches the Verifier, for request processing in $\tau$ units of time after receipt by the Verifier, and for ticket validity before the next access, $\Delta + \delta_r$. For tickets authorizing multiple accesses, the Verifier maintains a cache of tickets seen within the current time window and the number of accesses already exercised within that window for each ticket, to prevent ticket reuse by clients beyond $w$ times. Since the time windows are relatively small, the ticket cache will be held in primary memory and much, if not all, of it will fit in the processor cache.

Ticket validity checks by the Verifier require that requests must be placed within the allocated time window, must not exceed the allowed access count, $w$, must be issued from the IP address for which the ticket was issued, and must pass the MAC check. All these checks can be performed at a faster rate than that of all network access points. To prevent ticket theft and/or reuse from spoofed IP addresses, typical *ingress filtering* at network edges already required for other reasons (e.g., to prevent IP address spoofing by virus and worm propagation) must be performed at all clients' access points to the network.

*Transparent Operation.* The Verifier monitors the aggregate rate of clients' requests starting with the first possible client request (e.g., the client-server binding request). Whenever this rate exceeds a threshold, the Verifier enters the rate-control mode by directing each client to RCS. In rate-control mode, each client request must present a valid RCS-issued ticket. When rate-control mode can be exited, the Verifier returns an exit notification to the client along with the server's response. The client (i.e., the proxy code of the server running on the client machine) implements client-RCS and the client-Verifier request-response protocols, and thus these protocols are completely transparent to the client and server application code. However, an additional API is implemented by the client proxy so that the application can test whether the server operates in rate-control mode and retrieve the MWT. Hence, the client application can execute other concurrent threads while waiting for a server response.

*Session Cookies.* Although RCS and the Verifier can ensure that the aggregate request rate of all clients, legitimate or not, stays below the server's rate, an adversary can start a large number of clients on different machines, obtain valid tickets, and exercise them in an authorized manner. This would cause legitimate users' ticket start time and end times to be pushed into the future increasing the MWT for legitimate clients, possibly beyond reasonable values. To prevent uncontrolled client proliferation by an adversary and large MWT

values, we require that a client request for a ticket to RCS be accompanied by a cryptographic cookie attesting that the client had a human user behind it. The cookie can be obtained by the client in two ways: (1) after passing a reverse Turing test (or CAPTCHA [24, 1]), or (2) after verification of human-activated, hardware-implemented, trusted path on a TCPA-equipped client machine [20] that is registered with designated Internet servers. A cookie has a similar structure as a ticket, and includes: (1) a start time, (2) an end time, (3) a list of IP addresses from which ticket requests can be issued, (4) the time at the RCS, which issues the cookie, and (5) the MAC for the cookie. The time window in which the cookie is valid is comparable to that of a login session, and consequently the reverse Turing test or the verification of human-activated trusted path is required only once, upon session start by a human user. Also the cookie allows a limited number of clients to operate on behalf of the same human user from different IP addresses. Ingress filtering would also ensure that cookies cannot be passed around and used from addresses not included in the IP address list of the cookies.

*Performance Considerations.* The request overhead of this scheme is significantly lower than that of client puzzles. For instance, assuming that the typical round trip network delay is 140 milliseconds, the overhead for a request that otherwise would take one second is 14% instead of over 170% for the five-bid sequence of the puzzle auction mentioned in the previous section. The per-request overhead of a protocol requiring two to ten accesses each taking only 100 milliseconds would be between 14% and 70% when using a multi-access ticket, as opposed to over 1000% in the five-bid sequence of puzzle auctions.

For all applications, the question of how many accesses should be specified in a ticket arises naturally. Typical services that implement infrastructure protocols require between two and ten client accesses per application. In contrast, typical client server accesses for content distribution networks range from about six to sixty per application. If a single access is allowed, the communication cost for the client increases due to multiple RCS round trips for tickets. In contrast, if all accesses are included in single ticket windows, unused tickets by adversary's clients could decrease server utilization significantly due to reserved but unused windows. Hence, an optimal window size could be determined, at least theoretically, that trades off request overhead against server under-utilization.

Let $c_1$ be the unit cost of a round trip to RCS for ticket retrieval, $c_2$ be the unit cost of lost application server utilization caused by requests not issued by adversary's clients in a reserved time window, $A_r$ be the access count per application, and $l$ be the percentage of legitimate clients in the system, $0 \leq l < 1$. The optimal access count per window can be computed as a simple minimization of the total cost $C_{total} = C_{client} + C_{server} = c_1 A_r/w + c_2(1-l)w$. Setting $\frac{\delta C_{total}}{\delta w} = 0$, we obtain $w_{opt} = \sqrt{\frac{c_1 A_r}{c_2(1-l)}}$, where $1 \leq w_{opt} \leq \min(L, A_r)$. The corresponding time window size is $L/S \leq t_{opt} = w_{opt}/S \leq L/S_{min}$, where $S_{min}$ is the minimum request rate of the server. The unit costs $c_1$ and $c_2$ can be determined in terms of computation and communication latencies for a typical server and network. These costs must also take into account the communication

and computation costs relative to a particular application. Other considerations, such as the size of the Verifier cache, can also play a role in determining the optimal window size. We note that even unoptimized widow sizes would lead to significantly lower request overhead than any client puzzle mechanism.

## 5 Conclusions

The vulnerability of public services to flooding attacks in the Internet will continue to grow as the gap between network line rates and server rates continues to increase. Whether this vulnerability materializes as a significant service threat depends on whether new Internet services (e.g., voice over IP) will provide sufficiently attractive economic targets for attacks. Countering this vulnerability requires end-to-end solutions, yet those currently available based on client puzzles are inadequate as they offer very weak access guarantees at high per-request overhead. The solution presented in this paper is one of a class of user agreements that can be implemented without new network layer support, and thus offers practical advantages over current proposals.

Future research includes (1) the determination of rate-control parameters and policies for different types of Internet services, (2) the performance evaluation of different types of rate-control servers and request verifiers, (3) the experimental deployment of several rate-control servers and verifiers for different types of application services, (4) the investigation of MWT guarantees for client requests to multiple servers as might be necessary is distributed transactions over the Internet, and (5) the prevention of network-link flooding.

**Acknowledgments**

Discussions with V. Bharadwaj, R. Bobba, L. Eschenauer, O. Horvitz, R. Koleva, and H. Khurana over the past year helped sharpen many of the arguments presented in this paper. This work was supported by the Defense Advanced Research Projects Agency and managed by ITT Industries under the NRL contract N00173-00-C-2086. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency, NRL, ITT Industries, or the United States Government.

## References

1. L. von Ahn, M. Blum, N. Hopper, and J. Langford, "CAPTCHA: Using Hard AI Problems for Security," Advances in Cryptography - EUROCRYPT 2003, Warsaw, Poland, May 2003.
2. T. Aura, P. Nikander, and J. Leiwo, "DOS-Resistant Authentication with Client Puzzles," Proc. of the 8th Int'l Security Protocols Workshop, Cambridge, U.K., April 2000, LNCS vol. 2133, Springer Verlag, pp. 170-178.
3. D. Bertsekas and R. Gallager, *Data Networks*, second edition, Prentice Hall, 1992.
4. D. W. Bromley, *Making the Commons Work: Theory, Practice and Policy,* ICS Press, San Francisco, 1992 (Part 2, describing case studies; cf. [17], p.22, 272).

5. T. Darmohray and R. Oliver, "Hot Spares for DoS Attacks," *;login:*, Vol. 25, No. 7, July 2000.
6. D. Dean and A. Stubblefield, "Using Client Puzzles to Protect TLS," Proc. of the USENIX Security Symposium, August 2001.
7. C. Dwork and M. Naor, "Pricing via Processing or Combatting Junk Mail," Proc. of Advances in Cryptography: CRYPTO '92, LNCS Vol. 740, Springer Verlag, 1992.
8. V.D. Gligor, "A Note on the Denial-of-Service Problem," Proc. of the IEEE Symposium on Computer Security and Privacy, Oakland, California, April 1983. (also in *IEEE Transactions on Software Engineering*, SE-10, No. 3, May 1984.)
9. V.D. Gligor, "On Denial of Service in Computer Networks," Proc. of Int'l Conference on Data Engineering, Los Angeles, California, February 1986, pp. 608-617.
10. V.D. Gligor and P. Donescu, "Fast Encryption and Authentication: XCBC Encryption and XECB Authentication Modes," 8th International Workshop on Fast Sofware Encryption, FSE 2001, M. Matsui (ed.), Yokohama, Japan, April 2001, pp. 92-108.
11. T. Hagerup and C. Rub, "A Guided Tour of Chernoff Bounds," *Information Processing Letters*, vol. 33, 1989-90, North-Holland, pp. 305-308.
12. G. Hardin, "The Tragedy of Commons," *Science*, vol 162, (1968): 1243.
13. J. Henessy and D. Patterson, "Computer Architecture: A Quantitative Approach," Morgan-Kaufmann, 1990, pp. 8-9.
14. J. Ioannidis and S. Bellovin, "Implementing Pushback: Router-Based Defense Against DDoS Attacks," Proc. of Network and Distributed Systems Security Symposium," San Diego, California, Feb. 2002, pp. 79-86.
15. A. Juels and J. Brainard, "Client Puzzles: A Cryptographic Defense Against Connection Depletion Attacks," in Proc. of Network and Distributed Systems Symposium, San Diego, CA, Feb. 1999.
16. B. Lampson, "Software Components: Only Giants Survive," in *Computer Systems: Papers for Roger Needham*, A. Herbert and K. Sparck Jones (eds.), Microsoft Research, February 2003, pp. 113-120.
17. L. Lessig, *The Future of Ideas: The Fate of the Commons in a Connected World,* Random House, N.Y., 2001.
18. J. K. Millen, "A Resource Allocation Model for Denial of Service," Proc. of IEEE Symposium on Security and Privacy, Oakland, CA. 1992. (also in the *Journal of Computer Security,* vol. 2, 1993, pp. 89-106).
19. D. Moore, G. Voelker, and S. Savage, "Inferring Internet Denial of Service Activity," Proc. of 2001 USENIX Security Symposium, Washington D.C, August 2001.
20. *Trusted Computing Platforms – TCPA Technology in Context*, S. Pearson, B. Balacheff, L. Chen, D Palquin, G. Proudler (eds.), Prentice Hall PTR, 2003.
21. J.H. Saltzer, D.P. Reed, and D.D. Clark "End-to-End Arguments in System Design," *ACM Transactions on Computer Systems*, Vol.2, Nov. 1984.
22. L. Wang, V. Pai, and L. Petersen, "The Effectiveness of Request Redirection on CDN Robustness," Proc. of the 5th Symp. on OS Design and Implementation (OSDI), Boston, Mass. December 2002.
23. X. Wang and M. Reiter, "Defending Against Denial-of-Service Attacks with Puzzle Auctions," Proc. of IEEE Symp. on Security and Privacy, Berkeley, CA, May 2003.
24. J. Xu, R. Lipton, and I. Essa, "Hello, Are You Human," Technical Report, Georgia Institute of Technology, November 2000.
25. C.-F. Yu and V.D. Gligor, "A Formal Specification and Verification Method for Preventing Denial of Service Attacks," Proc. of the IEEE Security and Privacy Symposium, Oakland, CA., April 1988, pp. 187-2002 (also in *IEEE Transactions on Software Engineering,* vol. SE-16, June 1990, pp. 581-592).