This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2008 proceedings.

Adaptive Selective Verification

Sanjeev Khanna and **Santosh S. Venkatesh**, University of Pennsylvania
Omid Fatemieh, **Fariba Khan**, and **Carl A. Gunter**, University of Illinois Urbana-Champaign

Abstract—**We consider Denial of Service (DoS) attacks within the province of a shared channel model in which attack rates may be large but are bounded and client request rates vary within fixed bounds. In this setting it is shown that the clients can respond effectively to an attack by using bandwidth as a payment scheme and time-out windows to adaptively boost request rates. The server will be able to process client requests with high probability while pruning out most of the attack by selective random sampling. Our protocol, which we call *Adaptive Selective Verification (ASV)* is shown to be efficient in terms of bandwidth consumption using both a theoretical model and network simulations. It differs from previously-investigated adaptive mechanisms for bandwidth-based payment by requiring very limited state on the server.**

I. Introduction

Denial of service (DoS) attacks are a growing concern as they continue to pose an elevated threat to the reliability of the Internet. Such attacks can occur at all levels in the protocol stack and threaten both routers and hosts. Many attacks aim to deplete scarce resources (*e.g.* CPU, memory, disk) by generating illegitimate requests from one or many, possibly compromised, attacker-controlled hosts [16], [2], [5], [6], [3], [4]. The time required to process these requests degrades the service to available clients to an unacceptable degree or forces costly over-provisioning by the service provider. Instances of potentially vulnerable services include IKE key exchanges for gateway security association setup [13], legacy and digitally-signed DNS services [10], large file retrievals from web servers, computationally expensive query processing at database front-ends, and the capability allocator in network architectures that aim to handle DoS attacks by issuing capabilities to clients [24], [25].

A variety of counter-measures have been proposed to address these problems. *Currency-based mechanisms* are ones in which a server under attack demands some type of payment from clients in order to raise the bar for provoking work by the server beyond the capacity of the attacker. Classic currency examples in this context are *money* [15] and *CPU cycles* [23], [7], [9]. Our focus in this paper is on recent work that has proven the effectiveness of *bandwidth* as currency. In order to get service, the clients are encouraged to spend more bandwidth by either sending repeated requests from which the server selectively verifies (processes) some [11], [19], or dummy bytes on a separate channel to enable a bandwidth auction [22]. Currency-based mechanisms impose a cost on the system, particularly on the clients, so it is desirable to have *adaptive* counter-measures that are deployed dynamically and proportionally to blunt attacks at minimal cost. [22] describes how to do this for auction-based bandwidth payments but the proposed solution potentially requires significant server state such as tens of thousands of TCP sessions. The selective verification algorithm in [11], [19] requires almost no server state, but does not include any mechanism for adaptation.

In this paper we introduce *Adaptive Selective Verification* (ASV) which is a distributed adaptive mechanism for thwarting attackers' efforts to deny service to legitimate clients based on selective verification. Our scheme uses bandwidth as currency but the level of protection employed by the clients dynamically adjusts to the current level of attack. At a high level, the clients exponentially ramp-up the number of requests they send in consecutive time-windows, up to a threshold. The server implements a reservoir-based random sampling to effectively sample from a sequence of incoming packets using bounded space. This enables adaptive bandwidth payments with server state whose size remains small and constant regardless of the actions of the attacker. While the protocol itself is both natural and simple, analyzing its performance turns out to be a rather intricate task. A primary contribution of this work is a novel theoretical analysis of ASV whereby we evaluate its performance as compared to an "omniscient" protocol in which all attack parameters are instantaneously made known to all clients as well as the server. Surprisingly, we show that ASV closely approximates the performance of this omniscient protocol. The performance is measured in terms of success probability of each client, and the total bandwidth consumed by the clients. We also perform a simulation evaluation of the adaptive selective verification protocol with the aim of understanding its performance in practice. Besides validating our theoretical guarantees, our simulations show that under a time-varying attack, the performance of ASV protocol adjusts quickly to the prevailing attack parameters.

The rest of the paper is organized as follows. In Section II we position and compare this work against related work. Section III details our modeling of the client and attacker behavior. In Sections IV, V, and VI we introduce basic and extended versions of our protocol along with their theoretical analyses. Section VII presents our experimental simulations and finally Section VIII concludes the paper.

II. Related Work

Protection mechanisms that assure availability remain a difficult challenge for the Internet. Attacks can come from many sources in Distributed DoS (DDoS) or from a single source where spoofed addresses create the appearance of a DDoS attack. They can occur at link, network, transport, or application layers. They can be sudden and dramatic or gradual and subtle. Intentional attacks, aimed at disabling

978-1-4244-2026-1/08/$25.00 © 2008 IEEE

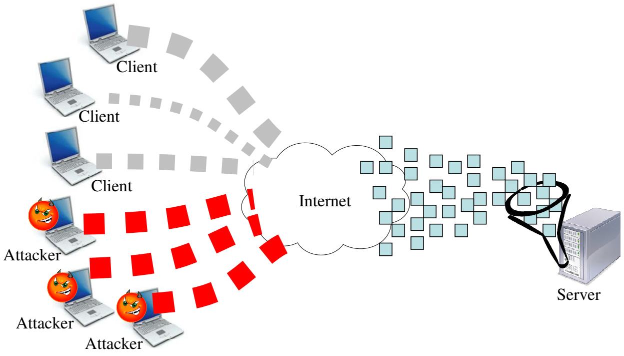

Fig. 1. Selective Verification

services, are easy to confuse with mis-configuration errors or heavy legitimate use. Most network protocols were designed without DoS protection and are vulnerable by design. This rich collection of attack vectors combines with various options for what can be changed to effect a counter-measure. For instance, is it possible to involve the routers or do solutions need to work exclusively at hosts? Is it possible to change packet formats and protocols or must these be handled in a transparent manner? Can one assume the attackers have limited knowledge of network state (such as eavesdropping on packets) or must they be assumed to have global knowledge? Solutions vary according to the types of attacks envisioned and these options for counter-measures. For instance, there are trace-back methods [18], [20] to find attackers so action can be taken to cut them off, and filter mechanisms to rate-limit of remove the attack traffic from the network [14], [12], [17]. There are capability-based mechanisms to allow parties that can prove their legitimacy to gain priority or to limit the impact of unproven parties [24], [25]. And there are currency-based schemes that aim to limit attackers by making them sacrifice a valuable resource like money or CPU cycles in order to get access to server resources [15], [23], [9], [7].

Perhaps the most counter-intuitive currency-based strategy is the use of *bandwidth* as payment. In such a scheme, clients use additional bandwidth to get access. The idea is that attackers are using all of the bandwidth available to them (or the maximum bandwidth they can afford to use without being detected by other mechanisms) to execute an attack, whereas legitimate clients are using only the resources they require to accomplish their less-demanding objectives. Hence legitimate clients have bandwidth to spare and can use this fact to differentiate themselves from attackers. This strategy was introduced in [11] in the context of authenticated broadcast and extended to general Internet protocols in [22]. At least two general strategies are possible. *Selective verification* allows clients to send extra requests and the server selects from these probabilistically. The idea is illustrated in Figure 1 where attacker and client requests are mixed together and filtered before being addressed by the server. *Bandwidth auctions* allow clients to build credit by sending bytes to an accounting system and the server takes requests from clients that have built the most credit. Bandwidth auctions provide a natural adaptive mechanism since clients send packets until they are serviced and, if the attack is modest, this occurs quickly so little bandwidth is wasted. On the other hand, this entails potentially significant requirements for accounting state whereas selective verification is essentially stateless. Since [11] does not propose any adaptive approach to selective verification, it remains an open question whether there is a good adaptive, stateless bandwidth payment scheme.

There are several works that address adaptive measures for DoS protection in other contexts. [27] provides ideas that are effective for filter schemes, although it is unclear how they can be applied to bandwidth payments. [21] shows how to use information available in the application layer to identify and differentiate between low and high utility clients to provide better service to more valuable customers. The solution requires more feedback from the application than selective verification and is more applicable to scenarios where the clients have a history of interactions with the server. [23] shows how to provide adaptation for client puzzles. Because of the nature of the client puzzle schemes, where the cost factor of the defense on the server is minimal, the proposal mainly focuses on cost minimization for the clients. However bandwidth payment schemes must account for costs to the server and network as well as the client. Finally, [26] proposes an adaptive solution for installing router throttles in the network. The main focus of the proposed approach is on network flooding attacks and router-based distributed defense against them, but it shares many of the same high-level adaptation concerns as bandwidth payment.

III. The Setting

Consider the following one-round client-server protocol. The first step of the protocol is an REQ packet from a client $\mathcal{C}$ to the server $\mathcal{S}$. In response, the server sends back an ACK to the client. Each client employs a *time-out window* of duration T determined by the worst case expected round-trip delay between the clients and the server: if after transmission of an REQ a client does not receive an ACK within T seconds he assumes that the attempt has failed. The parameter T is known to the clients as well as the server.

It will be convenient to partition time into a sequence of windows $W_1, W_2, \ldots,$ each of duration T. We suppose that the server $\mathcal{S}$ can process requests at a rate of S REQ packets per second so that the number of requests that it can process in any given window is ST. In any given window W, new clients arrive at a rate of $\mathsf{R}(W) = \rho(W)\mathsf{S}$ clients per second. The *client request factor* $\rho(W) = \mathsf{R}(W)/\mathsf{S}$ determines the fraction of the server's computational bandwidth that is required to process new clients in the window W. We suppose that the client request factors are uniformly bounded away from both zero and one, $0 < \rho_{\min} \leq \rho(W) \leq \rho_{\max} \leq 1$, for some fixed $\rho_{\min}, \rho_{\max}$ in the unit interval. We are particularly interested in the situation where $\rho_{\max} \ll 1$ as it may reasonably be expected that REQ requests are typically small packets with most of the server capacity dedicated to servicing the bulk of the communication to follow.

We will assume that a diffuse, distributed, denial of service attack $\mathcal{A}$ on the server takes the form of a potentially time-varying flood of spurious REQ packets aimed at overwhelming the server's capacity to process new REQs. We suppose that, in any given window W, the attack $\mathcal{A}$ sends spurious REQs at a rate of $A(W) = \alpha(W)S$ packets per second. The *attack factor* $\alpha(W) = A(W)/S$ determines the excess computational bandwidth that will be required of the server to process the illegitimate requests in window W. In keeping with the guiding philosophy of the shared channel model that was articulated by the authors to model DoS attacks [11], we assume that the attack factors are uniformly bounded, $0 \leq \alpha(W) \leq \alpha_{\max}$, for some fixed $\alpha_{\max}$, though the upper bound on the attack factors may be very large. Clearly, when $\alpha(W) > 1$ the attack *prima facie* overwhelms the server's capacity to process all requests and, abeyant a protocol to handle overflows, there is the potential for the successful execution of a DoS attack. Our interest is in the particular case where $\alpha_{\max} \gg 1$ and the attack can occur on a scale much larger than the available server computational bandwidth.

In order to focus on the DDoS attack at the receiver, in the next two sections we idealize the situation and assume that REQ and ACK packets are transmitted instantaneously, the round-trip delay occasioned solely by processing time at the server, and that no REQ or ACK packets are lost in transmission. Packet drops at the server are then occasioned only because the arriving request stream from clients and attackers combined exceeds the server's computational bandwidth. Thus, if $\rho_{\max} + \alpha_{\max} > 1$ then it cannot be guaranteed that an individual client's REQ will be processed by the server. If $\alpha_{\max} \gg 1$ it is in principle then possible to almost completely throttle the clients of service and effect a successful DoS attack.

In the sequel, logarithms are to base e. Long proofs are eliminated due to space constraints and will appear in the full version of the paper.

IV. The Omniscient Protocol

Consider any time-out window W. Suppose that $0 < \rho = \rho(W) < 1$ and $0 < \alpha = \alpha(W)$ denote the client request factor and the attack factor, respectively, over the window W. If clients and server clairvoyantly know ρ and α then it is easy for them to thwart the DDoS attack using a modicum of repetition combined with selective randomized sampling at the server. This simple, if unrealistic, situation is worth analyzing as it provides benchmarks for more realistic situations.

Omniscient Client Protocol: Given α and ρ, each new client in a given window W transmits $\lceil \alpha/\rho \rceil$ copies of the REQ packet in that window. Clients who do not receive an ACK from the server within T seconds leave never to return.

Omniscient Server Protocol: Given α and ρ, the server accepts an arriving REQ packet in the window W, independently of other arriving packets, with probability

$$p = \frac{1}{\alpha + \rho\lceil\frac{\alpha}{\rho}\rceil}$$

and discards it with probability $q = 1 - p$. The server sends out an ACK for each accepted REQ.

The total number of REQs transmitted by clients in window W is $\lceil\frac{\alpha}{\rho}\rceil \rho ST$. It follows that, *for any given window W, the cumulative mean transmission bandwidth consumed by client REQs in the omniscient client-server protocol is approximately αS packets per second.* As the number of attack packets received in this window is αST, the total number of REQs received by the server during window W is given by

$$N = N(W) = \lceil\tfrac{\alpha}{\rho}\rceil \rho ST + \alpha ST = \bigl(\alpha + \rho\lceil\tfrac{\alpha}{\rho}\rceil\bigr)ST.$$

Accordingly, the expected number of packets processed by the server in window W is given by $pN = ST$ so that the server processes REQs at its rate of S packets per second.

Theorem 1 (Omniscient Connection Confidence) *Suppose $0 < \delta < 1$ is a given confidence parameter. If*

$$\rho_{\max} \leq \frac{1}{-2\log\delta} \tag{1}$$

then the probability that a given client has an REQ accepted is at least $1 - \delta$ under the omniscient client-server protocol.

Proof: A given client $\mathcal{C}$ transmits $\lceil\alpha/\rho\rceil$ REQs in a window W. The probability that each of these REQs is discarded by the server is given by

$$Q := q^{\lceil\alpha/\rho\rceil} \leq e^{-1/2\rho} \leq e^{-1/2\rho_{\max}} \leq \delta \tag{2}$$

in view of the elementary inequality $1 - x \leq e^{-x}$. ■

Thus, for all sufficiently small client request factors $\rho_{\max}$, the omniscient client-server DDoS protocol accepts REQs from all but a small fraction of at most δ of all clients at a cost in transmission bandwidth of (about) αS client packets per second.

V. The Adaptive Protocol

The assumption in the omniscient client-server protocol that clients are continuously aware of the client request factor and the attack factor current in each window is clearly unrealistic, especially given the distributed and—until connection is established—as yet unknown location and legitimacy of the clients and, more critically, the ability of the attack to vary rates continuously and unpredictably. Designing a protocol for the worst-case attack is, of course, possible, but unnecessarily congests the network during periods when the attack is quiescent or at low levels. Our goal, hence, is to design a client-server DDoS protocol which adapts to the behavior of the attack $\mathcal{A}$ *without clients having access to explicit current information about the nature and intensity of the attack.*

In view of our experience with the omniscient protocol, on the client side we are led to seek a replicating protocol where the replication rate used by the clients should ideally be proportional to the current attack factor (and inversely proportional to the current request factor though this is likely to be under better regulation). While the client does not have direct access to this information, he can infer the state of the attack indirectly based on whether he receives an ACK or not in response to REQ(s) sent in the previous window. The failure to receive an ACK in response to transmitted REQ(s) can be construed provisionally as evidence of an attack in progress and the client can then ramp up his replication rate in an effort to counter current attack conditions. Experience with doubling algorithms (or, on the flip side, exponential back-off in TCP protocols) suggests that it would be profitable to have the replication rate grow exponentially with repeated connection failures (up to a worst-case maximum).

On the server side, a more detailed picture about current conditions can be directly obtained from the ensemble of packets arriving in each time-out window. The server can now very simply maintain the advertised service rate by reservoir sampling to generate a random sample of the sequentially arriving packets. The randomized sampling of incoming packets helps obviate timing attacks or the exercise of other overt control by the adversary over the decision making process at the server, while the adaptive changes in sampling rates that reservoir sampling accords allows the server to respond to changes in attack factors across windows while staying within the budgeted service bandwidth. These considerations lead to the following adaptive client-server protocol.

Adaptive Client Protocol: Given $\rho_{\max}$, $\alpha_{\max}$, and T, after each unsuccessful attempt the client adaptively increases the number of REQs sent in the succeeding time-out window up to a maximum number specified by the given parameters.

C1. [*Initialize replication count.*] Set $j \leftarrow 0$ and $J \leftarrow \left\lceil \log\left(\frac{\alpha_{\max}}{\rho_{\max}}\right) \big/ \log(2) \right\rceil$.

C2. [*Double replication.*] Send 2^j REQ packets to the server.

C3. [*Time-out.*] If no ACK packet is received within T seconds, set $j \leftarrow j+1$; if an ACK packet is received, exit the initiation protocol and proceed to the next phase of communication.

C4. [*Iterate till exit condition.*] If $j \leq J$, go back to step C2; else exit without communicating with the server.

Adaptive Server Protocol: The server performs reservoir sampling on incoming REQ packets during each time-out window. Given S and T, the server processes a random subset of the arriving REQs at a rate not exceeding S packets per second.

S1. [*Initialize window count.*] Set $k \leftarrow 1$.

S2. [*Form reservoir.*] Store the first $\lfloor ST \rfloor$ REQ packets arriving in window W_k in a reservoir. If time-out expires without filling the reservoir, go to step S4. Else, set REQ packet count to $j \leftarrow \lfloor ST \rfloor + 1$.

S3. [*Randomly sample incoming packets.*] If there is an incoming REQ numbered j, accept it for placement into the reservoir with probability $\lfloor ST \rfloor / j$ and discard it with probability $1 - \lfloor ST \rfloor / j$. If the REQ is accepted for placement in the reservoir, discard an REQ from the reservoir uniformly at random and replace it with the accepted packet. Set $j \leftarrow j + 1$ and iterate until the time-out window expires.

S4. [*Time-out*] Accept the packets in the reservoir and send out an ACK for each accepted REQ.

S5. [*Iterate.*] Empty the reservoir, set $k \leftarrow k+1$, and go back to step S2.

We have streamlined the protocols to focus on the critical ideas. In particular, we adopt the convenient fiction that step S4 in the server protocol occurs *instantaneously*. Thus, there is no gap in time between the expiration of a time-out window expires and the identification of the random subset of packets that is accepted by the server over that window. The reservoir sampling procedure is due to Fan, Muller, and Rezucha [8] and guarantees that if N REQ packets arrive in a given time-out window and $N > ST$, the server accepts a subset of size ST REQ packets uniformly at random from the N packets, and discards the remaining packets.

We call the quantity

$$J = \left\lceil \log\left(\tfrac{\alpha_{\max}}{\rho_{\max}}\right) \big/ \log(2) \right\rceil \tag{3}$$

the *retrial span* of a client. In the event that the attack is launched at maximum severity, a client can replicate packets over a period of J windows until he achieves a maximum replication rate of $\alpha_{\max}/\rho_{\max}$ matched to the peak attack.

Blocking Probabilities: Our results say that each client succeeds in establishing a connection with essentially the same confidence guarantee as in the omniscient case at the expense of some added delay.

THEOREM 2 (ASV CONNECTION CONFIDENCE) *Suppose* $0 < \delta < 1$ *is a given confidence parameter. If*

$$\rho_{\max} \leq \frac{1}{-5 \log \delta} \tag{4}$$

then the probability that a given client has a REQ accepted within JT *seconds is at least* $1 - \delta$ *under the adaptive client-server protocol.*

Note that the bound on $\rho_{\max}$ as given in inequality (4) differs from the one in inequality (1) by only a small constant factor.

The theorem above can be proved using the following two lemmas. We omit the proofs of these lemmas due to space considerations. Let us say that a client is in *generation* g *with respect to window* W if it initiated the protocol g windows in advance of W.

LEMMA 1 (TRAFFIC BOUND) *The total number of REQs, legitimate and illegitimate, received by the server during any window* W *can be bounded by*

$$N(W) \leq 5\alpha_{\max} ST. \tag{5}$$

The uniform bound on the number of packets that can be received during any window allows us to bound the probability that a client repeatedly fails to establish a connection with the server.

LEMMA 2 (BLOCKING PROBABILITY) *The probability that a client $\mathcal{C}$ in generation g with respect to a window W fails to establish a connection with the server by the end of W is bounded by*

$$Q_g(W) \le \exp\left(\frac{-2^g ST}{N(W)}\right)$$

for each $0 \le g \le J$.

If we set $g = J$ and bound $N(W)$ by (5) using the Traffic Bound Lemma 1, we obtain a bound for the probability $Q(\mathcal{C})$ (with a hopefully transparent abuse of notation in the reuse of Q) that $\mathcal{C}$ fails to make a connection and leaves the protocol (that is to say, the *blocking probability for* $\mathcal{C}$). In particular, the Blocking Probability Lemma 2 shows that.

$$Q(\mathcal{C}) \le \exp\left(\frac{-2^J}{5\alpha_{\max}}\right). \tag{6}$$

Noting that $2^J \ge \alpha_{\max}/\rho_{\max}$, the ASV Connection Confidence Theorem 2 follows.

This analysis is essentially tight for our protocol as can be seen by consideration of the setting where the attack factor is $\alpha = \alpha_{\max}$ in every window, the request factors satisfy $\rho_{\max} = \rho$, and $\alpha/\rho = 2^\kappa$ for some integer κ. In this case, with high probability each client would have to wait for κT seconds to consummate a connection.

Bandwidth Considerations: The Traffic Bound Lemma shows that if the attack factor is maintained at $\alpha_{\max}$ then the cumulative mean transmission bandwidth consumed by client REQs is $\mathcal{O}(\alpha_{\max} S)$ packets per second. The estimate is rather pessimistic, however, and we anticipate that the adaptive protocol does much better in periods of lulls in attack.

The key idea is provided by the Blocking Probability Lemma which shows that the probability that a client is blocked decreases very rapidly with generation; for moderate attacks, any given client is likely to be accepted by the server well before the end of his retrial span J; indeed, he is likely to form a connection within $J + \log(\rho_{\max})/2\log(2)$ generations. (Bear in mind that $\rho_{\max} < 1$ so that $\log(\rho_{\max}) < 0$.) This suggests that there is relatively little client traffic build up due to unconsummated connections near the end of the retrial span. It follows that the upper bound on traffic given by (5) may be much too generous in periods where attack rates are low. We claim that, indeed, the transmission bandwidth required by clients in the adaptive protocol is essentially of the same order as that commandeered in the omniscient protocol.

In order to present the results as compactly as possible, in the sequel we shall assume that the maximum client request rate satisfies $\rho_{\max} = \mathcal{O}\big(1/\log(\alpha_{\max})^2\big)$. (We refer the reader to the long version of the paper for general statements of the results and proofs.) We may also assume, without loss of generality, that $\alpha_{\max}$ is at least 1 as the problem admits a trivial statement otherwise.

We formalize the intuition that client requests are typically accepted relatively quickly by considering successive windows forming *generational slices* of width $\lambda := \lceil -\log(\rho_{\max})/2\log(2)\rceil$ windows. For any window W, we call the swath of $\sigma := \lceil \log(\alpha_{\max})/(\lambda - 1)\log(2)\rceil$ windows preceding it the *segment preceding* W and denote it $\mathcal{S}(W)$. Let $\overline{\alpha} = \overline{\alpha}(W)$ denote the largest attack factor of any window in the segment $\mathcal{S}(W)$.

THEOREM 3 *Under the stated conditions on the client and attack rates, the expected cumulative transmission bandwidth consumed by client REQs in window* W *under the adaptive client-server protocol is bounded by* $\mathcal{O}(\overline{\alpha} ST)$.

We have opted to present slightly conservative bounds on the client request factors to keep the theorem statement as uncluttered as possible to permit easy comparison with the omniscient strategy. The range of validity of the client request factors may be expanded at the cost of increased algebraic tedium. The proof of this theorem entails a recursive pruning of unabsorbed generations and is somewhat involved. We defer it to the full paper in view of space considerations.

For comparative purposes, one observes that if the attack factor in a given window is $\overline{\alpha}$, omniscient clients aware of both the current attack and request factors will occupy a bandwidth of $\overline{\alpha}S$ REQs per second. The adaptive protocol achieves essentially this order working from *tabula rasa* with no specific state knowledge. The simulations of Section VII provide additional confirmation of the stability and efficiency of the ASV protocol.

Since $\sigma = \mathcal{O}\big(\log(\alpha_{\max})\big/\log(\frac{1}{\rho_{\max}})\big)$ we obtain the following general theorem as a corollary of Theorem 3.

THEOREM 4 (ASV BANDWIDTH) *The expected bandwidth consumption of the adaptive client-server protocol is only* $\mathcal{O}\big(\log(\alpha_{\max})\big/\log(\frac{1}{\rho_{\max}})\big)$ *times larger than the bandwidth consumed by the omniscient selective verification protocol.*

It is worthwhile to contrast this bound with a non-adaptive approach that stays in the high protection mode at all times. The bandwidth consumed by such an approach can be $\mathcal{O}(\alpha_{\max})$ times larger than the bandwidth consumed by the omniscient selective verification protocol. Thus the adaptive scheme can improve the bandwidth consumption *exponentially*.

VI. EXTENSIONS

In this section we focus on extending the basic adaptive protocol from two aspects. First, we identify and address two concerns with respect to the practical deployment of the protocol. Specifically these relate to the possibility of an unreliable server and network. Second, we elaborate on the ways by which the clients and the server could use the protocol in a more flexible and cost-effective fashion. Specifically we consider regulation of bandwidth devoted to ASV by the server and client, respectively.

Suppose that a server being protected by ASV goes down. Under the current ASV protocol, this would cause a flood of requests from the clients, which would only aggravate the situation. It would be desirable to avoid this unwanted side-effect of the protocol. A potential solution is for the server to provide a special type of ACK, *DACKs (Drop ACK)* at step S3 in Section V for every request it receives but is not able to process. DACKs serve as an encouragement mechanism which communicates a "please retry more aggressively" message to the clients. A client in round i (which sends 2^i requests), waits for the server's ACK or DACK(s) before moving to round $i+1$. He quits upon receipt of an ACK; else, if he receives any DACK in T time units, he moves on to the next round; failing these two possibilities, he quits. In addition, in order to obstruct the opportunity of creating smurf-type attacks [1], the clients can put a nonce (as a weak authenticator) in each of their REQs, and expect the server to return it in DACKs.

We have so far assumed that the network is reliable. But, what if the network is lossy (*e.g.* due to congestion) and REQs and/or server responses are sometimes dropped? This could result in undesirable scenarios such as a client quitting under perception of a down server. A potential solution is to modify the client's protocol as follows. If no DACK is received for K consecutive packets sent by the client, he quits. This check is only performed at the beginning of each round. Therefore, if the path from a client to the server experiences a drop rate of d, this modification reduces the probability of a client incorrectly quitting to the order of d^K. In addition, this can serve as a crude congestion control mechanism, particularly if K is set to low values. We do not consider this addition a full-fledged congestion control mechanism. In particular, we intentionally do not want the clients to be overly reactive to low-rate packet drops, since this would easily make them back off and provide an opportunity for the attackers to build an advantage. However, if due to a network link attack (which is out of the scope of this work), or any other reason there exists *heavy* congestion in the network, it would be desirable for ASV clients to back off and not further flood the network. We believe by properly setting the value of K, the above mentioned effect could be achieved. We experimentally investigate the effect of network congestion on ASV in Section VII.

We can also envision situations in which a server that uses ASV for protection would be interested in devoting less bandwidth to the ASV process. For instance, consider a number of co-located services that share bandwidth. Under certain circumstances, it may not be economical for one service to consume too much bandwidth just to prevent denial-of-service attacks. Another example could be a server that would prefer to reduce the ASV bandwidth consumption in favor of having more bandwidth available for a period of heavy bulk transfers. Concerns of this type may be addressed by having the server inform the clients *not* to send too much traffic. However, this comes at the cost of a potential degradation in service guarantees to the legitimate clients. Translated to our protocol, this means that the server provides clients with a new retrial span J when it is desired that the clients reduce bandwidth consumption.

Consider a function $F\colon r \mapsto (BW, SuccProb)$ where r is a candidate retrial span, BW is an upper-bound on bandwidth consumption, and $SuccProb$ is the expected probability that each client succeeds in getting a request served. Now, consider a utility function $U\big((BW, SuccProb), P\big)$ which takes these bandwidth consumption and success probabilities together with a *system priorities* input P, and outputs the expected utility for the system as a value between 0 and 1. P is a generic input for system parameters that encode the current priorities and constraints of the system such as the ones discussed above: The goal is to pick a retrial span that maximizes the utility of the system. The new J calculated below is communicated to the clients to be used as their retrial span: $J = \arg\max_r \big\{U\big(F(r), P\big)\big\}$.

Suppose the server has successfully communicated the value of the retrial span J to the clients. In simple words, J is the maximum bandwidth (price) that the server is willing to accept from each legitimate client. However, what if the client is not willing to spend this much bandwidth to get service? We use a client cost-benefit analysis inspired by [23] to formulate this problem. Suppose each client has a *valuation function* V indicating the value of receiving the service in *some* units (for simplicity *dollars*). Also suppose that the client knows a non-decreasing success function that maps the retrial span r of the client to the probability of succeeding in getting service when the server is heavily loaded. Such a function could be obtained from the server through DACKs. Based on this the client can compute its retrial span as in [23]. On the other hand, there could be a client that is willing to send the required number of repeated requests, but its low bandwidth connection does not allow it to. Suppose that at a round i its connection allows it to send (in a single window) only $\frac{2^i}{f}$ packets for some integer $f \geq 1$. It is easy to verify that if the client continues to send $\frac{2^i}{f}$ packets, in $\mathcal{O}(f)$ windows it will succeed with the same probability as if it sent 2^i packets in a single window (provided that neither the attack rate nor the client traffic changes).

VII. Experimental Evaluation

To measure the effectiveness of the proposed adaptive mechanism, we perform several network simulations. The simulations provide an opportunity to test the full protocol in settings that reflect the real world situations more closely. Specifically, the simulations aim to evaluate the effectiveness of the adaptive scheme against its non-adaptive counterparts and verify the accuracy of our analytical results. In addition, we study ASV's behavior in the presence of network congestion, as well as its effect on TCP-based cross traffic.

Simulation Setup: The simulations are performed using the *NS-2* network simulator for the topology shown in Figure 2. The topology shown is dynamic in the sense that, in each simulation scenario, the number of clients increases with time. Every second, 50 new clients join the topology and start sending REQs toward the server. Each client that joins the topology needs to get one REQ served. The number of attackers can range between 1 and 100 to represent different attack rates. Each attacker constantly issues 400 REQs/s. So, depending on the number of attackers for each scenario, the attack rate would range between 400 and $40,000$ REQs/s. S, the number of requests that the server can process in a second, is set to 600. Translated into our notation $\alpha_{\max} = 66$, and $\rho = \rho_{\max} = 0.08$. RTT is 60ms and T is set to 0.4s. REQs are 200 bytes, and server DACKs and ACKS are 50 and 200 bytes,

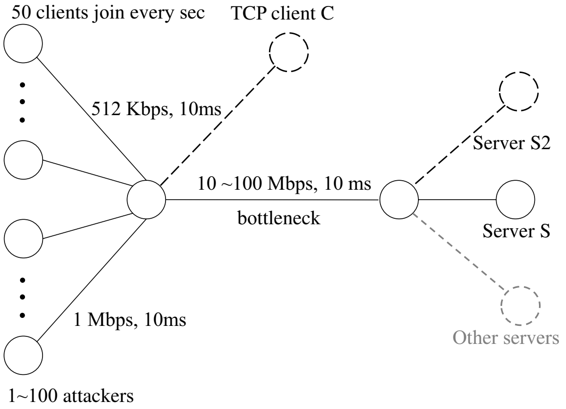

Fig. 2. Simulation topology.

respectively. All the communications are over UDP unless otherwise noted. In most of the experiments, the capacity of the bottleneck link is over-provisioned to 100Mbps to avoid any network congestion. However, in particular experiments, we reduce this capacity and create network congestions as needed. The arrival times of the clients and attackers, as well as the inter-packet intervals for attackers are randomized in order to avoid any undesirable deterministic pattern. Given the parameters above, based on the theoretical guidelines in Section V, the retrial span J should be set to 10. We performed separate experiments (not reported in this paper) using $J = 10$ which easily verify the theorems in that section. In view of practical cost-benefit considerations outlined in Section VI, we present here experiments for a retrial span of $J = 7$.

Comparing Adaptive and Non-Adaptive: In order to evaluate ASV against its non-adaptive counterparts, we implement three different client behaviors and compare them against each other in various attack conditions (solid line topology in Figure 2). The three client behaviors are:

- Naive: Send one REQ every T seconds. Quit if an ACK is received or JT seconds pass.
- Aggressive (Non-Adaptive): Send 2^J REQs. Quit if an ACK is received or JT seconds pass.
- ASV: Implement ASV for one REQ (which means for a maximum of JT seconds).

Each experiment is performed with one type of client and a fixed average attack rate for 30 seconds which proves to be sufficiently long for the system to stabilize. The results are obtained by changing attack rates across different simulation runs.

Figure 3 shows the ratio of clients that succeed in getting one REQ served against different attack levels. Figure 4 illustrates the expected time to service for clients that succeed in getting service. And finally, the aggregate bandwidth consumption from legitimate client REQs is depicted in Figure 5. We only report on the *client* bandwidth overhead since in each scenario, the number of REQs issued by attackers (and therefore the respective bandwidth consumption) is fixed and the same across all three cases. Client bandwidth consumption numbers are used to compare the bandwidth overhead (cost) introduced by each of the three client behaviors.

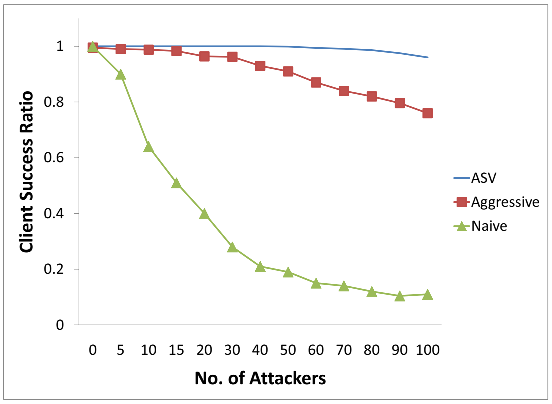

Fig. 3. The ratio of the successful clients to all clients (1500 in 30s) vs. attack rate.

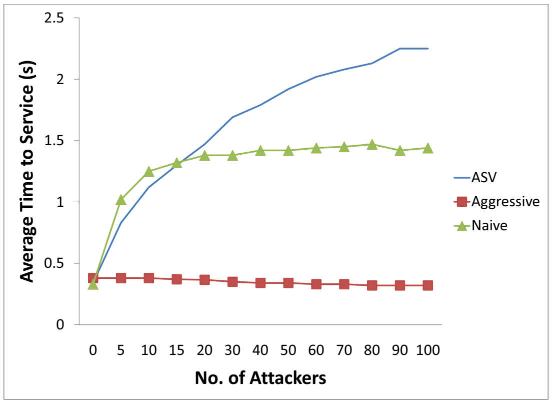

Fig. 4. Average time to service (for clients that succeed in getting service) vs. attack rate.

It can be immediately concluded from Figures 3 and 5 that ASV outperforms the Aggressive scheme in terms of success ratio and bandwidth consumption. This proves the effectiveness of the adaptation strategy in raising the costs (bandwidth) in accordance with the attack rate. As expected, the average time to service is lower for Aggressive, however,

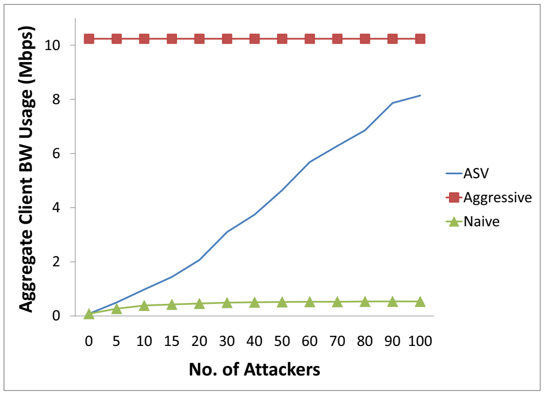

Fig. 5. The aggregate bandwidth consumption for all the clients vs. attack rate.

given the ASV's benefits in terms of success probability and bandwidth consumption, service latencies of at most 2.3s for the fiercest attacks should be considered an acceptable trade-off. The Naive clients, as expected, suffer serious failure rates, which underscores the effectiveness of ASV. The results also quantify the overhead of ASV, which is a factor of 16 in terms of bandwidth, and 1.5 in terms of service latency in the worst attack scenarios.

Pulse and Variable Rate Attacks: In each of the previous experiments the attack rate was fixed during the simulation. Here, we explore the effect of varying attack rates on clients that implement ASV. In the first set of experiments we subject the system to *pulse* attacks. In these experiments we observe the system's behavior under a 5 second no-attack period, followed by 10 seconds of heavy (but fixed rate) attack, and another 10 seconds with no attack. We performed this experiment for 25, 50, and 100 attackers. The detailed results are omitted due to space constraints. However, the most important outcome is that in all three scenarios, in less than two seconds the system fully adapts itself to attack conditions, *i.e.* success ratio, time to service, and bandwidth consumption numbers converge to the corresponding values in Figures 3, 4, and 5. In addition, after the attack stops, the system recovers to its pre-attack conditions in less than two seconds.

To better understand the effect of highly variable-rate attacks we simulate 45 seconds of variable rate attacks, preceded and followed by 5 second periods with no attack. The number of attackers changes and is $\lceil \exp(r) \rceil$ where r is a floating point number chosen at random from $[0, \ln(100))$ each second. The results are depicted in Figure 6.

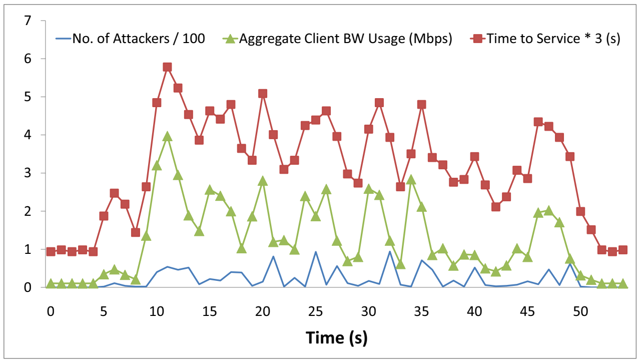

Fig. 6. The effect of 45 seconds of variable rate attacks on success ratio and aggregate client bandwidth consumption. Success ratio for clients is always 1. Note that clients *joining* the system between times i and $i+1$ are represented in front of Time $=i$.

These experiments show how quickly the system adapts and then recovers to the pre-attack pattern in the presence of pulse attacks. This significantly reduces the attackers' ability to disrupt the operation (and bandwidth consumption) of multiple ASV protected servers' at the same time by attacking them in rotation. The variable rate attack experiments (Figure 6) show how ASV preserves success ratio, time to service, and bandwidth consumption within reasonable bounds.

Lossy Network: So far, we assumed links are over-provisioned, and thus there is no packet loss in the network. In order to assess the effect of a lossy network, we make the bottleneck link drop packets at different rates. The experiments are performed for $K = 3$ and $K = 7$ with 50 attackers present. In brief, in both cases, for drop rates of up to 30%, there is almost no quitting and client bandwidth consumption stays approximately fixed. However, for network drop rates of 40% to 80%, the quit ratio ranges from 0.08 to 0.71 for $K = 3$, and from 0.01 to 0.32 for $K = 7$. The corresponding client bandwidth consumption ranges from 4.26Mbps to 1.04Mbps, and 4.62Mbps to 4.08Mbps respectively.

Even though enforcing a cap on the maximum number of outstanding REQs (with no DACK) is not meant to be a full-fledged congestion control mechanism, it would still be desirable for the ASV clients to react to very serious network congestions by backing off (please refer to Section VI for details). Additional simulations that we do not report here for lack of space provide evidence that if (for any reason) clients face heavy congestion in the network, they eventually react and stop aggravating the situation.

Effect on TCP Cross Traffic: To measure the effect of ASV on cross traffic we set up the following simulation scenario. We create a client C that is communicating with a data-backup server S2, co-located with the ASV-protected server S behind the bottleneck link. The capacity of the bottleneck link is set to 10Mbps (see the shaded lines in Figure 2). Client C is backing up data on S2, and thus uploads data on S2 at the rate of 512Kbps over TCP. In parallel, we simulate DDoS attacks on S with a clientele of 50 clients per second (Naive, and ASV with $K = 3$). As before, Naive behavior represents a no-defense base for comparison reasons. The attack rates and the queueing disciplines used in the bottleneck link vary in different scenarios. The queueing disciplines in the bottleneck link are DropTail and Stochastic Fair Queueing (SFQ) with 80 buckets. The amount of data that C can upload to S2 in 30s in each scenario is plotted in Figure 7.

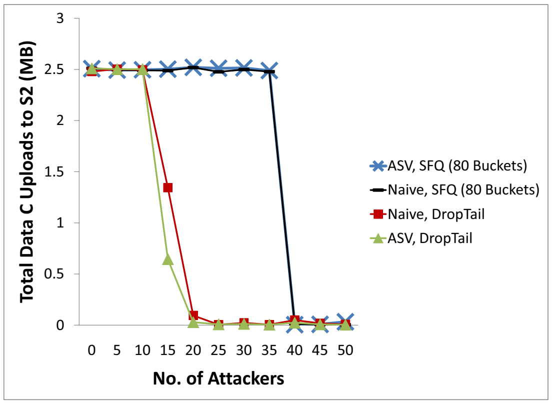

Fig. 7. The amount of data that client C can upload to sever S2 in 30s. The lines that are close to the horizontal axis represent values in the 7.5KB to 15KB range.

The figure shows that when TCP cross traffic shares a bottleneck link with non-congestion controlled traffic from attackers, it could be seriously throttled. It confirms that unless the network links around a UDP-based service are highly over-provisioned and protected against network link attacks, TCP cross traffic would be seriously harmed in the face of fierce attacks. In addition we observe that Stochastic Fair Queueing (SFQ) would provide better guarantees compared to DropTail *only* until client C's traffic is hashed into the same bucket as attackers packets. This results in C's traffic being dropped, which in turn causes it to back-off. However, the main result of this experiment is that the attack traffic is the major cause of the TCP client's suffering, and thus compared to Naive (which represents no-defense attack-only scenarios) ASV does not cause any significant *extra* harm to TCP cross-traffic.

VIII. Conclusions

In conclusion, ASV advances the state-of-the art in bandwidth based DDoS defense mechanisms by introducing a distributed adaptive solution based on selective verification. In ASV, the clients exponentially ramp-up the number of requests they send in consecutive time-windows, up to a threshold. The server implements a reservoir based random sampling to effectively sample from a sequence of incoming packets using bounded space. The novel theoretical analysis of the protocol proves that the performance of ASV (in terms of client success probability and bandwidth consumption) closely approximates an "omniscient" protocol in which all attack parameters are known to clients and the server. NS-2 network simulations of the protocol verify and quantify the effectiveness of ASV against its non-adaptive counterparts and illustrate that under highly variable-rate attacks, the performance of ASV adjusts quickly to prevailing attack parameters. In addition, it is shown that the effect of ASV on internet cross traffic is minimal, and comparable to that of its naive non-adaptive counterpart, which represents no-defense attack-only scenarios.

IX. Acknowledgements

We thank Michael B. Greenwald, Jose Meseguer, Ravinder Shankesi, and the anonymous reviewers for their valuable comments. This work was supported in part by NSF CNS05-5170 CNS05-09268 CNS05-24695, NSF CNS 07-16421, ONR N00014-04-1-0562 N00014-02-1-0715, DHS 2006-CS-001-000001 and a grant from the MacArthur Foundation. The views expressed are those of the authors only.

References

[1] CERT CC. smurf attack. http://www.cert.org/advisories/ca-1998-01.html.

[2] Two root servers targeted by botnet. *PC Advisor (pcadvisor.co.uk)*, 02/07/2007.

[3] Phish fighters floored by DDoS assault. *The Register (theregister.co.uk)*, 02/20/2007.

[4] Surge in hijacked PC networks. *BBC (bbc.co.uk)*, 03/19/2007.

[5] Telegraph floored by DDoS attack. *The Register (theregister.co.uk)*, 05/22/2007.

[6] FBI busts alleged DDoS mafia. *Security Focus (securityfocus.com)*, 08/26/2004.

[7] M. Abadi, M. Burrows, M. Manasse, and T. Wobber. Moderately hard, memory-bound functions. *ACM Trans. Inter. Tech.*, 5(2):299–327, 2005.

[8] I. R. C. T. Fan, M. E. Muller. Development of sampling plans by using sequential (item by item) selection techniques and digital computers. *J. Amer. Statist. Assoc.*, 57:387–402, 1962.

[9] C. Dwork, A. Goldberg, and M. Naor. On memory-bound functions for fighting spam. 2003.

[10] D. Eastlake. *RFC: 2535, Domain Name System Security Extensions.*

[11] C. A. Gunter, S. Khanna, K. Tan, and S. S. Venkatesh. DoS protection for reliably authenticated broadcast. In *NDSS'04: Network and Distributed System Security Symposium*. The Internet Society, 2004.

[12] C. Jin, H. Wang, and K. G. Shin. Hop-count filtering: an effective defense against spoofed DDoS traffic. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 30–41, New York, NY, USA, 2003. ACM Press.

[13] E. C. Kaufman. *RFC: 4306, Internet Key Exchange (IKEv2) Protocol.*

[14] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *SIGCOMM Comput. Commun. Rev.*, 32(3):62–73, 2002.

[15] D. Mankins, R. Krishnan, C. Boyd, J. Zao, and M. Frentz. Mitigating distributed denial of service attacks with dynamic resource pricing. In *ACSAC '01: Proceedings of the 17th Annual Computer Security Applications Conference*, page 411, Washington, DC, USA, 2001. IEEE Computer Society.

[16] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage. Inferring internet denial-of-service activity. *ACM Trans. Comput. Syst.*, 24(2):115–139, 2006.

[17] G. Oikonomou, J. Mirkovic, P. Reiher, and M. Robinson. A framework for a collaborative DDoS defense. In *ACSAC '06: Proceedings of the 22nd Annual Computer Security Applications Conference on Annual Computer Security Applications Conference*, pages 33–42, Washington, DC, USA, 2006. IEEE Computer Society.

[18] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for IP traceback. In *SIGCOMM '00: Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 295–306, New York, NY, USA, 2000. ACM Press.

[19] M. Sherr, M. B. Greenwald, C. A. Gunter, S. Khanna, and S. S. Venkatesh. Mitigating DoS attacks through selective bin verification. In *IEEE ICNP Workshop on Secure Network Protocols (NPSec)*, pages 7– 12, 2005.

[20] D. X. Song and A. Perrig. Advanced and authenticated marking schemes for IP traceback. In *INFOCOM '01: Proceedings of the Joint Conference of the IEEE Computer and Communications Societies*. IEEE Press, 2001.

[21] M. Srivatsa, A. Iyengar, J. Yin, and L. Liu. A middleware system for protecting against application level denial of service attacks. In *Middleware*, pages 260–280, 2006.

[22] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker. DDoS defense by offense. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 303–314, New York, NY, USA, 2006. ACM Press.

[23] X. Wang and M. K. Reiter. Defending against denial-of-service attacks with puzzle auctions. In *SP '03: Proceedings of the 2003 IEEE Symposium on Security and Privacy*, page 78, Washington, DC, USA, 2003. IEEE Computer Society.

[24] A. Yaar, A. Perrig, and D. X. Song. SIFF: A stateless internet flow filter to mitigate DDoS flooding attacks. In *IEEE Symposium on Security and Privacy*, pages 130–, 2004.

[25] X. Yang, D. Wetherall, and T. Anderson. A DoS-limiting network architecture. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 241–252, New York, NY, USA, 2005. ACM Press.

[26] D. K. Y. Yau, J. C. S. Lui, F. Liang, and Y. Yam. Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles. *IEEE/ACM Trans. Netw.*, 13(1):29–42, 2005.

[27] C. C. Zou, N. Duffield, D. Towsley, and W. Gong. Adaptive defense against various network attacks. In *IEEE Journal on Selected Areas in Communications*, volume 24, pages 1877–1888. IEEE Press, 2006.