This paper was presented as part of the main technical program at IEEE INFOCOM 2011

QoSoDoS: If You Can't Beat Them, Join Them!

Moti Geva
Computer Science Department
Bar Ilan University
Ramat-Gan, 59200, Israel
Email: moti.geva@gmail.com

Amir Herzberg
Computer Science Department
Bar Ilan University
Ramat-Gan, 59200, Israel
Email: amir.herzberg@gmail.com

Abstract—**We present QoSoDoS, a protocol that ensures QoS over a DoS-prone (Best-Effort) network. QoSoDoS ensures timely delivery of time sensitive messages over unreliable network, susceptible to high congestion and network flooding DoS attacks. QoSoDoS is based on scheduling multiple transmissions of packets while attempting to minimize overhead and load, and avoiding self-creation of DoS. We present a model and initial empirical results of QoSoDoS implementation. Our results show that under typical scenarios, QoSoDoS can handle high congestion and DoS attacks quite well.**

I. Introduction

The Internet is a Best-Effort delivery network, that is, there is no allocation of resources to connections, and hence the Internet provides no guarantee for message delivery, and no bounds on delay. If the network is congested, then routers drop arbitrary packets instead of transmitting them to their destinations. This is often exploited for *flooding/clogging Denial-of-Service (DoS)* attacks, where an attacker intentionally causes excessive traffic and hence congestion. In many cases, DoS attacks and especially clogging attacks are carried out using multiple zombie hosts, and are referred to as Distributed DoS (DDoS) attacks [1]. Such clogging DoS attacks are hard to prevent and mitigate in an open, best-effort network such as the Internet.

Since the Internet is subject to congestion and clogging attacks, critical applications, such as inter-bank clearing, usually resort to private networks, or use only routers that support *Quality of Service (QoS)* protocols [2], [3] to connect source to destination. This can be expensive, even for services which can tolerate large (but bounded) delays, if their availability is critical. Examples for such services include financial and other contractual obligations, in which each contract counterparty must stand up to her obligations within a given time frame, which is usually within the order of seconds, minutes, hours or even days.

This appears frustrating; the Internet provides low-cost, ubiquitous communication which usually works fine, even for high bandwidth applications such as voice and video calls. Yet, for critical applications such as financial transactions, which require low bandwidth and allow significant delays, we must resort to specialized, expensive infrastructure. *Can we use the unreliable-but-high-bandwidth connectivity of the Internet, to provide reliable-but-modest-bandwidth requirements of financial and other sensitive applications?*

We answer this question in the affirmative, by presenting QoSoDoS, an end-to-end transport protocol that ensures (modest) Quality of Service, among peers connected via an unreliable, clogging-prone network, which *usually* has much higher bandwidth, such as the Internet. The principle beyond QoSoDoS operation is simple: when QoSoDoS detects frequent packet losses, it begins sending each packet many times, until the packet is received correctly by the destination. Assuming a known bound on packet loss probability, and target for allowed probability of loss (by application), we can easily calculate the required number of retransmissions. This mechanism assumes that even under clogging attack, we can bound the probability of successful transmission; we have experimentally validated this assumption. This approach stands in contrast to existing reliable transport protocols such as TCP [4], which responds to packet loss by reducing its window (and rate), and even aborting the connection.

Therefore, QoSoDoS goal is to maintain crucial communication, even when the underlying network is clogged. This is complementary to existing research on clogging and other DoS attacks, which mostly attempts to prevent the clogging (or other attack), identify the attack and/or its source, and so on; see [1] for a taxonomy of DDoS attacks and defenses, and discussion of related work in Section V.

An important concern is that QoSoDoS must not clog the network itself, by sending more packets than the network can handle, esp. as a response to momentary congestion (or attack). Our experiments show that even when many QoSoDoS connections are sent over the same bottleneck link, they do not clog the link, and have comparable performance to 'regular' TCP connections. Furthermore, even large number of QoSoDoS clients, recover quickly from short-lived clogging attacks, even with better performance than TCP connections.

The rest of the paper is organized as follows. Section II describes the QoSoDoS model. Section III presents the design of QoSoDoS. In section IV we present our experimental evaluation of QoSoDoS. In Section V we discuss related work, and in Section VI we conclude, with discussion of future directions.

II. The QoSoDoS Model

Much of the research on networking considers one of two very different models: *QoS (Quality of Service) networks* and *DoS/BE (Denial-of-Service prone, Best Effort) networks.*

QoS networks are expected to always ensure some Quality of Service guarantees, e.g., no losses; DoS/BE networks are usually modeled as completely unreliable, i.e., can even drop all packets.

We believe, however, that it is reasonable to regard most DoS/BE (Best-Effort) networks, e.g. the Internet, as assuring QoS properties, but only with very *low* probability P_D of packet delivery, i.e. $0 \lesssim P_D \ll 1$. We believe - and confirmed it experimentally as reported later on - that the delivery probability P_D holds even when the connection is undergoing a clogging DoS attack.

Furthermore, we believe that in reality, packet losses are possible also with QoS networks; of course, in QoS networks, packets are delivered correctly with very high probability P_Q. Nevertheless, $P_Q < 1$, namely, there is some probability, such as for equipment failure, where packets get lost; i.e. $0 \ll P_Q \lesssim 1$.

To make a DoS/BE network assure packet delivery with high probability (P_Q), we may need to retransmit packets. Each time we resend a packet, increases the probability for successful delivery. Generally, if a packet is sent n times, and the delivery probability for each transmission is always (independently) P_D, then the probability that at least one copy of the packet is delivered is $1-(1-P_D)^n$. Hence, to ensure packet delivery with high probability P_Q, we need to resend the packet n times, where

$$n = \lceil \log_{(1-P_D)}(1-P_Q) \rceil \quad (1)$$

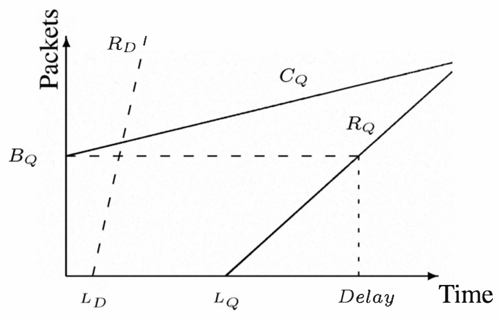

Fig. 1. QoSoDoS delay analysis. L_D and R_D (dashed) are the DoS/BE network's service latency and rate respectively, which are assured in low delivery probability (P_D). L_Q and R_Q are the latency rate assured parameters in high delivery probability (P_Q). Packets are transmitted in an average rate of at least $R_Q = \frac{R_D}{\alpha \cdot n}$, where $\alpha \geq 1$ is a transmission rate relaxation parameter, and $n \gg 1$ is the number of required packet retransmissions to assure packet delivery in probability P_Q. B_Q and $C_Q \leq R_Q$ are the assured leaky bucket parameters.

QoSoDoS is designed to implement a (high reliability) QoS network service over a (poor reliability) DoS/BE network service. We model both QoS and DoS/BE network services as *rate-latency (LR) servers* [5]. For DoS/BE, we assume maximal latency L_D and minimal rate R_D. For QoS, we require QoSoDoS to ensure maximal latency $L_Q \geq L_D$ and minimal rate $R_Q \geq R_D$. In addition, in DoS/BE we assume (low) delivery probability P_D ($0 \lesssim P_D \ll 1$), i.e. high probability for packet loss, and in QoS we require (high) delivery probability P_Q. The rate we can assure cannot be any larger than transmitting n packets at the maximum rate, R_D, i.e. $R_Q \geq \frac{R_D}{n}$, or:

$$R_Q = \frac{R_D}{\alpha \times n}, \ \alpha \geq 1$$

To bound the delay, QoSoDoS can limit the arrival process of packets received to be sent. Specifically, QoSoDoS implements a *Leaky Bucket arrival curve* [5]. The Leaky Bucket model describes a bucket, with a capacity for holding up to B units and a leaking hole which leaks units in a rate of C units every time unit. Flow into the bucket, in rate higher than C, fills the bucket up to its capacity - B. Any additional packets are poured out, i.e. discarded. In the network QoS analogy, C_Q represents the rate by which packets can be transmitted over the network, and B_Q represents the burstiness, i.e., the number of packets that can be queued for deferred transmission. The B_Q parameter affects two things: the buffer needed for packet transmission and the delay for the *entire* burst delivery. Specifically, the delay for leaky-bucket traffic is bounded by $Delay = L_Q + \frac{B_Q}{R_Q}$. In Table I we compare the DoS/BE and QoS parameters.

Parameter	DoS/BE	QoSoDoS
Latency	L_D	$L_Q \geq L_D$
Rate	R_D	$C_Q \leq R_Q = \frac{R_D}{\alpha \times n}, \ \alpha \geq 1$
Packet delivery probability	$0 \lesssim P_D \ll 1$	$P_D \ll P_Q \lesssim 1$

TABLE I
COMPARISON BETWEEN DOS/BE (BEST-EFFORT) AND QOSODOS LATENCY-RATE SERVER AND PACKET LOSS PARAMETERS (SEE FIGURE 1). $n = \lceil \log_{(1-P_D)}(1-P_Q) \rceil$ IS THE NUMBER OF RETRANSMISSIONS REQUIRED TO ASSURE SUCCESSFUL DELIVERY IN HIGH PROBABILITY P_Q WHILE TRANSMITTING PACKETS IN LOW DELIVERY PROBABILITY P_D.

III. QoSoDoS Design

A. Basic design: use of ART and TCP

Congestion and clogging attacks are relatively rare events; most of the time, the network delivers packets with high probability. Therefore, in design of a new transport mechanism, it is critical to ensure good performance in this typical, benign scenario. One desirable goal, would be to obtain comparable performance to that of TCP [4], which is by far the most established reliable transport protocol, and is also very efficient.

To achieve this goal while keeping QoSoDoS as simple as possible, we simply use TCP to send all packets whenever possible, i.e. initially, and whenever QoSoDoS determines that packet loss probability is sufficiently low. However, QoSoDoS sets a timer for TCP sending; if timeout occurs, and it appears that TCP cannot transmit data, then QoSoDoS suspends TCP usage and begins using a novel *ART (Automated Redundant*

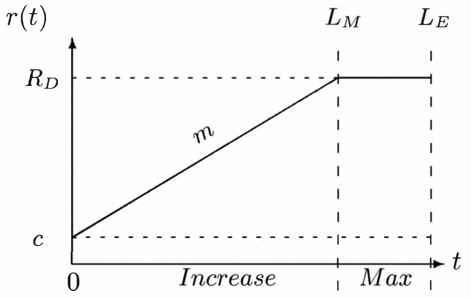

Fig. 2. 'Linear' Automated Redundant Transmission (ART) algorithm, in which the rate, $r(t)$, increases linearly up to the maximal rate - R_D. $L_E = {R_Q}^{-1}$ is the retransmission deadline, by which all n retransmissions must be made (see Section III-B). L_M is the point in time from which the algorithm sends in rate R_D. c is the initial transmission rate. m is the rate increment slope. The rate as a function of time is $r(t) = \min(m \cdot t + c,\ R_D)$

Transmission) mechanism, over a connectionless protocol such as UDP.

The ART mechanism is an important component of QoSoDoS, automatically retransmitting each packet, up to n times (as per Eq. 1), until the packet is delivered to the destination. ART mechanism should be contrasted with the well known class of ARQ (Automated Repeat reQuest) algorithms, which are designed for more benign environments with lower loss probability; hence, ARQ algorithms send each packet only once until detecting delivery or loss, while ART algorithms send multiple copies of a packet, before receiving indication of loss or delivery.

ART can use different algorithms to schedule retransmissions of a packet. Figure 2 depicts a 'linear' ART algorithm, which begins by sending a small number of copies, and gradually increasing the rate of redundant transmissions. In this paper, we will only use the 'linear' ART algorithm; in the future, we plan to compare it to other possible ART algorithms, as the choice of ART algorithm can have significant impact on the performance of QoSoDoS.

In addition, we require a mechanism to identify when the DoS attack is over, and resume TCP. Next section describes how QoSoDoS is facilitated with the capability to decrease ART transmission rate and resume TCP. To that end we define two additional parameters, R_E and P_E, which denote the *effective* rate and delivery probability respectively, by which the congestion controlled TCP[6] (or other protocols, such as DCCP[7]) would transmit packets, when not under DoS attack. R_E is set by the protocol as a function of the measured P_E. Ideally $P_E \lesssim 1$. If P_E decreases, a TCP friendly protocol is assumed to decrease its R_E respectively, thereby increasing P_E back into a stable state of $P_E \lesssim 1$. On the other hand, during high congestion and DoS attacks, TCP cannot provide any assurance for packet delivery. TCP's behavior of reducing the rate to practically zero, does not achieve the desired effect. In such cases, QoSoDoS have a distinct advantage over TCP as it assures communication to the congested destination host.

B. Lowering QoSoDoS's Transmission Rate By Using TCP

A fundamental requirement from QoSoDoS is that it must refrain from self created DoS, i.e. QoSoDoS clients must refrain from aimlessly congesting the network, especially when no attack is present. We therefore facilitate QoSoDoS with such mechanisms, preventing it from self-initiating DoS attacks, as well as recovering from the high rate transmission of ART. In this section we describe both the usage of TCP within the QoSoDoS core design, as well as relaxing the use of ART's high rate transmissions back to TCP.

As stated in section III-A, QoSoDoS prefers transmitting messages using TCP rather than ART. To achieve this, and still maintain the assured QoS, QoSoDoS uses a TCP user-timeout, which we refer to as L_T. L_T serves as an indicator of the need to suspend TCP and begin transmitting in ART scheme. L_T limits the time between a TCP packet transmission and its successful delivery. L_T must be large enough to allow at least the transmission of a single packet, taking into consideration an occasional packet loss and the need for retransmission. Hence, L_T must have an order of *at least* a few RTTs. As we want QoSoDoS to begin with TCP we refine L_Q (the assured latency) definition to include L_T, hence:

$$L_Q = L_D + L_T$$

A reasonable value for L_T should be around a few seconds. Larger values allow TCP to perform under some states of congestion. Nevertheless, a too large value will prolong L_Q, as well as the recovery time after an attack as described in Section III-D.

To achieve the design goal of relaxing the high retransmission rate, we design QoSoDoS as follows. Instead of assuring a rate based on transmitting all n packets at the highest rate R_D, thereby minimzing the time for packet acceptance, we relax the time requirement and allow packets to be transmitted over a longer period of time, yielding a lower assured rate of $R_Q = \frac{R_D}{\alpha \times n}$, $\alpha > 1$. Finally, we define L_E, a single packet's retransmission deadline (i.e. retransmission *end* time) as:

$$L_E = {R_Q}^{-1}$$

C. Lowering Average Transmission Rate of ART Algorithms

As discussed in Section III-A, reducing the average transmission rate down to R_E, should cause most packets to be delivered. As we cannot determine R_E a priori, we split each single packet transmission in ART algorithm into two periods, namely *increase* and *maximal*, similar to the depiction in Figure 2. In the increasing period, ART increases transmission rate *starting at very low rate*, such as $c = RTT^{-1}$, up to R_D. Next, ART should continue transmitting at the maximal allowed rate - R_D - until the deadline L_E is reached.

The average transmission rate produced by ART algorithms can be calculated as $\frac{N}{T}$ where N is the expected number of retransmitted packets and T is the expected transmission time. If, for simplicity, we assume transmission terminates immediately upon success, then N and T can be approximated

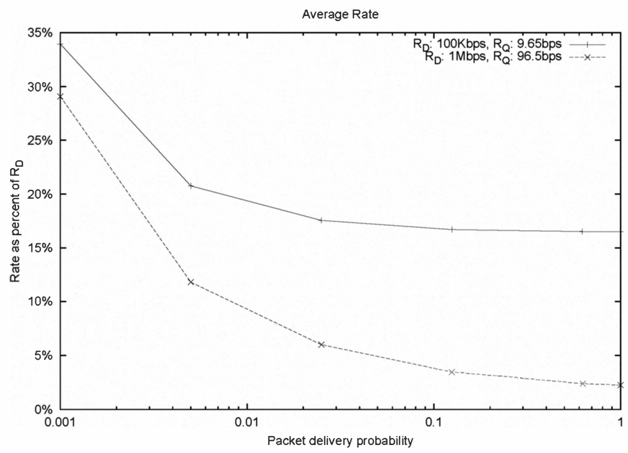

Fig. 3. Average rate (as percent of R_D) during a single ART transmission vs. delivery probability (log scale) using the 'linear' ART algorithm (see Figure 2). Both lines were produced numerically and took into consideration the additional RTT between the accepted packet transmission and the ceasing of retransmissions. The parameters were taken from Table II. The top line used $R_D = 100Kbps$ and the bottom line used $R_D = 1Mbps$. The average rate when $P_E = P_D$ is $\sim 1/3$ of R_D. When $P_E = 1$, the rates are $\sim 16Kbps$ and $\sim 22Kbps$, which correlates to 7-10 packets per second. Note that in most cases QoSoDoS will try to resume TCP between ART transmissions (see Section III-D), therefore further reducing the minimal average rate.

by:

$$N = \sum_{x=1}^{n} (1 - P_D)^{x-1}, \quad T = \sum_{x=1}^{n} (1 - P_D)^{x-1} \cdot art(x)$$

where $art(x)$ is the time interval added for packet x transmission. E.g. for the 'linear' ART (see Figure 2), $art(x)$ can be extracted from $\int_{art(x-1)}^{art(x)} (\min(m{\cdot}t{+}c,\ R_D))\,dt = 1$, $art(0) = 0$. Figure 3 depicts the average QoSoDoS client transmission rate vs. probability during ART.

Having low packet retransmission rate at the beginning of each packet's transmission, helps QoSoDoS recover from high congestion periods. Based on the results presented in Figure 3, Figure 4 depicts a theoretical macro view of multiple QoSoDoS clients in a DoS attack scenario where TCP transmissions are aborted by QoSoDoS, followed by ART transmissions and recovery period until TCP is resumed. Figure 5 depicts a correlating theoretical micro view of a single QoSoDoS client's multiple packet transmissions within the context of Figure 4. In our experiments (see Section IV) QoSoDoS clients resumed TCP almost immediately after the attack has stopped.

There are two reasons for the actual rate decrease. First reason is that the DoS attack (or high congestion) is over, which implies an increase in packet delivery probability. Second reason is QoSoDoS's retransmission behavior. When a DoS attack is over, the remaining congestion is created by legitimate clients, due to QoSoDoS's ART. This means that most, if not all, packets that reach the destination host are legitimate. Hence, even in the worst case, where all legitimate clients are transmitting packets at rate R_D, one legitimate packet gets an acknowledgement by the destination host. Consequently, at least one client drastically reduces its retransmission rate for its next packet transmition. This probabilistically assures, that the next accepted packet is a packet transmitted at higher rates than others. This attribute makes the client reduce its transmission rate for the next packet transmission. In general, the faster a packet retransmission rate is, the higher its accetptance probability is, therefore lowers the average retransmission rate for the following packet.

To simplify our discussion we regard the a model as depicted in Figure 6. Assuming that all legitimate clients are similar, we can bound the successful delivery probability as follows. The worst case is when all clients transmit packets in R_D, hence have the same probability for being queued by the network's routers, hence have the same probability for being forwarded to the destination. If m clients are transmitting at R_D, $P_E \geq \frac{m \times R_E}{m \times R_D} = \frac{R_E}{R_D}$. The number of packets needed to assure transmission is therefore, $n' = \log_{(1-P_E)}(1 - P_Q)$. We assume $P_E \gg P_D$, hence $n \gg n'$. This suggests that much less packets need to be transmitted in order to assure delivery, and therefore much less time passes until packet acceptance. This correlates to results presented in Figure 3, which suggest that the average rate will indeed be reduced when the attack is over, down to a minimum congestion which is a function of the number of QoSoDoS clients. Therefore as long the number of QoSoDoS clients is policed, self-DoS can be avoided.

Finally, we conclude that ART's average rate should be carefully controlled by making sure that $L_E \gg \frac{n}{R_D}$ and $L_E \gg RTT$ to avoid high minimal transmission rates.

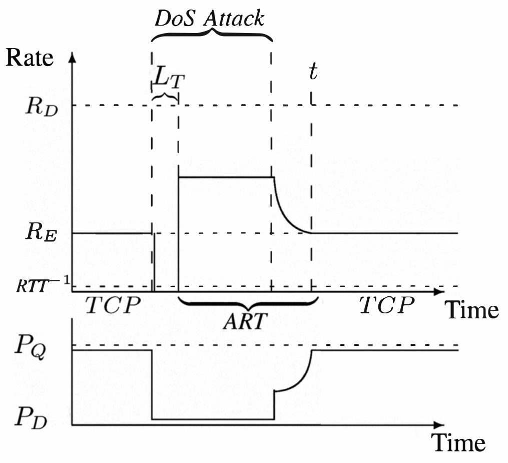

Fig. 4. QoSoDoS average rate vs. time (theoretical macro view based on Figure 3). Top figure represents rate vs. time of multiple QoSoDoS clients and the bottom figure is a corralating single packet's effective delivery probability (P_E). In the top figure, the solid line represent the average rate by which legitimate clients would transmit packets. R_E is the effective average rate by which TCP would transmit packets under congestion control (w/o attack). At the beginning of the DoS attack TCP practically stops transmitting data. L_T is QoSoDoS's TCP-timeout interval, which, when expires, results in the abortion of TCP and initiation of ART transmission. When the DoS attack is over, QoSoDoS starts a recovery period. At time t QoSoDoS resumes TCP.

D. Resuming TCP/UDP

Now that we have defined the ART scheme, and utilized it to decrease the transmission rate we describe when TCP can

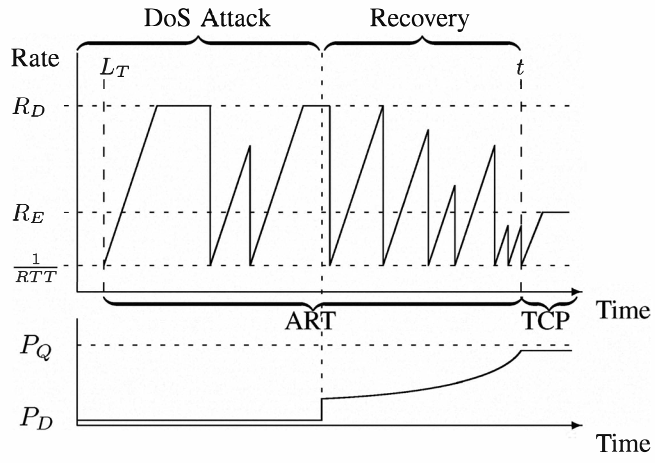

Fig. 5. QoSoDoS multiple packet transmission rate over time (theoretical micro view based on Figure 3). Top figure represents rate vs. time of a single QoSoDoS client and bottom figure represents a corralating single packet's effective delivery probability (P_E). In the top figure, the solid line is the rate by which legitimate clients transmit packets. At L_T QoSoDoS aborts TCP transmission and intiates ART. R_E is the rate by which TCP would have transmitted (w/o attack). Each distinct climb is a different packet transmission (see figure 2). At time t QoSoDoS resumes TCP.

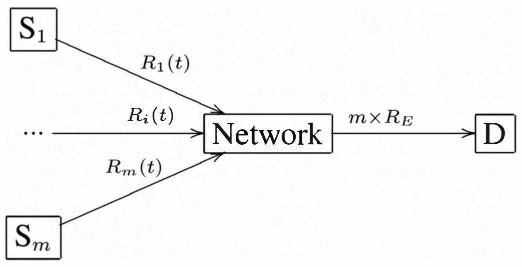

Fig. 6. A simplified transmission model from multiple senders to a single destination. $S_1..S_m$ represents m senders, D represents the destination, $R_i(t) \leq R_D,\ 1 \leq i \leq m$ represents the $i'th$ sender's transmission rate at time t and $m \times R_E$ represents the available rate by which the network transmits packets to the destination host.

be safely resumed while still assuring QoS. Before resuming TCP, QoSoDoS must assure that it has enough time for both the re-suspention of TCP, in case it fails or timeouts, and re-initiation of ART, without harming the assured QoS. In order to do so QoSoDoS inspects the assured parameters, and the actual packet acceptance state. It compares the current time and the next packet's guaranteed time of delivery. This is done by inspecting the next packet in the burst, if such exists, or the minimal time needed for transmission of a potential newly arriving packet. If the time until the guaranteed time of delivery for the next packet is greater than the time needed for re-suspending TCP (L_T) and its transmition using ART (L_E), then QoSoDoS can safely try to resume TCP, otherwise, it must continue transmitting in the ART scheme, so that the assured QoS will not be jeopardized. If there are no packets waiting for transmission, i.e. no burst, we can resume TCP immediately, as $L_Q = L_D + L_T$, and enough time exists between next packet arrival and the QoS commitment.

Finally, the mere *trying* to resume TCP is helpful in reducing the average QoSoDoS rate cf. to ART rate (see Figure 3), since it adds an interval L_T, in which very few packet are sent.

E. QoSoDoS: The Big Picture

QoSoDoS basically works in two modes derived from the state of the network, namely low congestion mode and high congestion mode. In low congestion mode, QoSoDoS uses TCP as is, thereby taking advantage of TCP mechanisms such as congestion and flow control. In high congestion, in which the probability for packet delivery drops below a point where TCP practically cannot function properly, QoSoDoS switches to high congestion mode.

The principle on which QoSoDoS relies during high congestion periods is that successful packet delivery is essentially statistical, even when a flooding DDoS attack is launched. During an attack, the attacker tries to clog the victim host's bandwidth by transmitting numerous futile packets and thus preventing legitimate packets from reaching that host, therfore most (legitimate) packets sent to the host are dropped. Nevertheless, probabilistically, once in a while a legitimate packet should come through and reach its destination. During an attack, the probability of packet successful delivery (assuming independence, and same packet size) drops down to about $P_D \sim \frac{R_N}{R_A+R_C}$, where $R_N,\ R_C,\ R_A$ are network, (legitimate) client and attacker rates respectively. The analysis details will be presented in the full version.

Following is a numerical example. Assume $R_N = R_C \cdot 100$ and $R_A = R_C \cdot 999\,999$, hence $P_D = 0.0001$. Assume C wants its packet delivered with $P_Q = 99\%$ certainty, hence C needs to retransmit the packet $n = \lceil \log_{(1-P_D)}(1 - P_Q) \rceil = 46\,050$ times. If C can transmit $R_D = 1\,000$ packets/second, C can assure delivery within $L_E = 46.05$ seconds. As C wants to refrain from self-created DoS, it will start transmission using TCP. After some time, say $L_T = 3$ seconds, C aborts TCP, and starts using ART. By relaxing the average ART rate and setting $\alpha = 1.5$, all n retransmissions end by $L_E = \frac{46\,050 \times 1.5}{1\,000} = 70$ seconds. Assuming independence, the packet sent by C should get accepted after an expected $\frac{1}{P_D} = 10\,000$ retransmissions, implying much shorter latency than 70 seconds, and higher expected assured rate.

F. Algorithm Design

Algorithm 1 describes a transmission algorithm, based on the QoSoDoS model. Algorithm 1 uses a general ART transmission algorithm, which can be implemented by the algorithm depicted in Figure 2, or by other conforming algorithms.

IV. Experimental Evaluation

We have implemented an initial version of QoSoDoS and used it to test our assumptions regarding a heavily congested network similar to a DDoS attack. We have setup a network in a Dumbbell topology as depicted in Figure 7, in which we emulated various attack sizes and compared QoSoDoS with TCP. The network topology is constructed so that all transmissions towards the destination host D congest the same link.

```
Input:
    $R_D$, $L_D$, $P_D$ - Best effort LR-Server parameters
    $R_Q$, $L_Q$, $P_Q$ - QoSoDoS LR-Server parameters
    $L_T$ - TCP deadline
    $Queue$ - Transmission Queue
Data:
    $n \equiv \lceil \log_{(1-P_D)}(1-P_Q) \rceil$ (max retransmissions)
    $L_E \equiv {R_Q}^{-1}$ (Single packet's deadline)
    $State$ - $TCP$ or $ART$
    $Pkt$ - Packet to transmit
    $t_{art}, t_p$ - ART and per $packet$ timers
    $i$ - ART transmissions counter (cf. $Queue$)
    $j$ - Single packet retransmission counter (cf. $n$)
$State \leftarrow TCP$;
while True do
    if $Queue \neq \varphi$ then
        $Pkt \leftarrow Dequeue(Queue)$;
        if $State = TCP$ then
            $t_p \leftarrow time()$;
            Async-TCP-Transmit(P);
            Wait for TCP abort or $Pkt$ ACK or
            $(time() - t_p) = L_T$;
            if $Pkt$ not ACKed then
                Abort TCP connection;
                $State \leftarrow ART$;
                $t_{art} \leftarrow time()$;
                $i \leftarrow 1$;
            end
        end
        if $State = ART$ then
            $j \leftarrow 0$;
            $t_p \leftarrow time()$;
            while $Pkt$ not-ACKed and $j < n$ and
            $(time() - t_p) < L_E$ do
                UDP-Transmit(Pkt);
                $j \leftarrow j + 1$;
                sleep($art(j)$);
            end
            if $Pkt$ not-Acked then
                Notify: Failed to deliver packet;
            end
            $i \leftarrow i + 1$;
            if $(time() - t_{art}) + L_T + L_E < i \cdot L_E$ then
                Try establishing a TCP connection ;
                if TCP connection established then
                    $State \leftarrow TCP$;
                end
            end
        end
    end
end
```

Algorithm 1: QoSoDoS's multiple packet scheduling algorithm. $art(j)$ is an ART algorithm such as the 'linear' algorithm depicted in Figure 2. See Section III-C.

All the machines, including routers, are Linux machines with kernel version $\geq$ 2.6.18. Both clients and server used Ubuntu 10.04 with the default TCP congestion control (CUBIC[8]). Each machine has 100Mbps Ethernet NIC(s). R_0 and R_1 are routers with three and two Ethernet interfaces respectively. S_1 and S_2 connect to the first NIC of R_0, whereas S_3 and S_4 connect to the seconds. The third NIC of R_0 is connected to the first NIC of R_1. The second NIC of R_1 is connected to D using a rate-limiter rating it to 10Mbps with buffer size of 750Kbps.

Each source node consists of several QoSoDoS clients and at least twice as many attackers with higher scheduling priority, transmitting as many packets as required to consume enough bandwidth according to the attack scenario. Each client transmits packets with a payload of 262 bytes which emulates 12 bytes QoSoDoS header and 250 bytes of payload. All attackers transmit a 12 byte payload over UDP, hence producing a minimal Ethernet packet with a size of 64 bytes. Such a packet size is typical to TCP packets carrying no payload, such as SYN, ACK, FIN and RST. Motivation for having such a small attacker's packets size is to give the attacker's packets better queuing probability in routers.

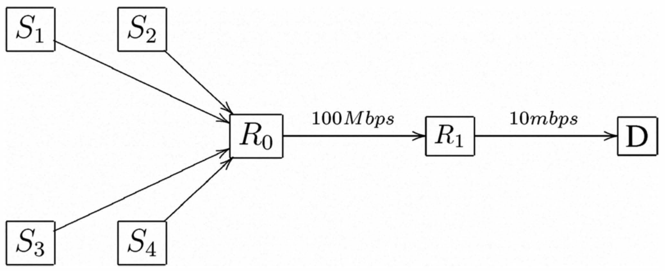

Fig. 7. Experiment setup. $S_1..S_4$ are sources of packets containing both QoSoDoS clients and attackers. Each QoSoDoS client is configured to use no more than 100Kbps when executing QoSoDoS-ART scheme (see Table II). The amount of packets and bandwidth created by the attackers changes to emulate various probabilities for QoSoDoS's successful packets delivery. R_0 is a router with a 100Mbps rate connected to router R_1 which limits the rate towards D to 10Mbps. Each single source does not transmit any more than 40Mbps so that the underlying Ethernet links won't affect the results.

A. Various Attack Sizes

On each source machine, $S_1..S_4$, we have executed six concurrent QoSoDoS clients with parameters as described in Table II. In addition, we have executed 14 attackers on each machine which produce the bandwidth-flooding of the link.

Each experiment was executed as follows. 30 seconds after the clients were all executed, an attack was launched for 30 minutes, followed by 60 seconds for clients recovery and test whether TCP was resumed properly. For all attack sizes we examined, all the QoSoDoS clients resumed TCP almost immediately.

We have tested attack sizes ranging from 10Mbps to 150Mbps, which provided an effective acceptance probability (P_E) ranging from 4.5% to 0.3% as described in Figure 10.

Figure 8 and 9 present the average latency and rate (respectively) vs. packet acceptance probability. As expected, packets were accepted at lower latencies and higher rates than

Parameter	Description	Value
L_D	Network max latency	75ms
L_Q	Assured latency	3s
L_T	TCP (user) timeout	2.925s
R_D	Client max rate	100Kbps
P_D	Packet delivery probability	0.1%
P_Q	QoS delivery probability	99.9%
α	QoSoDoS Relaxation Parameter	1.5
n	Required number of retransmissions	6905
R_Q	QoSoDoS assured rate	9.65 bps
B_Q	QoSoDoS assured burst	10Mbits
L_E	QoSoDoS packet timeout	255.2s

TABLE II
QOSODOS'S PARAMETERS USED IN EXPERIMENTS.

assured by QoSoDoS, since the packets' expected number of retransmissions until delivery is $\frac{1}{P_E}$.

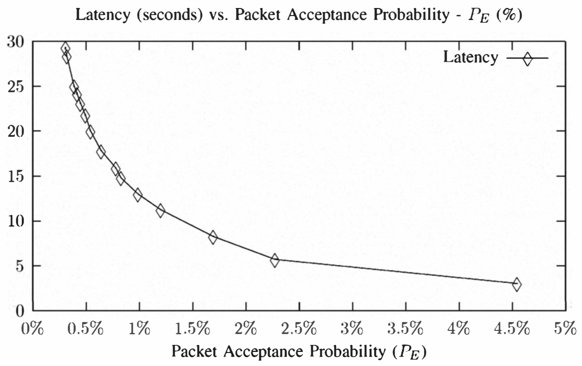

Fig. 8. Latency (seconds) vs. acceptance probability (P_E). The effective latency ranges between 29.3 and 3 seconds. Note that the latency values are lower than the assured value L_E (see Table II), as the average packet is accepted by the mean value of the effective probability ($\frac{1}{P_E}$), producing *much lower* latencies than the assured worst case.

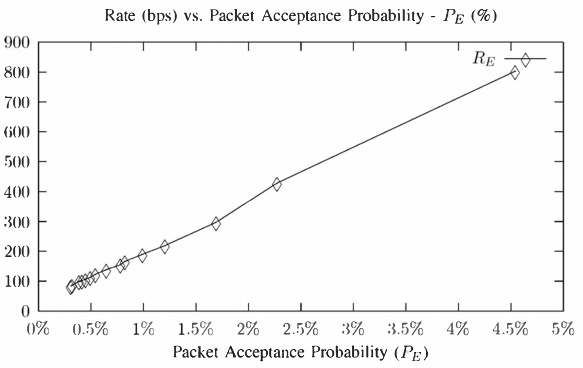

Fig. 9. Rate vs. acceptance probability (P_E). The effective rate ranges between 803 bps and 84 bps. The rate values are much higher than the assured R_Q (see Table II), as the average packet is accepted by the mean value of the effective probability ($\frac{1}{P_E}$), producing *much higher* rates than the assured worst case.

B. QoSoDoS and TCP Comparison

We conducted two experiments to compare TCP and QoSoDoS. In the first experiment we tested the goodput of TCP

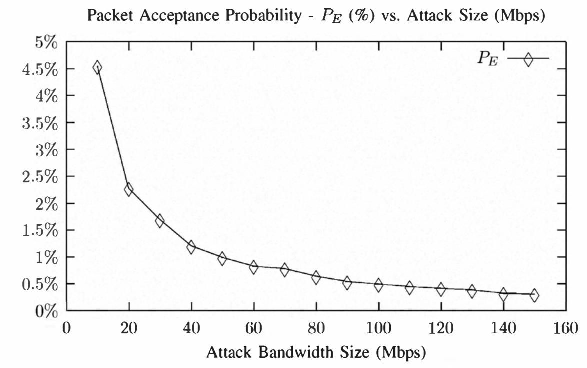

Fig. 10. Effective packet acceptance prob. (P_E) vs. attacker's bandwidth. Even for a strong attacker (×15 link bandwidth) the acceptance probability is 0.3%.

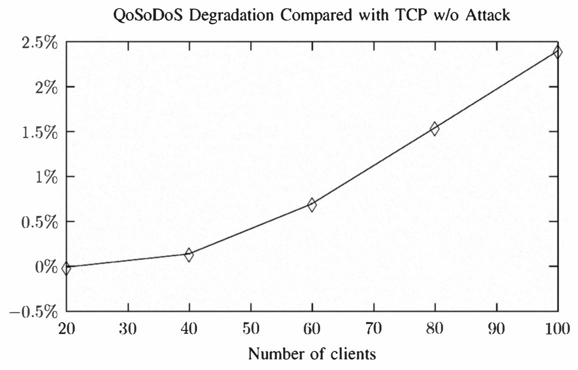

Fig. 11. QoSoDoS degradation compared with TCP. The X-axis is the number of concurrent clients and the Y-axis is the performance degradation ranging between 0 and 2.5%. QoSoDoS performs quite well even when no attack is launched. This seems like a reasonable price for assured QoS during flooding DoS attacks.

vs. the goodput of QoSoDoS. We have executed the same number of clients for 5 minutes and tested how much data was delivered using TCP and much data was delivered using QoSoDoS. For each QoSoDoS client we used the same configuration as described in Table II. Figure 11 shows that using up to a 100 concurrent clients, QoSoDoS performs within 2.5% of TCP, which seems like a reasonable price to pay for the assured QoS. In Figure 12 we show that the amount of QoSoDoS packets using ART (rather than TCP) is negligible; less than 0.12% (about one promil) of the accepted packets were ART packets when hundred QoSoDoS clients were executed concurrently. This demonstrates that QoSoDoS does not self-create DoS even when many clients run concurrently.

The second experiment we have conducted to compare QoSoDoS and TCP included a short lived attack. Like the previous experiment, we start without an attack. After 60 seconds of running TCP, we launch a 10Mbps attack (P_E = 4.5%) which lasts 40 seconds. After the attack is over we continue running the experiment for additional 200 seconds (2.3 minutes). This kind of attack is intended to simulate flash crowd, or an attacker with low and/or short lived capabilities. Low-rate attacks[9] can be regarded as an extreme case for

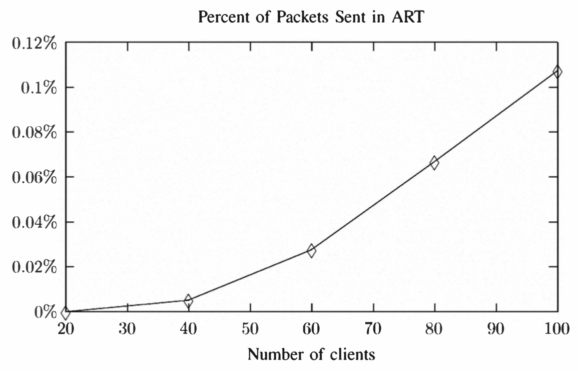

Fig. 12. Percent of packets sent in ART. The X-axis shows the number of concurrent clients and the Y-axis shows the percentage of packets accepted using ART. The values range between 0 and 0.12%, i.e. ART overhead is negligible. Note that this means that QoSoDoS has almost no overhead, adds only negligible amount of non-TCP traffic, and does not become source of congestion, let alone self-created DoS.

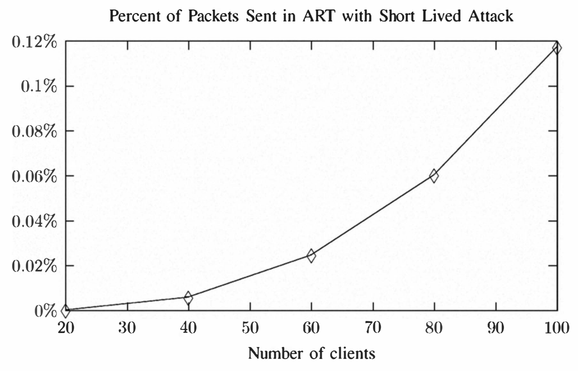

Fig. 14. Percent of packets sent in ART with short lived attack. Examining the percent of ART packets in the context of a short-lived attack yields that ART presents almost no overhead. This is comparable with the result in Figure 12. This result further supports the claim that QoSoDoS does not become a source of self-created DoS.

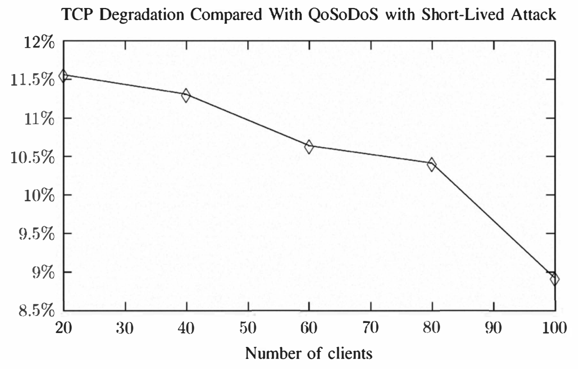

Fig. 13. TCP performance degradation compared with QoSoDoS, under a short-lived attack during a connection. *TCP performs worse than QoSoDoS* in the range of 11.5% to 9%. We believe that this phenomenon is due to QoSoDoS's ability to recover faster than TCP from DoS attacks, which is mainly the result of its TCP timeout mechanism L_T.

this type of an attack, which are very hard to detect and filter. In this case, *QoSoDoS performs better than TCP*. Figure 13, shows *TCP degradation* compared with QoSoDoS. In small number of clients QoSoDoS performs 11.5% better than TCP. This is reduced to about 9% when the number of clients is increased to a 100. Figure 14 shows that the overhead of QoSoDoS-ART packets remains almost as small as without an attack (about one promil). These results suggests that QoSoDoS recovers very well from such small sized and/or short lived attacks which targets TCP timeouts and recovery time. We believe that the main reason behind these results is mainly due to TCP timeout - L_T - which helps the fast recovery when an attack is over. The results presented in Figure 14 supports our claim that even after a DoS attack, QoSoDoS does not self create DoS.

V. Related Work

Extensive work has been carried out to mitigate various flooding DDoS attacks. We focus on network-flooding and mostly survey works done in that area. The lion's share of papers focus on varios ways to filter out attack traffic, and as far as we know none has taken our approach. Some publications try to identify special characteristics of an attacking packet such as IP spoofing. For example, in [10] the authors proposed a spoofing filter based on hop-counts differntiation between spoofed and real packets. Ingress filtering[11] aims to block spoofed packets at the source ISP, by identifiying that the spoofed packet cannot have originated from that ISP. LOT[12] tunnels packets between two hosts using a nonce, preventing non-MITM attacker from spoofing packets between these hosts. This type of mechanisms can not filter out packets that are not subverted, such as various zombie based attacks.

Pushback[13] schemes such as ACC[14], AITF[15] and StopIt[16] are core-based mechanisms trying to prevent DoS attacks by blocking attack streams close to attack source. This is done by identifying the attack near destination and propogating a block request upstream towards attack source(s).

Capabilities schemes such as TVA[17] and SIFF[18] propose a DoS limiting network architecture in which destination and routers use secure capability information, indicating the destination's willingness to receive information from the source. Packets with capabilities are prioritized over other packets and the entire architecture allocates bandwidth fairly to avoid various DoS attacks on the capability mechanism. SIFF, like QoSoDoS, builds on the idea that the probability that a packet gets through is non-zero, even when the destination is under attack.

Overlay schemes such as SOS[19], Mayday[20] and Phalanx[21] use an overlay architecture, which hides the destination host location using several layers. The overlay nodes verify the source client and forward packets to the hidden server via selected nodes. If any overlay node misbehaves it is forced out of the overlay.

Rate limiting schemes collect traffic statistics and shape traffic accordingally during an attack. D-WARD[22] rate-limit at the source-end. PSP[23] collect traffic statistics in core routers between OD (origin-destination) pairs and shape traffic during an attack, which might harm some of the clients which use the same OD route.

dFence[24] introduces middle-boxes into the ISP core, which upon DDoS activity, transparently add DDoS defense to victim servers. DefCOM[25] introduces a core-framework which enables cooperation between source, core and destination defenses during an attack, by using both shaping and pushback schemes.

Application-level DDoS defense such as Speak-Up[26], requires legitimate source host to "pay" for the service using bandwidth. Speak-up uses a *congestion controlled* bandwidth as the currency, and make an auction for each request before passing it to the server. We argue that during a flooding network-level DoS attack, on which our work is focused, Speak-Up, as well as other application level defences, will not be able to perform at all.

QoSoDoS is different from the above as it is end-to-end solution, requiring no changes to network core and deployed merely at connections' end hosts.

VI. Conclusions

In this paper we have presented QoSoDoS, an end-to-end transport protocol assuring QoS over a DoS prone networks. Experimental results show that QoSoDoS has good performace during large scale network flooding attacks. Compared to TCP under normal conditions QoSoDoS presents little overhead, whilst during and after an attack its performace is superior. QoSoDoS does not self create DoS attacks and recovers well after such attacks. In future work we plan to investigate various QoSoDoS ART algorithms and the QoSoDoS model itself. In addition we plan to further test QoSoDoS over various topologies and attacks, using large scale testbeds.

VII. Acknowledgments

We would like to thank Yehoshua Gev for his useful comments regarding this paper. We also thank the anonymous reviewers for their insightful ccomments which helped improving this paper.

References

[1] J. Mirkovic and P. L. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," *Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.

[2] D. Grossman, "New Terminology and Clarifications for Diffserv," RFC 3260 (Informational), Internet Engineering Task Force, Apr. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3260.txt

[3] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture," RFC 3031 (Proposed Standard), Internet Engineering Task Force, Jan. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3031.txt

[4] J. Postel, "Transmission Control Protocol," RFC 793 (Standard), Internet Engineering Task Force, Sep. 1981, updated by RFCs 1122, 3168. [Online]. Available: http://www.ietf.org/rfc/rfc793.txt

[5] J.-Y. L. Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, ser. Lecture Notes in Computer Science. Springer, 2001, vol. 2050.

[6] M. Allman, V. Paxson, and E. Blanton, "TCP Congestion Control," RFC 5681 (Draft Standard), Internet Engineering Task Force, Sep. 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5681.txt

[7] E. Kohler, M. Handley, and S. Floyd, "Datagram Congestion Control Protocol (DCCP)," RFC 4340 (Proposed Standard), Internet Engineering Task Force, Mar. 2006, updated by RFCs 5595, 5596. [Online]. Available: http://www.ietf.org/rfc/rfc4340.txt

[8] H. Sangtae, R. Injong, and X. Lisong, "Cubic: a new tcp-friendly high-speed tcp variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, pp. 64–74, July 2008.

[9] A. Kuzmanovic and E. W. Knightly, "Low-rate TCP-targeted denial of service attacks and counter strategies," *IEEE/ACM Trans. Netw*, vol. 14, no. 4, pp. 683–696, 2006.

[10] C. Jin, H. Wang, and K. G. Shin, "Hop-count filtering: an effective defense against spoofed ddos traffic," in *ACM Conference on Computer and Communications Security*, 2003, pp. 30–41.

[11] P. Ferguson and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing," RFC 2827 (Best Current Practice), Internet Engineering Task Force, May 2000, updated by RFC 3704. [Online]. Available: http://www.ietf.org/rfc/rfc2827.txt

[12] Y. Gilad and A. Herzberg, "Lightweight opportunistic tunneling (LOT)," in *Computer Security - ESORICS 2009, 14th European Symposium on Research in Computer Security, Saint-Malo, France, September 21-23, 2009. Proceedings*, ser. Lecture Notes in Computer Science, M. Backes and P. Ning, Eds., vol. 5789. Springer, 2009, pp. 104–119.

[13] J. Ioannidis and S. M. Bellovin, "Implementing pushback: Router-based defense against DDoS attacks," in *NDSS*. The Internet Society, 2002.

[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.*, vol. 32, pp. 62–73, July 2002.

[15] K. Argyraki and D. R. Cheriton, "Active internet traffic filtering: real-time response to denial-of-service attacks," in *Proceedings of the annual conference on USENIX Annual Technical Conference*, ser. ATEC '05. Berkeley, CA, USA: USENIX Association, 2005, pp. 10–10.

[16] X. Liu, X. Yang, and Y. Lu, "To filter or to authorize: network-layer DoS defense against multimillion-node botnets," in *Proceedings of the ACM SIGCOMM 2008 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Seattle, WA, USA, August 17-22, 2008*, V. Bahl, D. Wetherall, S. Savage, and I. Stoica, Eds. ACM, 2008, pp. 195–206.

[17] X. Yang, D. Wetherall, and T. E. Anderson, "A DoS-limiting network architecture," in *Proceedings of the ACM SIGCOMM 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Philadelphia, Pennsylvania, USA, August 22-26, 2005*, R. Guérin, R. Govindan, and G. Minshall, Eds. ACM, 2005, pp. 241–252.

[18] 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*. IEEE Computer Society, 2004, p. 130.

[19] A. D. Keromytis, V. Misra, and D. Rubenstein, "SOS: an architecture for mitigating DDoS attacks," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 176–188, 2004.

[20] D. G. Andersen, "Mayday: Distributed filtering for internet services," in *USENIX Symposium on Internet Technologies and Systems*, 2003.

[21] C. Dixon, T. E. Anderson, and A. Krishnamurthy, "Phalanx: Withstanding multimillion-node botnets," in *5th USENIX Symposium on Networked Systems Design & Implementation, NSDI 2008, April 16-18, 2008, San Francisco, CA, USA, Proceedings*, J. Crowcroft and M. Dahlin, Eds. USENIX Association, 2008, pp. 45–58.

[22] J. Mirkovic and P. L. Reiher, "D-ward: A source-end defense against flooding denial-of-service attacks," *IEEE Trans. Dependable Sec. Comput.*, vol. 2, no. 3, pp. 216–232, 2005.

[23] J. C.-Y. Chou, B. Lin, S. Sen, and O. Spatscheck, "Proactive surge protection: a defense mechanism for bandwidth-based attacks," *IEEE/ACM Transactions on Networking*, vol. 17, no. 6, pp. 1711–1723, Dec. 2009.

[24] A. Mahimkar, J. Dange, V. Shmatikov, H. M. Vin, and Y. Zhang, "dFence: Transparent network-based denial of service mitigation," in *NSDI*. USENIX, 2007.

[25] G. C. Oikonomou, J. Mirkovic, P. L. Reiher, and M. Robinson, "A framework for a collaborative DDoS defense," in *ACSAC*. IEEE Computer Society, 2006, pp. 33–42.

[26] M. Walfish, M. Vutukuru, H. Balakrishnan, D. R. Karger, and S. Shenker, "DDoS defense by offense," in *SIGCOMM*, 2006, pp. 303–314.