Responsive Aggregate Defence for Denial of Service Attacks

Arkaitz Bitorika, Ciarán Mc Goldrick, Meriel Huggard
Department of Computer Science
Trinity College Dublin, Ireland
{bitorika, ciaran.mcgoldrick, meriel.huggard}@cs.tcd.ie

***Abstract*—Denial-of-service attacks have become a regular occurrence on the Internet. The current architecture of the Internet, coupled with the relative ease with which software bugs can be exploited on Internet-connected hosts, provides a fruitful environment for the creation of malicious traffic attacks. This article proposes a novel DoS defence scheme based on Active Queue Management (AQM) principles. Responsive Aggregate Defence (RAD) provides effective penalisation of flooding DoS attacks at the router queue. RAD combines protection of TCP-like traffic behaviours, with a lightweight architecture that allows scalable implementation on a wide variety of network scenarios.**

I. Introduction

RAD is a router queueing scheme that works in conjunction with existing Active Queue Management approaches to protect responsive TCP traffic from bandwidth-flooding DoS attacks. RAD is designed as an additional queueing mechanism located before the AQM algorithm at the router queue. The RAD architecture comprises an AQM-like design philosophy using relatively coarse grained queueing for scalability to provide protection from flooding DoS attacks. RAD provides aggregate-based active queueing, while the associated AQM scheme manages the queue globally.

II. Literature Review

This section will discuss the current state of the research literature regarding three main areas: Denial of Service attacks, Active Queue Management schemes and analytical TCP modeling. These three subjects come together in the design of the RAD algorithm.

A. Denial of Service Attacks

Denial of service (DoS) attacks are one of the most significant types of malicious traffic on the Internet. DoS attacks can be defined as any attack whose goal is to render the victims services inaccessible to other parties. A DoS attack renders a host, or its network services, unavailable to other hosts. Internet DoS attacks can be split in two categories [1]: *software exploits* and *flooding attacks*. The former take advantage of specific vulnerabilities in the software running at the victim to make it unavailable. The latter send traffic to the victim with the goal of exhausting the bandwidth or computing resources available to it.

Distributed DoS attacks (DDoS) increase the attack intensity and lower the possibility of effective response by using multiple attacking machines, called *zombies* or *bots*. In distributed reflector DoS (DRDoS) attacks [2] malicious hosts send requests with the source address of the victim to "reflector" hosts that respond to requests. These reflectors will flood the victim with response packets for requests that it never made. There are many schemes proposed for attack detection; for instance, MULTOPS [3] focuses on the incoming and outgoing packet rates, WADeS [4] uses an LRU cache filter together with Wavelet variance analysis while Hussain *et al.* [1] use header analysis and ramp-up behaviour. Other researchers focus on discovering the source hosts of the attack traffic, or "traceback" [5].

B. Active Queue Management

Active Queue Management denotes a class of algorithms designed to provide improved queueing mechanisms for routers. These schemes are called *active* because they dynamically signal congestion to sources; either explicitly, by marking packets (e.g. Explicit Congestion Notification [6]) or implicitly, by dropping packets. This is in contrast to Drop-Tail queueing which is passive: packets are dropped if, and only if, the queue is full. The first AQM scheme, RED, [7] estimates the current congestion level on the network by keeping track of the exponential weighted moving average (EWMA) of the queue size. Sources are notified of congestion with a probability that is an increasing function of the average queue length. Many alternative AQM algorithms have been proposed since the IETF recommendation [8], [9].

The more complex, fairness-oriented AQM schemes are not necessarily effective against DoS traffic that can spoof IP packet headers to conceal its presence. Aggregate Congestion Control (ACC) [10] and RED-PD [11] are two AQM proposals that aim to protect responsive TCP traffic from aggressive non-responsive traffic. However, ACC and RED-PD are unable to deal with malicious traffic that spoofs any of the packet headers. There is a clear need for AQM-based algorithms that are effective against malicious spoofed IP traffic.

C. TCP Modelling

TCP is currently the dominant transport protocol on the Internet [12]. For DoS attack defence it is helpful to be able to predict TCP behaviour for DoS attack defence. One indicator of "good behaviour" of traffic on the Internet is the extent to which it can be viewed as being TCP compliant. Thus,

malicious attack traffic may be identifiable, at least to some degree, by comparing it with expected TCP-like behaviour.

TCP modelling has historically focused on the steady-state behaviour of the TCP congestion control algorithm. Mathis *et al.* provide a well known and relative simple TCP model [13], with Padhye *et al.* [14] using a stochastic model of steady-state TCP that offers better accuracy.

Other authors have focused on predicting the latency of finite-length TCP transfers [12] by accounting for connection establishment and closing. In [15] Ehsan *et al.* use a Markov Chain model to more accurately model the TCP congestion window after the first packet loss.

III. Modeling Average TCP Behaviour

In this section a simplified model of the average packet rate of finite TCP flows is derived. This model captures the main features of short-lived Web-like TCP flows, while being dependant only on the number of transferred bytes, the experienced average loss probability and round-trip time of the connection. This allows for straight forward deployment of the model at the router queue where the amount of information available about the end hosts is limited.

The average packet rate during the data transfer period of a finite TCP connection can be modeled as $B = \frac{W_{avg}}{RTT}$, where W_{avg} is the average value of the congestion window maintained by the TCP source during the data transfer and RTT is the round-trip time experienced by the source in seconds. The RTT is approximated by a constant during the flow's lifetime; thus we ignore variations due to queueing delays or other network conditions.

If we do not consider connection establishment and tear-down, estimating W_{avg} is key to being able to predict average TCP packet rates. TCP sources update the congestion window ($cwnd$) during the data transfer period of each connection, with the value of the window determining how many TCP segments (packets) the source will send in each round-trip time (RTT). Thus, $W_{avg} = \frac{d_p}{L}$ can also be seen as the average packet rate, in packets per RTT, obtained by the flow; where d_p is the amount of data (packets) sent by the connection during its data transfer period and L is the latency in RTTs of transferring the data. To predict W_{avg}, we use recognized TCP models [12], [15] to obtain equations for d_p and L.

The predictive model described herein derives from the transient model proposed in [15] and the study of TCP latency in [12]. It divides the data transfer duration into two main parts: the slow start phase in which $cwnd$ grows exponentially, and the congestion avoidance phase in which $cwnd$ converges towards a steady-state value. The detection of the first packet drop is the event that causes the flow to transition from the slow-start to the congestion avoidance window regime.

The model described in this section derives from that proposed in [15], with $cwnd$ evolved continuously in time. For the remainder of the section, and unless explicitly stated, the unit of time (or latency) used is RTTs.

A. Slow start phase

Slow start is the initial phase of each TCP connection in which the congestion window is increased exponentially. The window evolution in this phase can be characterised by

$$W_{ss}(t) = W_i \times r^t \tag{1}$$

where $W_i = 1 \leq cwnd \leq 3$ is the typical initial value of $cwnd$ used by the flow, $r = 1 + \frac{1}{b}$ with b indicating the number of packets received before an ACK is sent ($b > 1$ when delayed ACKs are being used) and t is the time elapsed since the beginning of the transfer.

The number of packets already sent at time t_n by the flow in slow-start can be calculated by integrating $W_{ss}$from 0 to t_n:

$$D_{ss}(t) = \int_0^{t_n} W_{ss} = \frac{(r^{t_n} - 1)W_i}{\ln r} \tag{2}$$

From this we can also calculate the number of RTTs needed to send s packets in slow-start as

$$R(s) = \log_r \left(1 + \frac{s \ln r}{W_i}\right) \tag{3}$$

B. First packet loss

The initial slow-start phase ends either when the flow has completed sending all the packets without loss or it suffers the first packet drop. This event happens at time t_0 since the beginning of the transfer. The estimated number of packets sent in slow-start is given in [15] as

$$d_{ss} = \begin{cases} 1 + \frac{(1-(1-p)^d)(1-p)}{p} & \text{if } p > 0 \\ d & \text{if } p = 0 \end{cases} \tag{4}$$

where d is the number of packets needed to deliver the connection data excluding retransmits and p is the random loss probability suffered by the flow. The time t_0 can then be estimated by calculating the time at which d_{ss} packets have been sent in slow-start using (3) and (4),

$$t_0 = R(d_{ss}) = \begin{cases} log_r \left(1 + \frac{(1-(1-p)^d)(1-p)\ln r}{p \times W_i}\right) & \text{if } p > 0 \\ log_r \left(1 + \frac{d \ln r}{W_i}\right) & \text{if } p = 0 \end{cases} \tag{5}$$

C. Congestion avoidance

Once the first loss occurs, $cwnd$ converges towards a steady-state window W_{st} [15]. This transient phase is important for the modelling of mid-size transfers that are long enough to suffer packet losses but finish before $cwnd$ converges to W_{st}. The evolution of the congestion window after the first loss can thus be expressed as

$$W_{ca}(t) = W_{st} + (W_{t0} - W_{st})|\Delta_2|^{t-t0} \tag{6}$$

where $W_{ca}(t)$ represents the value taken by the average congestion window at time t from the beginning of the transfer, t_0 is the time of the first loss, W_{t_0} is the congestion window

at the time of the first loss and $\Delta_2$is the convergence rate of the system to the steady state window W_{st}.

The steady-state congestion window model used is based on the one proposed in [14], which models the TCP rate as

$$B_{st} = \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0 min(1, 3\sqrt{\frac{3bp}{8}})p(1+32p^2)} \quad (7)$$

where B_{st} is the estimated steady-state packet rate generated by the flow, RTT is the round-trip time, p is the loss probability and T_0 is the initial value of the retransmit timeout. The steady-state window W_{st} derived directly from (7) depends both on p and RTT.

Due to our application requirements, we need an equation for W_{st} that depends only on the loss probability p. With this goal, the equation has been further refined to remove the T_0 and RTT variables. From simulated scenarios, we have chosen a timeout loss component value of $T_0 = 0.1$ as a good approximation. The RTT parameter in W_{st} has also been fixed, to $RTT = 0.2$, which is a reasonable average taking into account that Internet round-trip times are typically between 50 and 500 ms. These changes allow us to have an estimated steady-state window that only depends on the loss probability p. The equation for W_{st} with convergence rate Δ_2 from [15] is then:

$$W_{st} = \frac{1}{\sqrt{\frac{2bp}{3}} + \frac{min(1,3\sqrt{\frac{3bp}{8}})p(1+32p^2)}{2}} \quad (8)$$

D. Latency

Let $L = L_{ss} + L_{t_0} + L_{ca}$ be the latency in RTTs to send the connection data, with L_{ss} being the latency of transferring the packets sent during the slow-start phase, L_{t_0} the expected cost of the retransmission timeout or fast recovery that happens after the first loss and L_{ca} is the latency to transfer any packet left after the first loss in the congestion avoidance regime.

The slow-start latency is equal to $L_{ss} = t_0$ (5). The impact of the initial loss on flow latency is given by L_{t_0}, estimated by the approach in [12].

The congestion avoidance latency L_{ca} is the time needed to send the remaining packets (if any) not sent during the slow-start phase. The number of packets sent in congestion avoidance is $d_{ca} = d_p - d_{ss}$, where d_p is the total number of packets to send including retransmissions (calculated below) and d_{ss} is given in (4).

To calculate L_{ca}, we first calculate the number of packets already sent in congestion avoidance at time t as $D_{ca}(t) = \int_0^{t-t_0} W_{ca}(t)$.

We can now obtain $L_{ca}$by substituting d_{ca} in D_{ca} and solving for t, which gives the solution

$$L_{ca} = \frac{(W_{t_0} - W_{st}) + d_{ca}\log(|\Delta_2|)}{W_{st}\log(|\Delta_2|)} - \frac{W_{st}W(\frac{(W_{t_0}-W_{st})|\Delta_2|^{\frac{(W_{t_0}-W_{st})+d_{ca}\log(|\Delta_2|)}{W_{st}\log(|\Delta_2|)}}}{W_{st}})}{W_{st}\log(|\Delta_2|)} \quad (9)$$

where W is the Lambert W function.

E. Average congestion window

The average congestion window W_{avg} of a TCP flow can also be seen as the average number of packets sent by the flow per RTT. Given the number of packets the flow is expected to send during its transfer period, d_p, and the latency L of completing the data transfer, an estimate for W_{avg} can be obtained from $W_{avg} = \frac{d_p}{L}$.

We now estimate the number of packets sent, d_p, with loss probability p, assuming no duplicate retransmissions and flows that transfer I bytes with average packet size S.

Let A be the average number of times each data packet will have to be sent,

$$A = \sum_{k\geq 1} k\, P(n-k) = \sum_{k\geq 0} p^k \quad (10)$$

with the latter converging to $A = \frac{1}{1-p}$ for $-1 < p < 1$. We can now estimate the average number of packets sent by the flow, d_p as

$$d_p = d\, A = \frac{\lceil \frac{I}{S} \rceil}{1-p} \quad (11)$$

with d_p from (11) and L, W_{avg} can be obtained using $W_{avg} = \frac{d_p}{L}$.

IV. RAD

This section describes the RAD algorithm and its implementation. RAD works at the granularity level of traffic *aggregates*. An aggregate is composed of a set of flows traversing the link that share a common attribute. A flow will be identified as the set of packets that share the same source and destination IP address and port numbers.

Combining flows into aggregates provides a flexible model in which fine-grained per-flow information is not maintained at the queue and decisions are taken at the aggregate level. Having a relatively low number of aggregates increases the likelihood of penalising "unlucky" traffic but also allows for the support of high-bandwidth links that can have many simultaneous flows traversing them at any given time. The destination IP address is used as the key to map packets to aggregates, as it is the most difficult to spoof in an IP packet header while still providing routing towards a destination.

The RAD architecture does not require the scheme to be implemented at any specific location on the network. RAD functionality will be utilised during periods of high congestion and will prove most effective when implemented close to the attack victim.

A. Measurements

To be able to calculate the time-averaged metrics, RAD maintains counters that hold observations for the current measurement period of Δ seconds - in_a, $drop_a$ and A_a keep track of the number of incoming packets, number dropped packets at the queue and the set of flow identifiers seen for

aggregate a respectively. in is a counter for the total incoming packets at the queue and $drop$ counts the total dropped packets.

The counters described above are used to obtain the averaged metrics that RAD updates at the beginning of each measurement period. These include r_a, $loss_a$ and f_a which provide (for each aggregate respectively) the average incoming rate in packets/second, experienced loss probability and number of distinct flows. The average total incoming packet rate in packets per second at the queue is kept as r.

B. Algorithm

At the end of each measurement period RAD updates the average metrics r_a, $loss_a$, f_a and r, and then calculates the new per-aggregate RAD drop probability p_a^r for use during the following measurement period. The moving average metrics are updated using the following equations:

$$\begin{aligned} r_a &= (1-\alpha) \times r_a + \alpha \times \frac{in_a}{\Delta} \\ loss_a &= (1-\alpha) \times loss_a + \alpha \times \frac{drop_a}{in_a} \\ f_a &= (1-\alpha) \times f_a + \alpha \times size(A_a) \end{aligned}$$

where α is the parameter that determines how fast the averaged metrics track the intantaneous counters.

The RAD drop probability p_a^r penalises aggregates that show a behaviour that deviates from what is expected from finite TCP connections under similar conditions. For this, the TCP rate equations from the Section III are used as the reference TCP behaviour.

Let $B(p)$ be the estimated average packet rate for a finite size TCP connection with average connection size b_t, round-trip time RTT_t and average loss probability p. We can obtain the target packet rate r_a^t and target number of flows f_a^t for each aggregate with

$$\begin{aligned} r_a^t &= f_a \times B(loss_a) \\ f_a^t &= \frac{r_a}{B(loss_a)} \end{aligned}$$

The value taken by RTT_t will be used as an RTT configuration for $B(p)$. The average RTT experienced by flows on a given aggregate cannot be easily determined at the router queue, so an approximation will be used to calculate the value of RTT_t. This value evolves from a minimum of $RTT_{\min}$ to a maximum of $RTT_{\max}$ as the loss probability p goes from 0 to $p_{\max}$. The constants $RTT_{\min}$ and $RTT_{\max}$ are set to 0.05 and 0.5 seconds respectively, which covers the typical range of RTTs experienced by Internet traffic flows. $p_{\max}$ will be set to 0.4, as a value for loss probability at which TCP congestion control proves ineffective. Given these parameters, an approximation for RTT_t can be calculated as follows:

$$RTT_t = RTT_{\max} - \min\left(1, \frac{loss_a}{p_{\max}}\right) \times (RTT_{\max} - RTT_{\min})$$

An initial RAD drop probability is calculated for each aggregate as follows:

$$p_a^i = \min\left(\left[\max\left(\frac{f_a}{f_a^t}, \frac{r_a}{r_a^t}\right) - 1\right] \times \frac{r_a}{r}, 1\right)$$

The next step allocates RAD a global "drop budget" of $\frac{r-1}{r}$ at the queue for the next measurement period. Thus the target global drop rate applied by RAD at period i will be $\frac{r_{i-1}-1}{r_{i-1}}$. The goal of this adjustment is to allow RAD to strongly penalise malicious traffic whilst providing full link utilisation to the ensuing AQM module to avoid traffic starvation. RAD then scales its drop probability to achieve this predicted global loss probability in the subsequent period, based on the incoming rate over the current period. This scaling is also weighted by the historical packet rate of the aggregate. The final RAD drop probability for each aggregate is thus calculated as follows:

$$p_a^r = p_a^i \times \frac{\frac{r-1}{r}}{\sum p_a^i \times \frac{r_a}{r}} \tag{12}$$

The RAD algorithm described above presumes that the attack source cannot be notified of packet drops. If the attacker were capable of detecting congestion notification, it could adjust its attack rate and source address spoofing to minimise the effectiveness of RAD. This is an unlikely scenario in distributed DoS attacks, as the attacker relies on spoofed source IP addresses, and thus cannot receive loss notifications via ACKs.

The likelihood of RAD penalising non-malicious traffic is another important consideration in its use as a DoS defence mechanism. RAD utilises a conservative strategy - only impacting traffic when indicated by the "drop budget" or during periods of high congestion. It is possible for RAD to misclassify aggressive high-bandwidth TCP flows at attack time if the congestion rate is very high and the value used for RTT_t is close to RTT_{max}. Even in this scenario, these flows would be penalised based on their congestion responsiveness, so RAD should only rate-limit them up to the rates that conform with the behaviour predicted by the model.

C. Implementation

It is critical that an algorithm such as RAD be resilient and robust when handling traffic from all sources, including malicious ones. The RAD implementation uses a fixed number of operations per incoming packet and has constant memory usage. Upon packet arrival RAD updates the instantaneuous counters, which are stored in fixed-size data structures.

It is possible for an attacker to generate sufficient packets for RAD to overload the router CPU. This issue can be easily mitigated through the inclusion of rate dependent sampling. With this approach, RAD will process incoming packets with probability $p_s = \min(1, \frac{R}{r})$, where R is maximum bandwidth provided by the outgoing link and r is the average incoming rate experienced at the queue, both in packets per second.

RAD replicates its measurement data structures for each one of the aggregates it tracks. Each aggregate is associated with an $AggregateCounter$ data structure, which keeps track of in_a,

$drop_a$, A_a, r_a, $loss_a$ and f_a. All of these metrics except A_a are implemented as simple integer or floating point counters.

For A_a, Bloom filters [16] are used as the data structure. These randomised data structures can be used for set membership queries, providing very high space efficiency at the cost of some false positives. A_a must maintain information about the set of flows the queue has seen in a given measurement period. It is important that a fixed-space data structure be used in this case, as malicious sources that spoof IP headers may appear to generate any number of flows.

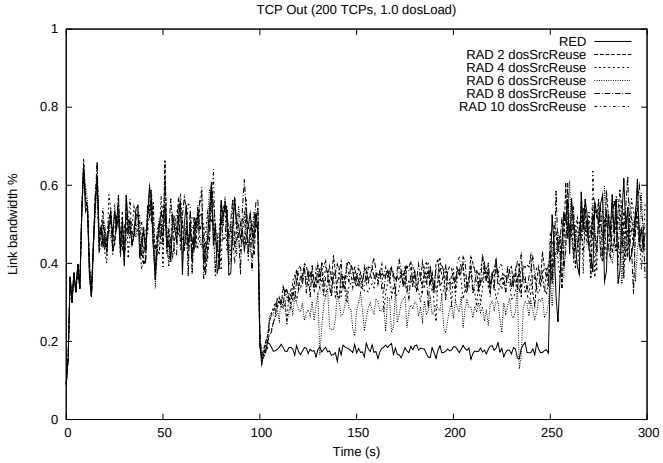
This implementation approach provides for a bounded-memory implementation per aggregate data structure. The data structures are also flexible in the sense that, if more memory is available, more aggregates can be tracked for finer grained filtering or larger Bloom filters can be used for more accurate flow counting.

V. RAD Evaluation

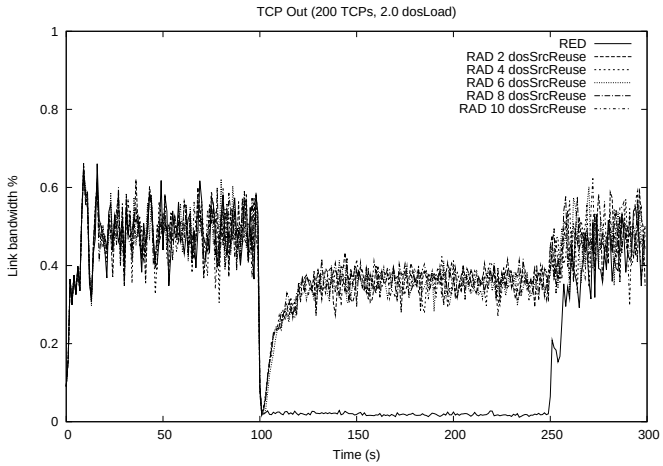
RAD performance was evaluated using the NS-2 simulator. The topology chosen is a single-bottleneck dumbbell with 50 traffic source nodes and one destination node for each TCP source. The bottleneck link has a 50 Mbps capacity.

The base traffic mix will consist of TCP Sack sources with a closed-loop on-off behaviour. The TCP sources transmit ON_b bytes of data during their ON period and when this transfer is finished they enter their OFF period, in which they wait OFF_t before starting a new ON period. The number of TCP sources will be 200 or 500 depending on the TCP load desired, ON_b will be set to 100000 bytes and OFF_t will be set to 10 seconds.

RED will be configured with the default settings of $qWeight = 0.001$, $maxProb = 0.1$, $minTh = 10$ and $maxTh = 50$. RAD configuration will be set as follows: $b_t = 100000$, $RTT_{min} = 50$ ms, $RTT_{max} = 500$ ms, $p_{max} = 0.4$, $\Delta = 0.5$, $\alpha = 0.25$, and the number of tracked aggregates will be 10.

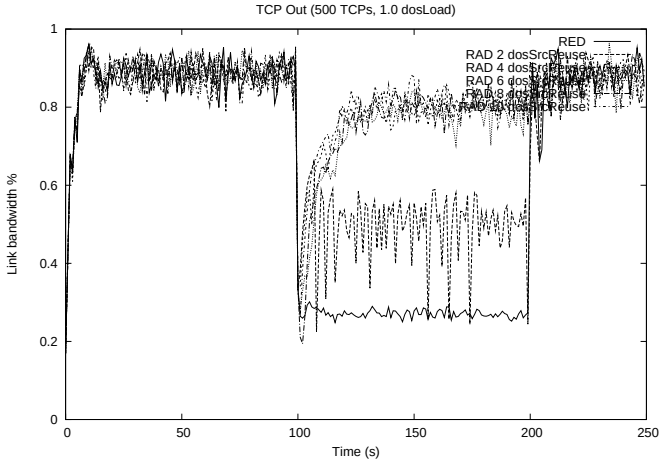


(a) 100% DoS traffic load

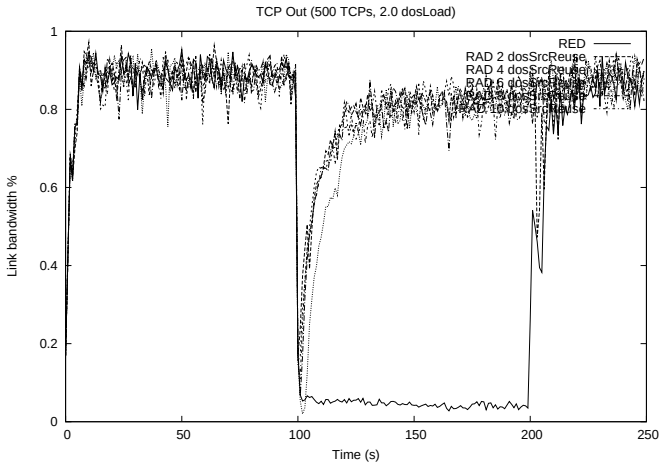


(b) 200% DoS traffic load

Fig. 1. RAD Performance with 200 TCP sources



(a) 100% DoS traffic load



(b) 200% DoS traffic load

Fig. 2. RAD Performance with 500 TCP sources

Figure 1 compares RED and RAD performance before, during and after a DoS attack with a packet rate of both 100% and 200% of the available bottleneck link bandwidth. This scenario uses a lightly-loaded background traffic mix of 200 TCP sources. Figure 2 shows the similar characteristics, using a high-load existing traffic mix caused by 500 TCP sources.

All scenarios show TCP throughput evolution over time, as DoS traffic is introduced in the mix between seconds 100 and 250 of the simulation. The $dosSrcReuse$ parameter is used to simulate the ability of malicious traffic to spoof IP packet headers. The DoS traffic generates a constant packet rate, and chooses a new spoofed source IP header every $dosSrcReuse$ packets generated, thus effectively being able to control how many different flows the attack traffic consists of from the queue's point of view.

The results in figures 1 and 2 show that RAD outperforms RED in protecting TCP throughput in the presence of malicious traffic, whilst not adversely affecting it when there is no attack in progress. RAD performs better with high throughput attacks, but still protects TCP at lower attack rates. Figure 1 shows how RAD is more sensitive to $dosSrcReuse$ values for low-rate DoS attacks. However, it always outperforms RED regardless of the number of flows the attack traffic consists of.

A scenario not covered by this evaluation is that in which the attacker is able to generate large amounts of conformant congestion responsive traffic that evades penalisation while still impacting on the throughput of "legitimate" traffic. At the network level, this is a scenario analogous to "flash-crowds" or traffic surges created by a sudden rise of interest in a Web site. RAD is not superior to other AQM schemes in this scenario, but it does ensure that the attack traffic will have to conform to a minimal congestion-responsive behaviour.

VI. Conclusion

This article proposes a novel AQM-based approach to defend against flooding denial-of-service attacks. RAD applies TCP modelling to protect congestion responsive traffic from malicious DoS traffic. It uses an AQM-inspired architecture that provides high scalability and bounded memory usage.

RAD has been shown to be an effective defence mechanism against flooding DoS in a variety of scenarios. It is also able to provide TCP protection in situations in which packet headers are spoofed by the malicious traffic.

VII. Acknowledgements

Elements of this publication have emanated from research conducted with the financial support of Science Foundation Ireland and Enterprise Ireland.

References

[1] A. Hussain, J. Heidemann, and C. Papadopoulos, "A framework for classifying denial of service attacks," *Information Sciences Institute Technical Report ISI-TR-2003-569*, 2003.

[2] V. Paxson, "An analysis of using reflectors for distributed denial-of-service attacks," *ACM Computer Communication Review*, vol. 31, no. 3, pp. 38–47, Jul. 2001.

[3] T. M. Gil and M. Poletto, "Multops: A data-structure for bandwidth attack detection," in *Proceedings of 10th Usenix Security Symposium*, Jul. 2001, pp. 23 – 38.

[4] A. Ramanathan, "Wades: A tool for distributed denial of service attack detection," M.Sc. Thesis TAMU-ECE-2002-02, 2002.

[5] D. X. Song and A. Perrig, "Advanced and authenticated marking schemes for ip traceback," in *Proceedings of IEEE Infocom 2001*, Apr. 2001, pp. 878–886.

[6] S. Floyd, "TCP and explicit congestion notification," *ACM Computer Communication Review*, vol. 24, no. 5, pp. 10–23, 1994. [Online]. Available: http://citeseer.nj.nec.com/article/floyd94tcp.html

[7] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," vol. 1, no. 4, pp. 397–413, Aug. 1993. [Online]. Available: http://www.icir.org/floyd/papers/red/red.html

[8] S. Floyd, R. Gummadi, and S. Shenker, "Adaptive RED: An algorithm for increasing the robustness of RED," Aug. 2001. [Online]. Available: http://www.icir.org/floyd/papers/adaptiveRed.pdf

[9] C. Hollot, V. Misra, D. Towsley, and W. Gong, "Analysis and design of controllers for AQM routers supporting TCP flows," vol. 47, no. 6, pp. 945–959, Jun. 2002. [Online]. Available: http://ieeexplore.ieee.org/

[10] R. Mahajan, S. M. Bellovin, and S. Floyd, "Controlling high bandwidth aggregates in the network," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 3, pp. 62–73, Jul. 2002.

[11] R. Mahajan and S. Floyd, "Controlling high bandwidth flows at the congested router," AT&T Center for Internet Research at ICSI (ACIRI), Tech. Rep. TR-01-001, Apr. 2001. [Online]. Available: http://www.cs.washington.edu/homes/ratul/red-pd/

[12] N. Cardwell, S. Savage, and T. Anderson, "Modeling tcp latency," in *Proc. IEEE INFOCOM*, vol. 3, 2000, pp. 1742–1751.

[13] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the tcp congestion avoidance algorithm," *ACM Computer Communication Review*, vol. 27, no. 3, pp. 67–82, 1997.

[14] J. Padhye, V. Firoiu, and D. F. Towsley, "Modeling tcp reno performance: A simple model and its empirical validation," *IEEE/ACM Transactions On Networking*, vol. 8, no. 2, pp. 133–145, April 2000.

[15] N. Ehsan and M. Liu, "Analysis of tcp transient behavior and its effect on file transfer latency," in *Proc. IEEE ICC'03, Anchorage, Alaska*, vol. 3, May 2003, pp. 1806–1811.

[16] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2005.