Bar Ilan University - Department of Computer Science

Network Security Group

Technical Report TR-12-01

Backward Traffic Throttling to Mitigate Network Floods

Yehoshua Gev, Moti Geva, Amir Herzberg

March 2012

Backward Traffic Throttling to Mitigate Network Floods

Yehoshua Gev Moti Geva Amir Herzberg
Computer Science Department, Bar-Ilan University, Ramat Gan, Israel

Abstract

We present *Backward Traffic Throttling* (BTT), an efficient, decentralized mechanism, coping with congestion situations and mitigating network-flooding attacks. Upon congestion detection, a BTT node throttles the traffic of its incoming links, and notifies neighboring BTT nodes of the throtting. The notification allows the upstream nodes to perform additional throttling closer to the traffic source. The throttling parameters (e.g., the rate limit) are determined separately at each node, using typical traffic estimations. BTT can serve as a barrier against a variety of network-based attacks and congestion conditions.

Both simulation and emulation experiments were performed to asses the effectiveness of BTT during distributed denial-of-service (DDoS) attacks. Results show that even limited BTT deployment alleviates attacks damage and allowing legitimate TCP traffic to sustain communication, whereas larger deployments maintain large portion of the original bandwidth.

1 Introduction

Denial-of-service attacks, and in particular network-flooding distributed denial-of-service (DDoS) attacks, are hard to prevent and mitigate in a decentralized, best-effort network such as the Internet. Mitigation is hard, also due to the huge resources of DDoS attacks, usually launched by large botnets. A recent study [3] showed that dramatic growth of DDoS attack volumes, e.g., during 2010, volumes exceeded 100 Gbps.

Flooding-DDoS botnets usually send traffic to victim hosts, causing packet losses on bottleneck links, thereby making TCP and TCP-friendly [9, 33] flows drop their transmission rates. New attack types utilizing peer-to-peer (P2P) traffic between compromised nodes, such as Coremelt [30], pose an additional threat. Depending on the botnet size, P2P attacks can utilize small traffic rate between many pairs of zombies (bots), to cause congestion on a link between two routers (possibly at the core of the network); such coremelt packets can be hard to detect and block (using existing techniques).

Defenses against flooding DDoS attacks fall into two categories: *end-host* defenses and *router* defenses. End-host anti-DDoS mechanisms are much easier to deploy, but cannot prevent congestion on routers, and seem to be limited to two strategies. The first strategy is to try to communicate 'around' the congested routers/links, by sending traffic via one or more relaying hosts; see, e.g., [1]. This strategy works only when such relay paths exist. Another circumvention strategy is to hide the destination host, and construct many routes to it using overlays, e.g. [2, 7, 16]. The second strategy is to make sufficient redundant transmissions, to get the traffic to the destination in spite of the loss due to the congestion; this must be done carefully to avoid self-inflicted congestion, see [10], and can only be done for very limited amounts of traffic. We conclude that there is also a need to develop router-based DDoS defenses, which is our focus in this paper.

This paper details the design, simulation and testbed emulation of *Backward Traffic Throttling (BTT)*, a novel *distributed* mechanism integrated into network routers, for protecting legitimate traffic by minimizing the impact of DDoS flooding attacks. Since BTT is decentralized, it can be adopted independently by one or (preferably) multiple autonomous systems (ASs), with incrementally improving defenses against DDoS attacks.

BTT design is based on carefully combining two mechanisms: *throttling* of excessive, congestion-causing traffic at each router, and a cooperative *pushback protocol* allowing a router aware of congestion, to request upstream routers to perform corresponding throttling closer to the traffic sources. Routers throttle packets only based on their direct detection of congestion, or if these packets will be routed via the router who issued the pushback request. Furthermore, throttling is adjusted to preserve both *fairness* and *utilization*; notice that a challenge here, is to preserve fairness to TCP and TCP-friendly connections, since these connections may reduce their rates considerably upon detection of loss; BTT addresses this by using estimates of typical amounts of traffic. When the excessive traffic causing congestion subsides, BTT carefully reduces and then stops throttling.

BTT is activated when links are heavily loaded, that is, when link capacity is nearly fully utilized. When BTT is activated, each router uses typical traffic estimates[1] to prioritize traffic from incoming links. Next, the router uses a pushback protocol to requests upstream routers to limit their traffic rate, instead of having it reach the router and being dropped by the router. This allows these upstream routers to employ a similar throttling and pushback mechanism, and shape their own traffic, instead of the likely arbitrary discard at the requesting router. The upstream routers ability to have control over which packets are discarded, provides an incentive to collaborate. Note that under normal load conditions, BTT has no effect on the network's behavior.

BTT rate shaping implements a fairness mechanism, in which we share the link based on the typical traffic rate requirements, i.e., links get proportional share, even when under attack. The decision on the rate limits is completely decentralized, and is both measured and decided locally in each router. Such rate limiting does not require flow classification, tagging or monitoring; instead, the incoming links' rates are considered and throttled based on their compliance to the typical traffic rates, which are estimated in advance. A notable effect of such mechanism is that BTT would not punish any specific flow characteristic, such as limiting large flows just for being large. BTT achieves this automatically, without requiring manual interventions (as suggested, e.g., in ACC [20]).

Several works have already tried to apply rate shaping [6] and pushback [4, 19, 20] to defend against network flooding DDoS at the core routers. BTT takes the following new design choices.

Fairness BTT uses estimates of the typical traffic to divide the available bandwidth among the incoming links, achieving fair allocation. Assuming that the forwarding of traffic is governed by business relationships, and that the payment is correlated with the typical traffic rates, our mechanism could be considered as dividing the bandwidth by the fiscal importance of the links.

Decentralization BTT is decentralized in the throttling decision making, i.e., a downstream router only requests upstream routers to comply with the rate limitation it sets, and lets the upstream router decide what packets to discard, if any, and which further upstream throttling to announce. In addition, the typical rate estimation is also carried out in a decentralized manner.

[1]BTT assumes the use of some reliable mechanism to provide reasonable estimates of typical traffic volumes. Such mechanisms were proposed in previous works.

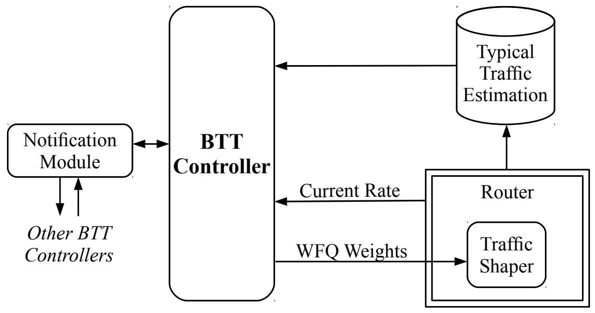

Figure 1: BTT system architecture.

Flow obliviousness BTT throttles traffic based on weights derived from the typical traffic rates, avoiding the need to identify the actual attacking flows, making BTT robust and not attack-specific, e.g., BTT is resilient to spoofing attacks since it does not try to identify the flows' sources, which might be impractical in case of spoofing, but instead limits the excessive flows based on its real flooding source.

Evaluation. As discussed in [22], evaluation of defenses against DDoS is challenging; emulation and testbed experiments are limited to rather small topologies, while simulations of a realistic network are challenging computationally - even with gross simplifications. Therefore, we performed both simulations as well as emulation-experiments (in the DETERlab testbed). For the simulations, we used a realistic AS topology.

Both simulation and emulation results show that BTT significantly prioritizes legitimate traffic over attack traffic, even in a massively distributed scenarios, such as Coremelt attacks. The results show that even in limited deployments, BTT proves to be effective. Furthermore, as BTT adoption increases, an increase is observed in the amount of legitimate traffic, demonstrating that even partial and gradual deployment of BTT proves valuable, and provides an effective defense against flooding DDoS.

The rest of the paper is organized as follows. Section 2 presents BTT design. In Section 3 we present empirical evaluation of BTT. Section 4 discusses related work. Conclusions and future work are presented in Section 5.

2 Design

Figure 1 depicts the system architecture. The router continuously sends traffic information to the BTT controller, as well as traffic statistics to the traffic estimator. The traffic estimator, the BTT controller, and/or notification modules, can be integrated within the router, however, more probably at least the traffic estimator should be external or even AS-wide database, which can contain significantly more information, used on-demand by the BTT controller. The BTT controller controls the router's modus operandi, which by default is unchanged. However, whenever the BTT controller detects excessive traffic rate on the router, or receives a rate-limiting request from a peer BTT, it would initiate traffic prioritizing and shaping at the router. The traffic weights installed on the shaper are based on the typical traffic estimates, retrieved from the traffic estimator. Next the BTT will propagate rate limiting notifications to additional upstream routers. If additional upstream routers support BTT, they will perform corresponding traffic shaping, hence limit attacking streams closer to their source.

This paper we describe the BTT controller design and the notification module, which is used to communicate with other BTT controllers.

2.1 Network Model

The network is modeled as a directed graph, with Autonomous Systems (ASes) as nodes. We assume that each node knows the AS-level route each outgoing packet will traverse and that no routing changes occur while the throttling is active.

The knowledge of the routes can be learned, for example, from the AS_PATH sequence of the Border Gateway Protocol (BGP) [27]. The AS_PATH contains a sequence of ASes between the current AS and one destination, and defines a possible path. The routing algorithm in each AS determines the AS_PATH used for routing to each destination, and hence can predict the path each packet it routes will take. In cases the routing decision is not based on BGP, a route-descovery protocol, similar to LOT [12], could be employed to detect the routes.

We denote a link from AS a to b as an ordered pair $L = \langle a, b \rangle$. The set $\mathrm{UP}(L)$ contains all neighboring links upstream of L, which contribute to the traffic from a to b. Each AS can measure the traffic rate, $\mathrm{CurRate}^{L_O}(L)$, received on an incoming link L and destined to be routed to a downstream link L_O.[2] Note that L_O might be several hops away from L.

Whereas in the design we treat each AS as a single node, parts of the system are operated within the routers, specifically traffic monitoring and shaping, and possibly the BTT controller as well.

[2]For readability, time is omitted from our notation.

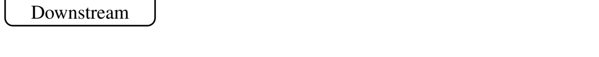
Congestion Detection
Noticiation Reciept from Downstream
Activate Throttling
Send Notification to Upstream Neighbors

Figure 2: Flow diagram of BTT.

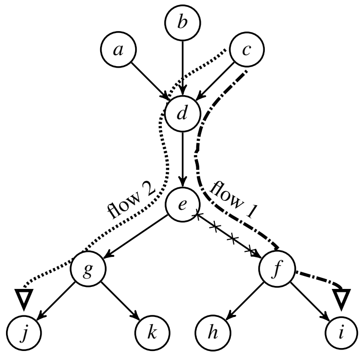

Figure 3: Backward throttling example. Operation steps: (1) Node e detects congestion on link $\langle e, f \rangle$. (2) Node e activates throttling on incoming links. (3) Node e notifies node d. (4) Node d creates a virtual link of the traffic destined to $\langle e, f \rangle$ and throttles it. The traffic of flow 2 will not be throttled as it is not destined to $\langle e, f \rangle$. Note that the throttling of a flow only depends on its downstream route.

2.2 Typical Traffic and Weights

The basic idea of BTT is to utilize an estimation of the *typical traffic rates*, as a guideline for fairly splitting the available bandwidth among the upstream neighbors . Similarly to other papers, which have proposed to use rate limiting based on typical traffic estimation, we assume that it is indeed possible to collect "typical traffic" statistics. In this paper we rely on existing methods, e.g., as proposed by [6].

Specifically, upon identification of congestion in an outgoing (downstream) link L_O, the node will fairly allocate a part of its downstream rate for each of the upstream links L_U, which send traffic to L_O. The weights are derived from the typical traffic estimation, as in Eq. 1. In the current design of BTT, we use wighted fair queuing (WFQ) to allocate the typical rate.

$$\phi_{L_U}^{L_O} = \mathrm{Typ}^{L_O}(L_U) \Big/ \sum_{L \in \mathrm{UP}(L_O)} \mathrm{Typ}^{L_O}(L) \tag{1}$$

The first task of BTT is, therefore, the estimation of the typical traffic from each link L_U into the AS, flowing to each destination link L_O, denoted $\mathrm{Typ}^{L_O}(L_U)$. We call this module of BTT the *typical traffic estimator*.

The typical traffic estimator runs continuously, collecting statistics of incoming traffic during normal operation, prior to attack detection. Using these statistics, the node can estimate $\mathrm{Typ}^{L_O}(L_U)$, the typical rate it would receive on incoming link L_U to be routed to any link L_O.

More specifically, for each incoming link, and for each time of the day, the node stores the typical rate it routes to every destination. That rate is updated periodically, e.g. daily, based on current traffic rates, possibly using techniques such as exponential smoothing that gives low weight to recent data, as to prevent an attacker from easily messing with the statistics.

2.3 Backward Traffic Throttling

The *Backward Traffic Throttling* (BTT) algorithm is the core of our mechanism. It allows an AS to confine attack traffic sent from different incoming links, and allows the upstream ASes to further confine the attack.

BTT is triggered when the congestion on an outgoing link exceeds a threshold or upon receipt of a NOTIFY message from a downstream AS (see Figure 2). Each AS runs an independent instance of the system. If BTT was initiated due to congestion in the current router, than it will immediately commence shaping and send a NOTIFY to neighboring BTT controllers. If the throttling was activated due to a NOTIFY message, then the BTT creates a virtual link, L_O, containing all traffic flowing through the the congested link (see Figure 4). The virtual link means that a new output buffer is allocated to traffic going through L_O, allowing the AS to shape it. We refer to the outgoing throttled link as L_O, in both virtual and physical cases, as both are treated similarly. The rate capacity of L_O, denoted C^{L_O}, is either the bandwidth of L_O, if it is the regulating link, or the requested rate restriction of the virtual link as described in the NOTIFY message.

The BTT procedure is described in Algorithm 1, which activates weighted fair queueing (WFQ) on L_O, determines the restrictions on the upstream links, and sends NOTIFY messages with those restrictions to neighboring upstream ASes.

The WFQ weight, $\phi^{L_O}_{L_U}$, assigned to traffic from upstream link L_U routed to L_O is given by Equation 1.

The next step is the calculation of the restrictions for the incoming links. For simplicity, we first describe a basic allocation scheme, limiting each incoming link to a fraction of C^{L_O}. Specifically, an incoming link L_U is

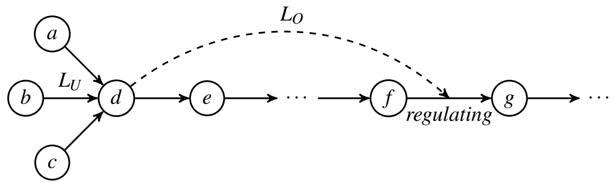

Figure 4: Schematic view of topology. L_O is a virtual link containing all traffic from d to the regulating link $\langle f, g\rangle$. If the regulating link is adjacent to d, L_O is the link itself. L_U is a neighboring upstream link.

Algorithm 1 Backward Traffic Throttling

```
   ▷ Activates the throttling mechanism on outgoing
   link $L_O$, using total rate capacity $R_{\text{cap}}$.
1: procedure BTT$_t$($L_O$, $R_{\text{cap}}$)
2:     Activate WFQ on $L_O$
           using weights $\{\phi^{L_O}_{L_U}\}_{L_U \in \text{UP}(L_O)}$    ▷ see Eq. 1
3:     for $L_U \in \text{UP}(L_O)$ do
4:         if $\text{CurRate}^{L_O}_t(L_U) > 0$ then
5:             SendNotifyMsg($L_U$, $L_O$, $Rest^{L_O}(L_U)$)
                                                    ▷ see Eq. 5
6:         end if
7:     end for
8: end procedure
```

restricted to $\phi^{L_O}_{L_U} C^{L_O}$, which corresponds to the typical rate on L_U multiplied by the typical overcapacity factor of L_O. For example, if L_O is a 100 MB link, with total typical traffic of 50 MB, the overcapacity factor will be $100/50 = 2$. An incoming link with typical rate of 10 MB, will be restricted to $\frac{10}{50} \times 100 = 2 \times 10 = 20$ MB.

These *base* restrictions fail to fully utilize the link, for when an incoming link falls below its restriction, the remainder of the bandwidth is not shared among the other links. To overcome this issue, an adaptive overbooking factor (OBF) is maintained for each outgoing link, adjusting the restrictions over time. To calculated the OBF, the ratio, ρ^{L_O}, between outgoing capacity and current incoming rate is calculated using equation 2.

$$\rho^{L_O} = \frac{C^{L_O}}{\sum_{L \in \text{UP}(L_O)} \min\left\{\text{CurRate}^{L_O}(L), Rest^{L_O}(L)\right\}} \tag{2}$$

This ratio considers all upstream nodes as complying to their restrictions, preventing non-complying nodes from obtaining bandwidth that might be assigned to others. From this ratio, the desired change factor, $\bar{\rho}^{L_O}$, of the OBF is caluclated in equation 3. The calculation restricts the agresiveness of the change using a parameter, α, which was set to 2 in our experimets, and a maximum level, $\rho_{\max}$, which was also set to 2.

$$\bar{\rho}^{L_O} = \begin{cases} \min\left\{\rho_{\max}, 1 + (\rho^{L_O} - 1)/\alpha\right\}, & \text{if } \rho^{L_O} \geq 1 \\ 1 + (\rho^{L_O} - 1) \cdot \alpha, & \text{if } \rho^{L_O} < 1 \end{cases} \tag{3}$$

Using this change factor, every τ seconds the OBF is adjusted, as appear in equation 4.

$$OBF^{L_O} \leftarrow \max\left\{1, \bar{\rho}^{L_O} \cdot OBF^{L_O}\right\} \tag{4}$$

When sending NOTIFY messages (algorithm 1), the restrictions are calculated by multiplying the base restrictions by the overbooking factor, as depicted by equation 5.

$$Rest^{L_O}(L_U) = \phi^{L_O}_{L_U} \cdot OBF^{L_O} \cdot C^{L_O} \tag{5}$$

In case multiple links request throttling, the throttling will first be performed on the virtual link connected to the farthest requesting link. Than, that throttled traffic will enter the throttling function of the nearer virtual link as any other traffic entering it.

Upon change of calculated current traffic rates, the BTT procedure would be activated again. However, new NOTIFY messages will only be sent if they further restrict the allocated bandwidth, or if a certain period of time elapsed. This behavior enables the system to respond quickly to new congestion situations, but delay relaxation of an active restriction.

2.4 Notification Protocol

The communication between BTT nodes is preformed by sending NOTIFY messages using TCP connections. Each NOTIFY message comprise of

1. Notification sender
2. Notification recipient
3. Regulating link identifier
4. Rate restriction (bps)
5. Expiration time.

For security reasons, authentication of the message source is required, however for simplicity we assume that this is done using well established techniques, such as [17]. If the message source is validated, it is passed to the BTT controller as described in the previous section. If the restriction was set to infinity or the expiration time has passed, the throttling mechanism is disarmed.

Since BTT does not actually require cooperation from other BTT controllers for the shaping, the sender need not wait for acknowledgment, moreover, the neighboring AS might not support the protocol, and as the sender's operation is not dependent on the message acceptance.

2.5 Recovery from Attack

In general, when the node that originated the throttling identifies that the attack had ceased, it will deactivate the throttling mechanism and send NOTIFY messages upstream with infinity rate restriction. However, to avoid exploitation by the attacker, two practices are being used.

First, the restrictions must not be removed too quickly, to prevent low-rate attacks such as [18]. This is achieved by limiting the ratio by which restrictions are reduced within each message, as well as the time between such messages.

Second, the activation of BTT is made at a high kick-in threshold Θ and its deactivation process begins only if the utilization of the link drops below a lower threshold θ. It is not enough that there are no packet drops on the link. Particularly, the lower threshold should be between the typical utilization, indicating that the network resumed its normal functioning, and the BTT's high kick-in threshold Θ. In any case, even if BTT ceases the shaping, and then the rate re-increases, whenever it crosses the kick-in threshold, it will automatically initiate the shaper, whereby limiting the impact made by such an attack, and maintain the typical/legitimate traffic.

2.6 Implementation Considerations

2.6.1 Communication

The mechanism is intended to be deployed at AS level, with the communication protocol described above being implemented between adjacent ASes. However, the ASes internal networks are comprised of many routers, requiring a separate interior protocol for transferring the restrictions from the activating router to the routers connected to the incoming links.

For NOTIFY messages are sent between adjacent ASes, there are no security concerns, such as message-origin spoofing or man-in-the-middle attacks. In addition, the trust relationship between the ASes is based on the fact that the downstream AS is already in position to drop its incoming packets, giving it no motivation to make a false restrictions.

2.6.2 Gradual Deployment

Another important issue is the operation of BTT during the transition phase, having been only deployed in a limited number of ASes. The mechanism is designed to operate even if no neighboring ASes support the protocol, by only activating the WFQ. We can use tunneling to enable distant supporting node to coordinate. In particular, [11] presents a lightweight protocol that both detects remote supporting nodes and constructs a tunnel between them.

As shown in Section 3, even small deployment of BTT can prove valuable in many scenarios. Generally, BTT can be useful when having a node with large number of connections. The reason for this is that each incoming link in AS leverl has a typical behavior in regards to where it sends its packets. Since it is very likely that the botnet will not have the exact deployment of the legitimate clients, at least some of the flows, which are now caught in the "cross-fire" can be saved. The flows coming without any attack in them can be fully forwarded even if they make a sudden surge of traffic, assuming that the link has overcapacity (without an attack), as described in Section 3. For other flows, who are mixed with attack traffic, the rate would simply decrease as a function of the ratio between legitimate and attack mix. Other flows, carrying only attack traffic may now only have the rate of the original typical ratio on that link.

For larger deployments, the throttling notification mechanism ("pushback") can kick into action, alerting upstream nodes about the throttle. In this case, as no drops currently happen on the upstream link, it can help prioritize legitimate traffic – which is assumed to be of typical rate – over attacking surges.

Consequently, gradual deployment should prove effective, however, for full impact, the larger the deployment is, the more legitimate traffic can pass and less mixed attack-legitimate links are found. Also, downstream nodes not supporting BTT, may still present congestion, hence the rate of TCP-friendly flows would decrease, whereas attack traffic might take over.

2.6.3 Resources Requirements

The estimations of the typical traffic rates are in the order of the number of different paths packets exiting the AS traverse. The number of such paths is bounded by the number of ASes, which today amounts to about 40 000 [5]. In case we would like to store this in the router itself, assuming the typical values are collected per the hours of the day, each using 4 bytes, such an array will take at most 3.75 MB. Such an array is stored for each incoming link. The use of sketching algorithms (e.g., [13]) would permit us to use pre-defined more limited space, trading off the accuracy and the calculation time of estimations. However, those algorithms are out-of-scope of this paper.

3 Evaluation

In this section we describe the experimental design and results, that we conducted in order to evaluate our proposed mechanism. We performed both an emulation and a simulation—each intended to measure different aspects of BTT's behavior. The emulation was designed to test how BTT handles real traffic and to measure the affect it has. In particular, we show BBT's response to attack start and end, on a timeline.

The simulation is meant to test BTT on an Internet-scale topology, showing its steady state. The simulation also enabled us to check how BTT behaves only partial deployed.

3.1 Emulation Setup

To evaluate the performance of BTT in a real environmet, we developed a prototype in Python that emulates a router with all the features of BTT (typical traffic estimator, BTT controller and traffic shaper). The prototype was tested on the DETERlab testbed, using the topology shown in figure 5.

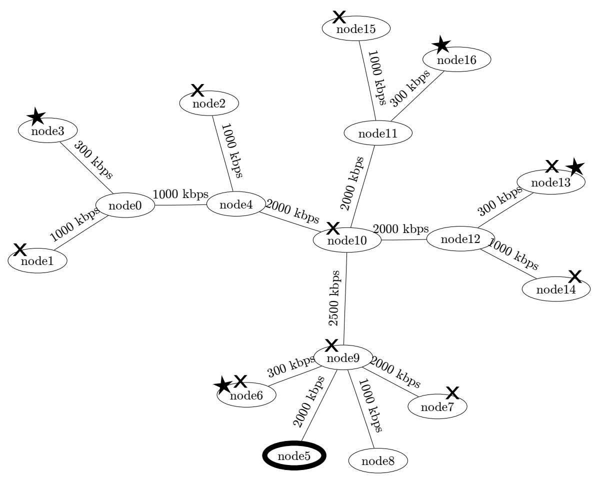

Figure 5: The emulation topoloy. A star marks legitimate TCP source. An *X* marks a UDP attacker. The destination of the legitimate traffic and of the attack is *node*5.

The attack traffic was generated using Deterlab traffic generation tools, as constant-bit-rate 1KB-packets UDP traffic. The nodes from which attack traffic is generated are marked with an *X* in figure 5. All the attack traffic is directed at *node*5.

The legitimate traffic is FTP connections uploading a large file to *node*5. Each node marked with a star maintains 3 FTP connections. Prior to the experiment, we measured the rates of the TCP connections, to be used as the typical traffic rates. In order to test how BTT handles changes of the legitimate traffic, the bandwidth of *node*16 was doubled to 600kbps after the typical rates were calculated.

We used Linux (version 2.6.32-31-generic-pae) machines for each node, running the standard protocol stack. The nodes where connected via a 20-packet-queue drop-tail links with rate limiters (see the topology).

3.2 Emulation results

Figures 6–11 show traffic measurements between *node*16 and *node*5. Figures 6 and 7 show the behavior without activating BTT. Figure 6 presents the aggregated rate of the 3 FTP connections from *node*16, before, during and after the attack, and figure 7 show the delay and packet looses as measured by ping. Both figures represnet a baseline for the following results.

Figures 8 and 9 present the results with BTT enabled but without overbooking.

Figures 10 and 11 present the results with BTT and overbooking enabled.

As can be seen in the above figures, the activation of

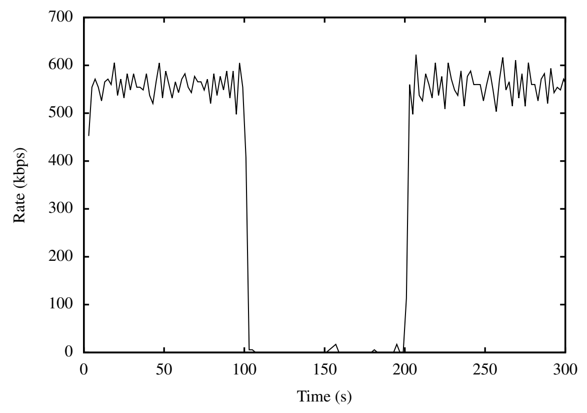

Figure 6: The legitimate traffic rate with BTT disabled. The attack took place between $t = 100$s and $t = 200$s. The TCP traffic drops to zero during the attack.

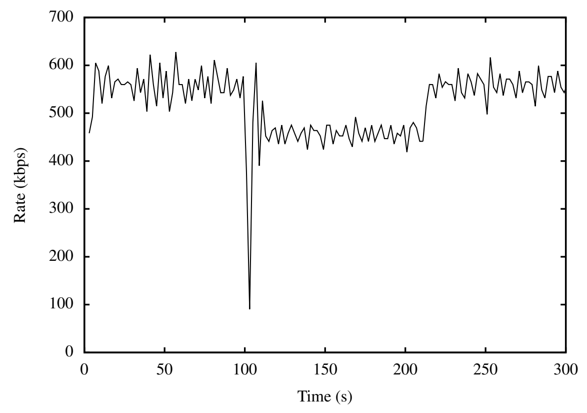

Figure 8: The legitimate traffic rate with BTT enabled, without overbooking. Two seconds after the attack begins, BTT succeeds in maintaining the TCP connection at about 90% of its maximal rate.

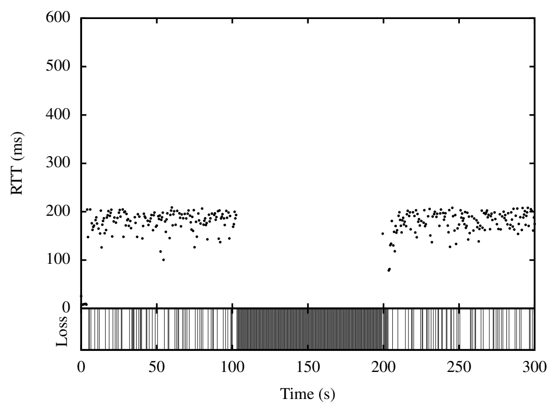

Figure 7: Ping results with BTT disabled. During the attack, all the ping packets were dropped. The parameters were measured using ping in a rate of 10 packets of 1KB per second. If an echo-reply packet was received, the round-trip-time (RTT) will appear in the upper section. In case no reply was receieved, a line will be marked in the lower section.

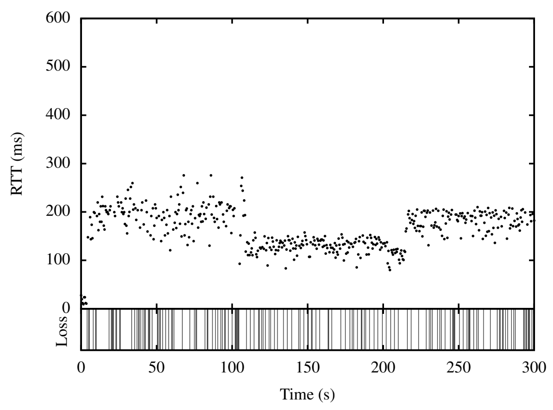

Figure 9: Ping results with BTT enabled, without overbooking. The delay decreases during the attack, as BTT without overbooking restricts the traffic rates to below their normal rates.

BTT succeeds in maintaining a good throughput of the legitimate streams, even without the use of overbooking. Nonetheless, overbooking enables the system to achieve its normal rate even during the attack.

It should be noted here that the presented rate originate from a source the generate no attack traffic. Nodes that generate attack traffic (e.g., *node*13) cannot maintain their TCP connection with current implementation, without introducing a filtering mechanism.

3.3 Simulation Setup

We performed the evaluation of BTT using PAWS simulator [31], which was originally designed for worm propagation simulation, with adaptations for testing DDoS attacks. The simulation runs on a full-scale AS-level topology of the Internet, including links' bandwidths and routing information, as constructed by Wei and Mirkovic [32].

The simulation aims to find the stable state of the network for each setup of attackers, legitimate sources and BTT nodes. To accomplish that, the simulation runs in rounds until the system stabilizes. At each round, the pass ratios are calculated for all the links in the network, based on the rates transferred on the previous round. Using these ratios, the traffic of the current round is adequately adapted.

Legitimate traffic is simulated as TCP-friendly flows, altogether consuming, at the stable state, the minimum of the full capacity of the paths they traverse and the flow's maximal rate. About 350 000 flows are created for each run of the simulation, by randomly choosing pairs of ASes and setting their maximal rate to 1% of the available bandwidth of the path between them. This method

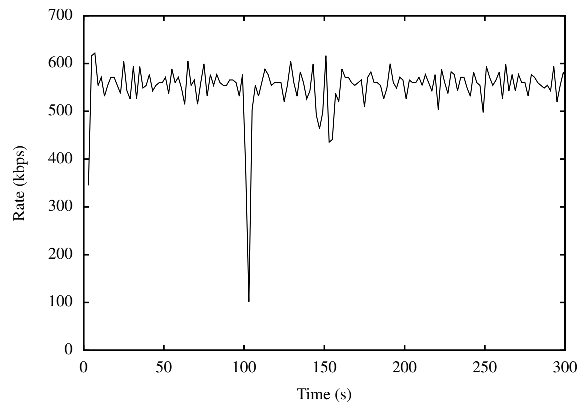

Figure 10: Legitimate traffic rates, with BTT and overbooking enabled. The legitimate TCP maintain its normal rate even during the attack.

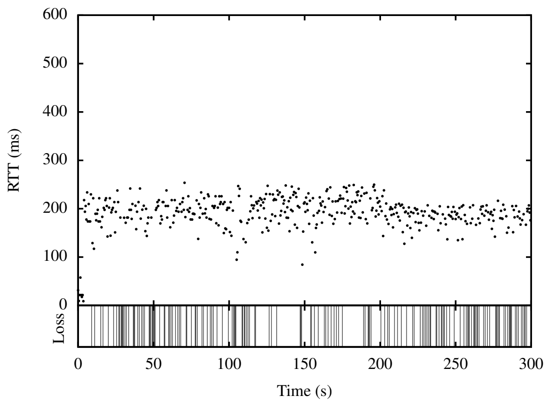

Figure 11: Ping results, with BTT and overbooking enabled. The delay is not affected by the attack.

results in a moderate utilization of the links.

Attack traffic is generated by a botnet, created in the distribution of the Slammer worm [25] or by selecting random IP addresses for the bots. For each run, a single target link is chosen at random, having all the bots generate traffic passing that link. When using the Slammer disribution, 50 000 bots are deployed, such that the bandwidth of all the bots is configured to create a certain overload of the target link. In the random distribution case, the bandwidth of the bots is preconfigured to 100 kBps, but the number of bots depends on the desired overload.

Our simulation setup was made of the following.

Attack overload factor The attacks are designed to congest a single target link, while possibly causing colateral congestion in the peripheral links. The rate of attack traffic at the incoming queues of the router at the head of the target link is reffered to as the attack rate. The *attack overload factor*, is the attack rate divided by the link bandwidth. An factor above 1 causes packet losses on

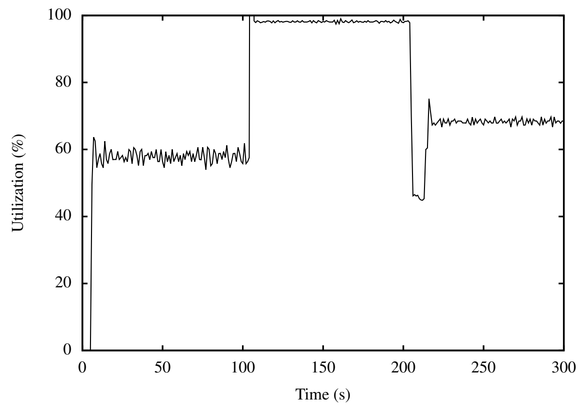

Figure 12: Utilization of the target link, with BTT and overbooking enabled. BTT succeeds to fully utilize the target link.

the link, even without any legitimate taffic. In the simulation we used an attack factor of 2 which should simulate at most 50% legitimate packet delivery probability, which means that any TCP connection would be completely disabled.

BTT deployment ratio BTT is deployed to a random set of ASes. *BTT deployment ratio* is the ratio of the deployed ASes out of total ASes.

Finally, we measure the **Goodput ratio** . To measure the degradation of legitimate traffic rates during an attack, we calculate the ratio between the rate before and during the attack. This *goodput ratio* is calculated for the target link, such that a low ratio indicates attack success. Due the the TCP-friendliness of the legitimate flows, without a throttling mechanism, a ratio below 100% but above 0% means that the attack only consumes a portion of the link bandwidth, allowing the legitimate traffic the remainder. Given that, an attack overloading the link above its bandwith, results with a goodput ratio close to 0%.

3.3.1 Simulation Results – Attacks on Stub Links

A classic attack type is to target a single AS (e.g., an AS containing a Web server), usually connected by a single link to a provider AS. To test this scenario, we choose random target links for which the tail degree is 1 and the head degree is below 10. These links represent a connection between a small-size provider AS and a stub AS. In the simulated attack all the bots send traffic to the stub AS, causing packet losses on the target link.

Figure 13 show how BTT performance enhances, on this attack scheme, as its deployment widens. BTT behaves similar for the Slammer botnet distribution and for a random distribution, providing significant mitigation when deployed on all ASes.

When deployed only at the AS adjacent to the target

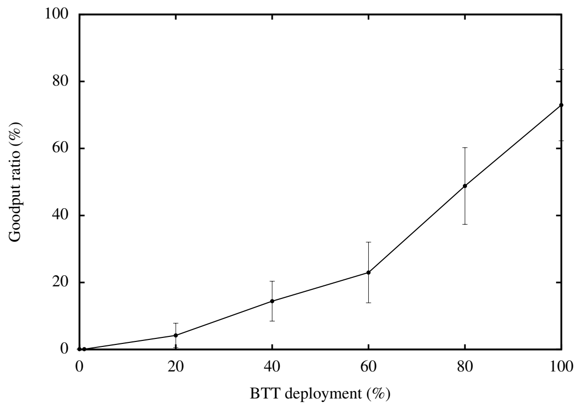

Figure 13: The goodput ratio vs. BTT deployment ratio, when the attack is against a stub link, using the Slammer distribution. The legeitimate traffic was randomized. The attack overload factor was set to 2.

(first point above 0 in the figure), poor performance is observed, as the attacking traffic overloads all the incoming links.

4 Related Work

Much work has been devoted in the recent years to protection against network-flooding DDoS attacks. Mirkovic and Reiher [23] presented a taxonomy of DDoS attacks and defenses. This paper is focused on network flooding DDoS attacks mitigation. In the current section we survey the related work in that area, we found relevant.

Some publications, e.g. [15, 28, 29], try to identify special characteristics of an attacking packet such as IP spoofing. An example for such technique is hop-counts differentiation between spoofed and real packets proposed in [15]. Ingress filtering [8] aims to block spoofed packets at the source ISP, by identifying that the spoofed packet cannot have originated from that ISP. LOT [11] 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 schemes such as Pushback [14], ACC [20], AITF [4] and StopIt [19] 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 propagating a block request upstream towards attack source(s). Unlike BTT, pushback schemes rely on identifying specific attack characteristic, such as flow volume, source IP etc. Another difference is that pushback routers require blocking assistance from upstream routers, otherwise the number of blocked flows may be too large to handle; this may also burden deployment and adoption. Finally, spoofing sources introduces additional difficulties in blocking streams at the right place.

Rate limiting schemes collect traffic statistics and shape traffic accordingly during an attack. D-WARD [24] rate-limit at the source-end. PSP [6] collects 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. PSP rate-limiting is designed for deployment in a single provider, such as an AS, and not Internet wide. In PSP, traffic statistics are collected centrally, and used by routers to tag excess packets for possible future discard by overloaded routers. Unlike PSP, BTT is fully decentralized and can be used between different providers, where no centralized authority controls the resources. BTT does not use any packet tagging, and does not rely on neighboring routers for cooperation. BTT requests traffic throttling from upstream routers, even when they are not loaded, which can distribute the workload from the overloaded router. Finally, BTT can be used as a complementary mechanism to PSP, where BTT is used as an inter-provider mechanism and PSP as an intra-provider mechanism.

Capabilities schemes such as TVA [35] and SIFF [34] 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.

Overlay schemes such as SOS [16], Mayday [2] and Phalanx [7] 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.

dFence [21] introduces middle-boxes into the ISP core, which upon DDoS activity, transparently add DDoS defense to victim servers. DefCOM [26] introduces a core-framework which enables cooperation between source, core and destination defenses during an attack, by using both shaping and pushback schemes.

Coremelt [30] is an attack type in which attackers, instead of targeting a specific destination, send traffic between their own bots. A part the possible $O(N^2)$ connections might be congesting a core link, however the communication between two bots seems legitimate and not unwanted, which makes it hard to detect, and therefore block. BTT can mitigate such attacks, by reinstating the typical rate, which should ultimately block, at least some of the attacking nodes, which create non-typical traffic rate, hence make such attacks significantly harder to launch.

5 Conclusion and Future Work

In this paper we have presented BTT, an efficient, decentralized mechanism to mitigate network-flooding attacks. BTT collects typical traffic statistics and uses them during DDoS attack to throttle the bandwidth of incoming links. Internet-based simulations indicate that BTT can throttle high percent of the attack rate and sustain a considerable amount of goodput.

As far as we know, we are the first to propose a defense mechanism to mitigate a Coremelt type attack.

Future work includes adding support for tunneled peers, that is not adjacent peers, as well as implementing BTT and testing it in testbeds for both temporal and performance evaluations.

References

[1] ANDERSEN, D., BALAKRISHNAN, H., KAASHOEK, F., AND MORRIS, R. Resilient overlay networks. *SIGOPS Oper. Syst. Rev. 35* (October 2001), 131–145.

[2] ANDERSEN, D. G. Mayday: Distributed filtering for internet services. In *USENIX Symposium on Internet Technologies and Systems* (2003).

[3] ARBOR NETWORKS. Worldwide infrastructure security report, 2010.

[4] ARGYRAKI, K., AND CHERITON, D. R. Active internet traffic filtering: real-time response to denial-of-service attacks. In *Proceedings of the annual conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2005), ATEC '05, USENIX Association, pp. 10–10.

[5] BATES, T., SMITH, P., AND HUSTON, G. CIDR report, July 2011.

[6] CHOU, J. C. Y., LIN, B., SEN, S., AND SPATSCHECK, O. Proactive surge protection: a defense mechanism for bandwidth-based attacks. *IEEE/ACM Trans. Netw. 17*, 6 (2009), 1711–1723.

[7] DIXON, C., ANDERSON, T. E., AND KRISHNAMURTHY, A. 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* (2008), J. Crowcroft and M. Dahlin, Eds., USENIX Association, pp. 45–58.

[8] FERGUSON, P., AND SENIE, D. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827 (Best Current Practice 38), May 2000. Some aspects updated by RFC 3704.

[9] FLOYD, S., AND FALL, K. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Trans. Netw. 7*, 4 (1999), 458–472.

[10] GEVA, M., AND HERZBERG, A. Qosodos: If you can't beat them, join them! In *INFOCOM'11* (2011), IEEE.

[11] GILAD, Y., AND HERZBERG, A. Lightweight opportunistic tunneling (LOT). In *ESORICS 2009*, vol. 5789 of *LNCS*. pp. 104–119.

[12] GILAD, Y., AND HERZBERG, A. Lightweight opportunistic tunneling (LOT). In *ESORICS* (2009), pp. 104–119.

[13] GILBERT, A. C., LI, Y., PORAT, E., AND STRAUSS, M. J. Approximate sparse recovery: optimizing time and measurements. In *Proc. 42nd ACM STOC* (2010), pp. 475–484.

[14] IOANNIDIS, J., AND BELLOVIN, S. M. Implementing pushback: Router-based defense against DDoS attacks. In *NDSS* (2002), The Internet Society.

[15] JIN, C., WANG, H., AND SHIN, K. G. Hop-count filtering: an effective defense against spoofed ddos traffic. In *ACM Conference on Computer and Communications Security* (2003), pp. 30–41.

[16] KEROMYTIS, A. D., MISRA, V., AND RUBENSTEIN, D. SOS: an architecture for mitigating DDoS attacks. *IEEE Journal on Selected Areas in Communications 22*, 1 (2004), 176–188.

[17] KRAWCZYK, H., BELLARE, M., AND CANETTI, R. HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational), Feb. 1997.

[18] KUZMANOVIC, A., AND KNIGHTLY, E. W. Low-rate TCP-targeted denial of service attacks and counter strategies. *IEEE/ACM Trans. Netw 14*, 4 (2006), 683–696.

[19] LIU, X., YANG, X., AND LU, Y. 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* (2008), V. Bahl, D. Wetherall, S. Savage, and I. Stoica, Eds., ACM, pp. 195–206.

[20] MAHAJAN, R., BELLOVIN, S. M., FLOYD, S., IOANNIDIS, J., PAXSON, V., AND SHENKER, S. Controlling high bandwidth aggregates in the network. *SIGCOMM Comput. Commun. Rev. 32* (July 2002), 62–73.

[21] MAHIMKAR, A., DANGE, J., SHMATIKOV, V., VIN, H. M., AND ZHANG, Y. dfence: Transparent network-based denial of service mitigation. In *NSDI* (2007), USENIX.

[22] MIRKOVIC, J., HUSSAIN, A., FAHMY, S., REIHER, P. L., AND THOMAS, R. K. Accurately measuring denial of service in simulation and testbed experiments. *IEEE Trans. Dependable Sec. Comput 6*, 2 (2009), 81–95.

[23] MIRKOVIC, J., AND REIHER, P. L. A taxonomy of DDoS attack and DDoS defense mechanisms. *SIGCOMM Comput. Commun. Rev. 34*, 2 (2004), 39–53.

[24] MIRKOVIC, J., AND REIHER, P. L. D-ward: A source-end defense against flooding denial-of-service attacks. *IEEE Trans. Dependable Sec. Comput. 2*, 3 (2005), 216–232.

[25] MOORE, D., PAXSON, V., SAVAGE, S., SHANNON, C., STANIFORD, S., AND WEAVER, N. Inside the Slammer worm. *IEEE Security Privacy 1*, 4 (2003), 33–39.

[26] OIKONOMOU, G. C., MIRKOVIC, J., REIHER, P. L., AND ROBINSON, M. A framework for a collaborative DDoS defense. In *ACSAC* (2006), IEEE Computer Society, pp. 33–42.

[27] REKHTER, Y., LI, T., AND HARES, S. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), Jan. 2006. Updated by RFC 6286.

[28] SAVAGE, S., WETHERALL, D., KARLIN, A., AND ANDERSON, T. Practical network support for IP traceback. *SIGCOMM Comput. Commun. Rev. 30*, 4 (2000), 295–306.

[29] SONG, D. X., AND PERRIG, A. Advanced and authenticated marking schemes for IP traceback. In *Proc. 20th IEEE INFOCOM* (2001), pp. 878–886.

[30] STUDER, A., AND PERRIG, A. The coremelt attack. In *ESORICS* (2009), M. Backes and P. Ning, Eds., vol. 5789 of *Lecture Notes in Computer Science*, Springer, pp. 37–52.

[31] WEI, S., KO, C., MIRKOVIC, J., AND HUSSAIN, A. Tools for worm experimentation on the DETER testbed. In *Proc. 5th TridentCom* (2009).

[32] WEI, S., AND MIRKOVIC, J. A realistic simulation of Internet-scale events. In *Proc. 1st VALUETOOLS* (2006).

[33] WIDMER, J., DENDA, R., AND MAUVE, M. A survey on TCP-friendly congestion control. *IEEE Netw. 15*, 3 (may 2001), 28–37.

[34] YAAR, A., PERRIG, A., AND SONG, D. X. SIFF: A stateless internet flow filter to mitigate DDoS flooding attacks. In *IEEE Symposium on Security and Privacy* (2004), IEEE Computer Society, p. 130.

[35] YANG, X., WETHERALL, D., AND ANDERSON, T. E. 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* (2005), R. Guérin, R. Govindan, and G. Minshall, Eds., ACM, pp. 241–252.