CICADAS: Congesting the Internet with Coordinated And Decentralized Pulsating Attacks

Yu-Ming Ke
National Taiwan University

Chih-Wei Chen
National Taiwan University

Hsu-Chun Hsiao
National Taiwan University
Academia Sinica

Adrian Perrig
ETH Zurich

Vyas Sekar
Carnegie Mellon University

ABSTRACT

This study stems from the premise that we need to break away from the "reactive" cycle of developing defenses against new DDoS attacks (e.g., amplification) by *proactively* investigating the potential for new types of DDoS attacks. Our specific focus is on *pulsating* attacks, a particularly debilitating type that has been hypothesized in the literature. In a pulsating attack, bots coordinate to generate intermittent pulses at target links to significantly reduce the throughput of TCP connections traversing the target. With pulsating attacks, attackers can cause significantly greater damage to legitimate users than traditional link flooding attacks. To date, however, pulsating attacks have been either deemed ineffective or easily defendable for two reasons: (1) they require a central coordinator and can thus be tracked; and (2) they require tight synchronization of pulses, which is difficult even in normal non-congestion scenarios.

This paper argues that, in fact, the perceived drawbacks of pulsating attacks are in fact not fundamental. We develop a practical pulsating attack called CICADAS using two key ideas: using both (1) congestion as an implicit signal for decentralized implementation, and (2) a Kalman-filter-based approach to achieve tight synchronization. We validate CICADAS using simulations and wide-area experiments. We also discuss possible countermeasures against this attack.

Keywords

DDoS attack; pulsating attack; distributed and decentralized coordination; Kalman filter

1. INTRODUCTION

Distributed Denial-of-Service (DDoS) attacks continue to be a major threat and have been evolving in both scale and sophistication [1]. As such, the defense community has been constantly playing "catch up" with attackers in this rapidly-changing attack landscape; e.g., in the last two to three years alone, new vectors like traffic amplification, the use of IoT devices as bots, and the targeting of core links have emerged [3,12,26]. Our work stems from the premise that to break out of this "catch up" game, it behooves the research community to *proactively* explore novel DDoS attacks that are stealthier and more destructive than ever before.

In this paper, we specifically focus on a class of DoS/DDoS attacks referred to as *pulsating attacks* [9,14,17,25]. These attacks target specific Internet links and can be more harmful than simple brute-force flooding, because they can effectively suppress legitimate flows by sending only intermittent bursts. For example, a series of carefully-crafted periodic pulses on a network link can deceive TCP flows over the same link to infer an extended period of congestion, thus forcing legitimate flows to repeatedly enter the timeout state.

To date, however, large-scale pulsating DDoS attacks have been mostly considered an academic curiosity because of two perceived limitations that either render the attack impractical or enable simple defenses:

- **Centralized coordination.** The adversary has to coordinate distributed bots such that their traffic streams add up to the desired waveform that resembles intermittent high-volume bursts at the target link, as Figure 1 illustrates. Prior work requires central coordination [8,24,34] and thus sacrifices resiliency and stealth, as this coordinator becomes a single point of failure for the attack.

- **Inaccurate delay estimation.** To produce the desired pulsating waveform, bots need to coordinate with each other so that their attack packets arrive at the target synchronously. Unfortunately, synchronizing clocks of hundreds of thousands of bots is difficult in a wide-area setting. To make matters worse, time synchronization itself is insufficient because the main challenge here is estimating the delay from each bot to the target link. Prior work assumes the delays remain unchanged throughout the course of an attack and estimates the delay values via a one-time measurement at the beginning of an attack. Due to the dynamic nature of Internet traffic, especially during attacks, bots fail to synchronize when relying on one-time measurements or even periodic measurements in a naïve way (see wide-area experiments in §5).

This paper indicates that it is possible to achieve decentralized pulsating attacks that are resilient to the delay variability introduced during attacks. Our proposed DDoS attack coordination mechanism, called CICADAS, builds on two key ideas.

ASIA CCS'16, May 30–June 3, 2016, Xi'an, China.
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4233-9/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2897845.2897866

- *Using congestion as an implicit coordination signal:* To synchronize bots without a central controller, we propose that bots use instants of congestion at the target link as a communication signal to synchronize among themselves. Using congestion for covert communication has two advantages. First, the signal can be observed by every bot. Second, because the same signal is critical to important networking protocols such as TCP congestion avoidance, defenses that attempt to interfere with the signal are rendered inapplicable.

- *Control-theoretic approach for accurate delay estimation:* To compensate for the delay variations to the target link, we design a feedback-based attack stream regulation algorithm that dynamically adjusts the phase and magnitude of each attack stream based on previous RTT measurements (i.e. the feedback). Applying a control theory concept, the algorithm can accurately estimate the attack sending time such that bots collectively generate a desired waveform at the target link.

Using simulations and real-world experiments, we confirm that CICADAS can successfully implement a decentralized pulsating attack. Particularly, our Internet-wide experiments show that with n bots and each bot sending 200ms-length and r-magnitude bursts, it can reliably generate 145ms-length and $0.9nr$-magnitude bursts on the target link with high probability, and the normalized throughput of the co-existing TCP flows is reduced to 0.3. In contrast, bots fail to synchronize when relying on one-time measurements. When using periodic measurements in a naïve way, bots are sensitive to delay variations and can only generate 60ms-length and $0.9nr$-magnitude bursts. We explore potential countermeasures to CICADAS in §6.

This paper makes the following contributions:

- Highlighting the challenges of achieving decentralized pulsating attacks on the Internet.

- Designing a novel decentralized coordination method that uses congestion as a signal for implicit communication among bots, such that no centralized controller is needed.

- Adopting a control-theoretic framework that allows a bot to accurately compensate for delay variations by dynamically adjusting the phase and magnitude of its attack stream.

- Evaluating the practicability of CICADAS using both simulations and Internet-wide experiments. The results confirm that CICADAS can successfully synchronize bots with high probability, whereas other approaches may be unstable or fail to synchronize.

2. BACKGROUND

In this section, we review 1) pulsating DoS attacks and 2) pulsating DDoS attacks using a central controller. We then discuss the challenges of executing pulsating attacks in a distributed and decentralized fashion.

We use the term *pulsating attack* to describe a class of (D)DoS attacks that send only intermittent bursts instead of flooding. They typically exploit network protocol specifications and thus can effectively reduce the availability of the victim service. As opposed to direct flooding, the pulsating attack is stealthy and destructive due to the low average sending rate and high damage-to-cost ratio.

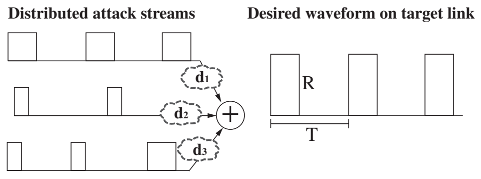


Figure 1: An illustration of distributed pulsating attacks. The desired waveform consists of periodic high-volume bursts of period T, burst length L, and burst magnitude R. Before the attack streams arrive at the target link, they experience dynamic delays (i.e., d_1, d_2 and d_3) introduced by network elements.

Pulsating DoS. The Shrew attack [14] is an attack that targets TCP and exploits the homogeneity of the minimum Retransmission Time Out (minRTO) in TCP. At a high level, the Shrew attack aims to cause short-term link congestion whenever legitimate TCP flows retransmit, which deludes them into inferring an extended period of congestion, cutting their sending rates down to almost zero. Because of the homogeneity of minRTO, which implies that flows experiencing simultaneous packet loss would likely retransmit at the same time, Shrew attacks can cause severe damage by sending a carefully crafted rectangular-wave DoS stream whose period approximates the minRTO (e.g., one second) and duration approximates the maximum Round-Trip Time of victim flows (e.g., 200 milliseconds). Worse yet, the analytical result of the Shrew attack shows that randomizing the minRTO parameter cannot fully mitigate this attack.

The RoQ attack [9] aims to degrade system performance rather than completely disable the service (the latter typically being the goal of DoS), and its effectiveness can be evaluated by the ratio between damage and cost. The RoQ attack can be implemented in a similar way to the Shrew attack: the attacker directs a periodic waveform to a target link in order to destabilize the adaptation process of the system, thereby preventing the system from converging to a stable state where performance is usually optimized. One well-known adaptive protocol is the Additive-Increase-Multiplicative-Decrease (AIMD) algorithm used in TCP congestion avoidance. The difference between the Shrew and RoQ attacks is that the Shrew attack exploits the timeout mechanism of TCP and aims to cause thorough denial of service, while the RoQ attack does not target a specific mechanism or protocol and degrades system performance by maximizing the ratio between damage and cost.

Pulsating DDoS using a centralized controller. Several works propose using a centralized controller to synchronize distributed attacks. Guirguis et al. [8] presented a distributed RoQ attack that obtains an unfair share of bandwidth on a target link. However, they assume a controller for DDoS coordination without specifying many details. Zhang et al. [34] studied the possibility of disrupting BGP with pulsating attacks. In their distributed version, bots are required to synchronize with a centralized controller that can observe all attack paths and is close to the target link, so as to reduce the impact of delay variance on time synchronization. Before the attack starts, each bot sends a timestamped packet to the controller, which then instructs the bots when to start the attack based on the time difference. However, it is difficult to accurately predict dynamic delays based on one sampled time difference, especially when the delay variance increases dramatically during attacks.

Challenges of distributed and decentralized pulsating attacks. It is natural to extend pulsating attacks to a distributed and decentralized setting, where multiple bot machines collectively create the periodic attack waveform without central controllers. A distributed pulsating attack is more lethal than its predecessor because it is capable of achieving a higher peak rate, thus making it possible to target high-capacity backbone links. A decentralized pulsating attack is more robust against takedown operations targeting the command and control servers. However, as mentioned above, one critical research challenge of constructing this type of pulsating attack is coordinating distributed bots such that their traffic streams can add up to the target waveform resembling intermittent high-volume bursts at the target link. While the adversary could attempt to synchronize the clocks of all bots and predict the delay from each bot to the target link, the delays are difficult to predict accurately due to the dynamic nature of Internet traffic.

3. CICADAS ATTACK OVERVIEW

In this section, we provide a high-level workflow of the CICADAS attack and highlight the key technical challenges. Before this, we formally specify the adversary's goals.

3.1 Goals and Requirements

We envision an intelligent adversary that wants to cause network congestion at a specific network link by inducing a pulsating traffic volume waveform on this link. We assume that the adversary has chosen the link through out-of-band mechanisms and through a combination of Internet topology maps [10–12].[1] As such, our focus in this paper is on the pulsating mechanism rather than the target selection.

The goal of the attack is to orchestrate n bots to synthesize a traffic stream of period T, burst length L, and burst magnitude R at the target link, as shown on the right hand side of Figure 1. Typical values are $T = minRTO = 1$ second, $L \approx maxRTT = 200$ ms, and R is slightly higher than the capacity of the target link to overflow the router queues [14].

As discussed earlier, there are two high-level requirements for the attack to be effective in practice. It should (1) require no central coordination point, and (2) achieve tight coordination despite high network delay variations.

3.2 CICADAS Intuition

The first-order goal of CICADAS is to coordinate the behavior of the bots in a decentralized manner and with minimal assumptions on the Internet status, such as delay variances. Here, the key insight behind CICADAS is that congestion at the target link can itself act as a "signal flare" that every bot can observe (e.g., through packet loss and delay) and synchronize with. Specifically, each bot infers the extent of congestion based on the variations in observed latency.

[1]One potential concern is early congestion, when the attack traffic congests a non-target link before reaching the target link. We assume that the adversary can avoid this on the chosen link via out-of-band bandwidth estimation techniques [10], similar to prior work [12].

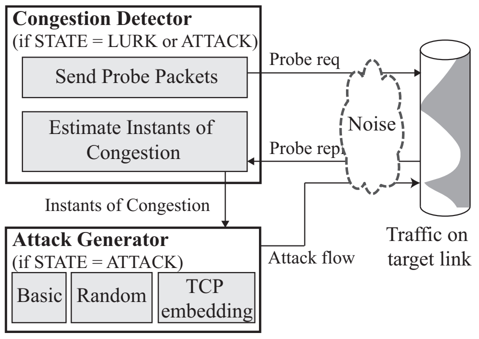


Figure 2: CICADAS adjusts the attack stream based on the estimated instants of congestion, which is inferred from RTT measurements.

Based on this insight, CICADAS can be viewed as a *feedback-based attack stream regulator* installed on each bot, as Figure 2 illustrates:

- **Congestion Detector** estimates instants of congestion at the target link based on observed RTTs.
- **Attack Generator** uses these estimations to regulate the attack stream, so that distributed attack streams can aggregate to periodic pulses at the target link without the help of a centralized online controller.

3.3 CICADAS Workflow

Given the insight above, we will describe the end-to-end workflow of CICADAS that has three conceptual stages: BOOTSTRAP, LURK, and ATTACK (Figure 3). That is, the CICADAS feedback loop can be intuitively viewed as a *state machine* that transitions between these three states. We describe the bot actions in each state below.

BOOTSTRAP In the BOOTSTRAP phase, each bot receives attack parameters, including the target link and target waveform (i.e., attack period, burst length, and burst magnitude). For instance, this bootstrapping may happen as part of the bot recruitment phase or other kinds of malware payload distribution schemes. At the end of this phase, the bot is not required to send any traffic. The bots may move to the LURK state (see below) at a predetermined time or when the estimated size of the botnet has passed a threshold [13,19]. Note that bots can enter the LURK state at different times; no time synchronization is needed.

LURK A bot in the LURK state stealthily monitors the target link via periodic Round-Trip Time (RTT) probing. Once congestion is sensed (e.g., high RTT is observed with high confidence), the bot switches to the ATTACK state. That is, each bot considers the first congestion at the target link as a signal for activating the attack stream. It is possible that legitimate/transient congestion at a non-target link falsely triggers a subset of the bots. CICADAS is designed to be resilient to such congestion. We will discuss this case in detail in §6.

ATTACK During the ATTACK state, a bot starts sending pulses that it hopes will contribute to periodic congestion at the target link. In addition, the bot adjusts the phase

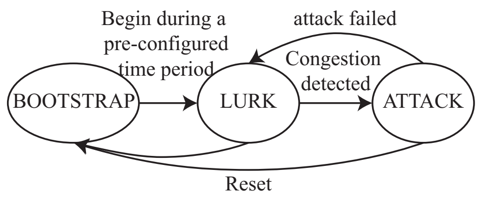


Figure 3: CICADAS's state machine view.

and magnitude of the next pulse based on the observations of past congestion so as to improve the level of coordination between bots and increase the severity of congestion at the target link. If a sufficient number of bots are triggered by the same congestion, the target link is anticipated to start periodically experiencing severe attack-induced congestion without any further legitimate congestion, as legitimate flows are throttled by the attack. If the attack coordination fails (i.e., there is a lack of severe congestion for an extended period of time), then the bot returns to the LURK state.

3.4 Practical Challenges

Given this high-level workflow, two key challenges remain to making CICADAS practical. They are addressed in §4.

1. How can we obtain a reliable feedback signal to detect instants of congestion at the target link with stealthy probe packets? Sending Ping packets at a high rate will raise suspicion.
2. How can we adjust an attack stream to improve coordination among bots? In particular, the challenge is eliminating noises introduced by dynamic network delays, the sampling of congestion signal, etc., and the noise filter should be able to learn from past observations to improve the results.

4. DETAILED ATTACK DESIGN

As depicted in Figure 2, CICADAS's control loop consists of two main modules: Congestion Detector (activated in the LURK and ATTACK phases) and Attack Stream Generator (activated in the ATTACK phase).

4.1 Limitations of Strawman Approaches

We present two strawman approaches to Congestion Detector. The impracticality of these approaches motivate our design using a control-theoretic approach for accurate congestion estimation.

Strawman: Estimate the sending time based on a one-time measurement at the beginning, and then simply send the attack bursts every T slots. Attack synchronization is likely to fail in this case because first, if the one-time measurement is inaccurate, there is no way to correct it later; second, even if the one-time measurement is very accurate at that moment, network delays can change over time (especially during attacks), and therefore there is no guarantee that the attack bursts can still arrive at the same time later.

Strawman: Send probe packets periodically but only use the latest measurement for estimation. Each bot sends an attack burst at $T_c + T$, where T_c is the time at which the latest congestion is observed. This approach is better than the first, but is sensitive to noise (mainly due to dynamic delays). Thus, bots take a longer time to converge and may be more unstable than in our scheme.

Essentially, we desire a filter to remove noise introduced by network delays and sampling errors, and the filter's capability should improve as measurements are collected, so that the degree of bot coordination can be improved progressively. Feedback-based control loops can achieve this by taking past measurements into account for estimation, and the Kalman filter is example of this. Our evaluation (presented in §5) confirms that using the Kalman filter significantly outperforms the above two strawman approaches.

4.2 Congestion Detector

The congestion detector, as illustrated in Figure 2, performs two tasks: sending probe packets to monitor congestion on the target link, and estimating when congestion occurs at the target link based on the probe packets.

Sending probes Each bot sends probe packets to the destination at a frequency higher than $\frac{1}{L}$, where L is the burst length (e.g., 200ms). This ensures that any congestion lasting longer than L can be sensed. Alternatively, the bot can send out each probe probabilistically, such that with a high probability, more than a threshold number of bots will sense the same burst. To improve stealthiness, CICADAS uses custom probe packets that are sent only between bots. Such probe packets are indistinguishable and consume a negligible amount of bandwidth (e.g., about 2.2 KB/s, when the probe packet size is 44 Bytes and probes are sent every 20 ms), and thus will likely stay under the radar without causing any alarm.

If the coordination succeeds, we expect to see periodic pulses (approximately every T units of time) in the RTT measurements. These measurements are smoothed out using statistical methods such as a simple moving average computed over the past few RTT samples.

Predicting next congestion event Taking the RTT measurements as the input, we now explain how to identify the instants of congestion. More specifically, we would like to know when the kth congestion occurs at the target link.

$z(k)$ denotes the observation of the kth congestion and $x(k)$ denotes the true state of the kth congestion. We first describe how to obtain $z(k)$ based on the probing results, and then formalize the relationship between $z(k)$ and $x(k)$, so that we can approximate $x(k)$ based on the observed $z(k)$.

Obtaining $z(k)$***:*** To obtain $z(k)$, CICADAS focuses on detecting the beginning (called the *on-edge*) and the end (called the *off-edge*) of each congestion event.

Let $OnEdge(i)$, $OffEdge(i)$, and $Cong(i)$ denote the indicators of probe i's correspondence to on-edge, off-edge, and congestion, respectively. Intuitively, an on-edge is detected by probe i if the ith RTT value is higher than a threshold Th_{on} and probe $i-1$ did not experience congestion. An off-edge is detected if the ith RTT value is lower or equal to another threshold Th_{off} and probe $i-1$ experienced congestion. Probes between a consecutive on-edge and off-edge thus correspond to congestion, as illustrated by Figure 4. Formally,

$$\begin{aligned}
&OnEdge(i) = \neg Cong(i-1) \wedge (rtt(i) > Th_{on}) \qquad (1)\\
&OffEdge(i) = Cong(i-1) \wedge (rtt(i) \leq Th_{off})\\
&Cong(i) = (Cong(i-1) \wedge \neg OffEdge(i)) \vee OnEdge(i)
\end{aligned}$$

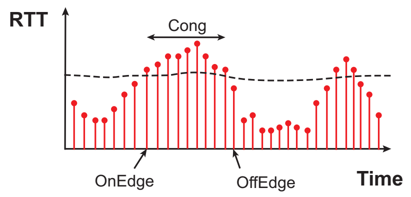


Figure 4: An example of on-edge and off-edge.

For example, the threshold values can be set such that $Th_{on} = Th_{off} = \overline{RTT}$. Since we expect to see the duration of malicious congestion to be approximately L, the congestion detector removes inadequately long congestion, e.g., $< L/2$, after being triggered. If multiple congestion events are observed within T, the detector keeps only the most "powerful" congestion, where the power is quantified by the average RTT during the period of congestion.

After the removal, the instant of the kth congestion, $z(k)$, is set to the **sending time of the probe packet** that corresponds to the kth on-edge. Next, we use the Kalman filter to eliminate network noises embedding in the observed state of congestion $z(k)$; the true state of congestion $x(k)$ can then be estimated by $z(k)$.

Approximating $x(k)$ from $z(k)$ using Kalman filter: Since we know that the target link will be congested approximately every T units of time when the attack succeeds, we can model $x(k)$ as a linear dynamical system:

$$\begin{aligned} x(k) &= x(k-1) + T + v(k) \qquad (2) \\ z(k) &= x(k) + w(k), \end{aligned}$$

where $v(k)$ is the process noise (i.e., the noise introduced by imperfect synchronization in our case) and $w(k)$ is the observation noise (i.e., noise due to delay variation).

This basic linear model provides us with a solid foundation to solve this congestion prediction problem. Although this formulation may be too simplified to capture the complex interactions between the network and bots, using simulations and Internet-wide experiments, we show that this basic model performs well for the purpose of coordinating attack streams. Given the model of the system, the Kalman filter performs the *prediction* and *update* recursively for each new observation.

Let $\hat{x}(k|k')$ be the estimate of $x(k)$ given observations $z(k'), z(k'-1), \cdots, z(0)$. Further, let $P(k|k')$ quantify the accuracy of the estimate $\hat{x}(k|k')$:

$$\begin{aligned} \hat{x}(k|k-1) &= \hat{x}(k-1|k-1) + T \qquad (3) \\ P(k|k-1) &= P(k-1|k-1) + V(k), \end{aligned}$$

and the bot updates $\hat{x}$ and P for each new observation $z(k)$:

$$\begin{aligned} \hat{x}(k|k) &= \hat{x}(k|k-1) + K_g(k)(z(k) - \hat{x}(k|k-1)) \qquad (4) \\ K_g(k) &= \frac{P(k|k-1)}{P(k|k-1) + W(k)} \\ P(k|k) &= (1 - K_g(k))P(k|k-1), \end{aligned}$$

where $K_g(k)$ is called the Kalman gain, and $V(k)$ and $W(k)$ are the covariances of $v(k)$ and $w(k)$, respectively. In practice, we can estimate the covariances using advanced tools, such as Autocovariance Least-Squares, or by calculating the variance of RTT via probing. The Kalman gain controls how much we should trust the new observation over the current estimate. Hence, the new estimate, $\hat{x}(k|k)$, is a weighted sum (determined by the Kalman gain) of the new observation and the current estimate.

In summary, using the Kalman filter, each bot predicts when the kth congestion will occur and sends a burst during the predicted instants of congestion, i.e., during intervals

$$[\hat{x}(k|k-1), \hat{x}(k|k-1) + L). \qquad (5)$$

4.3 Attack Stream Generation

There are many ways to combine n individual flows into the desired waveform. For example, there are three possible methods for generating individual attack streams: Basic, Random, and TCP Embedding. The latter two can be used to enhance the stealthiness of the attack.

Basic In the Basic method, each bot contributes a pulse of duration L and magnitude R/n to the predicated congestion interval derived in Eq. 5. This can be easily generalized to cases where each bot sends bursts of duration L/Δ and magnitude $\frac{\Delta \cdot R}{n}$.

Random The periodic nature of the pulsating attack suggests defense in the frequency domain [6, 17]. Although such approaches are often resource-intensive and it is unclear whether they can be applied to detect individual pulsating flows, the CICADAS adversary may still desire to obscure the periodic patterns for increased stealthiness.

The Random method probabilistically schedules a burst for each predicted congestion interval (Eq. 5), such that the expected value of the instantaneous rate at any point during an on-phase is $\frac{R+\epsilon}{n}$, where R is the magnitude of bursts at the target link, and n is the number of attack streams. The value of ϵ is set to ensure an instantaneous rate higher than R with high probability. The details of the parameter selection can be found in Appendix A.1.

TCP embedding So far, we have described attacks using unresponsive flows (e.g., UDP) that do not respond to congestion. Here, we present a novel technique called TCP Embedding, which allows bots to attack a target link and obtain an unfair share of bandwidth using responsive flows (e.g., TCP). This method not only enhances the stealthiness of CICADAS but also allows CICADAS to carry on the attack even when the TCP and UDP flows are segregated.

A flow is responsive if it faithfully adjusts its sending rate according to the TCP congestion control algorithm. Since bots know roughly when a link will be unobstructed (i.e., during the off-phases), the idea behind leveraging responsive TCP flows is that they can reliably increase the sending rates by sending packets during off-phases. In doing so, the resulting attack flow will be indistinguishable from a long-lived TCP flow that gradually increases its sending rate and happens to send traffic bursts during severe congestion from time to time. The details of how to construct such attack streams are presented in Appendix A.2

5. EXPERIMENT

To examine whether CICADAS can successfully orchestrate a distributed pulsating attack without a centralized controller, we conduct extensive experiments on both the Internet and the ns-3 simulation platform.

5.1 Evaluation Metrics

We consider three metrics for quantifying the effectiveness of a distributed pulsating attack.

Differential Time of Arrival A *majority* of attack packets arriving at the target link within a *short* time window is a direct indicator of synchronization. Formally, we define the *Differential Time of Arrival* (Δ_{TOA}) as:

$$\Delta_{TOA}(m, \alpha) = \min_{1 \le k \le n} [T_m(k + \alpha n) - T_m(k)] \tag{6}$$

where $T_m(k)$ is the arrival time of the kth bot (in temporal order) in the mth period, α is a parameter quantifying "majority", and n is the number of attack streams. For instance, $\Delta_{TOA}(m, 75\%) = 0.1s$ means that at the mth period, 75% of attack streams are arriving within 0.1 seconds. (The arrival time of an attack stream in the mth period is defined as the arrival time of the first packet in an attack stream within this period.) Thus, a smaller Δ_{TOA} value implies a more tightly synchronized attack.

Closeness of Synchronization The *closeness of synchronization* is defined as the average $\Delta_{TOA}(m, \alpha)$ over consecutive time periods in one run:

$$closeness\_of\_sync(m_1, m_2) = \frac{\sum_{i=m_1}^{m_2} \Delta_{TOA}(i, \alpha)}{m_2 - m_1}, \tag{7}$$

where m_1 and m_2 specify a given range. To focus on the steady state behavior, we will set m_1 to larger than 0 in our evaluation.

Normalized Throughput Apart from the synchronization of attack streams, the destructiveness of DDoS attacks can also be evaluated by the *normalized throughput* of co-existing legitimate flows.

5.2 Internet-wide Experiments

To show the feasibility of CICADAS in realistic settings, we conduct Internet-wide experiments using 64 geographically distributed machines, as summarized in Figure 5.

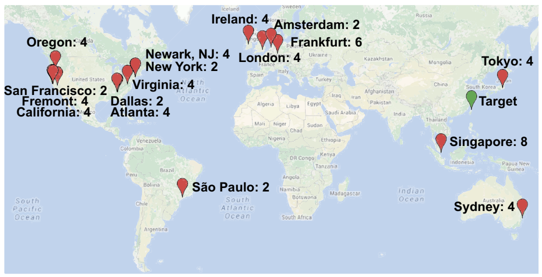


Figure 5: The locations and the number of machines used in our experiments.

Methodology We carefully design our experiments such that we can evaluate our DDoS coordination mechanism without affecting other services or inadvertently triggering the alarms of existing detection systems.

The bot machines reside in several cloud infrastructures, including Amazon EC2, DigitalOcean and Linode. Such a placement not only ensures that the inter-bot traffic can experience realistic delays, but also reduces the impact on individual cloud platforms. Each machine is either a bot-sender or a bot-receiver, and sends packets to each other at a very low rate to avoid flooding the Internet.

For monitoring ease, the target link is represented by a local machine under our control that performs three tasks:

- Forwarding packets between bot-senders and bot-receivers. Bot-senders and bot-receivers exchange packets via the target link, and NAT (Network Address Translation) is set on our machine for forwarding.
- Monitoring packets for deriving metrics of interest.
- Limiting the bandwidth of the target link such that bots can saturate the target link without disturbing other co-existing flows on the path. We use `tc` (traffic control) to constrain the bandwidth on the target.

In our experiments, each bot spends the first 10 seconds establishing a baseline of $\overline{RTT}$ before sending attack packets, and stops at 200s. The probing rate is 3KB/s and an attack burst is 20KB for each bot.

5.3 Results of Internet Experiments

First, one experiment instance is used to show how well the attack streams synchronize. Figure 6 illustrates the differential time of arrival $\Delta_{TOA}(m, \alpha)$ for one experiment when α is 75% and 90%. This result shows that attack streams are synchronized after about 25 seconds. $\Delta_{TOA}(m, 75\%)$ is less than 40ms and $\Delta_{TOA}(m, 90\%)$ is less than 60ms for most of the time periods. This indicates that the burst generated at the target link is about 160ms ($= 200 - 40$) in length and 75% of the maximum attack rate in magnitude; it can also be seen as 140ms ($= 200 - 60$) in length and 90% of the maximum attack rate in magnitude.

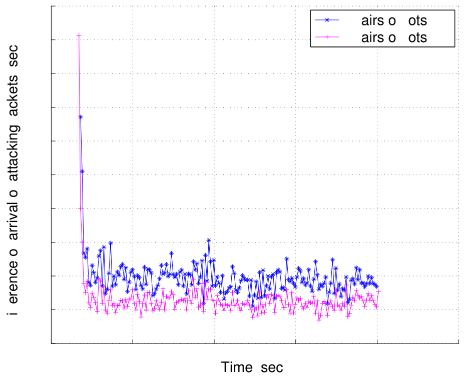


Figure 6: Differential time of arrival in one run.

We then investigate whether CICADAS can successfully coordinate bots with high probability. Figure 7 illustrates the cumulative distribution vs. the closeness of synchronization for 100 experiment instances. The results show that, in 90 out of 100 instances, the closeness of synchronization is less than 37ms and 54ms for $\alpha = 75\%$ and 90%, respectively.

CICADAS uses a feedback-based control loop to estimate the true state of congestion, and the Kalman filter serves as one example throughout the paper. To understand whether using Kalman filters is more effective than straightforward approaches, we substitute the Kalman filter in CICADAS with two other methods for comparison:

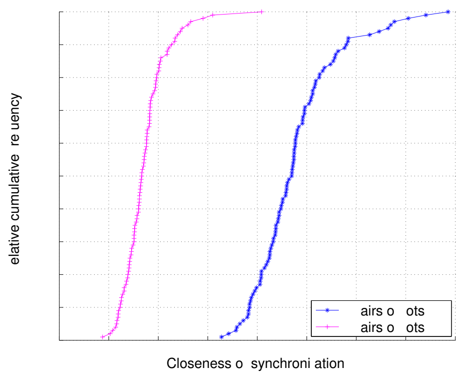


Figure 7: Relative cumulative frequency with multiple experiments.

- **No Feedback**: a one-time measurement. When bots observe the first congestion after establishing a baseline of $\overline{RTT}$, bots start attacking periodically at times $first\ congestion\ time + mT$, where m is mth period and T is the length of one period.
- **Simple Feedback**: a simple Kalman filter with the Kalman gain fixed to 1. That is, the bot estimates the true state of congestion based solely on the most recent observation. In other words, $\hat{x}(k|k) = z(k)$ when bots update the true state of congestion $\hat{x}$.

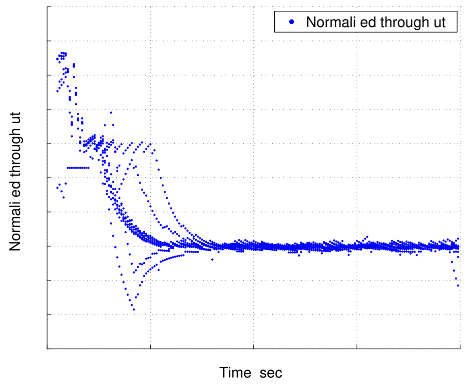


Figure 9: Normalized TCP throughput of 10 runs.

We run 100 instances for each method. For the No Feedback method, the closeness of synchronization is consistently higher than 200ms, which indicates failed synchronization. Figures 8(a) and 8(b) compare the results for the Simple Feedback and the Kalman Filter methods. In both figures, CICADAS (with the Kalman filter) outperforms the Simple Feedback method, as it is more likely to achieve lower values regarding the closeness of synchronization.

to show that CICADAS can obtain an unfair share of bandwidth, we use the *scp* command to download a large file via the target link and examine its throughput. Figure 9 shows the normalized TCP throughput of 10 experiments. Once the bots were synchronized, the normalized throughput was effectively reduced to 0.3.

5.4 Simulation

The ns-3 simulator allows us to investigate how well CICADAS can synchronize attack streams under different network conditions by changing corresponding parameters.

Methodology Since our goal is to test the sensitivity of various network conditions, we use a simple dumbbell topology with attack and TCP flows sharing a bottleneck link. Realistic topologies are tested in our Internet experiments.

Attack traffic consists of two types of packets: probe packets and burst packets. Each bot starts sending probe packets at roughly the same time. Once started, probe packets are sent periodically (default is 20 ms) to measure the RTT at a fine time granularity. The link delays are randomly selected at the beginning of a simulation.

5.5 Simulation Results

Similar to Figures 6 and 7, we first inspect the level of synchronization in one simulation instance, which allows us to cross-check the results of the Internet-wide experiments and simulations.

Figure 10(a) shows the differential time of arrival for one simulation instance. In this setting, we consider 10 legitimate TCP flows sharing the bottleneck link with 20 attack streams in a dumbbell topology. $\Delta_{TOA}(m, 75\%)$ and $\Delta_{TOA}(m, 90\%)$ are less than 20ms for most of the time periods, which means that the burst length can be as wide as 180ms $(= 200 - 20)$.

Figure 10(b) shows the cumulative distribution versus the closeness of synchronization for 100 simulation instances. The results show that in 90 out of 100 instances, the closeness of synchronization is less than 17ms and 24ms for $\alpha = 75\%$ and 90%, respectively.

The level of synchronization is generally higher in the simulation than in the Internet-wide experiments because the network delays in the simulation are fixed, whereas delays fluctuate a lot in the real network.

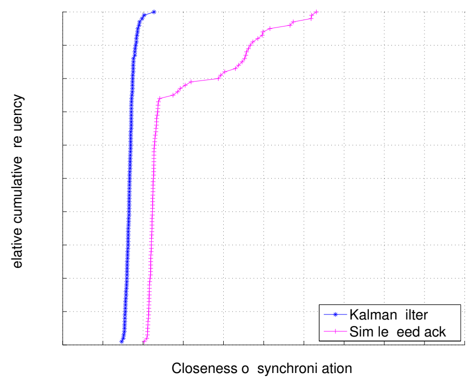
After confirming that CICADAS can effectively coordinate multiple attack streams, we study what factors affect the degree of coordination. We measure the closeness of synchronization for 1) different probe intervals, 2) different ranges of start times, and 3) different numbers of bot pairs.

5.5.1 *Probe Interval*

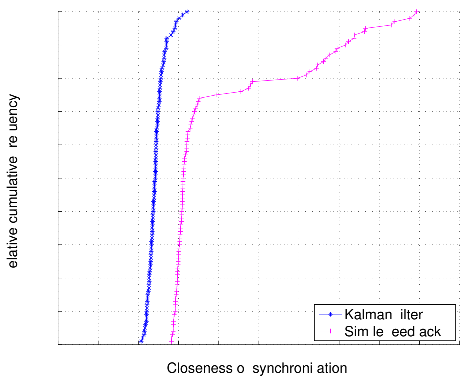
The closeness of synchronization can also be affected by the probing interval. As can be seen in Figure 11(a), the closeness of synchronization increases with the probe interval. There is a trend that the closeness of synchronization increases with the probe interval. This result confirms that a small probe interval provides bots with fine-grained information, resulting in a high degree of coordination. As the probe interval grows, each bot obtains fewer RTT samples per congestion, therefore making it difficult to differentiate attack-induced congestion from noise.

5.5.2 *Bot start time range*

Figure 11(b) shows the closeness of synchronization versus the range of start time of bots. We set the value $m_1 = 50$ in (7), because some bots start later and need time to synchronize. This result confirms that bots do not need to start at exactly the same time. On the other hand, when CICADAS attacks are in progress, bots can join the attack at a later time.

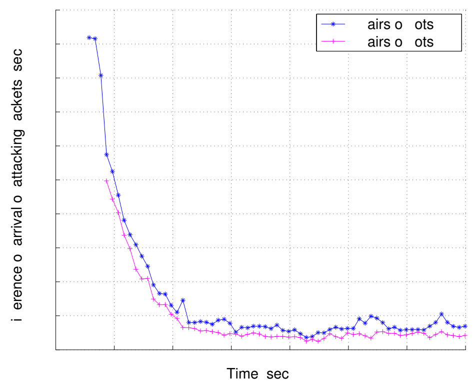


(a) 75% pairs of bots

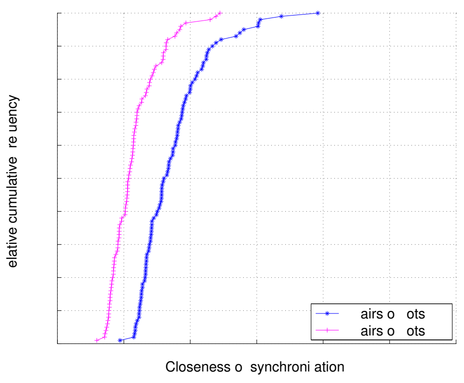


(b) 90% pairs of bots

Figure 8: The closeness of synchronization using different delay estimation methods. The No Feedback method failed constantly and thus its results were not depicted here.



(a) Differential time of arrival with one simulation on ns-3.



(b) Relative cumulative frequency with multiple simulations on ns-3.

Figure 10: Simulation results.

5.5.3 *Number of bot pairs*

Figure 11(c) shows the closeness of synchronization versus the number of bot pairs. CICADAS can reliably synchronize bots regardless of the size of the botnet.

5.6 Summary of Evaluation

The results of the wide-area Internet experiments confirm that CICADAS can effectively synchronize attack streams despite dynamic delays, whereas prior approaches cannot. The ns-3 simulation results confirm that CICADAS can reliably synchronize bots under different network conditions.

6. DISCUSSION

Congestion at non-target links There are two questions related to congestion at non-target links.

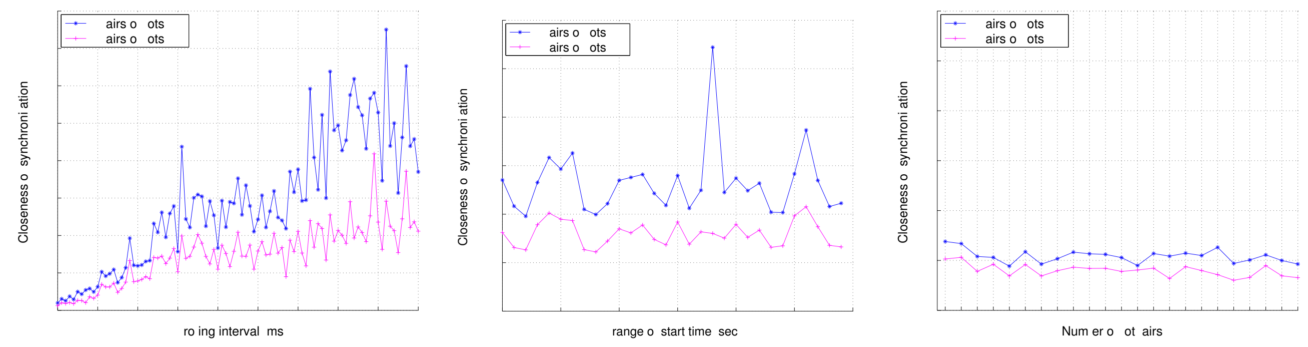
First, how does CICADAS react to transient congestion occurring at non-target links? Ideally, CICADAS should treat this type of congestion as noise, since CICADAS intends to regulate an attack stream based on the instants of congestion at the target link. To understand how such noise can be eliminated, we first note that our congestion detector is tuned to detect severe congestion only once it leaves the LURK state. Congestion of inadequate length or intensity will be filtered out quickly. Hence, we can minimize interference by legitimate congestion at non-target links, which typically has a short duration and small magnitude.

Second, will CICADAS cause congestion at non-target links? Similar to prior work [12], CICADAS can avoid causing malicious congestion at non-target links by estimating link capacity using existing bandwidth estimation tools [29].

Non-target congestion caused by an aggregate of legitimate and attack traffic will also be viewed as noise.

The above observation is confirmed by our Internet-wide experiments: In these experiments, bots may be triggered by congestion at non-target links and send bursts lasting for 0.2s during the next period. These bots either cause a slight congestion in a short period of time at the target link and create a cascading effect in the following time periods, or cause no effect on the target link.

Path change During a CICADAS attack, the communication path between bots may change. Such a path change may affect the observed delays and thus decrease the level of synchronization among bots in the short term. In the long term, CICADAS's feedback mechanism can help bots gradually converge toward the strongest attack force. If quick recovery is desired, the bot that detects a path change (e.g.



(a) Probe interval versus closeness of synchronization.

(b) Start time versus closeness of synchronization.

(c) Number of bot pairs versus closeness of synchronization with large queue.

Figure 11: Simulation results.

via traceroute) can switch back to the BOOTSTRAP state and re-synchronize with the other bots.

Trigger signal During the LURK state, a bot is triggered by the first observable congestion. In the absence of congestion, bots themselves can create triggers. For instance, bots can send normal TCP traffic to each other, since the TCP flow aggregate will naturally exhibit saw-tooth-like patterns, which saturate the link from time to time.

It is possible that bots may be triggered by different congestion instances on the target link due to their different start times, and thus attack at different times in the following period. However, once triggered, bots will converge over time as they observe similar congestion signals.

Potential countermeasures CICADAS is designed to circumvent common DDoS detection mechanisms because bot traffic in CICADAS is legitimate-looking: bots send low-volume traffic to each other using real IP addresses. Moreover, by design, the temporal pattern of the CICADAS attack is observable only at the target link, thereby rendering the redirection-based cloud traffic scrubbing service unusable. To mitigate such a stealthy DDoS attack, there needs to be support for DDoS-resilient Internet architectures [5] and efficient flow monitoring algorithms running at each router. However, none of these is easy to do with respect to design and deployment. As a proof of concept, we present a metric called *Disturbance Ratio*, by which a router can accurately distinguish legitimate flows and pulsating attack flows. The precision and recall rates are promising, but it remains to be seen how to compute such a metric at line rate without keeping per-flow state at routers. Countermeasure details can be found in Appendix B.

7. RELATED WORK

The most closely-related work is reviewed in §2. This section briefly reviews other DDoS attack and defense techniques. We refer interested readers to survey papers [20,23,33] for comprehensive studies.

To mitigate DDoS, existing solutions typically aim to throttle flows that exhibit characteristics such as spoofed IP addresses [4,7,16], undesired by destinations [31,32], and disproportionate bandwidth share [15,18,22,28]. However, the CICADAS attack appears legitimate to many detection algorithms since it can be performed by low-volume inter-bot traffic using real IP addresses. The periodic pattern of pulsating attacks is a distinguishing characteristic with regard to attack detection [6,17]. However, existing detection algorithms are too resource-intensive to be performed on routers.

The crossfire attack [12] aims to cut off a region from the Internet by flooding a few critical links that connect the two. To achieve this, algorithms are proposed for selecting the target links and attack paths. While our work also targets network links instead of end servers, it focuses on designing an attack coordination mechanism and thus is orthogonal to the crossfire attack. In Temporal Lensing [25], a single attack source uses paths with different latencies such that a flooding attack is transformed into a pulsating attack with a higher peak against the victim endhost. Its latency estimation relies on existing DNS infrastructure and an assumption that there are always DNS resolvers topologically close to end points. However, this assumption may not hold when the target is an intermediate network link rather than an endhost, and this approach may become less effective as more and more open DNS resolvers are closing [2].

8. CONCLUSION

Recent DDoS threats (e.g., amplification attacks and flooding core links) pose new research challenges for providing available communication in critical network infrastructures. The increased sophistication and impact of DDoS attacks motivates the study of future attack trends. This work focuses on the practicability of a new type of DDoS attack, referred to as the pulsating attack, which is considered powerful but has not yet been observed in a real-world setting. To show that it is possible to overcome the perceived limitations of pulsating attacks, we develop a novel attack coordination mechanism called CICADAS, which enables bots to synchronize with each other with no central controller in a highly dynamic network. Moreover, CICADAS coordinates bots in the *temporal domain* and can work in conjunction with other DDoS mechanisms that coordinate bots in the *spatial domain* [12,30] or temporally concentrate one flood into high-peaked pulses [25].

Acknowledgments

This work was supported in part by the Ministry of Science and Technology, National Taiwan University, and Intel Corporation under Grants MOST-103-2218-E-002-034-

MY2, MOST-104-2218-E-001-002, MOST-105-2633-E-002-001, and NTU-ICRP-105R104045.

9. REFERENCES

[1] ATLAS Q2 2015 Global DDoS Attack Trends. http://www.slideshare.net/Arbor_Networks/atlas-q2-2015final.

[2] Open Resolver Project. http://openresolverproject.org/.

[3] Spike DDoS Toolkit Threat Advisory. https://www.stateoftheinternet.com/resources-web-security-threat-advisories-2014-multi-platform-botnet-spike.html.

[4] K. Argyraki and D. R. Cheriton. Active Internet Traffic Filtering: Real-Time Response to Denial-of-Service Attacks. In *Proceedings of USENIX ATEC*, 2005.

[5] C. Basescu, R. M. Reischuk, P. Szalachowski, A. Perrig, Y. Zhang, H.-C. Hsiao, A. Kubota, and J. Urakawa. SIBRA: Scalable Internet Bandwidth Reservation Architecture. In *Proceedings of NDSS*, 2016.

[6] Y. Chen and K. Hwang. Collaborative Detection and Filtering of Shrew DDoS Attacks Using Spectral Analysis. *Journal of Parallel and Distributed Computing*, 66(9):1137–1151, Sept. 2006.

[7] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827 (Best Current Practice), May 2000. Updated by RFC 3704.

[8] M. Guirguis, A. Bestavros, and I. Matta. Bandwidth Stealing via Link-Targeted RoQ Attakcs. In *Proceedings on Communication and Computer Networks*, 2004.

[9] M. Guirguis, A. Bestavros, and I. Matta. Exploiting the Transients of Adaptation for RoQ Attacks on Internet Resources. In *Proceedings of IEEE ICNP*, 2004.

[10] N. Hu, L. Li, Z. Mao, P. Steenkiste, and J. Wang. Locating Internet bottlenecks: Algorithms, measurements, and implications. *SIGCOMM Comput. Commun. Rev.*, 34(4):41–54, 2004.

[11] M. S. Kang and V. Gligor. Routing Bottlenecks in the Internet: Causes, Exploits, and Countermeasures. In *Proceedings of ACM CCS*, 2014.

[12] M. S. Kang, S. B. Lee, and V. D. Gligor. The Crossfire Attack. In *Proceedings of IEEE Symposium on Security and Privacy*, 2013.

[13] D. Kostoulas, D. Psaltoulis, I. Gupta, K. Birman, and A. Demers. Decentralized Schemes for Size Estimation in Large and Dynamic Groups. In *Proceedings of IEEE International Symposium on Network Computing and Applications*, 2005.

[14] A. Kuzmanovic and E. Knightly. Low-Rate TCP-Targeted Denial of Service Attacks: the Shrew vs. the Mice and Elephants. In *Proceedings of the ACM SIGCOMM*, 2003.

[15] Y. Kwok, R. Tripathi, Y. Chen, and K. Hwang. HAWK: Halting Anomalies with Weighted ChoKing to Rescue Well-Behaved TCP Sessions from Shrew DDoS Attacks. *Networking and Mobile Computing*, 3169(August):1–10, 2005.

[16] X. Liu, A. Li, X. Yang, and D. Wetherall. Passport: Secure and Adoptable Source Authentication. In *Proceedings of USENIX/ACM NSDI*, 2008.

[17] X. Luo and R. Chang. On a New Class of Pulsing Denial-of-Service Attacks and the Defense. In *Proceedings of NDSS*, 2005.

[18] R. Mahajan, S. Floyd, and D. Wetherall. Controlling High-Bandwidth Flows at the Congested Router. In *proceedings of IEEE ICNP*, 2001.

[19] L. Massoulié and E. L. Merrer. Peer Counting and Sampling in Overlay Networks: Random Walk Methods. In *Proceedings of ACM PODC*, 2006.

[20] J. Mirkovic and P. Reiher. A Taxonomy of DDoS Attack and DDoS Defense Mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[21] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala. Path Splicing. In *Proceedings of ACM SIGCOMM*, 2008.

[22] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker. Approximate Fairness Through Differential Dropping. *ACM SIGCOMM Computer Communication Review*, 33(2):23, Apr. 2003.

[23] T. Peng, C. Leckie, and K. Ramamohanarao. Survey of Network-Based Defense Mechanisms Countering the DoS and DDoS Problems. *ACM Computing Surveys*, 39(1), Apr. 2007.

[24] P. Ramamurthy, V. Sekar, A. Akella, B. Krishnamurthy, and A. Shaikh. Remote Profiling of Resource Constraints of Web Servers Using Mini-Flash Crowds. In *Proceedings of USENIX ATC*, 2008.

[25] R. Rasti, M. Murthy, and V. Paxson. Temporal Lensing and its Application in Pulsing Denial of Service Attacks. In *Proceedings of IEEE Symposium on Security and Privacy*, 2015.

[26] C. Rossow. Amplification Hell: Revisiting Network Protocols for DDoS Abuse. In *Proceedings of NDSS*, 2014.

[27] A. Stavrou and A. D. Keromytis. Countering DoS attacks with stateless multipath overlays. In *Proceedings of ACM CCS*, 2005.

[28] I. Stoica, S. Shenker, and H. Zhang. Core-Stateless Fair Queueing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High-Speed Networks. *IEEE/ACM Transactions on Networking*, 11(1):33–46, Feb. 2003.

[29] J. Strauss, D. Katabi, and F. Kaashoek. A Measurement Study of Available Bandwidth Estimation Tools. In *Proceedings of ACM SIGCOMM IMC*, 2003.

[30] A. Studer and A. Perrig. The Coremelt Attack. In *Proceedings of ESORICS*, 2009.

[31] A. Yaar, A. Perrig, and D. Song. SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks. In *Proceedings of IEEE Symposium on Security and Privacy*, 2004.

[32] X. Yang, G. Tsudik, and X. Liu. A Technical Approach to Net Neutrality. In *Proceedings of Hotnets-V Workshop*, 2006.

[33] S. T. Zargar, J. Joshi, and D. Tipper. A Survey of Defense Mechanisms Against Distributed Denial of

Service (DDoS) Flooding Attacks. *IEEE Communications Surveys and Tutorials*, 15(4):2046–2069, 2013.

[34] Y. Zhang, Z. M. Mao, and J. Wang. Low-Rate TCP-Targeted DoS Attack Disrupts Internet Routing. In *Proceedings of NDSS*, 2007.

APPENDIX

A. ATTACK STREAM GENERATORS

This appendix describes the construction of seemingly-innocuous attack flows that are randomized and responsive to congestion, thus improving the stealthiness of the attack.

A.1 Random

We say that time t is during an *on-phase* if $t \in [\hat{x}(k|k-1), \hat{x}(k|k-1) + L)$ for some k, where $\hat{x}(k|k-1)$ is derived from Eq. 3; otherwise, t is in an *off-phase*.

The randomized attack stream generator probabilistically schedules a burst for each on-phase, such that the expected value of the instantaneous rate at any time point during an on-phase is $\frac{R+\epsilon}{n}$, where R is the magnitude of bursts at the target link, and n is the number of attack streams. The value of ϵ is set to ensure that the instantaneous rate of n attack streams is higher than R with high probability.

To achieve this, for each $\hat{x}(k|k-1)$, the generator selects p ($0 \le p \le 1$), y_1 ($y_1 \in \mathbb{N}^+$), and y_2 ($y_2 \in \mathbb{N}, 0 \le y_2 < y_1$) uniformly at random. This parameter selection process repeats until $\frac{(R+\epsilon)y_1}{np} \le r_{max}$, where r_{max} is the maximum instantaneous rate of an attack stream. Then with probability p, the generator sends traffic at rate $\frac{(R+\epsilon)y_1}{np}$ during interval

$$b_k = \left[\hat{x}(k|k-1) + \frac{y_2}{y_1}L, \hat{x}(k|k-1) + \frac{y_2+1}{y_1}L\right).$$

Analysis Let $R(t)$ be the instantaneous rate of n attack streams at time t, and $r(t)$ the instantaneous rate of a generator at time t. We derive the probability of $R(t) \ge R$. When t is in an on-phase, the expected value of $r(t)$ is

$$\begin{aligned} E[r(t)] &= Pr[\text{send burst}] \cdot Pr[t \in b_k] \cdot \frac{(R+\epsilon)y_1}{np} \\ &= p \cdot \frac{1}{y_1} \cdot \frac{(R+\epsilon)y_1}{np} = \frac{R+\epsilon}{n}. \end{aligned}$$

Since each generator selects parameters independently and $0 \le r(t) \le r_{max}$, we obtain

$$\begin{aligned} Pr[R(t) < R] &= Pr[n \cdot r(t) < R] \\ &= Pr[n \cdot r(t) < E[n \cdot r(t)] - \epsilon] \\ &< \exp\left(-\frac{2n^2\epsilon^2}{n \cdot r_{max}^2}\right) = \exp\left(-\frac{2n\epsilon^2}{r_{max}^2}\right). \quad (8) \end{aligned}$$

Eq. 8 is derived based on Hoeffding's inequality. Hence, for all $\delta \in (0,1]$, the randomized attack stream generator ensures that $Pr[R(t) \ge R] \ge 1 - \delta$ if $\epsilon \ge r_{max}\sqrt{\frac{-\ln\delta}{2n}}$.

Attack cost The cost of introducing this randomization technique (in terms of the increase in attack traffic per time unit per bot) is $\frac{\epsilon L}{nT}$, which is negligible for typical values of ϵ, L, n, and T. In sum, to ensure that the instantaneous rate of randomized attack streams $R(t)$ is higher than R with high probability, each bot only has to send at an average rate slightly higher than $\frac{R}{n}$ during each on-phase.

A.2 TCP embedding

We describe how to execute a DDoS attack using responsive TCP flows. The key idea is that since bots know roughly when the link will be unobstructed (i.e., during the off-phases), they can reliably increase their sending rates by sending packets during off-phases.

TCP increases its congestion window (*cwnd*) when no congestion is sensed. As the congestion window limits the maximum number of unacked segments, the sending rate is approximately the size of the congestion window divided by the RTT, i.e., $\frac{cwnd}{RTT}$. While the actual TCP implementations vary, and numerous suggestions have been made to improve the performance of TCP, our proposed mechanism can work with any TCP variants in principle. Hence, we consider the following standardized specification for illustration purposes. reader unfamiliar with TCP congestion control is referred to RFC 5681.

When the congestion window (*cwnd*) is smaller than the slow start threshold (*ssthresh*), TCP is in the slow start phase, where it exponentially increases the congestion window to probe for available bandwidth. In the slow start phase, the value of *cwnd* is doubled per RTT, or incremented by 1 Maximum Segment Size (MSS) per non-duplicate ACK. When $cwnd \ge ssthresh$, TCP enters the congestion avoidance phase, where *cwnd* is incremented by 1 MSS per RTT (or incremented by $\frac{MSS^2}{cwnd}$ per non-duplicate ACK). Upon a timeout, *cwnd* is reset to 1 MSS and *ssthresh* is set to max{FlightSize/2, 2 MSS}, where *FlightSize* is the size of all unacked segments in bytes.

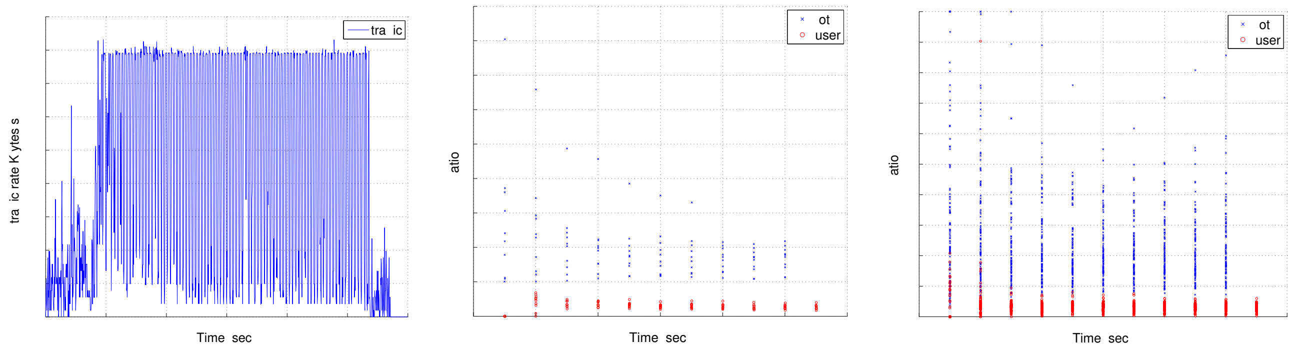
Analysis. Suppose each bot sends $\frac{cwnd}{MSS}$ segments per RTT during off-phases until $cwnd = x$. Then during an on-phase, the bot sends a burst of magnitude $\frac{x}{RTT}$ and duration RTT. If some packets in the burst are dropped due to congestion, *cwnd* is reset to 1 MSS and ssthresh to $\frac{x}{2}$. Assume no congestion experienced by cover traffic, the minimum number of periods before restoring *cwnd* to x is:

$$\Delta_{burst} = \left\lceil minRTO + (timeSS + timeCA) \cdot \frac{1}{T-L} \right\rceil,$$

where timeSS and timeCA are the time elapsed in the slow start phase and congestion avoidance phase, respectively. The current timeout value is minRTO, as the bot can ensure no repeated timeouts for the same packet. Since cover traffic is sent only during off-phases (which account for $T - L$ out of T time per period), the total time is divided by $T - L$. Furthermore, based on the TCP specification, we can derive that $timeSS = RTT \cdot \max\{\log_2 \frac{x}{2 \cdot MSS}, 1\}$ and $timeCA = RTT \cdot \min\{\frac{x}{2 \cdot MSS}, \frac{x}{MSS} - 2\}$.

Attack cost. We quantify the increase in cost with respect to the amount of cover traffic. Given that each bot sends $\frac{cwnd}{MSS}$ segments per RTT during off-phases, how much cover traffic does each bot need to send before reaching a congestion window of size x? When no packet loss occurs during off-phases, the average number of segments per second is bounded:

$$\begin{aligned} avg\_rate &\le \max\{avg\_rate\_SS, avg\_rate\_CA\} \\ &= avg\_rate\_CA = \frac{0.75x}{RTT \cdot MSS}. \end{aligned}$$



(a) Traffic on the router.

(b) Router detection with one experiment.

(c) Router detection with 10 experiments.

Figure 12: Countermeasure preliminary results.

B. POTENTIAL COUNTERMEASURES

We discuss potential countermeasures against CICADAS.

Throttle disturbing flows. As CICADAS aims to maximally disturb the communication of other flows while remaining stealthy, we can ideally define a new flow metric that quantifies the level of flow disturbance to other flows, and then throttle flows with a high disturbance value.

Following this line of thought, we observe that if a flow accounts for a larger share of traffic during congestion, the flow is likely to be an attack flow. (A legitimate flow, on the other hand, should slow down during congestion or at least behave consistently regardless of congestion.) Hence, when a router sees periodic patterns as in Figure 12(a), it can gather traffic statistics during congestion (i.e. the peaks) and non-congestion (i.e. the valleys) for further analysis. Specifically, we define the *Disturbance Ratio* of flow i as follows:

$$ratio(i) = \frac{c(i)}{C} / \frac{nc(i)}{NC}, \qquad (9)$$

where C and NC are the number of bytes received by the router during congestion and non-congestion, respectively. Similarly, $c(i)$ and $nc(i)$ are flow i's traffic (in bytes) received by the router during congestion and non-congestion, respectively. If a flow's disturbance ratio is greater than 1, it means that the flow contributes more to the overall traffic (in terms of percentage) during congestion than during non-congestion. As a result, a flow is classified as malicious if its disturbance ratio > 1.

As a proof-of-concept, we implement this method on a local machine (which serves as the target link), and run 10 pairs of bots and 10 normal TCP flows. Attacks start from 0s to 100s and the TCP flows start from 0s to 110s. The router computes $ratio(i)$ every 10 seconds.

Figures 12(b) and 12(c) show the Disturbance Ratio vs. time by running one and 10 experiments, respectively. The red dots represent users and the blue crosses represent bots. In Figure 12(c), the precision is $> 90\%$ during the first 20 seconds and increases to 100% in the subsequent 80 seconds. The recall is $> 90\%$ all of the time. Note that in the context of our study, precision means the proportion of detected malicious flows to all detected flows, and recall is the proportion of detected malicious flows to all malicious flows.

Although our proposed method has high precision and recall, a practical solution would require computing such a metric in an efficient manner. In the future, we aim to devise an efficient one-pass algorithm that computes disturbance ratios at line rate without keeping per-flow state at routers.

Disrupt CICADAS coordination. Another possible defense strategy is to explicitly disrupt the coordination of bots in CICADAS. For example, routers could attempt to rate limit probe packets, as our simulations show that the attack becomes less effective as the probe interval gets higher. However, since CICADAS can employ a custom probe protocol rather than the standard Ping utility, the probe packets are indistinguishable from other packets. As a result, such a simple rate limiter would fail to limit CICADAS's probe packets. Routers could also try to inject noise to the congestion signal – RTT and packet loss – which a bot relies on for regulating the attack stream. However, increasing end-to-end latency and packet drop rate is undesirable for many network applications.

Route around congested links. A promising general defense is multipath routing. Multipath communication can offer dramatic availability improvements, as legitimate senders can avoid maliciously-flooded links by switching between paths [21, 27]. A multipath protocol may be more resilient against link flooding if it supports fine-grained path selection; however, fine-grained path selection may actually grant too much power to bots. For example, these bots could construct loopy, lengthy paths that consume disproportionate amount of network bandwidth, jeopardizing the welfare of legitimate senders. In other words, multipath communication can be a double-edged sword that hurts benign flows in some adversarial environments. Thus, further research is required to study how much multipath protocols can help mitigate DDoS attacks and improve network availability.