2015 IEEE Symposium on Security and Privacy

Temporal Lensing and its Application in Pulsing Denial-of-Service Attacks

Ryan Rasti*†◇, Mukul Murthy*, Nicholas Weaver*†, Vern Paxson*†
*UC Berkeley, †ICSI, ◇VMware

***Abstract*—We introduce *temporal lensing*: a technique that concentrates a relatively low-bandwidth flood into a short, high-bandwidth pulse. By leveraging existing DNS infrastructure, we experimentally explore lensing and the properties of the pulses it creates. We also empirically show how attackers can use lensing alone to achieve peak bandwidths more than an order of magnitude greater than their upload bandwidth. While formidable by itself in a pulsing DoS attack, attackers can also combine lensing with amplification to potentially produce pulses with peak bandwidths orders of magnitude larger than their own.**

I. Introduction

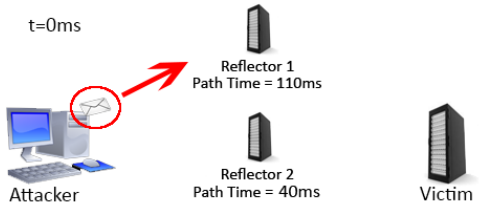
When conducting network-layer denial-of-service (DoS) flooding, attackers can either send traffic directly towards their victim, or bounce it off intermediary *reflectors* by spoofing the victim's address or otherwise sending the reflectors queries that will induce them to send follow-on traffic to the victim [16]. If an attacking system can send at a rate of ψ bytes/sec, then for direct attacks clearly the peak load it can impose on the victim is also ψ. In the second case, the reflectors might provide a factor k of *amplification*, depending on the relationship between the queries sent to the reflectors and the replies these result in the reflectors transmitting to the victim.

In the reflected case, it might thus appear self-evident that the peak load an attacking system can impose on the victim cannot exceed $k \cdot \psi$. Surprisingly, this turns out not to be the case. In this paper we introduce *temporal lensing*, which (a) exploits existing infrastructure (reflectors) in the novel way of *concentrating* a flood in time rather than simply *mirroring* it, and (b) can from these pulses produce a debilitating degradation of throughput for the victim's operational TCP traffic, such as explored in prior work on "shrew" and "pulsing" attacks [11], [14].

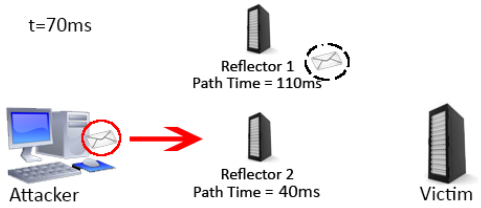
Previous work on pulsing attacks has left a significant opportunity for improvement: a majority of the attacker's bandwidth lies unused in between pulses. Therein lies the question of how to send packets during these idle times but have these packets still arrive at the victim all in a single burst. Doing so should allow an attacker to do significantly better than either brute-force flooding or pulsing attacks could do alone.

We draw an analogy to the military tactic "Time on Target" [10] for coordinating artillery fire. Using synchronized clocks and estimates of projectile flight times, a coordinated artillery battery can fire from different locations but have all their shells hit the target simultaneously. This technique played a key role in the efficacy of American artillery strikes during World War II [20].

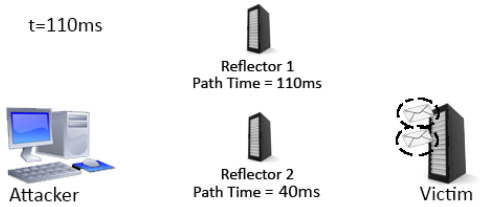
In a more sophisticated variant, "Multiple Rounds Simultaneous Impact" [12], a single ordnance makes multiple rounds rendezvous at the target by varying the angle of fire, charge, and thus flight time. By varying projectile paths, the artillery can make more shots arrive at the victim in one period of time



(a) At $t = 0\ ms$, the attacker sends one packet towards reflector 1



(b) At $t = 70\ ms$, the first packet is about 60% along its path to the victim and the attacker sends another packet to reflector 2



(c) At $t = 110\ ms$, both packets arrive at the victim

Figure 1: Attack illustration. Paths through reflectors 1 and 2 have *attack path latencies* of 110 and 40 milliseconds respectively. The attacker sends at a rate of 1 packet-per-70-ms, but concentrates the flow such that two packets arrive simultaneously at the victim. For an instant, the attacker has significantly increased the effective attack bandwidth at the victim.

DOI 10.1109/SP.2015.19

than it can send in that same amount of time.

To accomplish a similar feat, we leverage the wide range of paths and latencies on the Internet. If an attacking system can schedule its sending in such a way that it first sends packets that will take longer to arrive, and later sends those that take less time to arrive, the packets can all rendezvous at the victim within a small window of time; the attack compresses the original transmission into a sharp pulse of traffic that can completely fill the buffer at the victim and induce packet loss and thus congestion response. By repeating such pulses periodically, the attacker can realize *reduction of quality* [7] such as that resulting from a shrew attack [11].

Clearly, if the attacker sends directly to the victim, each packet will take roughly the same amount of time to reach the victim, since in general they will travel along the same (or at least similar) paths. Reflectors introduce the ability to obtain variable *attack path latencies*: the time from attacker through reflector to victim. Each reflector the attacker employs potentially introduces a new path for attack traffic and thus a different attack path latency. Figure 1 illustrates a simple example.

We term this technique *temporal lensing*, or simply *lensing*, as reflectors can temporally concentrate packets much like a lens focuses light. When describing how the attack works, we use the term *concentration* rather than amplification, as the former more directly matches the underlying dynamics, and the latter already has common usage in describing an orthogonal attack (which a lensing attack could also include).

After reviewing related work, we develop the attack in three main parts: determining attack path latencies through resolvers to the victim (§ III); building a sending schedule to create maximal lensing from these latencies (§ IV); and characterizing the efficacy of the attack (§ V). We experimentally assess an implementation of lensing (§ VI), and then address extensions (§ VII) and assess potential defenses (§ VIII).

II. Related Work

Kuzmanovic and Knightly [11] first described the concept of bursty, low average bandwidth pulses as "shrew" attacks. The attack aims to send enough packets in a short duration to cause a TCP retransmit timeout (RTO) in clients, and then induces additional RTOs with each subsequent, periodic pulse. They noted that due to their low average bandwidth, such attacks should prove harder to detect than traditional flooding. Their evaluation demonstrated that such attacks can effectively reduce throughput by an order of magnitude or more, and two potential defenses—use of RED to identify shrew pulses, and randomization of RTOs to avoid synchronization with subsequent pulses—require either a lengthy measurement period or loss of throughput in the absence of attack.

Zhang, Mao, and Wang explore shrew attacks with a focus on disrupting BGP [24]. They also discuss using multiple attackers to initiate a shrew attack, similar to lensing. However, attackers initiate lensing from a single host and use reflectors—entities not under the attacker's direct control—as the attack vector. In addition, lensing enables full utilization of an attacker's uplink bandwidth, while simply distributing a shrew attack still leaves the attacker machines idle most of the time.

Luo and Chang [14] generalized the idea of shrew attacks to disruption of TCP congestion control. Specifically, they also considered the AIMD congestion control response, showing that even without retransmission timeouts, such attacks can also severely degrade TCP flows. Guirguis et al. [7] further generalized low average bandwidth attacks as a type of RoQ (reduction of quality) attack, noting that pulsing exploits *transients* in a system (e.g., congestion response) instead of *steady state* capacity (e.g., victim's bandwidth).

While pulsing DoS boasts impressive theoretical and experimental efficacy, it appears to have seen little use in practice; perhaps attackers find little need to render their attacks more difficult to detect. If so, then they are better off with direct flooding: since senders are limited to their uplink bandwidth in creating pulses, simple pulsing cannot perform better than direct flooding in terms of damage inflicted. However, the idle time between pulses indicates room for improvement. In particular, our development of lensing aims to improve flooding *efficacy* rather than stealth, which may lead to attackers becoming more inclined to use it.

Kang, Lee, and Gligor [9] introduce another variant on stealthy, low-bandwidth DoS, called "Crossfire" attacks. Crossfire attacks use a set of attackers (presumably bots in a botnet) to flood at low bandwidths and has these flows converge at a few critical links to cut off the victim from the Internet. The result is a stealthy attack in which any given bot sends at a low-average rate, but the flows overflow a few critical links in a way not obvious to the victim or routers. Lensing shares this idea of concentration, but instead can be used to concentrate packets from a *single* flow *temporally* to create pulses as in a pulsing DoS attack on TCP congestion control. Further, as noted above, its primary goal is to buy more DoS efficacy for an attacker, rather than stealth. Theoretically, however, the two attacks could be combined (i.e., using lensing to better degrade the throughput on the target links).

Paxson [16] describes the role of reflectors in DoS attacks as amplifying the flooded traffic and helping attackers evade detection. He also notes the natural use of open DNS resolvers as reflectors. Our attack prototype takes advantage of this last fact and the abundance of such resolvers (estimated to be in the tens of millions [18]). However, we use reflectors in a new way, instead employing them to concentrate the arrival of packets at the victim, much like a lens focuses light. Thus, lensing works in a fashion complementary to traditional amplification attacks; it is not itself a way to amplify traffic *volume*.

Our approach requires the ability to calculate latencies between each reflector and the victim, an instance of the general problem of measuring latencies between arbitrary Internet end hosts, which has received significant study [4], [6], [15], [19], [21]. For our purposes, we leverage Gummadi et al.'s *King* [8], which offers the impressive advantage that

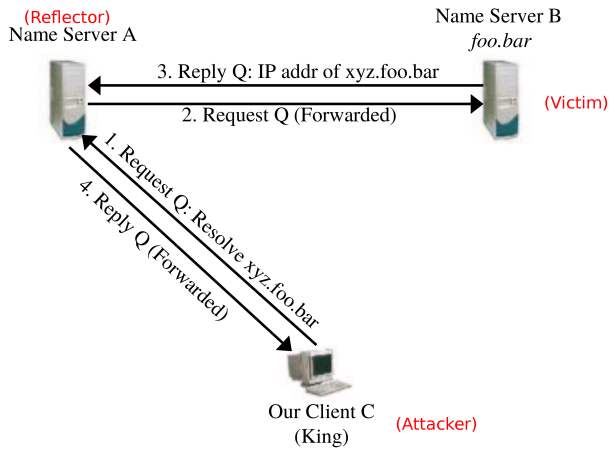


Figure 2: The operation of King (reprinted with permission), with the relevant actors for lensing added in red.

it can approximate the latency between arbitrary Internet hosts without requiring any additional infrastructure beyond what DNS already provides. King works by finding DNS servers close to the target hosts and estimating distances using recursive DNS queries. King suits our task particularly well because it has the greatest accuracy when either (or both) hosts are DNS servers. As a recursive protocol, DNS naturally lends itself to use in reflection.

Schomp et al. [18] estimate the number of open DNS resolvers in the tens of millions, with many running on (often outdated) commodity hardware. They note that many such resolvers are ephemeral, persisting at a given address on the order of days to weeks. In addition, they describe how many resolvers do not iterate the DNS hierarchy themselves, but instead *forward* the task to auxiliary resolvers. TurboKing [13] incorporates the possibility of such forwarders to improve its accuracy. As explained below, because when using open resolvers for lensing attacks the attack follows the same path as that used for determining latencies, such DNS nuances should not affect its precision.

III. Estimating Attack Path Latencies

Conducting the attack using a given set of reflectors requires us to first estimate each reflector's attack path latency, for which we employ the technique used by King. King operates by issuing recursive DNS queries between two DNS servers located close to the end servers in question. Figure 2, taken from [8], illustrates its operation, where we have overlaid labels representing the attacker, victim, and reflector, as used in our attack. With a single recursive query, an attacker can form an estimate for the attack path RTT by taking the difference in time between when the attacker sends a query to when the attacker receives a response.

King must address two conflicting caching issues. First, it "primes" the resolver so that it caches the fact that the victim is authoritative for its domain (i.e., cache the `NS` record). Doing so prevents the resolver from iterating through the DNS hierarchy for subsequent queries. Second, to obtain accurate measurements, the attacker must issue queries for different subdomains of the victim's domain (`foo.bar` in the example), lest the attacker's queries hit the resolver's cache and short-circuit the follow-on query (i.e., query for a different `A` record). By sending queries for distinct subdomains, each will result in sending the entire chain of packets 1–4 shown in Figure 2.

Lensing does not fundamentally require the use of recursive DNS resolvers as reflectors, but they serve very well in this role. By their recursive nature, they perform reflection naturally, and because of their direct co-location with the reflector, using them for measurement provides accurate attack path latency estimates, particularly so if we target the victim's DNS server with the attack.[1] Given an estimate of the attack path RTT, we then halve it to obtain the attack path latency. Halving the RTT might not in fact give an accurate estimate, due to the prevalence of asymmetric routing in the Internet. However, our positive experimental results on lensing in § VI experimentally validate this approximation.[2]

Short-term variation. A basic question for the accuracy of a lensing attack concerns the stability of attack path latency measurements over short periods of time. In particular, it may take a few minutes just to measure latencies to all of the reflectors. To investigate how attack path latencies vary over such time periods, we used a random sample of 44 resolvers from a public list [1] of a few thousand to measure the path latency through each every two minutes.[3] Figure 3 provides some examples of what path latency variation can look like over time. We show three cases, deemed (from the attacker's perspective) "good" latency variation, "bad" variation, and seemingly-good-but-not. In particular, the resolver in the third graph might at first appear good because it exhibits few timeouts and a fairly consistent latency over short periods of time. Over longer time periods, however, it appears to undergo repeated routing changes that abruptly alter the attack path latency. Using such a reflector could lead to packets sent it missing the pulsing window.

We can robustly characterize a given reflector's latency variation using the interquartile range (IQR) of the distribution of its measurements, i.e., the difference between its 75^{th} and 25^{th} percentiles. We find that misleading resolvers are fairly rare. For example, the one in the third plot had an IQR of 122 ms, while more generally we find that nearly half our resolvers had an IQR of under 12 ms.

In summary, for some resolvers, the attacker must either perform latency measurements immediately before launching

[1]The reader may correctly note that pulsing DoS attacks (which attack TCP congestion control) will likely have little impact on a UDP-based service such as DNS. We defer discussion of estimating attack path latencies to TCP-based hosts to § VII-A.

[2]It turns out that a constant error term (constant over all paths) in the approximation (i.e., Latency $= \frac{1}{2} \cdot \text{RTT} + \epsilon$) will still enable pulsing (see § VII-A).

[3]Here and throughout the remainder of the paper, our measurements always included warming DNS caches as necessary.

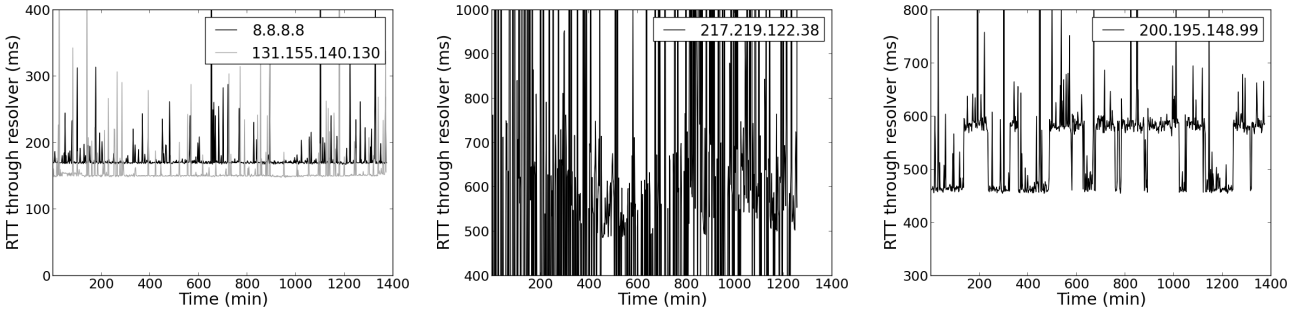


Figure 3: Two "good" resolvers (Google public DNS and Eindhoven University of Technology) with minimal path latency variation, an obviously "bad" resolver with high path latency variation, and a resolver that appears good over small samples of time but is actually bad for lensing, respectively. We took samples 2 min apart, and show timeouts with marks along the top of the plot.

an attack, or draw upon a longer period of statistics to identify the resolver as problematic. However, given that most resolvers do not suffer from widely varying latencies, even if the attacker does not account for misleading resolvers and simply assumes every resolver that appears good over a short period of time is indeed good, the efficacy of their attack will not significantly suffer.

Long-term variation and caching. We now turn to the degree to which attackers can fruitfully rely upon measurements taken further in the past than just a short period prior to launching an attack. An ability to use older measurements would enable attackers to better hide the "reconnaissance" activity necessary to set up an attack.

To this end, we used our same sample of resolvers and sent 50 packets through each every 4 hours for 10 days. For each resolver we computed the standard deviation of the median latency over each flight of measurements. We then divided these standard deviations by the mean of the median latencies, obtaining coefficients of variation (CoV). We found that many resolvers' path latencies exhibit very little variance over time, indicating an attacker could cache latency estimates for significant periods of time. Attackers could also compute such statistics over large groups of resolvers to identify substantial subsets to save on a "short list" of high-quality resolvers to employ in future attacks.

We also note that we did not find many resolvers with very high attack path latencies (many hundreds of msec). Such resolvers (if exhibiting low variance) would enable an attacker to increase their overall bandwidth gain because they would provide longer periods of time over which to send packets (and subsequently concentrate). However, we found that most such high latency paths also exhibited high jitter and inconsistency.

IV. Building an Optimal Schedule

Given path latency estimates for the reflectors, the attacker then needs to compute a sending schedule. This schedule divides up the sending window into a set of time slots T, listing for each slot which reflector the attacker should send to in that slot. The number of slots available for the attacker to send is a function of the attacker's maximum bandwidth (which determines the idle time between adjacent slots) and the range of path latencies measured for different reflectors.

We define the *pulse window* (or simply *window*) as the duration of the pulse as seen at the victim. In trying to create a maximal pulse, the attacker's goal is to maximize the expected number of packets that land in a predetermined window.

To do so, we use a greedy algorithm to compute an optimal schedule given an initial set of reflectors and estimates of their corresponding attack path latencies. According to our algorithm, at each time slot t we simply choose the reflector that provides the highest estimated probability of landing within the window. Absent any restrictions on how often an attacker can employ a given reflector, we can show that this greedy algorithm is indeed optimal (see below).

These problem statements and the ensuing proofs do not account for distortions in the attack path latencies due to the attack itself—such as those caused by effects of over-taxing resolvers, or congestion caused by the attack. By distributing the attack over geographically diverse resolvers, congestion should rapidly decrease at points further from the victim, so in fact self-congestion might not actually prove detrimental to the attack. In addition, our experimentation reveals little evidence of congestion actually inhibiting our emulated attacks.

In addition, our proofs cover the simplified case where the attacker can freely reuse any reflector for a given time slot. The more complex case where the attacker throttles overall use of any given reflector does not appear to readily lend itself to proofs of optimality.

Finally, an actual pulsing attack will consist of multiple, evenly spaced pulses. We can construct an optimal schedule for this scenario in a fashion similar to the single pulse case above. At each time slot t we choose the reflector that provides the highest estimated probability of landing in *any* window. We prove optimality for this case after first addressing the single-pulse case.

A. Proving Schedule Optimality: Single Pulse

For each time $t \in T$ when we consider sending, and for each reflector $r \in R$ (the set of reflectors), let $\Pr(t, r)$ denote the probability that if we send to reflector r at time t, the reflected packet will land in the desired window. Note that § III provides estimates for this probability. We assume that these probabilities are independent and time-invariant (for any reflector, a given latency will occur with a given probability regardless of when we send to the reflector, or what packets we send at other times).

Suppose we have chosen a schedule for which at each time t we send to reflector r_t. Let X denote the random variable representing how many packets arrive in the window. $X = \sum_{t \in T} X_t$, where

$$X_t = \begin{cases} 1 & \text{if packet sent at } t \text{ lands in window} \\ 0 & \text{otherwise.} \end{cases}$$

So, $E(X_t) = \Pr(t, r_t)$. Then, due to linearity of expectation,

$$E(X) = E(\sum_{t \in T} X_t) = \sum_{t \in T} E(X_t) = \sum_{t \in T} \Pr(t, r_t).$$

Given this, we claim that any schedule that optimizes $E(X)$ must have the condition that for each t, we send to the reflector with highest $\Pr(t, r)$ over $r \in R$. To see this, assume for the sake of contradiction that an optimal schedule S exists such that at time t^* we do not send to the reflector r^* that yields the highest probability; instead, we send to r^{**}. By construction, $\Pr(t^*, r^*) > \Pr(t^*, r^{**})$. Consider S', the same schedule as S except that at t^*, it sends to r^*. Let the expected number of packets landing in the window of S and S' be $E(X)$ and $E(X')$ respectively; then the difference between the expectation of the schedules is:

$$\begin{aligned} E(X') - E(X) &= \sum_{t \in T} E(X'_t) - \sum_{t \in T} E(X_t) \\ &= \left[\Pr(t^*, r^*) + \sum_{t \in T \wedge t \neq t^*} E(X'_t) \right] \\ &\quad - \left[\Pr(t^*, r^{**}) + \sum_{t \in T \wedge t \neq t^*} E(X_t) \right] \\ &= [\Pr(t^*, r^*) - \Pr(t^*, r^{**})] \\ &\quad + \sum_{t \in T \wedge t \neq t^*} E(X'_t) - \sum_{t \in T \wedge t \neq t^*} E(X_t) \\ &= \Pr(t^*, r^*) - \Pr(t^*, r^{**}) \end{aligned}$$

However, we already established that this last term is greater than 0. Thus, $E(X') - E(X) > 0$ and S' provides a better schedule than S, which contradicts our assumption that S is optimal.

B. Proving Schedule Optimality: Multiple, Evenly-spaced Pulses

We now extend the problem to scheduling optimally using multiple, evenly-spaced windows. For defining "optimal" in this case, we consider two intuitive notions:

- Having the largest amount of packets land in any pulse (that is, maximize the sum of packets in all pulse windows)
- Having the largest possible consistent pulses

We present an algorithm that trivially accomplishes the first and then show that it also accomplishes the second.

The algorithm proceeds as follows. For each time slot $t \in T$, choose the reflector with the highest probability of landing in *any* window. The justification that this satisfies the first criteria follows trivially from the previous proof, and we omit it here.

To see how the algorithm fulfills the second definition, we claim that an attacker can repeatedly execute an optimal schedule with the same period as the attack. Consider an arbitrary time slot t_i and the time slot p time units in the future, $t_i + p$, where p represents the period. Label the j^{th} window w_j. Suppose a packet sent at t_i to reflector r has probability $Pr_{w_j}(t_i, r)$ of landing in window w_j. Then, at $t_i + p$, reflector r has probability $Pr_{w_{j+1}}(t_i + p, r) = Pr_{w_j}(t_i, r)$ of landing in window w_{j+1}. In steady state, we can express the probability that a packet sent at t_i to r lands in *any* window as:

$$\sum_{j=-\infty}^{\infty} Pr_{w_j}(t_i, r) = \sum_{j=-\infty}^{\infty} Pr_{w_{j+1}}(t_i + p, r) \qquad (1)$$

Thus, the probability that it lands in *any* window is the same p time units in the future. Since our optimal algorithm for each t chooses the reflector r with the highest probability of landing in any window, and because this chance is periodic, it follows that its schedule is periodic.[4]

Since schedules are periodic, it follows that the expected number of packets that land in any window is constant. Thus, the algorithm produces a schedule that satisfies the second definition of optimality.

V. Characterizing Attacks

Armed with the ability to estimate attack path latencies and to build an optimal schedule from them, we now turn our development of an approach to experimentally validate lensing attacks.

A. Features measured

We explore the effectiveness of pulsing attacks in terms of three dimensions:

1) attacker bandwidth (sending capacity of originating system)
2) pulse window size (duration over which attacker wants packets to arrive at target)
3) maximum bandwidth to employ for each reflector

[4]There is the possibility that two equally good reflectors exist for time slot at t_i, and we could send to one at t_i and the other at $t_i + p$. We assume the schedule consistently chooses just one.

Regarding the last of these, along with considerations such as rate-limiting that a given reflector imposes on its activity, an attacker might want to throttle the bandwidth to any given reflector to avoid arousing suspicion. For our purposes, since we do not know the resources of the reflectors we employ, we err on the side of caution when using them, sending to each at a maximum bandwidth of 500 pps over a course of 20–100 ms (i.e., at most 5 KB per pulse).

We do not explicitly explore the number of reflectors used as a dimension, because the number of reflectors heavily depends on the existing dimensions of attacker bandwidth and maximum bandwidth to each reflector.

B. Metrics

In § IV we defined an optimal schedule as one that has the greatest expected volume of packets falling within the pulse window. This is intuitively a natural parameter to maximize. However, if we solely use *absolute* number of packets as our *metric* of efficacy, it can be artificially inflated just by increasing the uplink bandwidth of the attacker or increasing the target window size. This issue motivates us to we incorporate some additional, bandwidth-agnostic metrics:

- *bandwidth gain*: $\frac{\text{bandwidth in pulse window at target}}{\text{attacker's maximum sending bandwidth}}$
- *concentration efficiency*: $\frac{\text{\# packets landing in window}}{\text{\# total packets sent}}$

The first metric is the most important from the short-term point of view of the attacker. It gives the attacker a sense of how much extra bandwidth the attack can produce.

The second metric, however, provides a good basis for determining how the attack scales with the attacker's bandwidth. If the size of the reflector pool remains constant, upon increasing the sending (attacker) bandwidth, more time slots occur for which the schedule fails to provide an available resolver to send to (because we throttle bandwidth to any given reflector). In this case, the bandwidth gain will artificially decrease. For example, if (as an extreme case) we send a maximum of one packet to each of 100 reflectors, then we can send at most 100 packets to the target. Thus, as our uplink bandwidth, increases, the bandwidth gain will decrease (the bandwidth gain numerator is capped at 100)—not because of poor scaling, but because of an absence of suitable reflectors. However, all other things equal, the concentration efficiency will remain constant.

We note the importance of considering both of these metrics together. If we only assess one of them, we can easily inflate its value at the expense of the other: a large pulse window size will lead to a concentration efficiency of 1 (ignoring packet drops) but no bandwidth gain. A very small target window could result in an extremely high bandwidth gain (if one packet lands in it) but a very low concentration efficiency.

Lastly, we measure bandwidth in terms of packets per second. For our evaluation, the packets we send and that are reflected are small (around 100 bytes), which can make the quantities look artificially high. For a sense of scale, 10K pps translates to about 8 Mbps.

VI. Experimental Results

To assess the efficacy of lensing, we emulated attacks on machines under our control. We used an Windows Azure VM instance on the West Coast as our attacker. We employed another Azure VM instance along with an Amazon Web Services VM instance, both on the East Coast, as our targets. We used a publicly available list of 3,000 resolvers [1] as our reflectors.

We registered a domain name[5] that allows us to run authoritative DNS servers under our control. We made the AWS target instance authoritative for our domain and the Azure target instance authoritative for a subdomain of the original. This allows us to send recursive DNS queries through any open recursive DNS server to either of our targets.

Before an attack, the attack machine quickly scans the resolver list. It issues recursive queries to the resolvers (just as in an actual attack) to obtain latency measurements. We gather 10 samples from each resolver, which turned out well for our attacks. For each resolver, we construct a histogram for the distribution of each resolver's attack path latency, and use this along with a variant (discussed below) of the optimization algorithm in § IV to construct the sending schedule. During the attack, we simply send to the resolvers according to the schedule.

Figure 4 shows the results of pulses emulating attackers with different bandwidths using a relatively narrow pulse window of 20 ms. The emulation setup artificially capped the outgoing bandwidth by adjusting the minimum time between sending adjacent packets. The bandwidth gain corresponds to dividing the height of the pulse bucket by that of the tallest bucket for the attacker's sending. We see gains of $14x$ for the low-bandwidth case, $10x$ for moderate bandwidth, and $5x$ for high bandwidth. The efficiency corresponds to dividing the area of the pulse bucket by that of the sending buckets. We find efficiencies around 50% for the low- and medium-bandwidth cases, and just under 40% for the high bandwidth case.

The colors in Figure 4a map onto the reflectors used. We see that while a large number of reflectors contributed to the pulse, some do not at all, either due to misleading latency measurements or because of jitter occurring during the attack itself.

We observe what look like multiple pulses in Figures 4b and 4c. Packet traces reveal that these secondary spikes result from retransmissions by the resolvers. The target (an authoritative DNS server) could not keep up with the rate of incoming queries and failed to respond to many of them. The resolvers then timed out and retransmitted. Since many of them share a common retransmit timeout, their retransmissions rendezvous at time = (original pulse time) + (retransmit timeout). We could thus identify two common retransmit timeouts of 800 ms and 2 s. (We discuss ways an attacker could leverage retransmissions in § VII-D.) Retransmissions also caused the total number of packets received by the target to often exceed the total number sent by the attacker by about a factor of two,

[5] `pulsing.uni.me`

though we chose our metrics such that these additions do not affect our characterizations of the attack's efficacy.

Figure 5 shows how the lensing metrics vary with pulse window duration when we fix the attacker bandwidth at 10K pps and the maximum per-reflector bandwidth at 500 pps. As expected, the bandwidth gain (and absolute pulse bandwidth) falls as we increase the window size. This is not an indication of the attack performing poorer with larger window sizes, but instead an intrinsic consequence of choosing a larger window size (and thus a larger denominator in the pulse bandwidth calculation).[6] In fact, the increase in efficiency shows that, at larger window sizes, lensing performs closer to optimal, at the cost of a less sharp pulse.

While efficiency increases modestly with window size, the increase levels-off at larger window sizes. Much of this leveling off can be explained by the fact that many high-latency paths exhibit significant jitter (as discussed in § III). In fact, for a window size as large as 100 ms, resolvers with attack path latencies less than 250 ms (about half of those used) show an efficiency (in aggregate) of about 80%, while those with latencies over 250 ms show an efficiency of about 40% (which translates to an efficiency of about 60% over all resolvers).

Figure 6 shows lensing properties as a function of maximum bandwidth to any reflector. Here we have fixed the attacker bandwidth at 10K pps and the window size at 20 ms. The variation in the metrics of bandwidth gain and pulse bandwidth simply reflect high throttling of bandwidth to a constant-size pool of reflectors (discussed in more detail in § V-B). The illuminating metric is that of concentration efficiency. We see little variation, except at very high throttling (effectively sending only 1 or 2 packets per reflector), where we obtain efficiency. With these high efficiencies, however, comes no bandwidth gain, meaning that the attack failed to create a pulse because of the excessive throttling. Due to lack of variation in efficiency, we conclude that an attacker gains little by limiting the bandwidth to each reflector, so they would only do so for purposes of stealth, and not to avoid complications arising from reflectors rate-limiting or failing due to excessive loads (unless those effects only come into play above rates of 500 pps, the bound we used).

Figure 7 shows how the attack scales as a function of the attacker's bandwidth, where we have fixed the pulse window to 20 ms and the per-reflector bandwidth to 500 pps. The relatively constant efficiency at the beginning indicates that the attack scales well; we can explain the diminishing bandwidth gain by the fact that we throttle bandwidth to each reflector while keeping the pool of available reflectors constant (per § V-B). However, we see all metrics perform poorly at higher bandwidths. Plotting peak pulse bandwidth versus maximum attacker bandwidth (not shown) reveals a potential clue: we would expect scaling linear with the attacker's bandwidth, but instead we find it levels off, indicating that the largest pulse we can create of duration 20 ms has a bandwidth of 50–60K pps.

The apparent pulse degradation at scale could mean that the attack scales poorly. However, it could instead arise due to the attack working—increased jitter and queuing could cause pulse flattening or packet loss. (Apparent packet loss could also arise due to measurement loss. We did confirm, however, that `dumpcap` when recording our packet traces reported no drops.)

To determine the cause of the poor scaling for the Azure instance (the one we have been exploring in Figure 7), we stressed it for a short duration at a rate of 100K pps. After three trials, we found the maximum download bandwidth for small DNS packets fell between 57–62K pps. Thus, we conclude that the scaling issues indeed reflect the attack's efficacy—namely, it saturated our Azure instance's bottleneck resource in receiving these packets. We found that our AWS could accommodate a higher pps rate (Figure 8). Here, the pulse only starts to scale poorly at about 110–120K pps (further evidence that the attack was not exhibiting poor scaling for the Azure instance). We lacked sufficient attacker bandwidth to push the pulse higher than that level, so we could not we directly determine with certainty what causes the poor scaling at 120K pps.

However, the difference in behavior between the Azure and AWS instances with regards to *unaccounted* for packets provides a further hint. We define unaccounted for packets as those sent by the attacker but whose reflection never arrives at the target. Some subtleties arise here. Due to retransmissions, nearly all sent packets eventually arrive. Accordingly, we deem packets as "unaccounted" if they fail to arrive within 200 ms of the (20 ms-wide) pulse window.

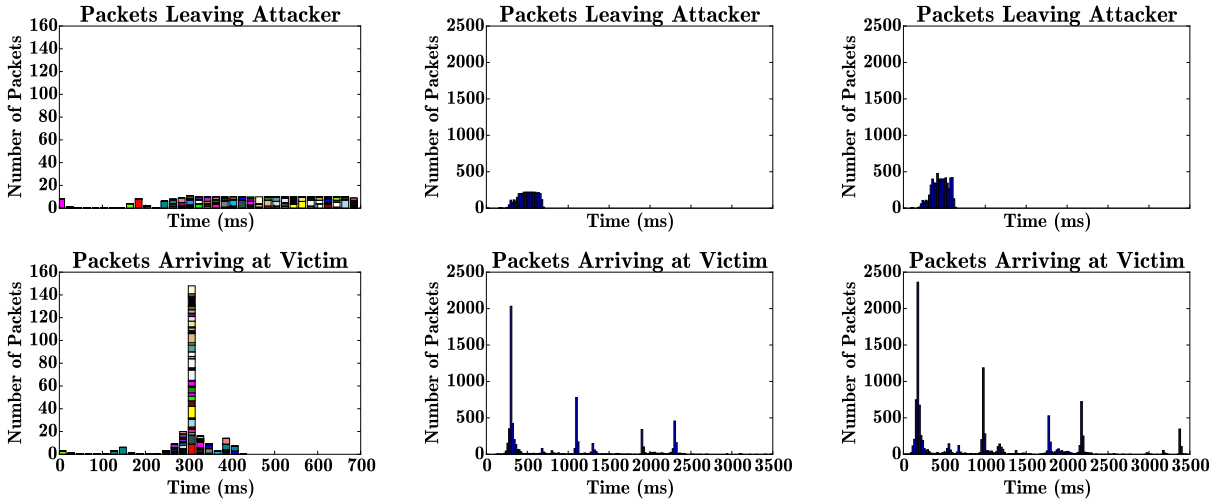
In Azure, it appears that packets beyond the VM's network quota arriving in large bursts get buffered, as evidenced by the pulse spreading in Figure 9. In contrast, the AWS instance does not apparently buffer them (cf. Figures 4b and 4c). Instead, the reflected packets never appear at the AWS end. Figures 10a and 10b quantify this difference. With the Azure instance as the target, we see a relatively constant proportion of unaccounted packets; the attack delivers most of the packets, even at high attacker bandwidths. However, at such higher bandwidths the AWS target receives far fewer such packets, per Figure 10b. It appears that AWS responds to excessive traffic incoming to an instance by dropping packets instead of queuing them as does Azure. This discrepancy between Azure and AWS indicates that the attack indeed worked effectively, and that the leveling-off at $\approx$ 120K pps mentioned above arises due to an AWS resource limit.

In short, the attack displays impressive numbers and scales well. As the peak pulse bandwidth nears the target's maximum capacity, however, the attacker sees diminishing returns.

VII. Extensions

In this section we assess a number of additions or potential "improvements" to lensing attacks.

[6] Similarly, if we focus light over a larger area, then its intensity at any point in that area diminishes.

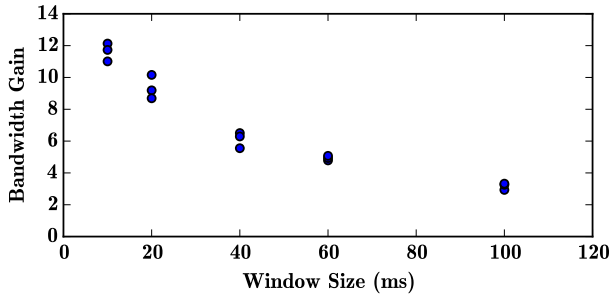


(a) Pulse from low-bandwidth (500 pps) sending to 75 reflectors.

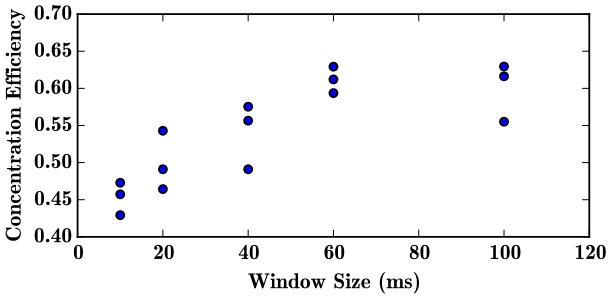
(b) Pulse from medium-bandwidth (10,000 pps) sending to 816 reflectors.

(c) Pulse from high-bandwidth (20,000 pps) sending to 1201 reflectors.

Figure 4: Selected pulses performed on AWS instance, using a maximum per-reflector bandwidth of 500 pps, a pulse window of 20 ms, and a plot bucket also equal to 20 ms. Time along the X-axis for the top and bottom figures does not reflect synchronized clocks, but instead starts (separately) shortly before the first appearance of attack packets.

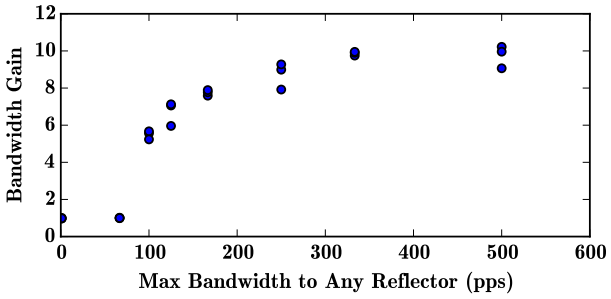


(a) Bandwidth Gain

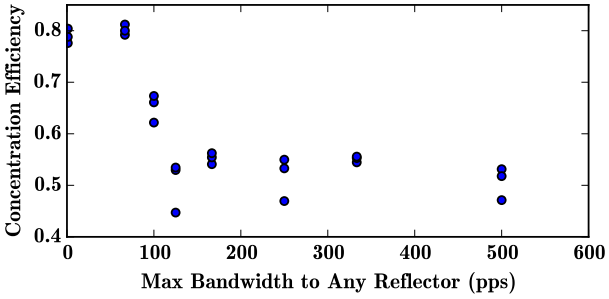


(b) Concentration Efficiency

Figure 5: Lensing metrics as a function of target *pulse window size*, with AWS instance as target. Attacker bandwidth = 10K pps; Max. per-reflector bandwidth = 500 pps.

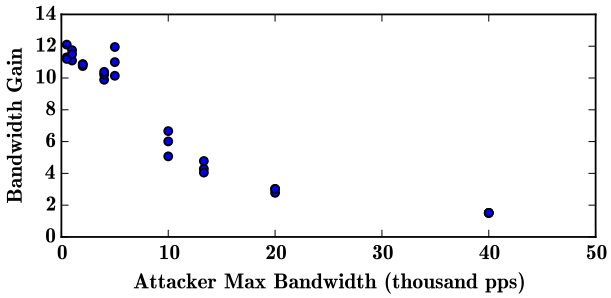


(a) Bandwidth Gain

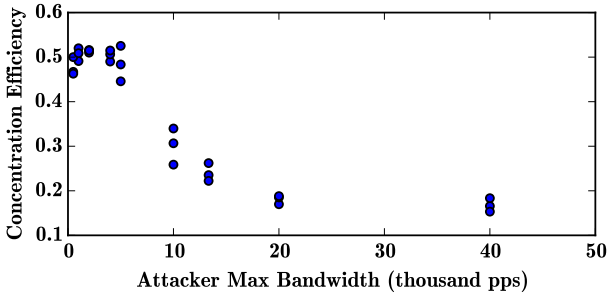


(b) Concentration Efficiency

Figure 6: Lensing metrics as a function of *throttled bandwidth to each reflector*, with AWS instance as target. Attacker bandwidth = 10K pps; window size = 20 ms.

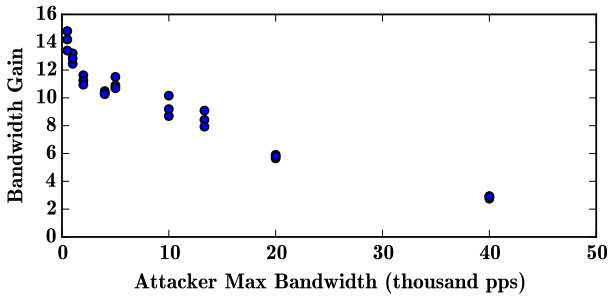


(a) Bandwidth Gain

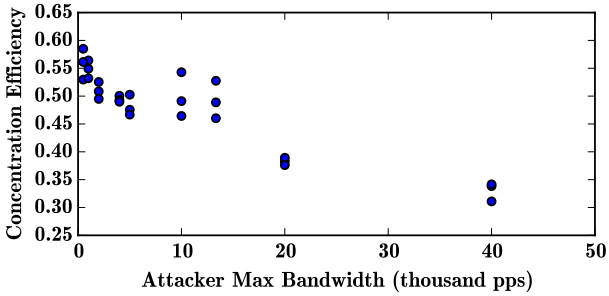


(b) Concentration Efficiency

Figure 7: Lensing metrics as a function of *attacker's maximum bandwidth*, with Azure instance as target. Max. per-reflector bandwidth = 500 pps; pulse window = 20 ms.



(a) Bandwidth Gain



(b) Concentration Efficiency

Figure 8: Lensing metrics as a function of *attacker's maximum bandwidth*, with AWS instance as target. Max. per-reflector bandwidth = 500 pps; pulse window = 20 ms.

A. Attacks on arbitrary end-hosts

We have framed our development of lensing attacks so far in the context of targeting DNS servers. Since DNS generally operates over UDP (and even for TCP has short flows), pulsing attacks—which primarily attack TCP congestion control—may have low efficacy against such servers. To target a more rewarding victim, an attacker must somehow calculate attack path latencies to that host. We present two methods, both heavily influenced by King [8].

DNS cache manipulation. We use manipulation of DNS cache contents[7] to calculate latencies between a DNS resolver and any other type of DNS server (not just an authoritative one), per Figure 11, which shows how to calculate the latencies between a resolver NS_A and any other DNS server NS_B. Along these lines, we create a DNS entry in *our own* authoritative DNS server—`mydomainname.com`—stating that NS_B (`10.0.0.0`) is authoritative for `10-0-0-0.mydomainname.com`. Then, if we issue queries to NS_A for subdomains of `10-0-0-0.mydomainname.com`, NS_A will have cached the NS record indicating NS_B as authoritative for `10-0-0-0.mydomainname.com`, and will query NS_B. NS_B will reply with an error, but the chain of queries still reveals the attack path RTT.

We note that we can extend this cache-poisoning technique of King's to arbitrary end hosts. By replacing NS_B with B (any arbitrary server—not necessarily for DNS), then when B receives a DNS query from NS_A, it most likely will not have a service running on port 53 (DNS). According to RFC 1122 [3], B "SHOULD" respond with an ICMP *Port Unreachable*, and, in response, the UDP layer of NS_A "MUST" pass an error up to the application layer. If NS_A's DNS implementation responds to this error indication by immediately responding to us, we again can calculate the attack path RTT.

Two issues arise when using the above method. First, B might not even receive packets sent to port 53 due to firewalling; or may have an explicit configuration not to respond. Second, the resolver implementation must respond to ICMP error messages propagated to it and deal with them appropriately. That said, we note that our server (using the default configuration of BIND9 on Ubuntu Linux) does in fact do both: it issues an ICMP error when port 53 lacks a running service, and, as a resolver, immediately responds back to the client when informed of an ICMP error. Further, we tested this method to build schedules and create pulses, and found that it indeed works with many resolvers. Some resolvers, however,

[7]Note that this manipulation—while called "poisoning" in the King paper—does not reflect a DNS cache poisoning *attack*. Rather, we simply return incorrect DNS records for domains over which we have legitimate control.

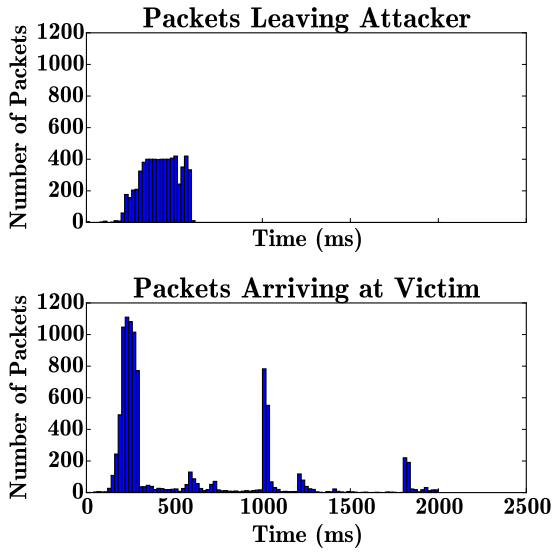
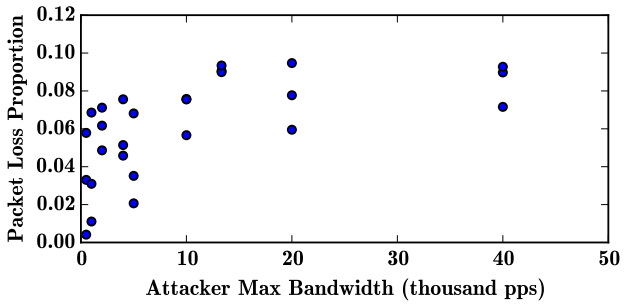
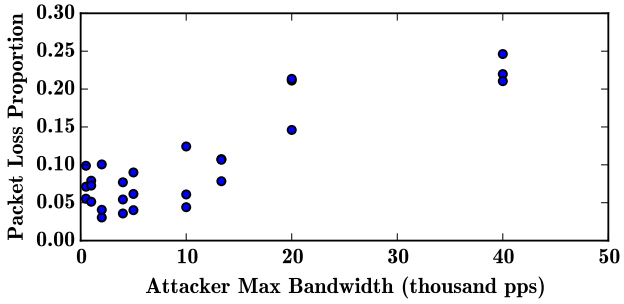


Figure 9: Illustration of pulse spreading at the Azure target. Attacker bandwidth = 20,000 pps; window size = 20 ms. Time along the X-axis for the top and bottom figures does not reflect synchronized clocks, but instead starts (separately) shortly before the first appearance of attack packets.



(a) Azure Instance



(b) AWS Instance

Figure 10: Proportion of unaccounted packets as a function of attacker's maximum bandwidth by instance.

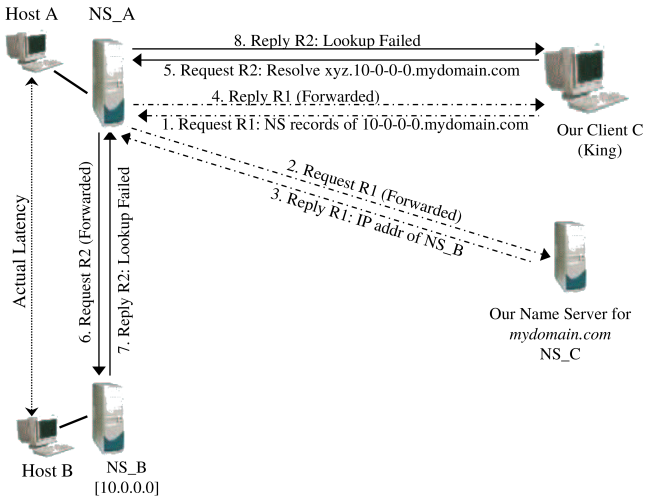


Figure 11: Queries for performing King's cache poisoning technique (reprinted with permission). Dotted lines represent queries to warm up caching of the NS record. Thick lines reflect measurement of the attack path RTT.

do not react to the ICMP, and instead time out, leaving us with no latency data. In short, we expect this refinement of King to allow better estimation of path latencies to many more types of hosts than just DNS servers.

DNS co-location. Lastly, attackers can attempt to find a DNS server co-located with the actual victim. As noted in [8], this occurs relatively commonly, but may introduce errors in latency measurements. However, if the same error arises for each attack path (as we would expect for a DNS server in the same network as the victim), then it will not disrupt the actual pulses. Each reflected packet will arrive at the victim at a constant offset from when expected, but still at the same time as the others.

B. Amplification

A natural extension to concentrating a flood in time is to additionally make the flood larger via amplification. Both amplification and lensing can compatibly leverage open DNS resolvers as reflectors—indeed, attackers already use resolvers in amplification attacks. Attackers could estimate attack path latencies using the methods we adapted from King and during their actual attack use the same reflectors for amplification.

In such a scenario, the attacker would gain the best of both worlds. For example, an amplification factor of 15 and a lensing bandwidth gain of 10 could, at its worst, allow an attacker to create pulses at 150 times the attacker's uplink bandwidth!

Note that the form of lensing we have explored does not need source address spoofing to enable reflection; instead it relies upon recursive queries. However, DNS-based amplification *does* require spoofing. While spoofing should readily work with lensing, we refrained from assessing it both for simplicity and to avoid confusing analysts potentially investigating our

traffic.

Forwarders (discussed in § II) introduce a potential difficulty for estimating attack path latencies in amplification attacks. If the resolver the attacker contacts is indeed a forwarder rather than a full resolver, then the attack path latency measurements will span two intermediary hops between the attacker and the target instead of just one. This problem did not arise for our experiments, since our emulated attack traffic followed the same path as the latency measurements (attacker ⇒ forwarder ⇒ resolver ⇒ victim). However, in an amplification attack we would expect the attacker to place a large query response in each reflector's cache. If cached at a forwarder, then the path of the actual attack traffic will skip the forwarder ⇒ resolver hop, which may or may not introduce a significant latency discrepancy.

Turbo King [13] identified this issue and added the ability to King to detect forwarders, essentially by positioning themselves at both ends of the DNS query (in our example, as attacker and target). An attacker who sets up a personal DNS server can do the same: identify potential forwarders, measure the error they introduce, and weed them out as needed.

C. Distributed attacks

Another natural extension would be to employ a number of geographically distributed machines to attempt to "add" their pulses together at the victim. Doing so would require relatively accurate time synchronization between the attack seeders, depending on the desired pulse width. Our preliminary experimentation with performing such synchronization with NTP found that its precision suffices to reliably create pulses for 40 ms window sizes for attackers located across continents (North America and Europe in our tests). For smaller windows, such as 20 ms, NTP synchronization becomes less reliable. Thus, an attacker can distribute lensing if they do not need a particularly narrow window, but will lose some efficiency squeezing distributed attacks into smaller window sizes.

D. Bolstering the bandwidth gain

Increasing the attack-path latency. Higher attack path latencies give an attacker a longer period over which to send and thus funnel bandwidth. In our measurements the longest attack path latencies we find were around 800 ms. However, a way exists to extend the time a query takes while still keeping the time predictable. If the attacker uses spoofing as the reflection mechanism, then for each resolver the attacker can send it a query that will cause it in turn to issue another query that will take a long time (for example, by needing to contact a distant[8] name server), thus delaying the resolvers final "reply" to the victim by a considerable amount of time. Similarly, the attacker might induce a query to a DNS server that does not respond, causing the resolver to time out and only then send a negative response to the victim. The attacker can likely measure the delay added by such timeouts with high precision.

Retransmits. As shown in Figures 4b and 4c, resolver retransmits can create secondary pulses of their own. The most prevalent timeouts we observed were 800 msec and 2 sec. An attacker can predetermine which timeouts predominate among their set of resolvers and arrange to send pulses at a period matching these timeouts. New pulses will coincide with pulses generated from by retransmissions for previous pulses, essentially superimposing the two and boosting the bandwidth gain.

This attack however does not work when combined with spoofing (for amplification), since in that case traffic reflected off of the resolver consists of responses (rather than queries), and resolvers will not retransmit responses.

[8]One can probably find many misconfigured DNS entries to aid in this regard. Also, an attacker can intentionally misconfigure a personal DNS server to this end, for example by adding an NS entry to a server that will not respond.

VIII. Defenses

In this section we analyze possible defenses against lensing attacks: preventing the initial reconnaissance phase, and undermining the attack itself.

Detecting and thwarting reconnaissance. A potential target could readily detect our King-style reconnaissance measurements (for obtaining attack path latencies) due to their noisy nature, as they exhibit a clear signature in terms of repeated queries for non-existent subdomains. Attackers could however potentially hide their presence by making queries for legitimate subdomains, since our experiences show that 10 or fewer queries per resolver suffices, so the attacker would just need to find a few unique domain names.

The victim could possibly *thwart* reconnaissance by poisoning attack path RTT measurements, for example by introducing artificial jitter. Simply adding random delay to each request will only slow the attacker down, since by gathering more measurements they can likely employ robust statistics to remove the noise. The victim might instead introduce an amount of jitter fixed as a function of the reflector's address, ideally using a function keyed so that knowing one jitter would not reveal information of another.

However, in response to either of these approaches the attacker might instead make measurements to a nearby server *not under the victim's control*. In this case, the victim may well lack any opportunity to detect reconnaissance or introduce jitter.

Resisting attacks. Significant work exists on defending against pulsing DoS attacks, much of which has application to lensing as well, including RTO randomization [5], [11], [22], extensions to RED [23], and increasing buffer sizes [17]. Increased buffering would likely manifest similar to Azure's pulse spreading (illustrated in Figure 9) and consequent decreased packet loss compared to AWS (in Figure 10). This defense is not complete, however, as (a) it requires significant buffer sizes, and (b) legitimate traffic will still suffer a latency hit.

Regarding defending specifically against the *lensing* side of our attack, a potential defense again revolves around introducing jitter. Routers might somehow add jitter during

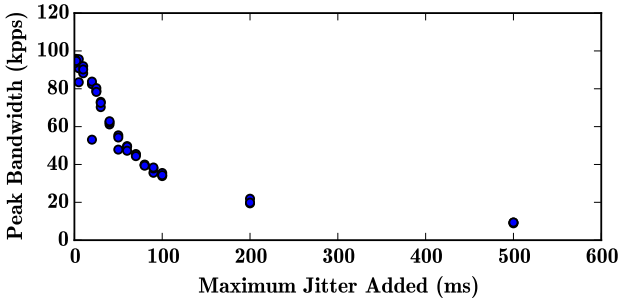


Figure 12: Pulse degradation upon the addition of artificial jitter (pulse window = 20 ms, sending rate = 10K pps).

an attack, possibly keyed off of flow 5-tuples to prevent intra-flow reordering of legitimate traffic. Indeed, in principle such an approach has effects similar to those of multipathing [2].

Figure 12 shows how this might play out. Here we added uniformly distributed jitter to an emulated attack's sending schedule, essentially providing the same effect as routers adding jitter would have. The graph indicates that cutting a relatively small window pulse of 20 ms in half would require adding jitter of 60 ms (i.e., uniformly distributed in the range 0–60 ms) to the attack path.

In short, it appears that attackers can readily hide their reconnaissance from their targets. Mitigating actual lensing attacks would require somehow changing attack path latencies during the attack, but requires the addition of a significant amount of jitter (which would itself severely impair real-time traffic). These issues suggest that more general defenses against pulsing attacks (such as improving TCP congestion control robustness) might offer the most promise, rather than specifically attempting to counter the timing-based nature of lensing attacks.

IX. Summary

We have introduced the concept of temporal lensing, which lends itself quite naturally to conducting pulsing DoS attacks. Using DNS recursion to both estimate attack path latencies and to create pulses from relatively low-bandwidth floods, we experimentally demonstrated its practicality and explored its scaling properties. In addition to its direct application to flooding a victim's DNS server, we sketch how attackers can likely distribute attacks and employ lensing for non-DNS targets. We also explored mechanisms for increasing the bandwidth gain further. We find that lensing by itself allows attackers to concentrate the bandwidth of a flood by an order of magnitude. Given these results, lensing's further compatibility with amplification, and the difficulties that arise in constructing defenses, the attack appears to pose a significant threat.

X. Acknowledgments

We are grateful to Mark Allman for much helpful advice especially on forwarders. We also thank Ethan Jackson for valuable insight on pinpointing packet loss on VMs and S. Zayd Enam for fruitful discussions and generously allowing use of his dedicated machine for bandwidth testing. Thanks too to our shepherd, Vyas Sekar, and the anonymous reviewers for their helpful comments. This work was supported by National Science Foundation grant 1237265, for which we are grateful. Opinions expressed in this work are those of the authors and not the sponsor.

References

[1] Public DNS Server List, May 2014. Available at `http://public-dns.tk/`.

[2] Augustin, B., Cuvellier, X., Orgogozo, B., Viger, F., Friedman, T., Latapy, M., Magnien, C., and Teixeira, R. Avoiding traceroute anomalies with Paris traceroute. In *Proc. ACM Internet Measurement Conference* (2006).

[3] Braden, R. RFC 1122: Requirements for Internet Hosts.

[4] Dabek, F., Cox, R., Kaashoek, F., and Morris, R. Vivaldi: A Decentralized Network Coordinate System. In *Proc. SIGCOMM* (2004), vol. 34.

[5] Efstathopoulos, P. Practical Study of a Defense Against Low-rate TCP-targeted DoS Attack. In *ICITST* (2009), IEEE, pp. 1–6.

[6] Francis, P., Jamin, S., Jin, C., Jin, Y., Raz, D., Shavitt, Y., and Zhang, L. IDMaps: A Global Internet Host Distance Estimation Service. *IEEE/ACM Transactions on Networking 9*, 5 (2001).

[7] Guirguis, M., Bestavros, A., Matta, I., and Zhang, Y. Reduction of Quality (RoQ) Attacks on Internet End-systems. In *Proc. IEEE INFOCOM* (2005).

[8] Gummadi, K. P., Saroiu, S., and Gribble, S. D. King: Estimating Latency Between Arbitrary Internet End Hosts. In *Proc. ACM Internet Measurment Workshop* (2002).

[9] Kang, M. S., Lee, S. B., and Gligor, V. D. The Crossfire Attack. In *Security and Privacy (SP), 2013 IEEE Symposium on* (2013), pp. 127–141.

[10] Kogler, T. M. Single Gun, Multiple Round, Time-on-Target Capability for Advanced Towed Cannon Artillery. Tech. rep., US Army Research Laboratory, Aberdeen Proving Ground, 1995.

[11] Kuzmanovic, A., and Knightly, E. W. Low-rate TCP-targeted Denial of Service attacks: The Shrew vs. the Mice and Elephants. In *Proc. ACM SIGCOMM* (2003).

[12] Le, H. B. Advanced Naval Surface Fire Support Weapon Employment Against Mobile Targets. Tech. rep., Naval Postgraduate School, Monterey, Calif., 1999.

[13] Leonard, D., and Loguinov, D. Turbo King: Framework for Large-scale Internet Delay Measurements. In *Proc. IEEE INFOCOM* (2008).

[14] Luo, X., and Chang, R. K. On a New Class of Pulsing Denial-of-Service Attacks and the Defense. In *Proc. NDSS* (2005).

[15] Ng, T. E., and Zhang, H. Predicting Internet Network Distance with Coordinates-Based Approaches. In *Proc. IEEE INFOCOM* (2002).

[16] Paxson, V. An Analysis of Using Reflectors for Distributed Denial-of-Service Attacks. *ACM SIGCOMM CCR 31*, 3 (2001).

[17] Sarat, S., and Terzis, A. On the Effect of Router Buffer Sizes on Low-rate Denial of Service Attacks. In *Proc. Computer Communications and Networks* (2005).

[18] Schomp, K., Callahan, T., Rabinovich, M., and Allman, M. On Measuring the Client-Side DNS Infrastructure. In *Proc. ACM Internet Measurement Conference* (2013).

[19] Sharma, P., Xu, Z., Banerjee, S., and Lee, S.-J. Estimating Network Proximity and Latency. *ACM SIGCOMM CCR 36*, 3 (2006), 39–50.

[20] Weigley, R. *Eisenhower's Lieutenants: The Campaigns of France and Germany, 1944–45*. Indiana University Press, 1981.

[21] Wong, B., Slivkins, A., and Sirer, E. G. Meridian: A Lightweight Network Location Service without Virtual Coordinates. In *Proc. SIGCOMM* (2005), vol. 35.

[22] Yang, G., Gerla, M., and Sanadidi, M. Defense Against Low-rate TCP-targeted Denial-of-Service Attacks. In *ISCC* (2004), vol. 1, IEEE, pp. 345–350.

[23] Zhang, C., Yin, J., Cai, Z., and Chen, W. RRED: Robust RED Algorithm to Counter Low-rate Denial-of-Service Attacks. *Communications Letters 14*, 5 (2010), 489–491.

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