Cluster-Based Analysis for Combatting DDoS Attacks

Bryan Cuccioli

Cornell University

Abstract

Distributed denial of service attacks constitute one of the most easily exploited and difficult to defend against risks to computer network security. We propose a route-based system for recognizing and mitigating DDoS attacks as they occur in real time. The system is designed with the primary goal of recognizing requests from illegitimate sources participating in a DDoS attack using time-based sequential k-means clustering, enabling the service to drop those requests while still handling and performantly responding to those representing legitimate users.

We implement the proposed design for a simplified use case in Java. We find that with the classifying gateway in place, the average latency of the service does not increase past the standard error margin when the system is exposed to multiple other hosts continuously sending requests.

1 Introduction

Distributed denial of service (DDoS) attacks are among the most common, devastating, and difficult to prevent security risks facing the Internet today. Such attacks consist generally of flooding a target service with a sufficiently high number of requests as to take processing time for legitimate requests, introduce instability into the system, and surface latent issues with the service that are only visible at large scale. Attackers are able to carry out DDoS attacks with large number of hosts under their control – organized into so-called *botnets* – or in some cases with a small number of hosts that can cause other services to output large amounts of information, as in the case of *DNS amplification* attacks.

DDoS attacks are particularly difficult to defend against because unlike most other computer security attacks, they are not dependent on and do not target specific vulnerabilities, either at the implementation level or protocol level. Rather, they are indistinguisable from legitimate user requests and traffic, except for their frequency. This has made quickly identifying and defending against DDoS attacks without interrupting the entire network service very difficult.

The goal of this project is to equip web services with a way to defend against DDoS attacks as easily and quickly as possible, with minimal disruption to the user experience. This mechanism should be completely transparent to the legitimate end-user, avoiding modifying their experience or the service's interface in the event of a DDoS attack. It should also be mostly transparent to the web service itself, requiring minimal feedback and overhead from the service so as to minimize the amount of extra processing power needed during a DDoS attack. Finally, this mechanism should not rely on extensive infrastructure modifications or those outside the control of the owner of the web service. Such modifications would require widespread adoption to work, and it would be difficult for the average service operator to have them implemented.

The design of this mechanism revolves around a network of gateway-like nodes between the web service and the rest of the Internet. These gateways will be responsible for distributing requests to the nodes of the service, using clustering techniques from machine learning to attempt to quickly separate legitimate hosts from those participating in the DDoS attack, silently discarding malicious requests and passing others on to the web service. This way, while the service will have to respond to some initial small number of malicious requests, it will be largely shielded from devoting processing power to them.

The remainder of this paper is as follows. In section 2, we discuss related work and how it compares with our proposal. In section 3, we give a detailed design overview for our system. In section 4, we give a more low-level overview of our implementation of the system. Finally, in section 5, we present our evaluation plan and empirical results, demonstrating the efficacy of the system.

2 Related Work

Because of the important security implications, a lot of work has already been done in this area using a variety of ideas derived from a number of differing perspectives. Below, we present a comprehensive cross-section of the techniques being researched and employed to prevent, protect against, and mitigate DDoS attacks.

2.1 Friction-Based Prevention

Some of the most basic ideas for mitigating DDoS attacks are based on a redesign at the level of the protocols governing network communications. In *DoS: Fighting Fire with Fire* [10], the authors propose limiting the rate of incoming requests and requiring users to aggressively retransmit their requests until they go through successfully; this is a version of the idea of requiring users to make micropayments to use the service, where the payments here come in the form of bandwidth.

Along a similar vein, the paper *Game-Based Analysis of Denial-of-Service Prevention Protocols* [7] proposes requiring the client to participate in a key exchange with the server before the server is able to handle any of its requests. This requires a micropayment in the form of client processing time, and has the added benefit of ensuring that the client is not spoofing the origin of their data. A very similar approach is taken in the paper *Efficient, DoS-Resistant, Secure Key Exchange for Internet Protocols* [1].

2.2 Route-Based Filtering

Other researchers have been more focused on designing and implementing systems to perform route-based filtering, similar to our proposal. The paper *On the Effectiveness of Route-Based Packet Filtering for Distributed DoS Attack Prevention in Power-Law Internets* [8] discusses how effective certain techniques are for router-based packet filtering based on the assumption that the Internet obeys certain mathematical power law relationships with respect to the connectivities and degrees of its nodes. A similar approach is taken in the paper *Mayday: Distributed Filtering for Internet Services* [2]. In addition, the paper *Using Active Networking to Thwart Distributed Denial of Service Attacks* [3] uses a software-based network infrastructure called Active Networking which can be easily reprogrammed and reshaped to minimize the illegitimate traffic's effects on the service. While these systems all use route-based filtering, they differ from our system in that they use certain mathematical assumptions about the structure of the network rather than an adaptive, learning-based solution.

2.3 Learning-Based Prevention

Some have adapted principles from data mining and machine learning to the problem of DDoS recognition and prevention. Some of these techniques draw from the mathematical disciplines of probability and statistics. The paper *Detecting DoS and DDoS Attacks by Using an Intrusion Detection System and Remote Prevention System* [6] uses a *cumulative sum algorithm* from statistical quality control to determine whether a group of packets is from a DDoS attack. The paper *Random Flow Network Modeling and Simulations for DDoS Attack Mitigation* [5] takes a more theoretical approach, constructing models based on network flow and running simulations based on idealized Internet topological models to analyse the emergent behaviour of DDoS attacks.

While these systems are based on learning techniques to help detect DDoS attacks, they are heavily theoretical and do not make much attempt to implement these ideas in practice. They use more complex mathematical models that may not be easily integrated into an existing system. Moreover, they do not break down the time-based data into dimensions reflecting normal operation versus operation under a DDoS attack.

2.4 Traceback-Based Prevention

One of the most useful changes in infrastructure to allow recognizing and defending against DDoS attacks would be the ability to trace a connection back to its source, mapping out the path it took to arrive at the target. One approach, taken in the paper *Research on Authentication Scheme for DDoS Attack Source Traceback* [11] uses the *reflection relay algorithm* and algebraic coding theory to mark packets and perform path reconstruction. The paper *Reverse Traceroute* [4] describes the implementation of a system for similarly reconstructing the path a given packet took to arrive from its original source. The paper *Network Support for IP Traceback* [9] describes a novel approach towards achieving this capability without necessitating infrastructure modifications. This works by probabalistically marking packets and using a large number of them to reconstruct partial paths.

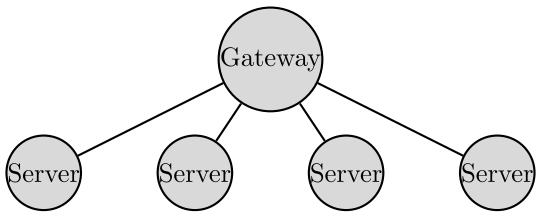

Figure 1: Topology of our system

3 Design

We structure our system as a gateway guarding the nodes comprising the web service (Figure 1). A larger or more sophisticated system may desire to partition the nodes behind multiple gateways linked to a load-balancing router. The gateway is responsible for enforcing a sharp upper bound on the rate of requests that get through to the web service for processing.

One of the central design tenets of our system is simplicity. The difficulty of implementing other anti-DDoS schemes has been a barrier to their adoption in practice. A system that is effective yet simple to integrate is more likely to be of use to administrators of web services.

3.1 Routing

For the sake of simplicity, the gateway performs simple routing of requests to each hosts in a round-robin-like fashion, based on the most recently reported load of the host. This load balancing scheme is furnished by the servers periodically communicating to the gateway information about what level of stress they are currently under. Each server i re-calculates a *load score* L_i. This embodies

1. the number of threads currently dispatched from their thread pools T_i to handle requests, and
2. for each thread t_j, the amount of time N_t the most recent request for the thread required to complete.

In particular

$$L_i = \sum_{t \in T_i} N_t. \tag{1}$$

This heuristic is intentionally simple: it provides a reasonable snapshot of the amount of load server i is under without requiring too much processing power, and hence behaving as transparently as possible while also not appreciably skewing the actual load of the server.

Each gateway maintains a priority queue P_j holding the servers (the *server queue*), subject to the invariant that if $L_i < L_k$ for the most recently received values of L_i and L_k, then $k \prec i$ in the priority queue. That is, a server has higher priority in the queue if its load score is lower. Moreover, there is a user-defined *load threshold* M that bounds above the amount of load a server can be under for it to be considered available. Explicitly, we do not add i to P_j if $L_i > M$ in the most recent update cycle.

In order to safely implement request dispatch, the gateway maintains a simple *request queue* Q_j of incoming requests. If P_j is non-empty, the gateway immediately dequeues a server i from P_j and dispatches the incoming request to server i. Otherwise, the request is added to Q_j. After every update cycle, the gateway checks P_j for a server, and if it finds one, dispatches the request to the server as described above.

3.2 Request Classification

The primary function of the gateway is to classify requests into two types: legitimate and malicious. We seek to perform the classification based on the relative times at which requests are seen. To this end, there are two dimensions of data we wish to measure: time and frequency of requests seen during normal operation; and time and frequency of requests seen when under a DDoS attack. This suggests using a cluster-based analysis to discern sources associated with normal operation from those associated with a DDoS attack.

Additionally, there are several necessary features for the classification process which we base our design around.

1. The classifier should mark request sources as malicious without having to see them a large number of times. Each time a classifier sees a request from a source and has not yet marked it as malicious, it will take up server processing time.
2. The classifier should over time lower its confidence that the source is malicious if no further

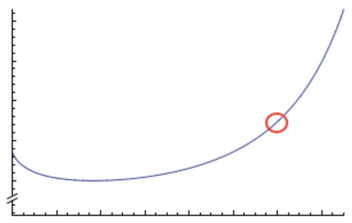

Figure 2: Illustration of the beginning of a DDoS event

attacks occur. This is necessary in the event that the source IP address is leased to a legitimate host or in the event that the attacker has hijacked a host that would be sending legitimate requests and is using it as a zombie. The relative likelihood of these events is fairly low, so slowly lowering the confidence level that a source is malicious is acceptable.

3. Symmetrically, the classifier should slowly lower its confidence that the source is legitimate if no new DDoS event occurs, for the same reasons.

With the above needs in mind, maintain a global coordinate plane which will hold unique request sources. When the service is not known to be experiencing a DDoS attack, we add the source of each incoming request to the plane with coordinates (α, α) that it is a legitimate source. At each time interval τ, we update α to $\alpha - \Delta\alpha$ for some small pre-defined value of $\Delta\alpha > 0$. This encodes requirement (3) above.

We also globally track the rate of incoming requests at the gateway level. At every pre-defined interval τ, we store $F(t)$: the number of requests that have been processed at time t. We also globally store the *linearized derivative*

$$\partial F(t) \approx \frac{F(t) - F(t-\tau)}{\tau} \tag{2}$$

and *linearized second derivative*

$$\partial^2 F(t) \approx 2\frac{F(t) - F(t-\tau) - \partial F(t-\tau)\cdot\tau}{\tau^2}. \tag{3}$$

(These are derived by interpolating F to a continuous function and computing its first- and second-order Taylor series.)

These heuristics give a precise measurement of when the rate of requests is sharply increasing. When $F(t)$, $\partial F(t)$, and $\partial^2 F(t)$ rise above certain pre-defined thresholds, we say a DDoS event has begun at time t (see Figure 2 for a possible illustration of the beginning of such an event, as recognized by the above heuristic). As new requests come in, we mark those requests with coordinates (β, β). As above, after every time interval τ, we decrease the confidence with which we believe a source to be malicious by $\Delta\beta$. This serves two purposes: it encodes requirement (2); and it biases malicious classification towards sources actually participating in the DDoS attack, since those sources are expected to send requests multiple times in fast succession.

When adding sources to the plane, we continuously perform a *sequential k-means clustering* algorithm using a low-pass filter (Figure 3). This algorithm is desirable because of its *forgetful* property: newer samples receive more weight than older samples. More precisely, we can see that each mean reaches its final state in terms of its nearest samples as

$$\mathbf{m}_i = \lambda \sum_{k=1}^{n} (1-\lambda)^{n-k} x_k. \tag{4}$$

This helps to encode properties (2) and (3). Moreover, the update step of the algorithm is very simple and can be implemented in only a few operations, minimizing the bottleneck effect of the classifier.

The end result of the above classification steps is we now have a strong notion of which sources are legitimate and which are actively participating in the DDoS attack against the web service. For example, the resulting clustering from a small-scale run is shown in Figure 4. Now, when a gateway node receives a request, it first consults the classifier to determine whether the source is believed to be malicious. If the source has been classified as malicious with sufficiently high confidence, the gateway node simply resets the connection and discards the request. Otherwise, it adds the request to the request queue Q_j as described above.

4 Implementation

For the sake of quickly prototyping the system, we implement the system using Java. We will use the implementations included in the Java standard library

```
function SequentialKMeans(input stream $\mathbf{x}$)
    $\mathbf{m}_1 \leftarrow (0, 0)$
    $\mathbf{m}_2 \leftarrow (0, 0)$
    while $\mathbf{x}$ contains more samples $x_i$ do
        $i \leftarrow$ index of mean closes to $x_i$
        $\mathbf{m}_i \leftarrow \mathbf{m}_i + \lambda(x_i - \mathbf{m}_i)$
    end while
end function
```

Figure 3: Sequential k-means clustering algorithm

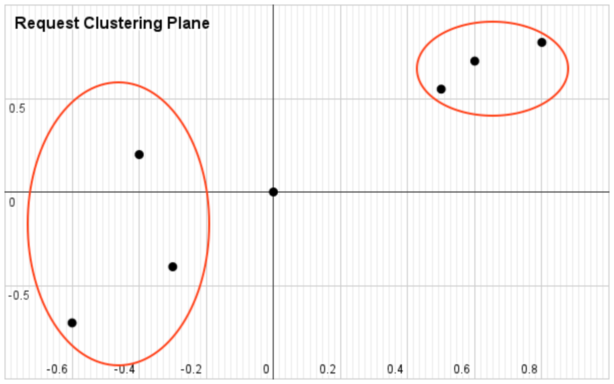

Figure 4: Sample clustering of requests

for concurrency, socket communications, and the necessary data structures.

4.1 High-Level Implementation

The class `Host` represents an abstraction for each server node in the system. Each `Host` instance has a load score, associated socket for communicating with the gateway, and a `Queue<Socket>` of incoming requests, the latter of which is not cached at the thread level. When writing a response back to the gateway, the `Host` also encodes the most recent load score in the response. Each thread running on a `Host` object repeatedly polls the queue for incoming requests, sends it to the server, reads the response, and writes the response back to the gateway.

Configuration variables are stored in a global static `Configuration` class as public static constants. This helps to easily expose the configuration settings and make the system easily tunable.

The `Clustering` class maintains a static `HashMap` which maps request sources to coordinates in the clustering plane, as well as a `BlockingQueue` of incoming samples to classify and current means for both clusters. It continuously polls the `BlockingQueue` for a new sample, at which point it updates its values in the `HashMap` and then updates the means accordingly – this entire operation runs in $O(1)$ constant time to reduce the latency caused by the classifier. It exports a public function `isGoodHost(String)` which tells in $O(1)$ time to which cluster the given host belongs.

The entry point for the application is the `Gateway` class, which represents the routing gateway guarding the server nodes. This class maintains a `PriorityQueue<Host>` of `Host` objects, sorted by lowest recent load score. It also maintains a global request count and request derivatives. A special thread is forked for the `Updater` class, which is responsible for continuously reading the number of requests and recomputing the linearized first and second derivatives. Upon initializing, the gateway reads the `Configuration` class for a list of services, and then sends an echo request to each one to check that they are operational and that they can communicate with the gateway. The `Gateway` object then waits for an incoming request, at which point it continuously polls the priority queue for an available host until one exists. It then consults the `Clustering` class and forwards the request on to the host if appropriate.

4.2 Toy Service

For evaluation purposes and to demonstrate the interface for communicating with the gateway system, we implement a *toy service*. This service listens on a network socket for a numerical input N, and upon receipt returns a tuple $(p, N/p)$ where $p > 1$ is the lowest number which divides N. For large values of N with only two prime factors, this process is almost entirely CPU bound. This makes it ideal for testing our system, as it has very little potential for latency due to I/O or network usage – communications both ways each consist of only a few bytes.

5 Evaluation

The evaluation of the system is carried out on a campus-wide local area network to minimize network latency.

5.1 Experimental Setup

The main difficulty in designing experiments to test the efficacy of our system is that real DDoS attacks consist of thousands of nodes sending huge amounts of packets for a long time period to a host that is normally able to handle large quantities of traffic. Perfectly replicating this situation requires an unrealistic amount of resources. Therefore, our experimental setup is focused on scaling down both the attack size and target size considerably so that it can be realistically run. Because of the distributed nature of the system, the behaviour in the scaled down setup should be faithful to the real-world situation.

To create a simple experimental situation, we implement a *toy service* which receives a TCP packet containing a single large integer N as the body, where N has prime factorization $N = pq$, and returns a packet containing p and q. This way, single requests will command a relatively large amount of processor time while associated communications will be kept very simple.

We build this service from four identical listening processes which peform this integer factorization, and initialize one gateway node to guard them.

5.2 DDoS Event Simulation

Once the system has been initialized, we begin by interacting with it by sending and receiving single requests from five different hosts on the LAN. We then initialize up to 20 processes sequentially which begin sending large volumes of requests to the service from different unique class C IP addresses. While this "attack" is taking place, we manually send requests to the service and measure the response time.

5.3 Experimental Results

We carried out our evaluation scaling from 0 to 20 DDoS attacker nodes a total of 10 times and averaged the results. We found that service latency is nearly constant with respect to the scale of the DDoS attack. In particular, we found a mean latency of $\mu \approx 26.26$ ms and standard deviation $\sigma \approx 2.57$ ms. This result is summarized in Figure 5.

Additionally, of the total of 521 legitimate requests sent manually to the system, we found that 508 of the requests were responded to (97.5%). That is, only 2.49% of requests were mis-classified as malicious and dropped.

6 Conclusion

Distributed denial of service attacks prove very difficult to recognize and mitigate, as they are indistinguishable from normal traffic except in the volume of activity. Our system aims to exploit this single characteristic to classify and withstand such attacks. We perform a cluster-based analysis of request times and classify sources based on their appearance during DDoS attacks. With this system, we are able to quickly and efficiently classify sources, resulting in effectively constant service latency in the face of a DDoS attack while dropping a minimal number of legitimate user packets.

7 Acknowledgements

We would like to thank Prof. Hakim Weatherspoon and Erluo Li for providing feedback and suggestions.

References

[1] William Aiello, Steven M. Bellovin, Matt Blaze, John Ioannidis, Omer Reingold, Ran Canetti, and Angelos D. Keromytis. Efficient, DoS-resistant, secure key exchange for internet protocols. In *Proceedings of the 9th ACM conference on Computer and communications security*, CCS '02, pages 48–58, New York, NY, USA, 2002. ACM.

[2] David G. Andersen. Mayday: distributed filtering for internet services. In *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, pages 3–3, Berkeley, CA, USA, 2003. USENIX Association.

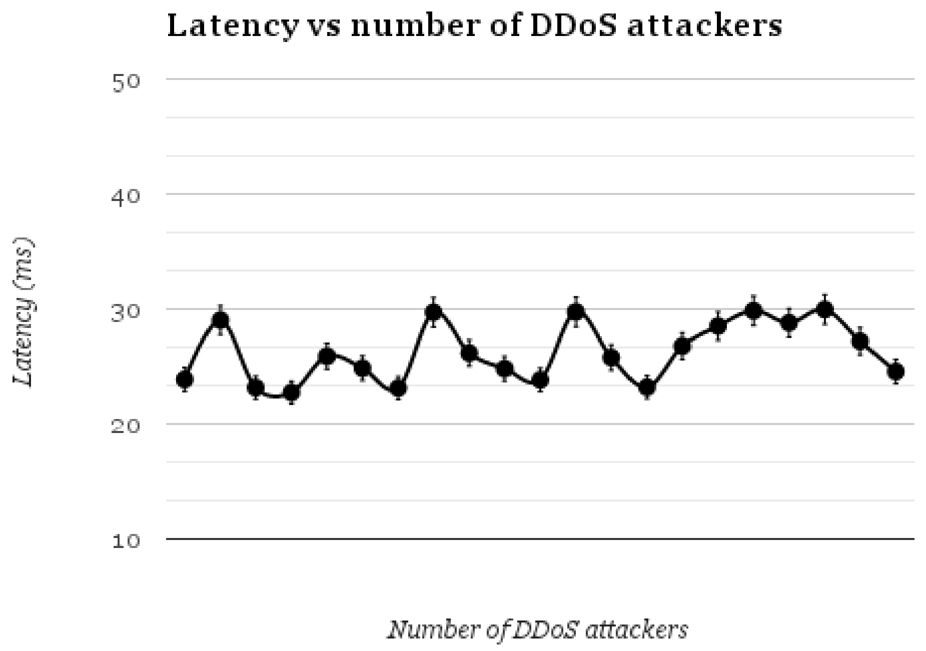

Figure 5: Variation in latency w.r.t. scaling DDoS attack

[3] Adam Hovak, Joseph Riolo, and Scott Shyne. Using Active Networking to Thwart Distributed Denial of Service Attacks. In *Proceedings of IEEE Aerospace Conference*, pages 1103–1108, 2001.

[4] E. Katz-Bassett, H. Madhyastha, V. Adhikari, C. Scott, J. Sherry, P. van Wesep, A. Krishnamurthy, and T. Anderson. Reverse Traceroute. In *USENIX Symposium on Networked Systems Design & Implementation*, 2010.

[5] Jiejun Kong, M. Mirza, J. Shu, C. Yoedhana, M. Gerla, and Songwu Lu. Random flow network modeling and simulations for DDoS attack mitigation. In *Communications, 2003. ICC '03. IEEE International Conference on*, volume 1, pages 487–491 vol.1, 2003.

[6] Fang-Yie Leu and Zhi-Yang Li. Detecting DoS and DDoS Attacks by Using an Intrusion Detection and Remote Prevention System. In *Proceedings of the 2009 Fifth International Conference on Information Assurance and Security - Volume 02*, IAS '09, pages 251–254, Washington, DC, USA, 2009. IEEE Computer Society.

[7] Ajay Mahimkar and Vitaly Shmatikov. Game-Based Analysis of Denial-of-Service Prevention Protocols. In *Proceedings of the 18th IEEE workshop on Computer Security Foundations*, CSFW '05, pages 287–301, Washington, DC, USA, 2005. IEEE Computer Society.

[8] Kihong Park and Heejo Lee. On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pages 15–26, New York, NY, USA, 2001. ACM.

[9] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Network support for IP traceback. *IEEE/ACM Trans. Netw.*, 9(3):226–237, June 2001.

[10] Michael Walfish, Hari Balakrishnan, David Karger, and Scott Shenker. DoS: Fighting Fire with Fire. In *4th ACM Workshop on Hot Topics in Networks (HotNets)*, College Park, MD, November 2005.

[11] J. Zhang and S.Q. Chen. Research on Authentication Scheme for DDoS Attack Source Trace-

back. In *Application Research of Computers*, volume 24, pages 131–134, 2007.