Lightweight Resource Management for DDoS Traffic Isolation in a Cloud Environment

Ibnu Mubarok, Kiryong Lee, Sihyung Lee, Heejo Lee

Lightweight Resource Management for DDoS Traffic Isolation in a Cloud Environment

Ibnu Mubarok[1], Kiryong Lee[1], Sihyung Lee[2] and Heejo Lee[1]

[1] Korea University {ibnu, krlee, heejo}@korea.ac.kr
[2] Seoul Womens University {sihyunglee}@swu.ac.kr

Abstract. Distributed denial-of-service (DDoS) attacks are one of the most difficult issues in network security and communications. This paper is a part of research project that applies distributed defense against distributed attacks. The aim of this project is to provide services by distributing load from one main server to an infrastructure of cloud-based replicas. This paper proposes a lightweight resource management for DDoS traffic isolation in cloud environments. Experimental results show that our mechanism is a viable approach for dynamic resource scaling under high traffic with distributed resource location.

Keywords: Cloud computing, DDoS attack, resource management

1 Introduction

Existing DDoS mitigation solutions can be categorized by their approach: victim-based and network-based. Victim-based solutions focus on the mitigation of an attack at the client side, while network-based solutions protect the attack at routers. Victim-based solutions suffer from several drawbacks, such as the need for oversized server resources. Network-based solutions often suffer from link congestion and other challenging issues. New types of attacks, such as Layer-7 DDoS attacks, also contribute to the difficulties with network-based solutions. Several methods use a signature-based approach, in which attack signatures are automatically shared among service providers [1]. However, the collection and analysis of these signatures are time consuming. Content delivery network (CDN)-based approaches, such as Akamai [5] and CloudFlare [3], typically use cache servers, and therefore, they are vulnerable to cache pollution attacks such as false-locality and locality-disruption attacks [8], [4]. Attackers might continue to send requests for unpopular data and pollute the cache stored in different locations within the network topology, leading to service disruption and incurring additional costs.

Enormous resource capacity and ease of scalability in cloud computing could be beneficial to protecting services from DDoS attacks. Many companies and researchers are considering the use of a cloud as a shield from DDoS attacks. However, one concern with this approach is the resource usage. Because one component of billing for cloud services is resource usage and most DDoS attacks aim to deplete a servers resource. However, if critical services are being targeted, and those services cannot be shut down, there is no other option but to keep the service alive using the clouds massive power.

This paper proposes a lightweight resource management for isolating DDoS traffic in a cloud environment. Our work focuses on resource allocation management and provides a mechanism for scaling resources under DDoS attack. In this paper, we describe a simple resource placement based on response time and a modified threshold-based scaling mechanism to ensure resource availability.

The main contribution of this research is to propose and evaluate a simple yet effective mechanism for resource scaling and allocation management in a cloud environment under a DDoS attack. Further, this mechanism can be used when the large bandwidth consumption from other legitimate applications lead to the degradation of the servers quality of service (QoS).

2 Distributed Defense Against Distributed Attacks

2.1 DROPFAST

This paper is a part of a research project that is attempting to apply distributed defense against distributed attacks. The project, DROPFAST, is based on the idea of providing efficient and secure service by distributing the load from one main server to an infrastructure of cloud-based replicas of the main protected server [7].

Our project is based on the use of the massive computing power of the cloud infrastructure as well as the distributed nature of the cloud to protect the system from DDoS attacks. Our goal is to distribute DDoS traffic inside the cloud environment, guaranteeing that the main server remains accessible to legitimate users. Anomalous users are directed to a replica server in the cloud, whereas an obvious attacker is directed to the quarantine server.

We define the main server as our primary server in which the services must be protected during a DDoS attack. Replica servers are virtual machine resources in the cloud comprising an image of the original server. Based on traffic classification, legitimate users can access the original server, whereas users are forced to access the replicas. We can not simply drop the suspicious traffic, as it is the gray area in which user might be legitimate. For DDoS detection, we use an open source IDS such as Snort system since detection is not our main research focus. To distribute traffic, we apply DNS-based traffic redirection.

As today's services depend on multiple resources, all the independent resources might be replicated in several locations. Our prototype implementation only mentions about one type of resource, which is a web server, but the concept and idea could be extended as well.

2.2 Problem Statement

In a cloud, computing resources must be allocated and scheduled in such a manner that providers achieve high resource utilization and users meet their applications performance requirements with minimal cost. This task is related to cloud resource management. Based on the idea of providing service by redistributing load from one main server to an infrastructure of cloud-based replicas, in addition to proper traffic isolation

and user classification, a clouds resources can be managed to efficiently mitigate DDoS attacks. Therefore, to guarantee service availability, a cloud service provider must provide effective resource management for handling DDoS traffic.

3 Lightweight Resource Management

3.1 Overview

An implicit goal of our work is to minimize cost when serving resource to client. In order to do so, we must ensure that our scaling mechanism works effectively and efficiently. One option is to significantly minimize resource consumption by selecting the best replica locations and allocate a minimum number of resources to each replica. Therefore, our mechanism consists of two main phases: (1) Replica Location Activation; responsible for selecting the best replica placement among these potential locations. (2) Dynamic Scaling; phase when the system triggers an increase or a decrease in the amount of allocated resources in the server, using modified-threshold based scaling.

3.2 Replica Activation and Deactivation

We assume that replica servers are deployed in N predefined locations. In this setting, our objective is to choose M replicas ($M < N$) and activate selected replicas, such that congestion is minimized near the main server. Figure 1 shows the ideal result when we activate one replica ($M = 1$). The high load of attack traffic is redirected to the location furthest from the main server, thus relieving the load near the main server. Legitimate traffic can still access the main server.

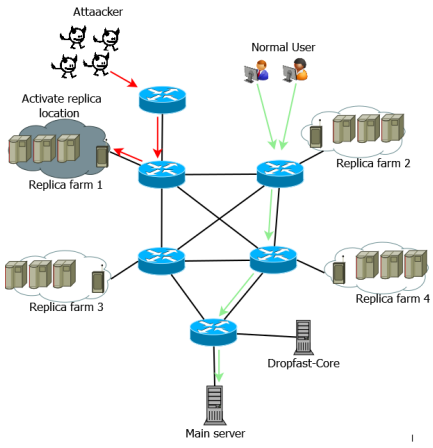


Fig. 1: Replica activation. After activating replica farm 1, traffic from attackers will be redirected from the main server to replica farm 1

Because our ultimate goal is to minimize congestion near the main server, we need to define a measure of congestion. For our purposes, we use the response time at the main server. The response time is a straightforward metric that represents a servers availability. A benefit of using this metric is that it can be measured by probing the server from the outside, and it does not require installing a probing agent at the server. The installation of additional agents in the server is often a sensitive issue, because such modifications may degrade the servers performance or lead to failure.

According to the objective and the measure of congestion, the optimal solution is to choose an M-subset of replicas with a minimal response time out of all possible M-subsets. However, identifying such a subset is not practical when N is large. The evaluation of all possible M-subsets can take a significant amount of time, and quick decisions are necessary when under DDoS attack. Therefore, we propose an approximation algorithm, a greedy method that performs M iterations and chooses one replica at each iteration. This greedy algorithm works as follows:

1. At the beginning of the iterations, none of the N replicas are activated.
2. At each iteration, we evaluate all of the replicas that are not yet activated. In particular, we compute two metrics: (i) the response time from each replica to the main server, and (ii) the hop count from the replica to the main server.
3. We choose the replica with the highest response time, assuming that the area around this selected replica is more congested than other areas. The response time is measured by sensor agents, which are deployed in every replica farm, and which send TCP SYN and simple HTTP requests to the main server on a regular basis.
4. In step (3), if more than one replica yields the same highest response time, we choose the replica with the highest hop count. Such a selection ensures that the attack traffic is redirected at the farthest possible location from the main server.
5. We activate the selected replica, and then repeat steps (2)-(4), until M replicas are activated.

To summarize, we activate one replica R at a time, selecting the replica with the highest response time and hop count. We expect that this will redirect the largest amount of traffic to R and that this redirection will occur at the location farthest from the server. One advantage of the proposed greedy algorithm is that it does not require the network topology information. Retrieving the latest topology information may not be practical, particularly when the network has thousands of nodes and the topology (and routing) changes frequently.

In the evaluation, we compare the proposed algorithm with two traditional methods that have been used for resource placement: (i) Random placement: randomly chooses M replicas among N potential sites ($M < N$) according to the uniform distribution, and it does not depend on traffic workload. (ii) Hotspot placement: This algorithm selects replicas near the clients that generate the highest load [6]. In particular, the algorithm sorts the N potential sites according to the amount of trafc generated within their area. The algorithm then selects the M sites that generate the largest amount of trafc.

In most cases, DDoS attacks only occur for a certain amount of time. To deactivate the replica location, we check the status of the attack using DDoS detection software/module, whether it is still under attack. The sensor agent then checks the response

time. If the response time is gradually returning to normal, there is no congestion, the traffic load returns to normal, and the server health check yields a normal status, the server is in a healthy condition. At this point, we deactivate the replica location. If there are many replica locations, we select the replica location where no congestion is detected or where there is the smallest response time. This purpose of the replica location deactivation is resource saving.

3.3 Scaling Condition

In this section we explain how our scaling mechanism works. Initially, we start with a low number of machines and gradually increase the load to determine the number of resources required. This approach helps us avoid a situation where we need to reduce the number of machines ($M-1$) at each iteration. The number of resources required must be estimated accurately so that they can be provisioned within the cloud infrastructure.

Normally, a simple threshold consists of two lines, the upper threshold and lower threshold. This simple threshold might not sufficiently capture traffic fluctuation and adapt to changes. Therefore, we use a four-line threshold-based mechanism. We use a modified threshold mechanism to dynamically change the resource allocation for certain threshold configurations. Using a threshold is a straightforward method to scale resources, and it is shown to be effective in any type of scenario. If the performance metric overpassed the invocation threshold, for a certain time, the system will perform invocation. Similarly, when the performance metric is under the revocation threshold, the system will automatically perform revocation. Figure 2 shows the threshold-based approach for invocation. Invocation is a process by which a system increases the resource allocation, whereas revocation decreases the current resource allocation.

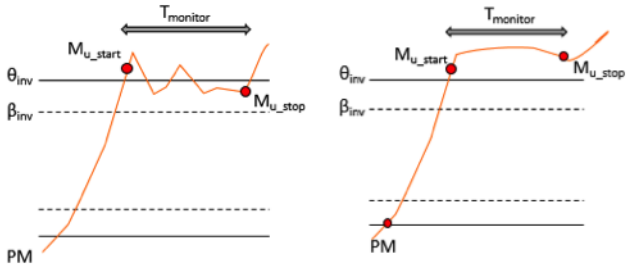


Fig. 2: Invocation condition

The four lines of threshold are: threshold invoke(θ_{inv}), bumper invoke(β_{inv}), bumper revoke(β_{rev}), and threshold revoke(θ_{rev}). We use four lines because a metric that goes beyond the original threshold might only stay momentarily above a certain point, and the additional threshold (bumper) can help us determine if the situation is lasting. For the first interval, if the performance metric value exceeds the invocation threshold, we start the timer to monitor the systems status. During this monitoring period, we collect

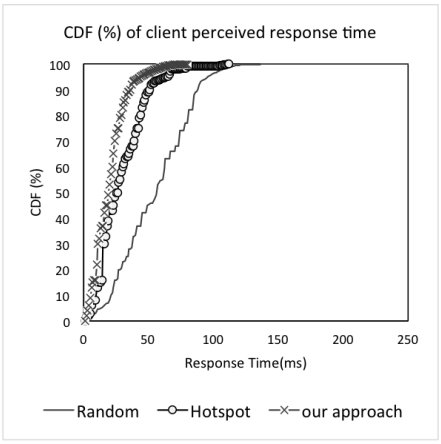
the performance metric value until the monitoring time ends. If the stop value is located above the bumper invocation threshold, there is a high probability that current resource allocation is stressed. If the last PM (M_{stop}) value is above βinv and below θinv, and the average PM value is also above βinv, we perform invocation with a one-by-one increase. If the average PM value is above θinv, and the $Mstop$ value is also above θinv, we conduct invocation with a two-fold increase. This is similar to TCPs slow start, we do 'x 2' to reach the top as soon as possible at the beginning of a DDoS. Once we are near the top, we do "+1" to gradually increase the number of replicas. In addition to the invocation condition, the revocation condition has the same principle for deciding the revocation step, but with a different direction value. The monitoring starts when the performance metric value goes deep below the θrev. This triggers collection of the performance metric value until the monitoring stops. We use response time as performance metric, as we found it sufficient to determine the degradation of a server's performance. Possible threshold and bumper values are 10% for threshold revocation, 25% for bumper revocation, 75% for bumper invocation, and 90% for threshold invocation. This percentage is proportionally scaled with range value from minimum to maximum for the performance metric.

4 Evaluation and Results

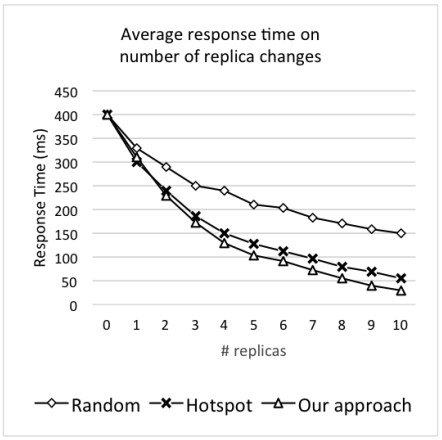
For our experiment, we use OMNET++, a simulation environment shown to be efficient for large networks [2]. We also use OverSim and HTTP component from INET module in OMNET++. To emulate Internet topology, we use GT-ITM and BRITE topology generator. We also derive the topology from BGP routing tables. We use the public Web trace of World Cup 1998 as input, which contains flash crowd traffic, a traffic pattern very similar to DDoS attack. The server and network was overloaded by a flash crowd event, and the aggregated volume resembles that of DDoS attack. In total, the web site received more than 1.35 billion requests during the collection period of three months (11,000 requests per minute on average), and almost 5TB of data was sent to clients (41 MB per minute on average). We vary the number of replicas from 1 to 4 for 10-node graphs, from 1 to 10 for 100-node graphs, and from 1 to 100 for 1,000-node graphs.

We ran the simulation several times to obtain a stable result, and these results resemble the normal condition. First, we analyzed the replica placement algorithm. We measured the response time from clients to replicas. As shown in Figure 3.a, which is generated by 100-node graphs, our mechanism achieves shorter response time than random and hot-spot placements. The results are similar in 10-node and 1,000-node graphs. Further, we measured the response time as we increased the number of replicas, starting from 1 to 10. Figure 3.b shows that our mechanism outperforms random and hot-spot placements, and the differences become greater as more replicas are used. This is because hotspot algorithm makes decisions solely based on traffic volume, whereas our mechanism considers multiple factors, (i) response time and (ii) furthest location based on the hop counts. We also measured the efficiency of our scaling mechanism. We assume that one replica can handle one million requests per hour and this may vary depending on the capacity of replicas. Figure 4 compares our approach with the sim-

ple threshold and shows that our mechanism much more closely follows the real traffic load, thereby eliminating the needs to invoke redundant replicas. Compared to the random and hotspot algorithms, our approach demonstrates the following benefits: it will only be active when there is an attack, and it is lightweight and easy to implement.



(a) CDF of client response time



(b) Average response time on number of replica changes

Fig. 3: Result of comparison between random, hotspot and our approach

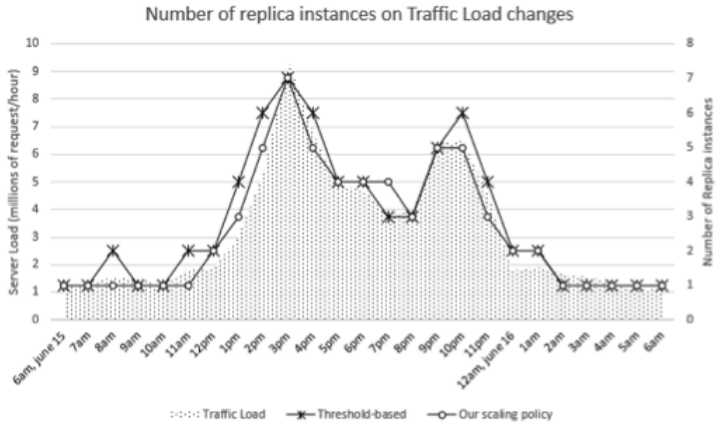


Fig. 4: Replica instances on traffic load changes

5 Conclusion

A lightweight scaling mechanism to guarantee service during a DDoS attack. We evaluate a simple and effective mechanism for resource scaling allocation management for DDoS traffic isolation in a cloud environment. Our work has demonstrated the compelling benefit of the cloud, which can handle high traffic and scale the service dynamically. This research motivation is not to entirely eliminate DDoS attacks, but to provide continuity of a service during a DDoS attack. Other topics, such as a pricing scheme for this mechanism, can be calculated fairly by the cloud service provider as a new type of service, despite the actual resource usage. Inconsistency and synchronization issues procedure would be another problem we should consider be to apply the resource prediction mechanism for dynamic resource scaling.

Acknowledgments. This research was supported by the R&D Support Center of the Seoul Development Institute and the South Korean government (WR080951).

References

1. ArborNetworks: Fingerprint sharing alliance (2012), http://www.arbornetworks.com/careers/53-products/4418-fingerprint-sharing-alliance-overview
2. Bless, R.: Using realistic internet topology data for large scale network simulations in omnet++. Tech. rep., University (2002)
3. Cloudflare: An overview of cloudflare (2012), https://www.cloudflare.com/overview
4. Muttik, I., Barton, C.: Cloud security technologies. Inf. Sec. Techn. Report 14(1), 1–6 (2009)
5. Nygren, E., Sitaraman, R.K., Sun, J.: The akamai network: a platform for high-performance internet applications. Operating Systems Review 44(3), 2–19 (2010)
6. Qiu, L., Padmanabhan, V.N., Voelker, G.M.: On the placement of web server replicas. In: INFOCOM. pp. 1587–1596 (2001)
7. Rashad, A., Dongwon, S., Heejo, L.: Dropfast: Defending against ddos attacks using cloud technology. In: International Conference on Security and Management (2013)
8. Xie, M., Widjaja, I., Wang, H.: Enhancing cache robustness for content-centric networking. In: INFOCOM. pp. 2426–2434 (2012)