Mitigating Denial-of-Service Attacks in Contested Network Environments

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Quan Jia
Master of Science
George Mason University, 2012
Bachelor of Science
South China University of Technology, 2006

Director: Angelos Stavrou, Professor
Department of Computer Science

Spring Semester 2014
George Mason University
Fairfax, VA

Dedication

To Xun and our beloved daughter.

Acknowledgments

This dissertation would not be possible without the help and support of many people, to whom I am greatly indebted.

First and foremost, I would like to express my heartfelt gratitude to my advisor, Angelos Stavrou, for his continuous guidance and support. Thanks to Angelos, I was given the opportunity to experience different research projects covering various aspects of information security and computer science. His advice and encouragement has not only been a great help to my graduate studies but will also remain invaluable for my future career. Additionally, I would like to thank other members of my dissertation committee, including Fei Li, Duminda Wijesekera, Brian Mark, and Damon McCoy, for their insightful comments and precious time in helping me improve this work.

My sincere thanks also go to Kun Sun, Dan Fleck, and other faculty members with whom I was fortunate to collaborate on many projects and research papers. Their knowledge, inspiration, and commitment have always made our working together a pleasure.

I am deeply thankful to my colleagues and friends Meixing Le, Fengwei Zhang, Sharath Hiremagalore, Jiang Wang, Zhaohui Wang, Rahul Murmuria, and Chaitanya Yavvari, for their unselfish input and help throughout my PhD research.

Last but not least, I would like to thank my lovely wife, Xun. The journey in pursuit of a PhD is a long and difficult one, and I would not have survived it without her constant support and encouragement. She has always been there for me at hard times, even when we were separated by oceans.

Table of Contents

List of Tables

List of Figures

Abstract

MITIGATING DENIAL-OF-SERVICE ATTACKS IN CONTESTED NETWORK ENVIRONMENTS

Quan Jia, PhD

George Mason University, 2014

Dissertation Director: Angelos Stavrou

In an increasingly connected world, ensuring availability of the services and applications carried by computer networks is critical. As the most far-reaching computer network, the Internet is home to millions of services that have profound influences on people's lives and work. Additionally, mobile ad hoc networks (MANETs), often used to support critical military and civilian projects, rely on the availability of all participating nodes to fulfill their missions. However, the availability of these networks and services are constantly threatened by denial-of-service (DoS) attacks with growing intensity and sophistication. In this thesis, we study different DoS combating mechanisms to protect services and hosts in the contested Internet and MANET environments.

To mitigate distributed denial-of-service (DDoS) attacks bombarded by powerful botnets on the Internet, we propose a moving target mechanism that progressively separates benign clients from the mingled attackers. This mechanism is achieved by endowing mobility to the defense system to evade naive attackers while smartly shuffling clients to quarantine advanced and persistent attackers. We present two mobile defense architectures tailored for different threat and application models. The first architecture, named MOTAG, is built on secret moving network proxies that act as the intermediate layer between authenticated

clients and protected services. By only disclosing the active proxies to the authenticated clients and quickly replacing the attacked ones, this intermediate layer becomes a moving target that continues to escape from network flooding attacks. The second architecture, enabled by the resource elasticity and network space of cloud computing, replicates open web servers to partition incoming workloads of various clients. By dynamically instantiating new replica servers scattered in the cloud and re-shuffling clients' assignments, we are able to quarantine flooding attacks targeting both network and computational resources. Under both architectures, advanced attackers may follow the moving targets to persist their attacks. To isolate the following attackers from benign clients, we perform elaborate shuffling operations that intelligently redistribute clients among different proxies or replica servers. For guiding the shuffling operations, we design an optimal dynamic programming algorithm that expectedly saves the maximum number of benign clients from each shuffle. We also introduce a much faster greedy algorithm that can generate near-optimal shuffling plans in realtime. Furthermore, a maximum-likelihood algorithm is employed to accurately estimate the number of following attackers. Results from extensive simulations show that our defense mechanism can save a vast majority of benign clients from persistent and intense DDoS attacks in a few rounds of shuffling. Experiments that study the overhead of the shuffling operation demonstrate that clients can be re-assigned among different proxies or replica servers in several seconds.

In addition, this thesis introduces a capability-based mechanism that inhibits MANET DoS attacks in the context of multi-path routing. Existing solutions are limited because they assume that a single path is used to route traffic for each flow. To prevent attacks that multiply their throughput by employing multiple different routes, we present CapMan, an enhanced capability-based mechanism that enforces per-flow limits across all employed routes. To ensure overall capability compliance, CapMan not only informs all intermediate nodes of a flow about the assigned capability but also provides them with a global throughput perspective via periodical summary exchange. Results show that CapMan is able to maintain flow-wide capability constraints consistently and distributedly, even when multiple colluding insiders attempt to exacerbate the attack. In the meantime, the impact

of CapMan on well-behaved flows is shown to be small.

Chapter 1: Introduction

1.1 Motivation

The emergence, evolution, and propagation of network technologies have revolutionized the world. To date, with the proliferation of smart mobile devices and applications, people's lives and work have become more dependent on data networks than ever. Undoubtedly, billions of people around the globe are using the Internet as their primary means to communicate, share information, and conduct business. More and more people are relying on cellular networks for ubiquitous connections to the Internet and to one another. Wireless ad hoc networks are quickly gaining popularity in vehicle-to-vehicle communications, military operations, and disaster rescues. Given the increasingly critical roles that these networks are playing, it is crucial to ensure their availability to the growing number of subscribers.

Over the past decade, denial-of-service (DoS) attacks have become the top threat to network and service availability. By definition, a DoS attack is an attempt by one or more machines to prevent the targeted victim from doing useful work [1]. A typical DoS attack aims to make networks and servers unavailable to their intended users. Most commonly, attackers flood the target with excessive network traffic or service requests to exhaust the victim's limited network bandwidth or computational resources (CPU, memory, etc.). To inflict profound damage and avoid being blocked, a DoS attack on the Internet is often performed by numerous geographically distributed attacker-controlled zombies and are thus addressed as distributed DoS (DDoS) attacks.

1.1.1 DoS Attacks On The Internet

DDoS attacks are one of the most common and potent security threats to Internet services. A recent survey by Arbor Networks revealed both increasing DDoS attack intensity and

sophistication over the past decade [2]. The same survey also showed that large-scale DDoS attacks have gained significant prevalence. In 2010, the highest throughput achieved by a flooding DDoS attack reportedly reached 100 Gbps. In contrast, the cost to perform a DDoS attack has proven to be surprisingly low. A Trend Micro white paper [3] about Russian underground market disclosed that one-week DDoS service costs as little as $150. Lately, ideology has become a new major driving force for DDoS attacks. The most well-known incidents are the series of retaliation attacks initiated by WikiLeaks' supporters to paralyze the websites of MasterCard, Visa, and others [4]. The observed upward trend of DDoS attacks can cripple the operation of critical infrastructures and online services, disrupting both civilian and military facilities.

A DDoS attack is usually bombarded by a botnet composed of large numbers of infected machines (i.e. bots) [5]. The actual attack can take many forms [6]. For example, TCP SYN flooding can be used to deplete the victim's available connections [7, 8]; UDP flooding [9], smurf attacks [10], and DNS reflector attacks [11] are able to saturate the victim's network bandwidth; and application-level flooding [6] may drain the victim's computation power. Although on the surface, an attack may exhibit different characteristics, internally the actions of the entire botnet are coordinated through a common command and control (C & C) channel [12].

DDoS attacks are hard to prevent not only because of their multitude and distributed nature but also due to the inherent designs of the Internet. Unlike circuit-switched networks in which every end-to-end communication is allocated with an exclusive channel, the Internet is based on packet switching where link resources are shared among many users [13]. A flooding DDoS attack can disturb or even shut out legitimate users by preempting as many shared resources as possible [14]. In addition, the Internet was designed following the end-to-end principle [15], which advocates building a simple network core that does little more than packet transmission, while having sophisticated end systems to ensure reliability, quality of service (QoS), and security. Undeniably, the end-to-end principle provides tremendous convenience for developing and deploying new applications, which contributed

greatly to the unprecedented success of today's Internet. However, following this principle also resulted in a lack of security and controlling mechanisms on the network intermediaries, making the Internet inherently vulnerable to distributed flooding attacks converging on end systems [16].

To make up for the vulnerabilities introduced by the Internet design principles, researchers proposed adding security functionalities, including filtering-based [17–19] and capability-based mechanisms [20–23], to the Internet core and edge routers to stop attacks from their sources. However, implementing these solutions calls for a universal upgrade on the legacy Internet infrastructures and collaborative efforts among different Internet service providers (ISPs). To date, none of them have received wide deployment due to the lack of incentives. Overlay-based defenses [24–30] are considered to be more feasible alternatives to address DDoS attacks because they enable unilateral and incremental deployment. Nevertheless, to maintain resilience against strong flooding attacks, an overlay network must own abundant resources to withstand and absorb the attack traffic, which inevitably drags overlay providers into an arms race with the ever-expanding botnets.

Among the deployed DDoS defense mechanisms, content delivery networks (CDNs) such as Akamai [31] and Limelight [32] are employed to increase the availability of bulk data services through widespread data replication. This has proven to be a viable and effective approach to protect web services involving only stateless transactions. However, plenty of other Internet services are either stateful or hosting data too sensitive to be vastly replicated, which calls for effective protection on centralized servers. As of today, practical DDoS mitigation for centralized Internet services still primarily depends on packet filtering using access control lists (ACLs), despite their functional and operational limitations [2].

1.1.2 DoS Attacks In Mobile Ad Hoc Networks (MANETs)

Unlike the Internet that lives on fixed network infrastructures, a MANET is a dynamic network that connects wireless mobile equipment without the support of established infrastructure. In MANETs, a wide range of radio-equipped devices, including laptops, smart

phones, and sensors, act as end-nodes and routers at the same time to enable multi-hop communications. Compared to wireline networks, MANETs are characterized by dynamic topology, constrained resources, and weakened physical security [33].

MANETs are exposed to a wider variety of DoS attacks. For example, signal jamming attacks can obstruct the victim's wireless radio communication [34]; route fabrication, modification, and impersonation attacks can disturb route discovery and maintenance, leading to sub-optimal and potentially incorrect routing decisions, or even network wormholes short-circuiting the flow of routing packets [35–37]; flooding attacks across different protocol layers may drain the victim's limited computational resources and restrained battery power [33].

MANETs are vulnerable to DoS attacks for several reasons. First, a MANET is composed of mobile devices that lead to dynamic topology, making it difficult to define a clear line of defense that separates trusted and untrusted nodes. Second, since no fixed infrastructure or centralized administration is available, a MANET user must rely on untrusted peers for packet delivery and feedback. Third, a MANET uses open air for radio communication, which is susceptible to physical interference and channel congestion. Last but not least, compared to Internet servers and desktops, wireless mobile devices are equipped with constrained CPU, memory, and especially battery power, all of which are subject to exhaustion attacks.

Many previous studies [36–44] endeavored to secure MANET routing and data forwarding, but flooding attacks on end nodes have not drawn enough attention. To protect end nodes against flooding, reactive mechanisms, including decentralized intrusion detection [45], distributed filtering [46] and attacker tracing [47], were proposed to detect and block flooding attacks. However, more proactive approaches are needed to preserve MANETs' limited bandwidth and energy. To that end, capability-based solutions were introduced to stop flooding DoS attacks by giving "control over resource usage to the owner of the limited resource" [20, 48–51]. In an end-to-end communication, a capability token refers to a ticket issued by the responder to the initiator, specifying the permitted traffic

volume within a pre-defined window of time. Intermediate nodes between the initiator and responder are responsible for performing traffic policing according to the given capability. Unfortunately, current capability-based mechanisms [48–51] are inadequate in that they have failed to account for multi-path routing that can be exploited by attackers to increase the throughput of flooding attacks.

1.2 Thesis Contributions

This thesis aims to combat flooding DoS attacks in both the Internet and MANET environments. To protect Internet services from flooding DDoS attacks, we propose a moving target defense mechanism to achieve progressive attack segregation [52–54]. In addition, to secure MANET communications from network DoS attacks, we present a capability-based defense to defeat flooding attacks that exploit multi-path routing to increase attack throughput [55, 56]. In summary, this thesis makes the following list of contributions:

- Created a proof-of-concept shuffling-based DDoS mitigation mechanism that realizes the moving target defense strategy [57]. The goal of the mechanism is to separate benign clients from attackers that inflict persistent flooding attacks. To progress toward attack isolation, the proposed solution dynamically brings in new proxies or servers into service, and intelligently re-assigns (i.e. shuffles) clients from the attacked nodes to the new nodes, as determined by the designed shuffling algorithms.

- Created an architecture to enable the shuffling-based defense on protecting critical Internet services mandating client authentication against network flooding attacks. This architecture employs dynamic network proxies as moving targets to relay traffic to and from the protected server. During a DDoS attack, the overwhelmed proxies are quickly replaced by new proxies at different network locations. Clients are re-directed as indicated by the shuffling mechanism.

- Created an architecture to enable the shuffling-based defense on protecting open Internet services hosting anonymous users against both network and computational

flooding attacks. This architecture leverages the scale and resource elasticity offered by cloud computing to realize server replication and mobility. Our in-cloud replica servers are turned into moving targets during DDoS attacks. When under attack, new replica servers are dynamically instantiated to replace the ones being bombarded. Clients are re-directed as indicated by the shuffling mechanism.

- Designed an optimal dynamic programming algorithm that expectedly separates the maximum number of benign clients possible from attackers in each round of shuffling. The algorithm was implemented in MATLAB [58] and evaluated via simulations using a wide array of parameters.

- Designed a faster greedy algorithm that can produce near-optimal shuffling plans. The greedy algorithm was compared with the dynamic programming algorithm analytically and through MATLAB based simulations. The results show that the greedy algorithm is almost as effective as the dynamic programming algorithm, but it reduces the time complexity by several orders of magnitude.

- Designed a maximum-likelihood algorithm that accurately estimates the number of sophisticated and persistent attackers that follow the moving targets constituted by our defense mechanism. The result of the maximum-likelihood estimation (MLE) algorithm is fed to the dynamic programming algorithm and the greedy algorithm as a key parameter in optimizing the shuffling plans. The accuracy and limitations of the MLE algorithm were thoroughly evaluated. A remedy was studied to resolve the limitations of the algorithm.

- Evaluated the integration of the algorithms against simulated DDoS attacks. The results show that the proposed mechanism can effectively mitigate intensive DDoS attacks in a few shuffles by separating a vast majority of benign clients from attackers.

- Implemented proof-of-concept prototypes of the two proposed architectures to evaluate the overhead of client redirection operations. By implementing both proxies and

clients in PlanetLab [59], we demonstrate that during a shuffling operation, a client can be re-directed between two proxies in less than one second. Using servers based in Amazon EC2 [60] and clients resided in PlanetLab, our experiments show that re-assigning clients to different replica servers can be accomplished in a few seconds at worst.

- Created the first capability-based mechanism to mitigate DoS attacks on multi-path MANET communications. The proposed solution empowers the responder of an end-to-end flow to issue and distribute a capability that regulates per-flow throughput within a specified window of time. To perform traffic policing across all employed routes, intermediate nodes on different routes will exchange bandwidth consumption reports periodically to track the flow-wide throughput, making sure the flow capability is universally enforced.

- Evaluated the capability-based mechanism both analytically and experimentally using a proof-of-concept prototype implemented in NS2 [61]. The results show that the proposed solution can effectively identify and limit the impact of a network flooding attack with a relatively small performance penalty.

1.3 Thesis Organization

In the rest of the thesis, we first provide an overview of the related work on combating flooding DDoS attacks in Chapter 2. After that, Chapter 3 describes the strategy, architectures, and algorithms of the shuffling-based moving target defense against Internet DDoS attacks. Chapter 4 introduces the capability-based mechanism to stop MANET DoS attacks exploiting multi-path routing communications. Chapter 5 concludes this thesis and discusses directions for future research.

Chapter 2: Background and Related Work

This chapter reviews the background and related research on combatting flooding DDoS attacks in the Internet and MANET environments. Much of the previous work on mitigating DDoS attacks across different network contexts reflects similarities in their concepts and design.

2.1 Defenses Against Internet DDoS Attacks

The Internet was originally designed for openness and scalability rather than security and accountability. Given the size and structural complexity of today's Internet, large botnets can easily initiate a DDoS attack to bombard any resource-limited target. Since the high-profile DDoS incident in 2000 that shut down several of the most visited websites including Yahoo, eBay, CNN, Amazon, E*Trade, and ZDnet [62], DDoS attacks have become one of the severest threats to Internet services. A recent CSI survey [63] reported that 16.8% of the respondents had fallen victim to a DDoS attack in 2010.

Combating Internet DDoS Attacks has been an active area of research over the past decade, resulting in a wide variety of solutions [64–66]. Many of the proposed mechanisms fall into three major classes, namely filtering-based defenses, capability-based defenses, and overlay-based defenses. The following subsections provide a comprehensive review on the existing proposals that reside in and out of these classes.

2.1.1 Filtering-based Defense

To stop DDoS attack traffic close to the source(s), filtering-based defenses [17–19] intend to have ubiquitous packet filters deployed on Internet core and edge routers. Depending on the specific protocols, the missions of the widely deployed filters are to block IP packets

with spoofed addresses, hosts emitting excessive traffic, or flows deemed as malicious by their receivers.

To thwart flooding attacks using packets with spoofed source addresses, ingress filtering [18] proposed to validate the source address of each packet at its network entry point. Ingress/Egress routers should only allow incoming/outgoing packets bearing source IPs within the expected range. Internet service providers (ISPs) are in the best position to implement ingress filtering because each device can be bound to a particular IP address as soon as it connects to the internet. Although ingress filtering empowers individual ISPs to regulate Internet users and prevent network misuse, this mechanism has not received wide enough adoption to eradicate IP spoofing. A recent study [67] showed that more than 30% of the tested Internet users can still spoof arbitrary IP addresses, and more than 75% of the rest can forge IP addresses within their subnet.

By building a large botnet comprising a huge collection of compromised hosts, attackers may be able to perform flooding DDoS attacks without IP spoofing. To mitigate flooding attacks with or without IP spoofing, Pushback [17] proposed to rate-limit inordinate senders via aggregate-based congestion control. An aggregate is a collection of packets sharing some common properties, such as address prefix, application, or packet type. Pushback argued that attack traffic can be grouped into different aggregates from legitimate traffic. Internet routers are expected to identify and rate-limit all attacking aggregates in a collaborative manner. However, differentiating well-masked attack traffic from legitimate traffic is a difficult job that requires deep packet inspection, costing precious CPU cycles from the routers. Moreover, given the lack of incentives, it is difficult to foster the collaboration among different ISPs on attack blocking.

Compared to Internet routers that can only understand up to the IP layer, the destination node of an end-to-end flow can spot a misbehaving source node more accurately. Therefore, Stopit [19] aimed to have Stopit servers on the destination side collect complaints about malicious source nodes and instruct Stopit servers on the source end to block malicious flows locally. Each Stopit server manages one autonomous system (AS) that is

considered a fate-sharing unit. Passport [68] is employed for AS-level source validation to prevent IP spoofing. Traceback technologies [27, 69–74] can be used to locate the malicious sources flagged by a destination node. Like Pushback, Stopit needs to be widely deployed on the Internet to be effective, but this is unlikely if no strong incentives are provided.

To conclude, filtering-based defenses assume that attack traffic, regardless of whether IP spoofing is used, are distinguishable from legitimate traffic. This is a strong assumption that is subject to dispute, given that sophisticated attackers may hide their intent by carefully crafting their strategy and packets to mimic the behavior of legitimate clients. In addition, to be adequately effective against DDoS attacks, packet filters have to reach a high degree of deployment to cover a broad range of ASes. The lack of incentives to encourage early adopters remains a significant and unsolved problem.

2.1.2 Capability-based Defense

To address the attacker identification problem encountered by filtering-based approaches, capability-based defenses [20–23, 75] let the destination node decide whether a particular source node is allowed to send packets to the destination. In addition, the destination node also sets the limit on the volume of traffic that can be sent by a flow within a specified time window. In essence, capability-based mechanisms give the control of resource usage to the owner of the resources. Source nodes that violate the assigned capability will be considered malicious.

The early research work that advocated capability-based defenses include [20] and [21]. Under these mechanisms, an Internet host that wants to send packets to another host must first request the receiver's approval. The source of a request to send is attested by the forwarding routers. The approval from the destination node is issued in the form of a flow-bound capability token that also defines the maximum throughput allowed. To be compatible with legacy communication protocols while encouraging early adopters, flows assigned with capabilities are treated preferentially during a DDoS attack. Meanwhile, a capability serves as the gauge for regulating the approved communication flow. The routers

between the source and destination are responsible for performing traffic policing according to the given capability.

Although only a small portion of the overall bandwidth is expected to carry capability requests under normal conditions, some early solutions are vulnerable to denial of capability (DoC) attacks [76]. Instead of targeting the data channels, DoC attacks simply flood the capability request channel to deny access to legitimate communications. To prevent DoC attacks, TVA [22] was introduced to ensure per-path fairness among all capability requests. Within TVA, the edge routers of trusted ASes will mark each capability request packet to identify the routing path along which the request is forwarded. A hierarchical fair-queueing system was proposed to make sure requests routed via different paths are fairly processed. On the other hand, Portcullis [75] used a proof-of-work (PoW) solution that guarantees per-computation fairness. Before capability requests from a particular source can be delivered to the destination, the source node must solve crypto-puzzles disseminated by well-provisioned services (e.g. DNS) and attach the answers to the requests. Participating forwarding routers are responsible for verifying the answers and dropping all invalid requests. Enforcing PoW will reduce attackers' traffic rate and increase the chances that legitimated capability requests are going to reach the desired destinations.

Like filtering-based methods, capability-based mechanisms also rely on high degrees of adoption to provide adequate protection. To implement capability-based solutions, new primitives, including packet marking and cryptographic operations, will need to be added to Internet routers. As in the case of filtering-based mechanisms, the need for such a large-scale and expensive infrastructural upgrade is inevitably hampered by the lack of incentives. Besides, incorporating capability in end-to-end communications also entails changes to client software, which will introduce compatibility issues with the already deployed Internet applications.

2.1.3 Overlay-based Defense

To eliminate network administrative boundaries and enable incremental deployment, secure overlay networks were proposed to provide authentication, filtering, indirection, attack tracking and tolerance [24–29] on top of the physical Internet infrastructure. In general, these solutions rely on a well-provisioned, distributed overlay network on the front end to diffuse and absorb attack traffic, shielding the hosts on the back end.

Among the early overlay solutions, SOS [24] introduced a three-tier network design with doubly indirect routing to assure DDoS resiliency. A small portion of all overlay nodes are employed to form each tier. Incoming flows are forwarded to the beacon nodes after successful authentication, and then routed to the secret servlets that eventually lead to the protected server. Overlay nodes can join and exit the network at runtime without affecting the overall stability. SOS is robust against pure flooding attacks in that attackers have to bombard a large number of overlay nodes to cause a service disruption. However, SOS is susceptible to sweeping attacks that target a subset of the overlay nodes at one time and gradually sweep through the entire overlay network. It is also insufficient to protect individual clients that are followed by attackers. To address the limitations of SOS, the spread-spectrum method [25] was proposed to encourage clients sending duplicated packets to multiple overlay nodes to ensure a high packet delivery rate during an attack. Similar to SOS, Phalanx [29] recommended using a swarm of proxy nodes to serve as mailboxes that bounce end-to-end traffic in the middle of the Internet. The destination hosts are expected to actively contact the mailboxes to pick up legitimate packets, leaving attack traffic to be filtered out. To generalize the discussion on secure overlay networks, Mayday [26] studied different overlay architectures and design options using an extended threat model. The same paper also presented a series of lightweight authentication schemes to help identify and screen out compromised overlay nodes.

TOR [30] is a well-known implementation of overlay network. It can offer anonymity to both the source and the destination of an end-to-end communication to mitigate DDoS attacks. With the help of encrypted onion routing, the source and destination can negotiate

a secret rendezvous point to exchange data packets. Unfortunately, insider attacks on TOR can potentially expose the hidden services to external attackers [77].

The composition of an overlay network is often published and relatively static. Therefore, the overlay itself may become a target of a flooding DDoS attack. Current overlay-based defenses assume that attackers are incapable of overwhelming the entire overlay network. This assumption inevitably drags overlay network providers into an arms race with attackers, forcing constant investments on the overlay infrastructure to match the ever-growing strength of botnets.

2.1.4 Other Defense Mechanisms

Some more recent studies proposed to separate the connection setup channel from the data transfer channel [78, 79]. WRAPS [79] takes advantage of the referral relationship between mutually trusted websites to provide privileged access for legitimate clients during an attack. As client authentication is done at multiple referrer sites, WRAPS relies on the aggregate capacity of all referral websites to resist flooding attacks. Similar to the overlay-based schemes, Epiphany [78] uses proxies to relay the communication between clients and servers. The difference is that Epiphany employs different sets of proxies for connection setup and data communication. Only setup proxies are published and addressable via anycast. Hidden data proxies are dynamically recruited to build up ephemeral paths that lead to the protected service.

2.2 Defenses Against MANET DoS Attacks

Compared to the Internet, mobile ad hoc networks are vulnerable to a wider range of DoS attacks. The focus of this thesis is on network flooding attacks that aim to deplete the limited bandwidth, computational resources, and battery power of the targeted node(s). To serve that goal, this section reviews the related work on detecting and resolving network flooding attacks in the MANET environment.

Wireless communication comes with limited physical security. Therefore, cryptographic

methods are usually used to implement node authentication, channel encryption, and packet validation. For example, to combat flooding DoS attacks performed by external attackers, [80] recommended that every packet should be digitally signed by the sender and verified by all forwarders. HEAP [81] proposed a hop-by-hop packet authentication method using keyed-hash message authentication code (HMAC) [82]. Asymmetric key based cryptography was used as the cornerstone to build various security mechanisms [83]. Given the wide application of cryptographic mechanisms, key management becomes a fundamental security issue in MANETs. However, due to the dynamic and autonomous nature of MANETs, it is unrealistic to assume a single static central authority that issues and maintains all public keys. To address that, [84–86] employed distributed and mobile certificate authorities (CAs) to provide reliable and tamper-proof key issuance and authentication.

Crypto-based solutions can effectively protect MANETs from malicious outsiders, but they do not offer resilience against attacks exerted by malicious insiders with valid cryptographic credentials. To detect an insider attack and identify the misbehaving node(s) behind it, intrusion detection systems (IDS) need to be deployed to trigger appropriate response mechanisms. Due to the lack of fixed vantage points for traffic monitoring, existing Internet IDS solutions are not directly applicable to MANETs. To realize global anomaly detection, Zhang et al. [87] designed a distributed IDS architecture that allows collaborative decision making in wireless ad hoc networks. In this solution, an independent IDS module is installed on every node to monitor local traffic while information gathered is also exchanged among nodes to enable broader investigations on suspicious events. To encourage node cooperation while penalizing selfish and misbehaving nodes, researchers have proposed token-based [88], credit-based [89], and reputation-based [90] schemes to record the historical behavior of MANET nodes and publicize their credibility.

As discussed in Section 2.1.2, capability-based defenses were originally designed to protect Internet hosts from DDoS attacks. Recently, Alicherry et al. introduced a similar mechanism [48–50] to address flooding attacks in MANETs. With the proposed solution,

data traffic are denied by default. However, trusted nodes can request for network capabilities that allow them to send packets to the desired nodes. A network capability is a policy token that encodes both access control rules and traffic-shaping parameters, which can be verified by every intermediate node. Although using capability can preserve resources by preventing unauthorized access, this solution is limited by its implicit assumption that a single static route is used by each authorized flow, which is unrealistic in most cases considering the dynamic nature of MANET topologies. In other words, the compliance to a per-flow capability cannot be guaranteed if multiple routing paths are employed. Attackers can swamp a target node by spreading the attack traffic to as many routes as possible and approach (while staying under) the throughput limit along each path. To counter such attacks, this thesis studies an improved capability-based mechanism that enables cross-path collaboration to enforce flow-wide capabilities under the context of network mobility and multi-path routing.

Chapter 3: Moving Target Defenses Against Internet DDoS Attacks

3.1 Introduction

This chapter describes a family of dynamic defense mechanisms and architectures that aims to protect Internet services from flooding DDoS attacks. The proposed solutions implement a common moving target strategy that is designed to quarantine the impact of a DDoS attack. To achieve that, we dynamically replace the nodes that are bombarded by the attack and intelligently re-assign clients from the attacked nodes to the new nodes. We call this procedure a shuffling operation. The purpose of the shuffling operation is to separate benign clients from DDoS attackers by casting them onto different nodes. Through multiple rounds of shuffling, our ultimate goal is to separate out and restore the quality of service (QoS) for as many affected benign clients as possible, thereby achieving attack isolation.

The basic idea of the shuffling operation can be illustrated by the simple example in Figure 3.1. In this example, there are initially seven clients (including attackers, denoted by $C_1, \ldots, C_7$) and three serving nodes (referring to proxies in Section 3.2 and replica servers in Section 3.3). C_3 and C_5 are attackers. First, clients are randomly assigned to all serving nodes as suggested by the dotted lines: clients C_1, C_2, and C_3 are assigned to node K_1; C_4 and C_5 are assigned to node K_2; and C_6 and C_7 are assigned to node K_3. Attackers (C_3 and C_5) will bring an attack to K_1 and K_2. However, since defenders cannot tell who the real attackers are, clients C_{1-5} have to be considered as equally suspicious. To separate affected benign clients from attackers, our defense mechanism will react by replacing K_1 and K_2 with new serving nodes K_4 and K_5. Benign clients and attackers previously connecting to K_1 and K_2 are re-assigned to K_4 and K_5, respectively. One possible way of re-appointing is to send C_1, C_3, and C_5 to K_4 while matching C_2 and C_4 with K_5, as indicated by the

solid lines. As a result, K_5 will be saved, but K_4 will remain under attack. The bindings between C_6, C_7, and K_3 stay unchanged because they are not affected by the attack. In this way, we can save C_2 and C_4 from an on-going attack.

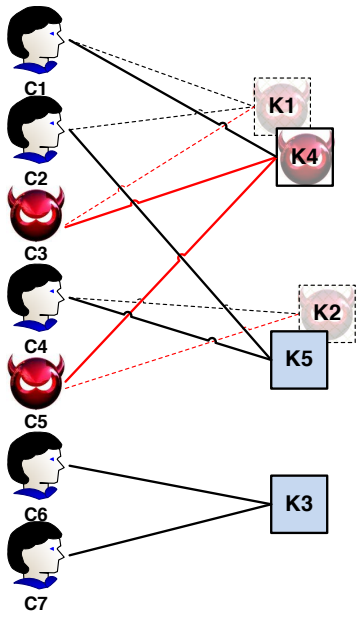


Figure 3.1: Client Shuffling

Our shuffling mechanism realizes moving target defense (MTD) against DDoS attacks. In contrast to most traditional security measures that are static in nature, MTD advocates for controlled changes on network, software, and other system dimensions to increase uncertainty and complexity for attackers [57]. Several existing security solutions have incorporated the idea of MTD. For example, network address space randomization [91] dynamically changes the IP addresses of Internet hosts to hamper the propagation of hit-list worms. Fast flux technique [92] employs a fast-changing proxy network of compromised machines to evade lawful blocking and increase the availability of illegal websites. On mitigating DDoS attacks, MOVE [93] migrates the attacked web server to a new network location. However, MOVE does not account for attacks targeting computational resources (CPU, memory, etc.)

or extended attacks inflicted by advanced attackers that continue to follow the migrated server.

This thesis introduces new architectures that create different types of moving targets to address both network and computational flooding attacks. In the meantime, novel shuffling mechanisms are proposed to segregate persistent and persistent attackers from benign clients. In this chapter, Section 3.2 presents a packet indirection architecture that employs moving proxies to protect critical services mandating client authentication against network attacks. Section 3.3 describes a cloud-enabled architecture that uses moving replica servers to mitigate both network and computational attacks on open services hosting anonymous users. Section 3.4 explains the methodology and algorithms adopted by the shuffling mechanisms to achieve attack isolation.

3.2 Protecting Critical Services Using Moving Proxies

This section introduces MOTAG, a dynamic DDoS defense architecture that adopts moving target strategy to protect centralized online services. In particular, MOTAG offers DDoS resilience for authorized and authenticated clients of security-sensitive services such as online banking and e-finance. MOTAG employs a layer of secret moving proxies to mediate all communications between clients and the protected application server. The network-level filters surrounding the application server only allow traffic from the valid proxy nodes to reach the protected server.

Proxy nodes in MOTAG have two important characteristics. First, all proxy nodes are *secret* in that their IP addresses are concealed from the general public and are exclusively known by legitimate clients after successful authentication. Each legitimate client is provided with the IP address of one working proxy at any given time to avoid unnecessary information leakage. Existing proof-of-work (PoW) schemes [75, 94–96] are applied to protect the client authentication channel. Second, proxy nodes are *moving*. As soon as an active proxy node is attacked, it is replaced by another node at a different location, and the associated clients are migrated to alternative proxies. These unique characteristics of

MOTAG not only enable us to mitigate brute-force DDoS attacks but also empower us to discover and isolate malicious insiders that divulge the location of secret proxies to external attackers. We do so by smartly shuffling (or repositioning) clients' assignment to new proxy nodes when their original proxies are under attack. The details of the shuffling mechanism will be discussed in Section 3.4.

3.2.1 Threat Model and Assumptions

The MOTAG architecture is designed to protect security-sensitive online services against *network flooding attacks*. The clients of the protected services are pre-authorized and authenticated before they are served.

We assume powerful attackers with high aggregate bandwidth that are capable of overwhelming any standalone machines on the Internet. However, we do not assume attackers that can saturate well-provisioned Internet backbone links for ISPs, data centers, or cloud service providers. Attackers, in case of uncertainty, can first perform a reconnaissance attack (e.g., IP and port scanning) to pinpoint the targets for the subsequent flooding attack.

With the knowledge of the MOTAG architecture, attackers can re-target their attack to flood the authentication channel through which legitimate clients are admitted. However, we assume that it is significantly harder for attackers to pass strong authentication and reach the proxies the way legitimate clients do. To uncover the network locations of our secret moving proxies, persistent attackers may play as insiders by compromising authorized machines or eavesdropping on legitimate clients' network connections. Given the technical difficulty of such advanced attacks, we assume the number of malicious insiders residing in a protected system is limited.

3.2.2 Architecture

Figure 3.2 shows the MOTAG architecture, which consists of four interconnected components: the authentication server, the proxies, the filter ring, and the application server. The application server provides the online services (e.g., banking or e-finance services) to

be protected and made available to authenticated clients. The IP address of the application server is concealed from all clients. The proxy nodes are a group of dynamic and distributed machines that relay communications between clients and the application server. The filter ring, similar to what was described in [26], comprises a number of high-speed routers placed around the application server, allowing inbound traffic only from valid proxy nodes. The authentication server is responsible for authenticating clients and assigning legitimate ones to active proxy nodes.

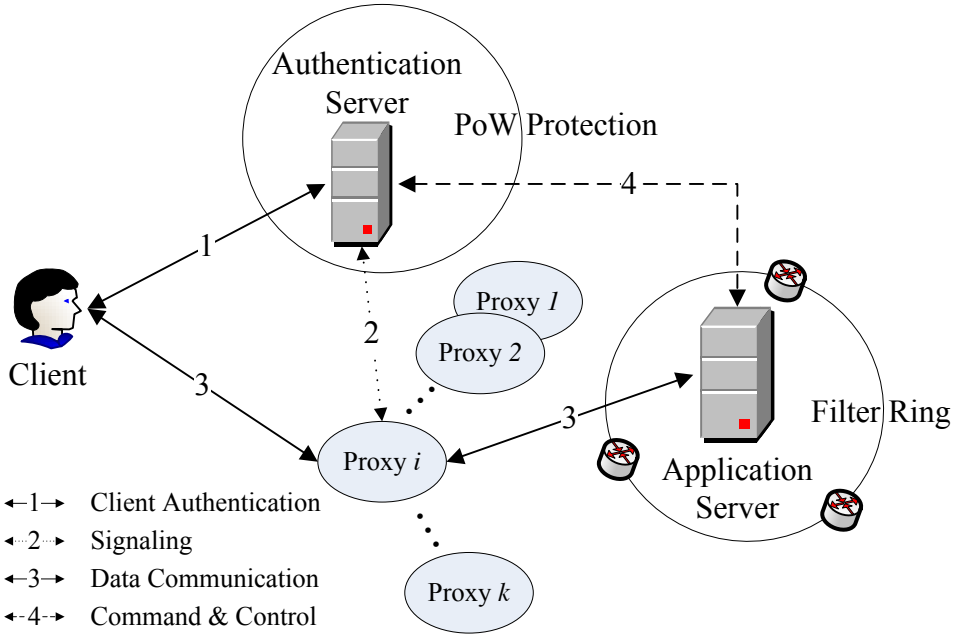


Figure 3.2: The MOTAG Architecture

MOTAG allows a client to access the application server only if the client can be successfully authenticated. One simple solution is to associate the application domain name with the IP address of the authentication server during DNS registration. Each successfully authenticated client is randomly assigned to one of the active proxy nodes whose IP addresses are not publicly known. The authentication server will inform each client about the address of the designated proxy, and in the meantime, it will also notify the proxy node about the

forthcoming connection from the client. The authentication server, as well as every proxy node, maintains a dedicated interface for the purpose of signaling. Through this signaling channel, proxies report to the authentication server when they are under attack; the authentication server informs proxies about each client assignment and coordinates proxies' actions against DDoS attacks. To prevent the authentication server from being flooded, we employ proof-of-work (PoW) schemes [75, 94–96] to ensure its availability for legitimate clients.

The authentication server also assigns a *capability token* for every client-to-proxy session. This token limits the client's throughput by specifying the number of packets (or the number of bytes) allowed for the session in the next time window (t seconds). A proxy should receive two identical copies of the capability token for every session: one from the authentication server to notify new client assignment, and one from the client as a proof of identity. Each proxy node maintains a per-session packet counter to regulate client traffic according to the associated capability. Capability-based traffic policing is the key for detecting external, brute-force flooding attacks by distinguishing authorized packets from illegal ones. Furthermore, abusive attacks such as sharing an assigned capability with external attackers are not going to consume more server-side bandwidth than the pre-defined limit. To secure communications between proxy nodes and the application server, a lightweight authenticator as described in Mayday [26] can be employed for proxy identity validation. The filter ring routers can perform fast lookups to verify such lightweight authenticators encoded in proxy-to-server packets. These authenticators can be dynamically altered to improve security. Once changes are made, active proxy nodes will receive timely updates via the dedicated signaling channel. Using the lightweight authenticator not only allows the filter ring routers to quickly discard unauthorized traffic but also enables us to revoke the access of compromised proxies.

Secret Moving Proxies

The MOTAG architecture creates a traffic indirection layer with a pool of geographically distributed proxies that attackers are incapable of compromising altogether. This layer of proxies can diffuse attackers' traffic and shield the application server. Not all proxies are active all the time. Instead, only a small subset of the proxies are working at any given time to keep costs down. The mapping from clients to the working proxies is many-to-one. An ideal source for the proxy pool is one or several cloud computing environments where customers are charged only for the running instances. The IP addresses of all proxies are withheld from the general public. The authentication server will match legitimate clients to each working proxy. Our proxies are resilient to reconnaissance attacks because packets from unauthorized IPs will be dropped.

If some proxy nodes are under attack, they will be shut down and a number of new proxy nodes at different network locations will be activated for replacement. Proxy substitution is a fast, lightweight operation because all proxies run the same simple traffic indirection logic and maintain no client state. All session information is centrally stored at the application server. A few hot spare proxies can be kept alive over time to quickly start functioning when necessary. All clients connecting to the attacked nodes will be intelligently re-assigned across the entire set of replacement proxies with the goal of separating legitimate clients from DDoS attackers. The new assignment can be pushed to the affected clients by the authentication server, or the clients can be forced to pass re-authentication for security assurance. We name the overall process of proxy replacement and client re-allocation as client-to-proxy *shuffling*, which will be discussed in detail in Section 3.4. When there is no attack, no shuffling is necessary. Only a small set of proxy nodes with constant IP addresses will be used to serve all legitimate clients.

MOTAG is different from existing overlay network solutions [24–26, 29], which rely on a fairly static network composition of overlay nodes to tolerate and filter out the attack traffic. Building and maintaining a static and resilient overlay network entails extensive and continuous investment to acquire more nodes and bandwidth. In addition, sweeping [25]

and adaptive [26] flooding attacks may still cause severe service disruptions. In contrast, MOTAG keeps its proxies confidential and mobile. Only authenticated clients are informed about their assigned proxies. In this way, MOTAG enhances defense agility against massive, sophisticated attacks while reducing its dependence on the volume of proxy resources.

Authentication with Proof-of-Work Protection

Maintaining high availability of the authentication server is essential to our moving target defense. The authentication server acts as the initial checkpoint to separate legitimate clients from illegal ones. User authentication is also used as the mechanism to bind a legitimate client to a specific network flow. Only with such unique binding are we able to track the behavior of each client throughout the shuffling process. Every client must pass authentication before being assigned to a working proxy that eventually routes traffic to the application server. The IP addresses of the authenticated clients are recorded and sent to the proxies that enforce IP-based filtering. The authentication server is also responsible for advertising subsequent client-to-proxy re-assignments during shuffling. MOTAG is agnostic to the specific authentication mechanism employed.

The authentication server is the only component of the MOTAG architecture that can be publicly addressed. Consequently, it can be considered as a new target of flooding DDoS attacks. To protect the authentication server from the re-targeted attacks, we employ existing proof-of-work (PoW) schemes [75, 94–96] to throttle the achievable throughput of attackers. PoW mechanisms force clients to solve cryptographic puzzles before allowing them to consume resources on the server side. In particular, enforcing PoW can realize per-computation fairness on bandwidth usage among all clients [75], prevent connection depletion attacks [96], and mitigate DDoS attacks on application-level authentication protocols [94, 95]. However, while suppressing attackers' throughput, mandating extra computational tasks also imposes considerable overhead on legitimate clients, making it a less preferable option for securing application data communication. Therefore, MOTAG only uses PoW approaches to protect client authentication because authentication packets are

infrequently sent and are more delay-tolerant compared to data packets. We propose to use the more efficient client-to-proxy shuffling mechanism to mitigate flooding attacks on the data communication.

3.2.3 Discussion

By hiding proxies while enforcing client authentication, MOTAG can effectively protect its packet delivery system from external attackers. Moreover, by keeping proxies mobile and performing guided shuffling on client-to-proxy assignments, MOTAG can also mitigate insider attacks that expose secret proxies to flooding attacks.

Advanced attackers can implant malicious insiders in the targeted system via social engineering, compromising legitimate clients, stealing clients' identities for authentication, and eavesdropping on clients' network connections. Although the number of installed insiders might be small considering the high technical sophistication required for such targeted attacks, the damages caused by these insiders can be quite significant. Once insiders uncover the IP addresses of some proxy nodes, they will notify external attackers who will carry out flooding attacks against these exposed proxies. We address such attacks as insider-assisted DDoS attacks, or simply insider attacks. Although insider attacks cannot be fully prevented, we aim to minimize their impact on innocent clients. To that end, Section 3.4 introduces a client-to-proxy shuffling mechanism that can restore service availability for benign clients and quarantine the impact of insider attacks over time.

The MOTAG architecture employs dynamic, private proxies as moving targets to mitigate network flooding DDoS attacks on services offered to authorized and authenticated clients. Our solution does not rely on global adoption on Internet routers or collaboration among ISPs to function, nor do we depend on resource-abundant overlay network to outmuscle high bandwidth attacks and provide fault tolerance. Instead, we take advantage of our proxies' secrecy and mobility properties to fend off powerful attackers. This entails lower deployment costs while offering substantial defensive agility, resulting in an effective DDoS protection.

However, MOTAG cannot offer resilience against computational flooding attacks or be applied to protect open Internet services that do not enforce client authentication. To address these problems, Section 3.3 presents a cloud-enabled architecture that offers wider DDoS protection for generic Internet services.

3.3 Protecting Open Services Using Moving Replicas

This section introduces a cloud-enabled, moving target defense architecture that confuses, evades, and isolates attackers that attempt to blend in with benign users and mount DDoS attacks against Internet services. Section 3.2 proposed MOTAG, a dynamic DDoS mitigation architecture for critical services for authorized and authenticated clients. Unlike MOTAG, we describe in this section a more generic solution that also protects services open to anonymous users. We focus our efforts on web services because they are the most popular yet vulnerable Internet applications. Our defense leverages the resource elasticity offered by cloud computing to intelligently replicate the attacked service when compromised. However, instead of blindly expanding system capacity on demand, we go one step further, attempting to isolate the source(s) of the attack. To that end, we employ the shuffling mechanism to scramble client-to-server assignments: when an attack occurs, we instantiate new replica servers at different network locations and compute an assignment plan to migrate affected clients to the new servers. All attacked servers are recycled to confuse attackers and reduce costs. The network locations of the new replicas are not published and are only known to the subset of clients assigned to them. By keeping track of the assignments and whether a new replica is also attacked, we can quickly tell benign client sessions from potentially malicious ones. The details of the shuffling mechanism and related algorithms will be provided in Section 3.4.

Under this architecture, attackers are forced to keep chasing the "moving" replicas in order to sustain their attack, which eventually enables us to isolate them onto a progressively smaller set of replicas while moving benign clients away from them. In this way, we can not only defeat blind network flooding attacks but also segregate attackers that act as insiders

to expose the network locations of our replica servers or cause a computational attack.

3.3.1 Threat Model and Assumptions

Network DDoS attacks employ voluminous traffic to deplete a victim's bandwidth. Computational DDoS attacks exhaust a victim's limited computation resources (CPU, memory, etc.). These attacks are usually performed by attacker-controlled botnets that are composed of a large number (millions) of infected machines. We assume botnets are capable of bombarding standalone physical servers on the Internet. To combat that, our solution proposes to deploy services in the cloud that enables dynamic server replication. We assume botnets can overwhelm individual cloud servers but not the aggregate of all cloud data centers that host our servers.

To access our servers in the cloud, all clients have to be redirected by the DNS servers followed by the cloud load balancing devices. We assume DNS servers are well-provisioned, and DDoS attacks against DNS service is beyond the scope of this paper. To bypass the redirection steps, attackers may use reconnaissance attacks to locate and gain information about our servers. Incorporating moving target defense in our solution will allow us to evade naive attackers that are unable to receive or parse our re-assignment messages. However, we assume that more persistent attackers, especially human-controlled bots that are equipped with browser-like software, are able to blend in with benign users and follow the moving replica servers. These persistent attackers can act as insiders to launch computational attacks or disclose the network locations of the replica servers to outsiders that mount network or computational attacks. For simplicity without loss of generality, any replica server assigned with one or more persistent bots are considered under attack in our modeling.

We assume the occurrence of a DDoS attack can be easily detected with signs such as sudden network congestion and abrupt surge of HTTP requests. Advanced traffic analysis techniques, such as [97,98], can also be used for early detection and the detection of sophisticated attacks. Once an attack is detected, the solution below can be applied for attack mitigation and isolation.

3.3.2 Architecture

To mitigate DDoS attacks, we propose a cloud-enabled moving target defense architecture to achieve attacker segregation. Our goal is to save benign clients by separating them from attackers via client-to-server shuffling: when an attack occurs, our defense dynamically instantiates new replica servers and intelligently re-assigns affected clients to the new servers. Our solution leverages the resource elasticity and scale offered by cloud computing to instantiate and hide new replica servers. Each client, identified by the IP address, is randomly assigned to one of the active replica servers. As a result, attackers will only be able to see the servers to which their bots are assigned. Through multiple rounds of shuffling as guided by the algorithms to be discussed in Section 3.4, we can gradually separate benign clients from attackers. When there is no attack, only a small number of static servers are maintained to meet the requirement of the regular workload.

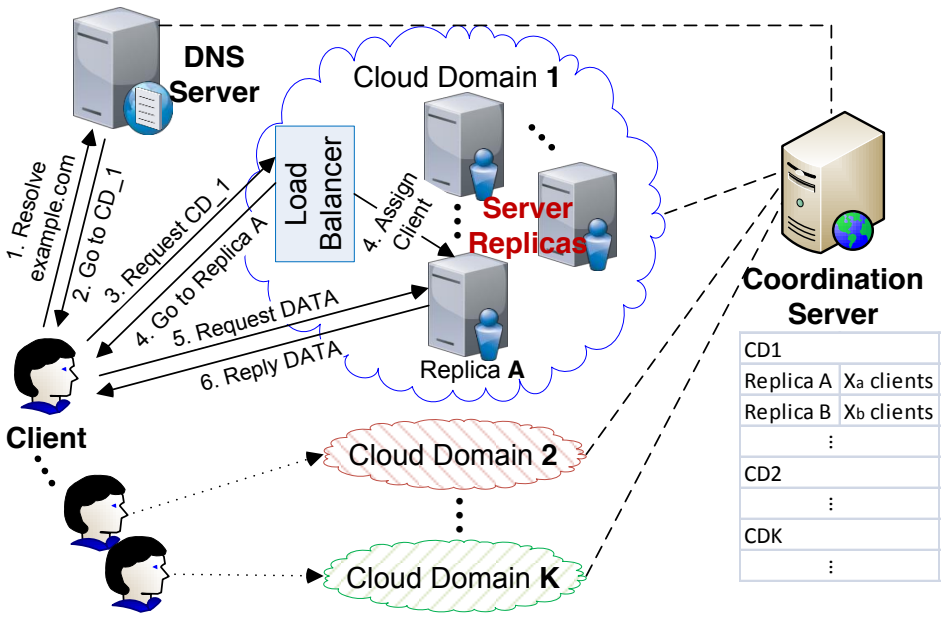


Figure 3.3: The Cloud-Enabled DDoS Defense Architecture

The system architecture and components that enable our moving target defense is shown

in Figure 3.3. This architecture presents a distributed connection redirection system deployed across multiple cloud computing domains. To reach the protected web service, a client must first resolve its domain name via DNS (step 1). The DNS server will send the client to a cloud domain where our DDoS defense is in place (step 2). One or more load balancers are established in each cloud domain to greet newly arrived clients (step 3). A load balancer keeps records about the active replica servers within the same domain and assigns new clients to these replicas according to selected load-balancing algorithms. The load balancer informs both the corresponding client and replica on every new assignment (step 4). As a result, the replica will add the client IP to its whitelist. Once the client contacts the replica, the replica will provide the client with the requested service (step 5, 6).

The coordination server is the central controller that maintains global client-to-server bindings and directs defensive reactions to DDoS attacks. Although the mechanism is centrally controlled at the backend, all client-facing components are fully distributed, forming a robust layer of defense against DDoS attacks. Next, we describe each key component in greater detail.

Load Balancer

A load balancer assigns each new client to a replica server. Clients are identified by their IP addresses. Each client IP is only matched with one replica server (i.e. sticky sessions). The load balancer will inform the server and the client on every client-to-server assignment.

At least one load balancer is installed in each cloud domain to keep track of all co-located, active replica servers. Here, different domains refer to groups of separately managed cloud servers that do not share common bottleneck links. They can be datacenters deployed at different geo-locations by the same cloud provider or different cloud systems operated by distinct providers. Deploying multiple load balancers per cloud domain and simply having more cloud domains can improve attack resiliency and fault tolerance. The DNS records of all load balancers can be registered under the domain name of the protected web server.

Techniques including round-robin DNS [99] can be used to send clients to different load balancers.

Each network load balancer maintains an up-to-date list of online replica servers in its local domain. Any load balancing algorithm can be applied to assign clients to replicas. We make sure our load balancers will redirect each client to a running replica. For web services, this can already be achieved via HTTP redirection messages (i.e. status code 301) [100] that are handled automatically by client browsers. The reasons for using redirecting load balancers are two-fold: first, traffic redirection resembles a two-way handshake, which can effectively stop junk packets with spoofed source IPs; second, by not making load balancers the middleman of client-server communication, we lower the risk of having them be a bottleneck during an attack.

To ensure the load balancers' availability, many cloud providers, including Amazon [60], enable auto-scaling of a virtual load balancer to provide on-demand capacity. It is reported that the throughput of a dedicated load balancer can reach 40 Gbps [101]. To avoid becoming a system bottleneck, multiple distributed load balancers can be deployed at the backbone network link of each cloud domain to increase resiliency against flooding attacks.

Replica Servers

Compared to load balancers, web servers are more vulnerable to flooding attacks. To combat such a threat, we replicate the protected service within the cloud environment. When there is no attack, only a small number of replica servers are maintained to meet the requirement of the regular workload. Each client is redirected to an active replica by a load balancer. Replica servers enforce whitelist-based filtering, only admitting clients whose IPs are confirmed by the referring load balancer.

Once some replicas are bombarded by a flooding attack, a number of substitute replicas will be instantiated at different network locations. All clients served by the attacked replicas will be re-assigned across the entire set of replacement replica servers. This process is called *client-to-server shuffling*. Replica servers that are used for shuffling are thus called

shuffling replicas. The attacked replica servers will notify associated clients about their new assignments. For stateless web applications, such notification can be simply implemented by HTTP redirection code (3xx) [100]. After sending out all notifications, the attacked replicas will be taken offline and recycled. To expedite the replica replacement process, a few hot spare replica servers can be maintained at runtime.

Dynamically replacing attacked replica servers constitutes moving targets that significantly raise the level of uncertainty for attackers. To inflict longstanding attacks, attackers have to keep chasing the "moving" replicas. Assuming attackers cannot overwhelm the underlying cloud infrastructure, they will need to send out scouts that play as normal clients to follow the moving replicas. However, as will be discussed in Section 3.4, our shuffling mechanism can expose and isolate such scouts, and save the affected benign clients over time.

Coordination Server

The coordination server maintains the global state of the defense system and directs realtime actions against DDoS attacks. In particular, it tracks the number of clients bound to each replica server and records which replicas are currently under attack. In response to a DDoS attack, the coordination server runs the shuffling algorithms to separate benign clients from bots. Based on the number of attacked replica servers and the current client distribution, the coordination server computes an optimal shuffling plan that maximizes the probability of attack segregation. To ensure efficient shuffling operations, the coordination server decides the number of clients to be re-assigned from an attacked replica to a newly instantiated replica. It does not control the specific assignments of individual clients. The coordination server communicates among cloud domains via a dedicated command and control channel that is inaccessible to clients. Section 3.4 provides detailed analysis on the shuffling procedure and related algorithms.

3.3.3 Discussion

This section proposed a cloud-enabled architecture to mitigate network and computational DDoS attacks against open web services. This architecture lays the foundation for client-to-server shuffling, which can effectively segregate the attackers that pretend to be benign but exfiltrate the network locations of the servers or exert computational attacks. Using client redirection and whitelist based traffic filtering can help defeat IP spoofing and reconnaissance attacks.

The architecture is designed to be deployed as a cloud-based end system, which can be independently implemented and managed by a non-ISP organization. The resource elasticity offered by cloud computing makes our solution scalable to withstand strong DDoS attacks. As long as the underlying cloud computing infrastructure is available, the designed system is expected to gradually segregate attackers and restore QoS for benign-but-affected clients. Resources are not allocated permanently but elastically, which incurs minimal maintenance costs compared to having dedicated defensive infrastructures.

In addition to this architecture and the MOTAG architecture introduced in Section 3.2.2, we apply optimized client shuffling operations to achieve attack isolation. The next section is going to provide detailed analysis and evaluation of the shuffling procedure.

3.4 Shuffling-based Attack Segregation

This section presents the modeling, analysis, and evaluation of the proof-of-concept client shuffling mechanism. The goal of shuffling is to separate benign clients from persistent attackers (i.e. bots) following and bringing attack to the moving targets constituted by our DDoS defense architectures. In the context of the MOTAG architecture described in Section 3.2, the moving targets are the secret moving proxies employed to mediate the communication between clients and the application server; persistent attackers refer to the malicious insiders that divulge the IP addresses of the secret proxies to external attackers. In the case of the cloud-enabled architecture for protecting open Internet services in Section 3.3, the moving targets become the dynamically instantiated replica servers residing in the cloud; persistent attackers are the advanced bots that exfiltrate the network locations of the replica servers or exert computational attacks. The shuffling mechanism to be discussed in this section is an abstracted methodology that can be applied to both architectures. For the convenience of discussion, we will address the proxies and replica servers of respective architectures uniformly as service nodes, and call persistent attackers persistent bots, or simply bots.

It is hard to achieve attacker segregation using an engineering method because persistent bots may behave exactly like benign clients. With our solution, new service nodes are dynamically instantiated during a DDoS attack to replace the ones bombarded by bots. The shuffling operation refers to a structured method of re-assigning the affected clients from the attacked service nodes to the new service nodes. Benign clients are saved when assigned to the service nodes with no bots. To maximize the number of benign clients saved from each shuffle, we need to make informed and optimal decisions based on the status quo. To that end, we developed algorithms to realize fast attacker segregation. The controlled and optimized shuffling process will enable us to mitigate an intensive DDoS attack in a few rounds of shuffling.

3.4.1 Shuffling Strategy

Before diving into the mathematical analysis, we first explain the underlying theory of the shuffling mechanism. Section 3.1 illustrated the basic idea through a simplified example. Here we provide a formal description about the rationale and methodology.

Our shuffling mechanism is triggered by the occurrence of a DDoS attack. At time of peace, only a small number of static service nodes are kept online to meet the requirement of a regular workload. As soon as an attack happens, new services nodes are instantiated to replace the attacked nodes and perform client shuffling. The set of active service nodes can be logically divided into two groups, namely *serving service nodes* and *shuffling service nodes*. Serving service nodes provide more reliable QoS to the known benign clients, while shuffling service nodes are employed for shuffling operations and only offer intermittent connections to suspicious clients. If attacked, shuffling service nodes will be replaced and the associated clients are flushed and reassigned.

At the beginning, all service nodes are unmarked. Clients are randomly assigned to different service nodes. If some service nodes are attacked after the initial assignment, they will be marked as shuffling service nodes while the others are regarded as serving service nodes. By employing the shuffling algorithms described below, we repeatedly shuffle the client assignment within the shuffling service node group to distinguish persistent bots from benign clients and segregate them.

After each shuffle, some shuffling service nodes may remain under attack while others may not. The shuffling service nodes that are no long under attack become serving service nodes and the associated clients are marked as trusted and considered as saved from the on-going attack. Clients connected to the attacked service nodes are considered untrusted because we cannot tell who the actual bots are among them. To save the benign-but-affected clients, we randomly re-distribute all the untrusted clients across the entire group of shuffling service nodes. Given the specific number of untrusted clients and available service nodes, new shuffling service nodes can be instantiated to help accelerate the shuffling operations. Generally speaking, the more shuffling service nodes available, the faster the bots will be

quarantined.

By repeating the shuffling operation for multiple rounds, keeping track of the attacked service nodes and untrusted clients, we can gradually identify most benign clients and narrow down the range of suspicious clients. The bots will eventually be quarantined, and the attack damage will be minimized.

Notice that the shuffling process is stateless, meaning each shuffle is considered independent. The tags (trusted/untrusted) placed on clients will be reset after every shuffle, to avoid being confused by bots' inconsistent behavior. These tags do not necessarily reflect the true identity of the clients. Plus, the roles of service nodes (shuffling/serving) are interchangeable across shuffles, depending on the behavior of the bots. The goal of the shuffling operations is to separate benign but attacked/suspected clients from true bots. Although some bots may make us believe in their innocence by staying inactive, we can ensure that they are not going to cause extra damage if they begin to attack later.

3.4.2 Theoretical Problem Modeling

We first constructed a mathematical model of the shuffling process. Notations used in this model are summarized in Table 3.1. We consider a snapshot of the entire system before shuffling is performed. There are a total number of N clients, M of whom are persistent bots that act as malicious insiders ($M \leq N$). There are a constant number of P service nodes used for shuffling. Notice that after certain rounds of shuffling, some service nodes may stop participating in the shuffling operations if they are no longer under attack. In that case, we can activate more service nodes to supplement the need for shuffling clients. We assume $P < N$; otherwise, each client can be allocated with an exclusive service node to accomplish complete attacker isolation. In practice, many clients are assigned to a service node.

For the purpose of fast attacker segregation, we endeavor to save as many benign clients as possible from each round of shuffling. Since shuffling is a stochastic process, we can only calculate the probability that a given service node is attacked. With that probability, we

Table 3.1: Notations used in section and their meanings.

Notation	Meaning
N	number of clients (including benign clients and bots)
M	number of persistent bots
P	number of shuffling service nodes
S	number of clients to be saved
p_i	probability that the i-th shuffling service node is not under attack
x_i	number of clients assigned to the i-th shuffling service node

derived $E(S)$, the expected number of benign clients to be saved in one round. Therefore, solving the following optimization problem will maximize the number of benign clients to be saved.

$$\max\ E(S) = \sum_{i=1}^{P} p_i \cdot x_i = \frac{\sum_{i=1}^{P} \binom{N-x_i}{M} x_i}{\binom{N}{M}} \tag{3.1}$$

$$\text{subject to} \sum_{j=1}^{P} x_j = N$$

In the above equation, $x_1, x_2, \ldots, x_P$ denote the numbers of clients we assign over these P shuffling service nodes. p_i denotes the probability that the i-th service node is not under attack, which is also the probability that all M bots are assigned to other service nodes. Hence,

$$p_i = \frac{\binom{N-x_i}{M}}{\binom{N}{M}}$$

where $\binom{N}{M}$ computes the total number of ways to distribute the M bots within the client population N, and $\binom{N-x_i}{M}$ gives the number of combinations that all bots are within the $N - x_i$ clients not connecting to shuffling service node i.

3.4.3 Optimal Solution

To solve the above optimization problem, we state the problem alternatively with regard to its key parameters. Let $S(N, M, P)$ denote the maximum expected number of benign clients that we can save in one shuffle, subject to a total number of N clients (including bots), M bots, and P shuffling service nodes. To solve $S(N, M, P)$, we can instead solve $\max\{S(a, b, 1) + S(N - a, M - b, P - 1)\}$, where a is the number of clients assigned to the last service node, and b is the number of bots among a. Although we do not have direct control over b, there is a probability $\Pr(b)$ associated with each possible value of $b \in [0, \min\{a, M\}]$. These probabilities will vary as we change the value of a. For each different a, we can compute the expected result of the decomposed problem. In this way, the original optimization problem can be recursively broken down into a series of sub-problems, until all potential solutions for the sub-problems can be readily enumerated.

We use a dynamic programming approach to solve these sub-problems in a bottom-up fashion, until eventually arriving at the optimal solution to the original problem. In particular, we incrementally build a lookup table, starting from using only one service node ($P = 1$). To compute the one service node case, we have

$$S(a, b, 1) = \begin{cases} a, & b = 0 \\ 0, & b > 0 \end{cases} \tag{3.2}$$

Solutions for larger service node numbers can be generated using

$$S(N, M, P) = \max_{1 \le a \le N-1} \{\sum_{b} \Pr(b) \times [S(a, b, 1) + S(N - a, M - b, P - 1)]\}$$

where

$$\Pr(b) = \frac{\binom{M}{b}\binom{N-M}{a-b}}{\binom{N}{a}}, b \in [0, \min(a, M)] \tag{3.3}$$

Algorithm 1, below, describes the implementation of the dynamic programming procedure. Two tables, assign_no and save_no, are created to store the number of clients to assign to each service node and the expected number of benign clients to be saved, respectively. The overall time complexity of the algorithm is $O(N^3 \cdot M^2 \cdot P)$, while the space complexity is $O(N \cdot M \cdot P)$.

Algorithm 1 Optimal-Assign(N, M, P)

Initialize $assign\_no[0 \cdots, N, 0 \cdots, M, 0 \cdots P]$ and $save\_no[0, \cdots, N, 0 \cdots, M, 0 \cdots, P]$
for $i \leftarrow 1, N$ **do**
    for $j \leftarrow 1, M$ **do**
        for $k \leftarrow 1, P$ **do**
            compute $S(i, j, k)$ using Equations 3.2 and 3.3, with $a \in [1, i-1]$ and $b \in [1, \min\{j, a\}]$;
            select $a = \alpha$ that maximize $S(i, j, k)$;
            update table entry $assign\_no[i, j, k] = \alpha$ and $save\_no[i, j, k] = S(i, j, k)$.

3.4.4 A Special Case

Although the dynamic programming algorithm is guaranteed to produce optimal client-to-service-node assignment, it is inadequate for making realtime decisions against on-going DDoS attacks due to its high time complexity. We hope to identify the common structure of an optimal solution that can be applied directly at runtime. As a special but practical case, we are going to show that it is always best to evenly distribute all clients to every service node when the number of bots (M) is relatively small.

Theorem 1. *Evenly distributing clients to all shuffling service nodes maximizes $E(S)$, when $M \leq P$ and client-to-service-node distribution is uniform.*

Proof. We prove Theorem 1 using the exchange argument. For simplicity without loss of generality, we consider an arbitrary pair of service nodes among all shuffling service nodes. Assume there are V clients on one service node and W clients on the other, where $V + W = T$. Given that client-to-service-node distribution is uniform and $M \leq P$, it is expected that there are one or two bots assigned to this pair of service nodes. Using

Equation 3.1, when $M = 1$, we have

$$E(S) = \frac{\binom{T-V}{1} \cdot V}{\binom{T}{1}} + \frac{\binom{T-W}{1} \cdot W}{\binom{T}{1}} = \frac{2 \cdot V \cdot W}{T}$$

When $M = 2$, we have

$$E(S) = \frac{\binom{T-V}{2} \cdot V}{\binom{T}{2}} + \frac{\binom{T-W}{2} \cdot W}{\binom{T}{2}} = \frac{V \cdot W \cdot (T-2)}{T \cdot (T-1)}.$$

In both cases, $V = W$ maximizes $E(S)$. Since this pair of shuffling service nodes is randomly chosen, we can apply an exchange argument on any pair of shuffling service nodes iteratively, and eventually arrive at the conclusion that all the shuffling service nodes should have the same amount of clients to maximize $E(S)$. □

As a result, in the special case where the number of bots is less than or equal to the number of service nodes, we can obtain an optimal client-to-service-node assignment in linear time $O(N)$ by randomly assigning N/P clients to each service node.

3.4.5 A Fast Heuristic Greedy Algorithm

The discussion in Section 3.4.4 covers the special case in which $M \leq P$. However, a runtime algorithm is still needed to solve more general cases where $M > P$. To that end, we designed a fast heuristic greedy algorithm that is capable of generating near-optimal shuffling plans.

Algorithm 2 describes each step of the greedy algorithm for computing the client-to-service-node assignment, with the main function called *GreedyAssign*. Instead of targeting the global optimization problem, the greedy algorithm tries to optimize for one service node at a time. The algorithm first computes the number of clients (ω) that should be assigned to the first service node under the original problem setting, so as to save the maximum number of benign clients expectedly from this node. To do that, the algorithm enumerates all possible values of x_1 (denoting the number of clients to be assigned to the first service

node), ranging between $[0, N-1]$. x_1 cannot be N. Otherwise, everyone will be attacked if there is one bot on board. For each possible value of x_1, the algorithm computes the result of $p_i \cdot x_1$. The value of $x_1 = \omega$ that yields the maximum result becomes the final selection. This subroutine is described in procedure *MaxOneNode* of Algorithm 2. Without further computation, the algorithm attempts to assign ω clients to as many service nodes as possible until running out of clients or service nodes. The computation will terminate under three conditions. First, when there are more service nodes left than clients, each client will be assigned to an exclusive service node. Second, when there is only one service node left, all remaining clients will be appointed to it. Third, when the expected number of remaining bots is rounded to 0, all remaining clients will be evenly distributed for load balancing. Otherwise, if the number of remaining clients is less than ω, the algorithm updates the problem setting (N', M', P') with the remaining unassigned clients and service nodes, and calls itself recursively. The computational complexity of the greedy algorithm is $O(N \cdot M)$ while its space complexity is $O(P)$.

To evaluate the optimality of the greedy algorithm, we compare it head-to-head with the dynamic programming algorithm through a series of MATLAB simulations. It is worth mentioning that the high computational complexity of the dynamic programming algorithm only permitted experiments with relatively small parameters (up to 1,000 clients and 200 service nodes).

Figure 3.4 shows the numbers of benign clients saved in one shuffle by running the two algorithms. As we can see, the curves denoting respective algorithms almost overlap in all cases, indicating that the effectiveness of the greedy algorithm closely approaches to the optimal dynamic programming algorithm.

The advantage of using the greedy algorithm is reflected by comparing Figure 3.5 and 3.6. Even though small client, bot, and service node numbers were used, the dynamic programming algorithm required tens of hours to compute the client-to-service-node assignments. In contrast, the greedy algorithm needed only a few milliseconds to complete the same tasks. Therefore, the greedy algorithm is preferred as the runtime algorithm to guide shuffling

Algorithm 2 Greedy algorithm for computing client-to-service-node assignment.

function GREEDYASSIGN($Client, Bot, ServNode$)
  if $Client \leq ServNode$ **then**
    Assign 1 exclusive service node to each client
  else if $ServNode = 1$ **then**
    Assign all clients to the last service node
  else if $Bot = 0$ **then**
    Evenly distribute $Client$ over $ServNode$
  else
    $\omega = MaxOneNode(Client, 0, Client - 1, Bot)$
    $NodeToFill = floor(Client/\omega)$
    if $NodeToFill \geq ServNode$ **then**
      $NodeToFill = ServNode - 1$
    $RemC = Client - NodeToFill * \omega$
    $RemP = ServNode - NodeToFill$
    $RemB = Round(\frac{Bot*RemC}{Client})$
    Fill $NodeToFill$ Proxies with ω clients each
    Fill the rest proxies according to
    $GreedyAssign(RemC, RemB, RemP)$

procedure MAXONENODE($Client, Lbound, Ubound, Bot$)
  Max=0, MaxAssign=0
  for $i = Lbound \rightarrow Ubound$ **do**
    $Save = \binom{Client-i}{Bot} i / \binom{Client}{Bot}$
    if $Save > Max$ **then**
      $Max = Save, \;\; MaxAssign = i$
  return MaxAssign

operations against on-going DDoS attacks.

3.4.6 Enhancing The Greedy Algorithm

As discussed above, the greedy algorithm aims to maximize the number benign clients expectedly to be saved from one service node at a time. For an arbitrary service node i, the number of clients assigned to it and the number of benign clients that can be saved are denoted as x_i and S_i, respectively. The expected value of S_i is computed as follows.

$$E(S_i) = p_i \cdot x_i = \frac{\binom{N-x_i}{M}}{\binom{N}{M}} \cdot x_i \tag{3.4}$$

The subroutine *MaxOneNode* of Algorithm 2 finds the value of x_i that maximizes $E(S_i)$ by enumerating all possible choices. For the purpose of this discussion, we name this

approach the *enumeration approach*. The enumeration approach can find the optimal value of x_i in $O(N \cdot M)$ time complexity, where N denotes the total number of all clients and M denotes the number of bots among the clients. Therefore, as N and M get larger, the running time of the subroutine *MaxOneNode* becomes notably longer.

To further improve the efficiency of the greedy algorithm, we designed an approximation approach that can find a near-optimal value of x_i in $O(1)$ time complexity. This is done by solving Equation 3.4 using Stirling's Approximation $n! \approx (\frac{n}{e})^n \sqrt{2\pi n}$ and a series of other approximations assuming $M << N$, as follows:

$$\begin{aligned} E(S_i) &= \frac{(N-M)!(N-x_i)!}{(N-M-x_i)!\ N!} \cdot x_i \\ &\approx \frac{(N-M)^{N-M}(N-x_i)^{N-x_i}\sqrt{(N-M)(N-x_i)}}{(N-M-x_i)^{N-M-x_i}N^N\sqrt{(N-M-x_i)N}} \cdot x_i \\ &\approx \frac{(N-M)^{N-M}(N-x_i)^{N-x_i}}{(N-M-x_i)^{N-M-x_i}N^N} \cdot x_i \\ &\approx (\frac{N-M}{N})^{x_i} \cdot x_i \end{aligned}$$

Let $x_i = \frac{N \cdot \lambda}{M}$, we have

$$E(S_i) = (1-\frac{M}{N})^{\frac{N\cdot\lambda}{M}} \cdot \frac{N \cdot \lambda}{M} = e^{-\lambda}\frac{N \cdot \lambda}{M} \tag{3.5}$$

After applying approximation $\lim_{n\to\infty}(1-1/n)^n \approx e^{-1}$ on the derivation of Equation 3.5, we get

$$\frac{\partial E(S_i)}{\partial \lambda} = \frac{e^{-\lambda} \cdot N}{M} \cdot (1 - \lambda) \tag{3.6}$$

Apparently, the derivation of $E(S_i)$ equals 0 if and only if $\lambda = 1$. In other words, $\lambda = 1$ maximizes $E(S_i)$. As a result, assigning $x_i = \frac{N}{M}$ clients to service node i will optimize the expected number of benign clients that can be saved, which is

$$Max(E(S_i)) = \frac{N}{M \cdot e} \tag{3.7}$$

To evaluate the accuracy and the optimality of the approximation approach, we compare it with the enumeration approach using a series of simulations in MATLAB. Figure 3.7 shows the numbers of clients that should be assigned to one service node to expectedly save the maximum number of benign clients, computed using the enumeration approach and the approximation approach respectively. Since the enumeration approach compares all possible choices, we know that it is always giving the optimal selection for one service node. In Figure 3.7, we notice that the results produced by the approximation approach overlap with those given by the enumeration approach in almost all cases. Therefore, as a faster approach ($O(1)$ time complexity) that can still generate near-optimal results, we believe the approximation approach is the preferred method in computing the number of clients to assign to one service node for the greedy algorithm.

3.5 Attack-Scale Estimation

When applying the greedy algorithm or the dynamic programming algorithm, the number of persistent bots M is a key parameter in computing the optimized client-to-service-node shuffle plans. In the earlier discussions, we assumed that M is known. However, in practice,

no such prior knowledge is provided. Since M has direct influence on the client-to-service-node assignments, accurately estimating M is critical. To achieve that, we designed a Maximum Likelihood Estimation (MLE) algorithm.

The core idea of the MLE algorithm is to evaluate the likelihood of each possible value of M with regard to the number of observed service nodes that are not under attack (denoted as X). The MLE algorithm computes the probability that a particular value of M is going to cause X service nodes not to be hit by an attack. All possible values of M are enumerated, and the one that corresponds to the highest probability becomes the final estimate of M.

In the following discussion, the likelihood that an arbitrary value of M is going to leave $X = \pi$ service nodes not under attack is denoted as $Pr(X = \pi)$. To compute $Pr(X = \pi)$, we build Equation 3.8 by employing the inclusion-exclusion principle under the balls-and-urns model [102].

$$\begin{aligned} Pr(X = \pi) = Pr(X \geq \pi) - \binom{\pi+1}{\pi} Pr(X \geq (\pi+1)) \\ + \binom{\pi+2}{\pi} Pr(X \geq (\pi+2)) - \dots \\ + (-1)^{P-\pi} \binom{P}{\pi} Pr(X \geq P) \end{aligned} \tag{3.8}$$

In Equation 3.8, $Pr(X \geq \Pi)$ stands for the probability that at least Π ($\Pi = \pi, \pi + 1, \dots, P$) service nodes are not under attack, P is the total number of all shuffling service nodes. In particular, these Π unattacked service nodes constitute the set $\mathbf{U} = \{u_1, u_2, \dots, u_\Pi\}$. Set $\mathbf{U}$ can be any Π sized subset of the P shuffling service nodes.

The key idea to compute $Pr(X \geq \Pi)$ is similar to the way that Equation 3.1 and Equation 3.4 were derived. If a particular set $\mathbf{U}$ of service nodes are not under attack, all persistent bots must be among the clients assigned to other service nodes (the complement of $\mathbf{U}$). Thus, we can derive Equation 3.9, in which $\sum_{\mathbf{U}}^{(\Pi)}$ denotes the summation over

all possible combinations of $\mathbf{U}$ (all Π sized subsets of the P shuffling service nodes), and $N - \sum_{j=1}^{\Pi} x_{u_j}$ gives the number of clients connecting to the service nodes outside of the set $\mathbf{U}$. Specifically, u_j is an arbitrary service node within the set $\mathbf{U}$, and x_{u_j} denotes the number of clients assigned to that node.

$$Pr(X \geq \Pi) = \frac{\sum_{\mathbf{U}}^{(\Pi)} \binom{N-\sum_{j=1}^{\Pi} x_{u_j}}{M}}{\binom{N}{M}} \tag{3.9}$$

Under a certain client-to-service-node assignment scheme $\mathbf{x}$, we can now correlate $Pr(X = \pi)$ with the potential values of M by combining Equation 3.8 and 3.9.

To evaluate the accuracy of MLE through example cases, we implemented the algorithm in MATLAB and ran a series of simulations. The actual bot numbers (X-axis), the estimated bot numbers (the solid line corresponding to the Y-axis on the left), along with the percentage of attacked service nodes (the dotted line corresponding to the Y-axis on the right), are plotted in Figure 3.8. In this figure, each data point is the mean of 40 repeated runs surrounded by a 99% confidence interval. From the results, we notice that MLE produces accurate estimation numbers unless all shuffling service nodes are under attack.

As discussed above, MLE always chooses the value of M that maximizes the likelihood function. For the special case where all shuffling service nodes are under attack, the likelihood is always greater with the higher value of M. Consequently, the largest possible M, i.e. the overall client number N, becomes the final estimate. Given this is an unrealistic overestimation, extra shuffling service nodes should be instantiated to make sure at least one service node is not under attack. Below, we theoretically determine the number of shuffling service nodes P needed to meet this requirement under uniform client-to-service-node assignment.

We abstract an approximate model as following: There are P shuffling service nodes and M bots. We assign M bots sequentially onto the service nodes.

Theorem 2. *If* $M > \log_{1-\frac{1}{P}}\left(\frac{1}{P}\right)$*, there is a high probability that all service nodes will be attacked.*

Proof. Let X denote the number of service nodes that are not under attack. Let

$$X_i = \begin{cases} 1, & \text{if service node } i \text{ is not under attack;} \\ 0, & \text{otherwise.} \end{cases}$$

Then the expected value of X is

$$E(X) = \sum_{i=1}^{P} E(X_i) = \sum_{i=1}^{P} \Pr[X_i = 1] = P\left(1 - \frac{1}{P}\right)^M$$

When all shuffling service nodes are attacked with high probability, the expected number of unattacked shuffling service nodes $E(X) < 1$. We have

$$E(X) < 1 \Rightarrow P\left(1 - \frac{1}{P}\right)^M < 1 \Rightarrow M > \log_{1-\frac{1}{P}}\left(\frac{1}{P}\right)$$

□

According to Theorem 2, when all shuffling service nodes are under attack, P must be increased such that $M \leq \log_{1-\frac{1}{P}}\left(\frac{1}{P}\right)$. The resource elasticity permitted by the underlying cloud infrastructure allows sufficient space for us to increase the number of service nodes. Therefore, we can always be confident in the estimations made by the MLE algorithm.

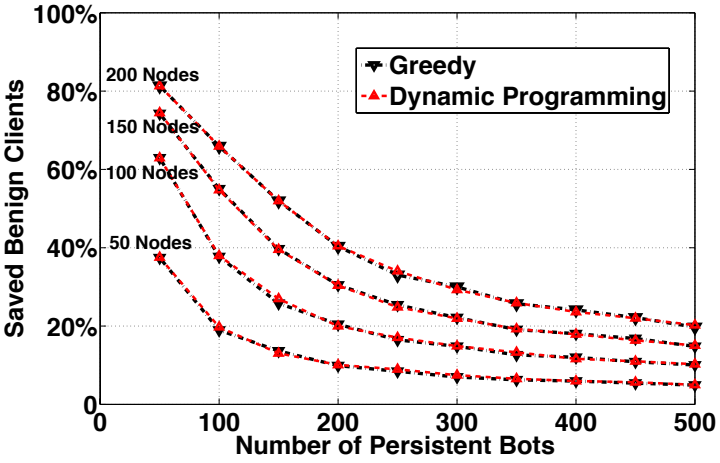


Figure 3.4: Comparison of the effectiveness of greedy algorithm and dynamic programming algorithm with 1,000 clients. (Curves are overlapping.)

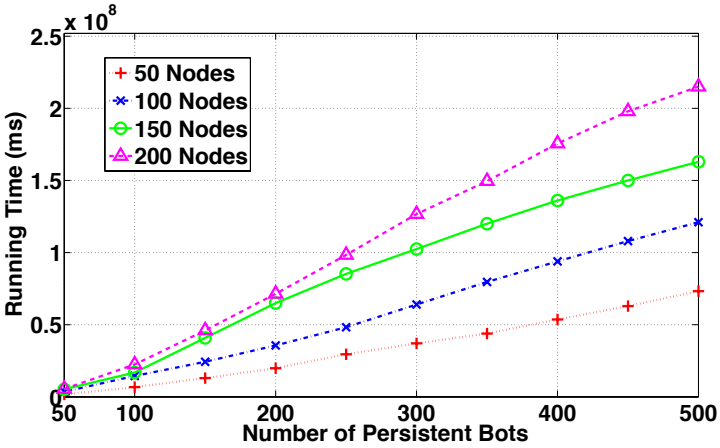


Figure 3.5: Runtime of the dynamic programming algorithm with 1,000 clients.

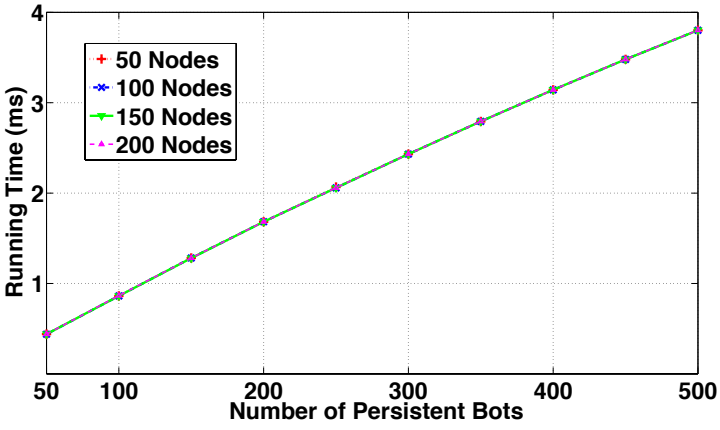


Figure 3.6: Runtime of the greedy algorithm with 1,000 clients.

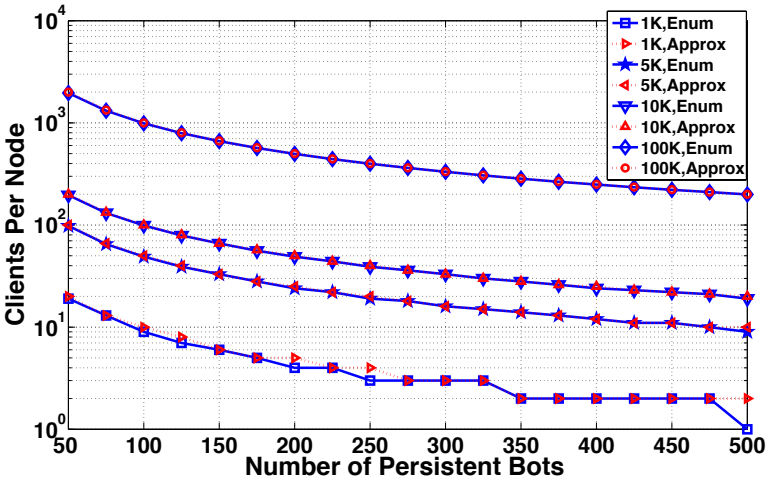


Figure 3.7: The optimal number of clients to assign to one service node computed using the enumeration approach and the approximation approach. The Y-axis is in log-scale, and the curves are overlapping.

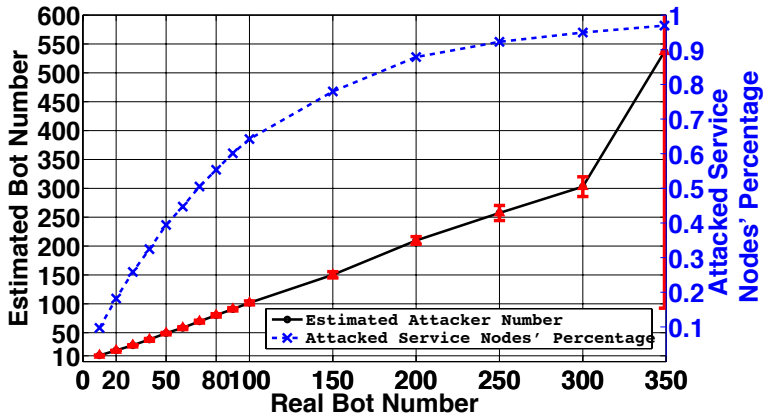


Figure 3.8: Evaluate the MLE algorithm through examples (10,000 clients, 100 shuffling service nodes).

3.6 Evaluating Moving Target Defenses

3.6.1 Attacker Quarantine Capability

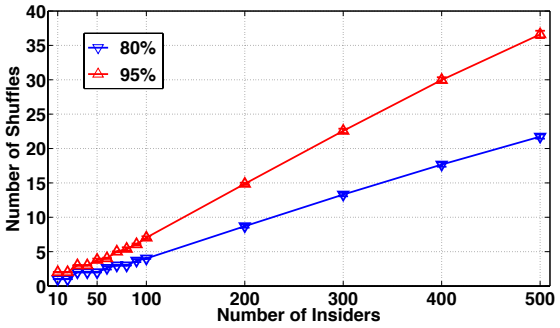
This section experimentally evaluates the effectiveness of the shuffling mechanism used in separating benign clients from attackers. As discussed in Section 3.4, the shuffling mechanism can be applied on the MOTAG architecture (Section 3.2) to segregate malicious insiders from the authenticated clients, and can also be combined with the cloud-enabled architecture (Section 3.3) to save benign clients from the moving replica servers followed by persistent bots. To better evaluate the shuffling mechanism with regard to different architectures and application models, we implemented the core algorithms in MATLAB and ran simulated attacks under different settings.

Segregating Insiders From Authenticated Clients

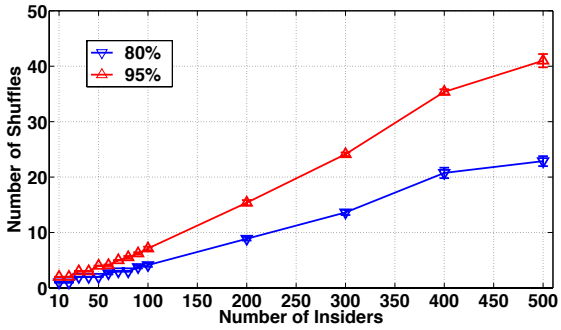
To simulate an attack scenario on MOTAG, we injected a small number of insiders simultaneously into the group of authenticated clients. These insiders will bring external attackers to some secret moving proxies, causing these proxies to be flooded. To respond to the attack, the applied shuffling mechanism will dynamically replace the overwhelmed proxies, and intelligently re-distribute affected clients among all shuffling proxies. For ease of comparison and evaluation, a fixed number of shuffling proxies are kept active throughout each experiment. We assumed that the attackers are aggressive and own infinite bandwidth. Consequently, all shuffling proxies that are assigned one or more insiders are regarded as under attack. However, as discussed in Section 3.2, we only assumed a limited number of insiders (hundreds) considering the difficulty of bypassing strong authentication. In all of our simulations, the MLE algorithm from Section 3.5 was used to estimate the number of remaining insiders. The greedy algorithm discussed in Section 3.4.6 was used to compute the optimized client-to-proxy assignment plan for each shuffle. Mersenne twister [103] was employed as the random number generator for all simulation runs.

Figure 3.9 shows the number of shuffles needed to save 80% and 95% of benign clients

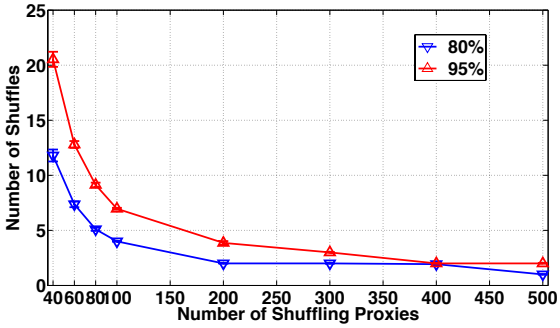
under various parameters. In particular, Figures 3.9a and 3.9b vary the number of insiders while keeping the total number of clients and shuffling proxies constant. Figures 3.9c and 3.9d only change the number of shuffling proxies while having the other two parameters fixed. 10,000 clients were simulated in the experiments that produced Figures 3.9a and 3.9c; 100,000 clients were simulated in the simulations that generated Figures 3.9b and 3.9d. For each setting, we ran the same simulation 30 times to get the mean with a 99% confidence interval plotted.



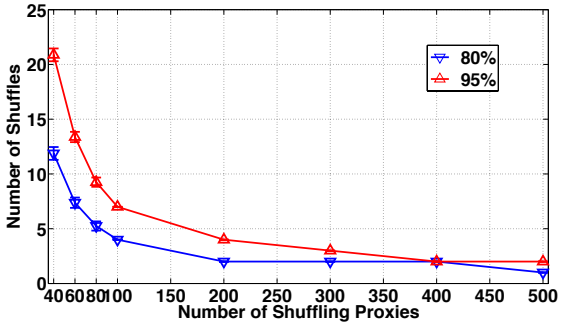
(a) Varying the number of insiders under 10K clients, 100 shuffling proxies.



(b) Varying the number of insiders under 100K clients, 100 shuffling proxies.



(c) Varying the number of shuffling proxies under 10K clients, 100 insiders.



(d) Varying the number of shuffling proxies under 100K clients, 100 insiders.

Figure 3.9: The number of shuffles needed to save 80% and 95% of benign clients from malicious insiders.

From Figures 3.9a and 3.9b, we can see that the number of shuffles needed to save the same percentage of benign clients grows almost linearly with the increase in the number of insiders. More shuffles indicate a longer time to mitigate an attack, but it also means that

attackers have to devote much more effort to recruit more insiders. Figures 3.9c and 3.9d reveal that when the numbers of clients and insiders are fixed, the number of shuffles needed drops as more shuffling proxies are used. The decline of the curves is much steeper when the proxy number is lower than the insider number, but it stabilizes after the proxies outnumber the insiders. Moreover, the narrow confidence intervals of all the data points indicate that the performance of our shuffling algorithm is reliable and predictable.

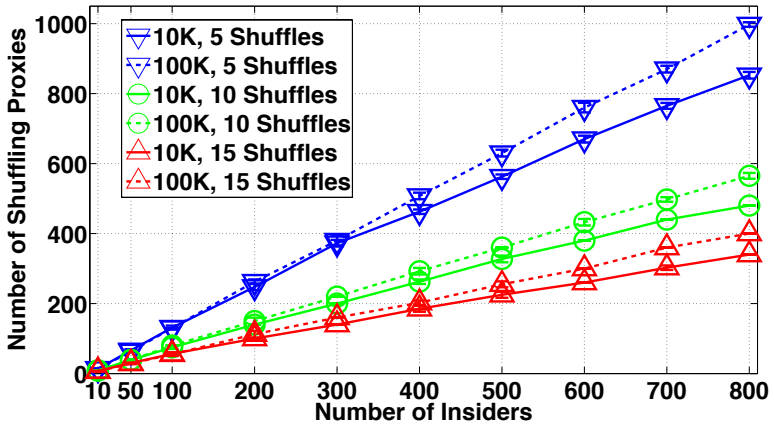


Figure 3.10: Number of proxy nodes needed to save 95% of benign clients within 5, 10, and 15 shuffles, with 10K and 100K clients and an increasing number of insiders.

Moreover, it is worth noting that the increase from 10,000 to 100,000 clients caused almost no difference in the simulation results. Instead, the figures show that the ratio between the number of shuffling proxies and the number of insiders is the primary factor deciding the number of shuffles needed to achieve a pre-defined goal. To further explore the relationship between the proxy number and the insider number, we performed experiments with varying insider numbers to find out the minimum number of shuffling proxies required to save 95% of benign clients within 5, 10, and 15 shuffles, respectively. The results are displayed in Figure 3.10, with the number of insiders ranging from 10 to 800. The solid lines represent experiments with $10,000$ clients while the dotted lines denote simulations using $100,000$ clients. This figure reflects a close-to-linear relationship between the number

of required shuffling proxies and the number of insiders in achieving a constant security goal. Again, a ten fold increase in the client population only showed minor differences in the results, and the 99% confidence intervals are almost negligible.

Isolating Persistent Bots Attacking Open Internet Services

To assess the effectiveness of the shuffling mechanism in mitigating DDoS attacks on open Internet services, we simulated intensive attacks initiated by botnets composed of large numbers of persistent bots. Instead of assuming all benign clients and bots are present at the start of the simulations, we adopted a more realistic approach that assumed both benign clients and bots arrive in a Poisson process. In our experiments, the arrival rate of bots was 5,000 per three shuffles while that of benign clients was 100 per three shuffles. We also simulated a number of replica servers to host and shuffle clients and bots. Similar to the previous simulations that used fixed numbers of shuffling proxies, constant numbers of shuffling replica servers were employed throughout individual simulation runs for the convenience of comparison. Replica servers assigned with any number of persistent bots were regarded as attacked. By observing the numbers of replica servers under attack, the shuffling algorithms decided the total number of clients to be assigned to each replica server. Benign clients, together with bots, were randomly assigned to replica servers to fill up all available spaces. Across our experiments, we varied the numbers of benign clients, persistent bots, and shuffling replica servers to study the performance of the shuffling algorithms in saving affected benign clients under different conditions. All simulations were repeated 30 times to plot the mean and 99% confidence interval.

First, we ran simulations with 1,000 shuffling replicas and varied numbers of benign clients and bots. The results are plotted in Figure 3.11. One can see that the number of shuffles required to save 80% and 95% of benign clients rises slowly with the increase in the populations of bots and clients. In the worst case, a ten-fold increase on the bot number results in a less than three-fold increase on the shuffle number; for a given number of bots, a five-fold increase on the benign clients only introduces less than 70% (40) more shuffles

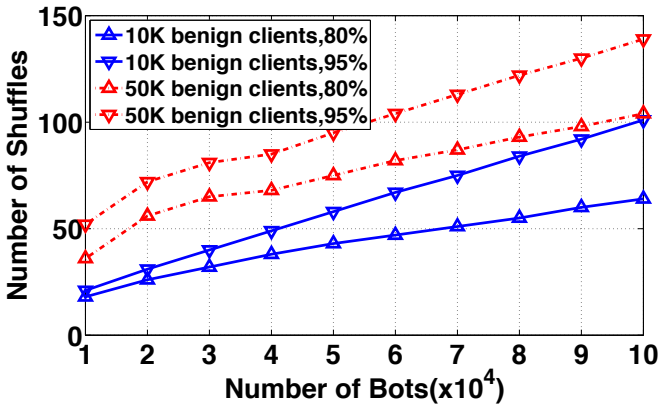


Figure 3.11: Number of shuffles to save 80% and 95% of 10^4 and 5×10^4 benign clients, with 1,000 shuffling replica servers and varying bot numbers.

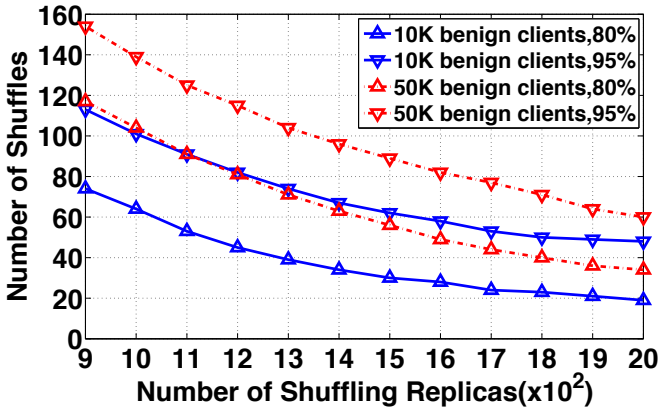


Figure 3.12: Number of shuffles to save 80% and 95% of 10^4 and 5×10^4 benign clients, with 10^5 bots and varying shuffling replica server numbers.

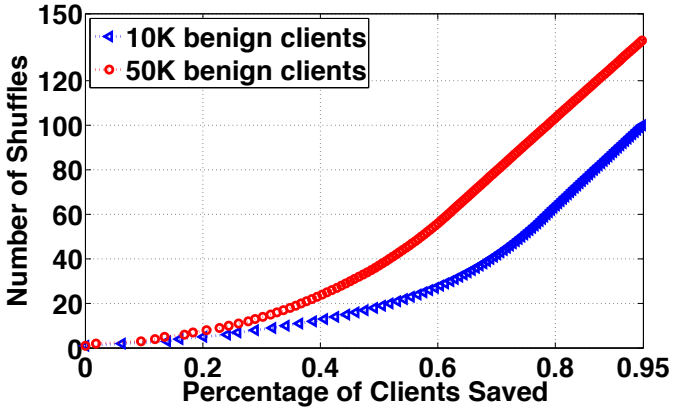


Figure 3.13: Cumulative percentage of saved benign clients vs. number of shuffles, with 10^5 bots, 10^4 and 5×10^4 benign clients.

to save the designated percentage of clients.

Next, we changed the number of shuffling replica servers while keeping the client population (10^4, 5×10^4) and bot population (10^5) constant. The curves in Figure 3.12 show that the number of shuffles needed to save the same percentage of benign clients drops steadily when we add more replica servers.

One interesting pattern consistent across both figures is that in most cases, the number of shuffles it takes to save 95% of benign clients is more than 40% higher than what is needed to save 80% of benign clients. To explore this pattern in greater detail, we recorded the number of benign clients saved in each shuffle and plotted a cumulative percentage graph in Figure 3.13. Apparently, the early shuffles were able to separate more benign clients from bots than the latter shuffles. The reason is that, as more benign clients were saved, it became harder to separate the remaining ones out because bots gradually accounted for a greater percentage of the remaining population.

3.6.2 Overhead Evaluation

The overhead introduced by our MTD solutions are largely determined by their respective architectures. To evaluate the overhead caused by the MOTAG architecture discussed in Section 3.2 and the cloud-enabled server replication architecture presented in Section 3.3, we implemented a proof-of-concept prototype for each and ran various experiments on them.

Evaluating The Overhead Of The MOTAG Architecture

The MOTAG architecture mainly introduces two aspects of overhead to the communication between clients and the application server, namely proxy-based communication indirection, and client-to-proxy shuffling.

First, to assess the overhead introduced by proxy-based traffic indirection, we selected 10 geographically distinct U.S. nodes from PlanetLab [59] to form five end-to-end flows. We also randomly picked 24 other nodes that spread across the country to serve as proxies. We measured the latency and throughput for both direct and indirect communications of the five

Table 3.2: Latency overhead introduced by proxy indirection.

	Direct	Indirect			
	RTT	Mean RTT	Overhead	Max RTT	Overhead
1	63ms	104ms	63.35%	143ms	125.41%
2	86ms	99ms	15.64%	128ms	49.45%
3	83ms	102ms	23.73%	133ms	60.47%
4	90ms	112ms	23.77%	131ms	45.18%
5	84ms	107ms	27.73%	120ms	42.48%

Table 3.3: Throughput overhead introduced by proxy indirection (Mb/s).

	1	2	3	4	5
Direct	90.66	83.46	86.24	123.30	121.20
Indirect	15.20	14.46	13.99	15.97	14.09

flows. The results are shown in Table 3.2 and Table 3.3, respectively. For these experiments, SSH tunneling through individual proxy nodes was employed to relay traffic between end nodes. Round trip time (RTT) numbers in Table 3.2 were obtained by bouncing short TCP messages back and forth between the end nodes of each flow 100 times to compute the mean. The throughput number of each flow in Table 3.2 is the average of ten Iperf [104] sessions. Apparently, the impact of proxy-based indirection on latency (mostly less than 30%) is much less significant than its influence on throughput. The drop on throughput is not only caused by the extra relay hop but is also the result of message encryption and decryption enforced by SSH agents. In fact, different crypto strategies, including no encryption, can be listed as options when implementing MOTAG-based systems. Users can make informed decisions based on the nature of the protected application.

The time needed to shuffle clients among different proxy nodes determines the agility and usability of MOTAG against insider attacks. Quick shuffles will make it harder for attackers to follow and have insiders quarantined faster. At the same time, benign but shuffled clients will suffer less severe service disruptions. To quantify the impact of our

Table 3.4: Time to switch between two proxy nodes (seconds).

	1	2	3	4	5
MEAN	0.514	0.512	0.509	0.546	0.530
MAX	0.677	0.773	0.693	0.714	0.753
MIN	0.291	0.208	0.249	0.357	0.214

system to the end users, we measured the time it takes to switch a client from one proxy node to another. For this experiment, we chose five geographically dispersed nodes from PlanetLab to serve as the destination servers. We randomly picked one other node to play the role of the authentication server. We timed the entire process by which our local client receives notification from the authentication server, discards the current proxy, and then connects to the new proxy to reach back to the destination server. During this process, the authentication server sends a session ticket to both the client and the new proxy node. The client will present this ticket to the proxy for identify validation. Only after that, the new proxy node will start forwarding packets for the client. We used another 8 PlanetLab nodes as proxies and switched between them. The average, maximum, and minimum proxy switching times corresponding to each destination are listed in Table 3.4. The numbers are fairly small yet consistent. A less than one-second proxy switching time will enable fast client-to-proxy shuffling without causing significant service disruption for most non-realtime applications.

Evaluating The Overhead Of The Cloud-Enabled Architecture

To study the overhead of the cloud-enabled architecture with moving replica servers, we built a proof-of-concept prototype as described by Figure 3.14. We implemented two replica servers and a coordination server/coordinator on separate Amazon EC2 [60] micro instances. We used 60 PlanetLab nodes as clients. Each replica server runs a simple web server that displays static webpages fetched from a network-mounted storage. The web server logic is

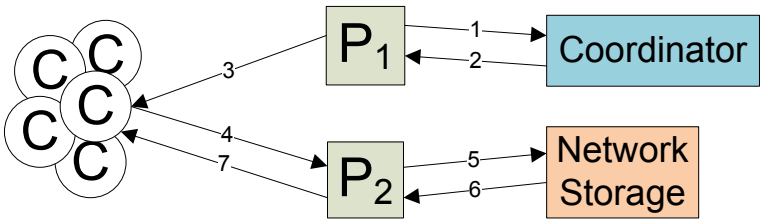


Figure 3.14: System prototype (C – Client, P – Replica Server).

written in Node.js [105].

Initially, all clients are served by replica P_1. When a simulated attack is triggered on P_1, P_1 notifies and consults the coordinator about the next step (step 1). In general, the coordinator will make shuffling decisions either by running the greedy algorithm or by looking up the pre-computed client-to-server assignment tables generated by the dynamic programming algorithm. The decisions are sent back to the attacked replica to guide subsequent shuffling operations. In our case, the coordinator will respond by asking P_1 to redirect all clients to replica P_2 (step 2). As a result, P_1 will proactively send redirection notifications to all clients (step 3). Upon receipt, clients will start to contact and reload the current web page P_2 (steps 4, 5, 6, and 7).

Unlike conventional HTTP-based communications that start with client requests, the redirection operation is always initiated by an attacked replica server. To send unsolicited messages to HTTP clients, we take advantage of the WebSocket [106] technology multiplexing HTTP(S) ports (80 and 443). WebSocket is supported by all major browsers. Therefore, the adoption of our mechanism does not depend on extra software being installed on the client side. For this prototype, our server injects a snippet of Javascript code (40 LOC) into the requested webpage to establish a WebSocket between clients and the replica server.

With this prototype system, we studied the network overhead of redirecting clients from the attacked replica server (P_1) to a new replica server (P_2). This prototype ran Firefox browsers (v17.0) on up to 60 geographically distributed PlanetLab nodes as clients, who all visited the same webpage (246KB) initially served by replica server P_1. The time for all

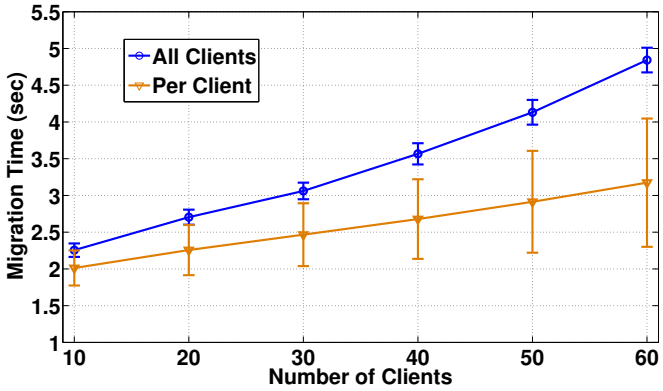


Figure 3.15: Client migration time between two replica servers.

clients to complete step 1-7 (i.e. redirection time from P_1 to P_2) is shown in Figure 3.15. In this figure, the upper curve shows the time it took to successfully redirect all clients, while the lower curve reveals the average redirection time per client. The measurement for each data point was repeated 15 times to obtain the mean and 95% confidence interval. The results show that we can re-assign 60 clients in less than 5 seconds. This indicates a low overhead of client re-assignment operations, but we still expect more room for improvement because our server program was single-threaded and not optimized at all. Compared to the proxy switching times displayed in Table 3.4, Figure 3.15 shows a longer delay of redirecting one client between two replica servers. The difference is largely due to the need to reload the webpage in the latter case. This cloud-based prototype was mainly developed to show the feasibility of the approach and provide a lower bound on performance. The system is not expected to slow down with more replica servers because all replicas act independently and in parallel.

Chapter 4: Protecting Multi-path MANET Communications Using Capability

4.1 Introduction

Compared to the Internet, mobile ad hoc networks (MANETs) lack a clear line of defense and are therefore more vulnerable to flooding Denial-of-Service (DoS) attacks. To protect unicast MANET communications from flooding attacks, Alicherry et al. introduced capability mechanisms [48–50] that enforce quota-based traffic control on each end-to-end flow. However, existing capability defenses for MANETs focus on preventing DoS attacks when a single path is used to route traffic of each flow; they do not offer protection against DoS attacks if end nodes communicate through multiple paths. In fact, multi-path routing is a desired function for MANETs because it provides better load balancing, improved fault tolerance, and more efficient use of bandwidth. Unfortunately, it also generates a new threat vector multiplying the potential impact of DoS attacks. Indeed, even if the bandwidth consumption on individual routing paths can be constrained within the assigned quota, the aggregated multi-path traffic may become much higher than that.

This section presents the design, implementation, and evaluation of *CapMan*, a capability-based security mechanism that prevents DoS attacks exploiting multi-path MANET communications. CapMan is designed to be installed on every node and enforce capability limits that effectively regulate all end-to-end network flows. CapMan consists of two main protocols: the capability distribution and the capability enforcement. The capability distribution protocol empowers the responder of an end-to-end flow to issue and distribute a capability to the initiator and the intermediate nodes along all the employed routing paths. To comply with the protocol, the initiator of a flow has to request the responder's explicit

permission before sending any data packets. To grant access to the initiator behind a received connection request, the responder sends a capability token back to the initiator. This capability token is not only the passport that allows the initiator to send data packets, but it is also cached by all intermediate nodes for restricting the throughput of the flow. To perform traffic policing, the capability enforcement protocol ensures the capability constraint is enforced on a per-hop basis across all employed routing paths. To achieve that under the context of dynamic topology and multi-path routing, intermediate nodes on different paths exchange bandwidth accountability and consumption reports periodically so that all cooperating nodes are informed of the global network state in a scalable and consistent manner. The broadcast of the bandwidth usage reports empowers individual nodes to regulate end-to-end traffic across multiple dynamic routing paths, as dictated by the per-flow capabilities.

The proposed mechanism can effectively identify and mitigate sophisticated DoS attacks that target multi-path routing protocols, even when both the initiator and the responder are colluding malicious insiders. Aside from the capability set by the responder, which is referred to as *flow capability*, we also introduce a *default capability* that defines the upper bound of all flow capabilities. If a malicious responder distributes a flow capability greater than the default capability, the default value will be used, and the suspicious responder can be detected and further investigated.

4.2 Threat Model & Assumptions

Due to the dynamic nature of MANETs, there is no infrastructure or other form of network separation as seen in wired networks. As a result, attackers are given great flexibility in selecting targets and attack methodology. For instance, attackers can simply jam the telecommunication channel, or dump massive junk packets onto a victim to exhaust its constrained computation, communication, and energy resources.

To prevent arbitrary attacks initiated by external attackers, cryptographic methods should be implemented to secure the physical communication channels. In particular,

public-key cryptography needs to be employed for the purpose of node authentication, channel encryption, and packet validation. Only authorized and authenticated hosts should be allowed to join a protected network and send packets to other members. For our research, we assume these fundamental security measures are already in place to prevent external attacks. In particular, we assume each authorized node in a MANET is identified by a unique and attested public/private key pair. However, given the dynamic and autonomous nature of MANETs, it is unrealistic to rely on a single CA (certificate authority) for reliable and continuous key management service. Instead, decentralized PKI architectures that employ distributed CAs [84–86] can be adopted to provide reliable and tamper-proof key issuance and attestation.

Cryptographic protocols can effectively block external attackers but are inadequate to prevent malicious insiders from attacking within the trusted group. In an effort to address insider threats, this part of the work focuses on protecting MANETs from flooding DoS attacks launched by malicious insiders. In particular, our solution enforces a capability-based mechanism on each unicast packet flow. For this work, we use the term "flow" to refer to a uni-directional packet stream traveling from the initiator to the responder. The initiator is the node from which the packet stream originates, while the responder is the destination node of the packet stream. Therefore, a UDP stream is considered one flow from the initiator (source) to the (responder), while a bi-directional TCP connection is regarded as two flows in two opposite directions.

Multi-path routing [107, 108] is a routing functionality that allows packets of a single flow to be delivered using multiple paths between the two end nodes. It is a desirable feature for MANET communications because it can lead to better load balancing, improved fault tolerance, and higher aggregate throughput. Unfortunately, it also exacerbates the possibility and intensity of flooding DoS attacks. In hostile environments, any node in a MANET can be malicious and start these attacks against others. Based on their targets, these flooding attacks can be classified into the following two categories:

- **End Node Targeted:** Flooding attacks targeting end nodes aim to saturate the

bandwidth or exhaust the CPU and power energy of a flow responder, making it unavailable to legitimate flows. Such attacks can be carried out by an individual malicious node or multiple colluding nodes. A malicious initiator can exploit multi-path routing to increase its traffic load, while malicious intermediate nodes can further augment the attack through packet forging or replay.

- **Intermediate Node Targeted:** Another form of DoS attacks aims to overwhelm the bottleneck intermediate nodes in the network. By purposely directing flows with high traffic volumes to the victim, malicious upstream nodes can congest or even disable the resource-limited target downstream. This can cause substantial packet loss and lead to debilitating results for other flows that share the same link. Although active queue management (AQM) schemes like [109] are available to provide fairness among flows, they are insufficient for differentiating normal high throughput communications from malicious traffic or prioritizing traffic of critical applications when a DoS attack happens. In addition, colluding malicious nodes can take advantage by forming large numbers of flows to achieve high aggregate throughput.

This work aims to mitigate DoS attacks targeting both end nodes and intermediate nodes in MANETs. Next, we will explain in detail our comprehensive, capability-based solution named CapMan.

4.3 Capability System Overview

A high-level flow control of the CapMan system is depicted in Figure 4.1. CapMan consists of two interdependent components, namely capability distribution and capability enforcement. Each node in the network runs an identical copy of the CapMan system. The operation of CapMan is triggered by packet reception. A flow responder executes the capability distribution module upon receiving a connection request packet. If the connection is granted, the capability distribution module then generates a capability token and sends it back to the initiator. An intermediate node, upon seeing a capability packet, updates

its local capability table and forwards the packet to the next hop until reaching the flow initiator. When a data packet reaches an intermediate node, the node runs the capability enforcement module to decide whether to forward this packet. Every node periodically puts together and broadcasts capability summary packets that advertise the node's local capability status. The receivers activate the capability enforcement component to process these packets and adjust their local capability settings accordingly.

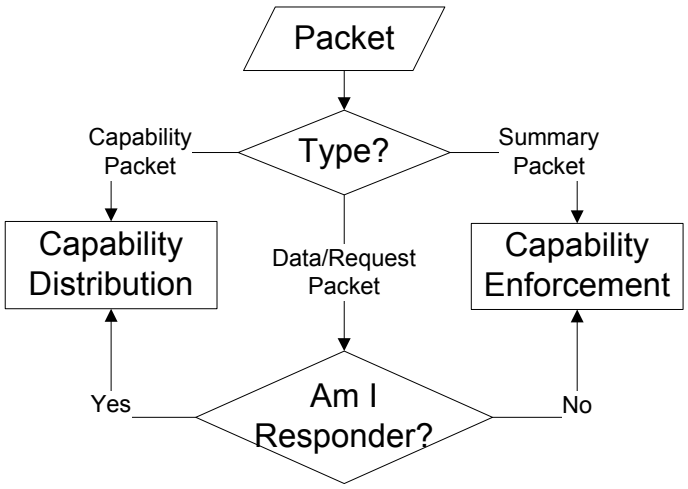


Figure 4.1: Overview of the flow control in CapMan system.

4.3.1 Capability Distribution

Capability distribution is the process of capability issuance from the responder to the initiator, which accurately distributes the capability among all employed routing paths. The solution to capability issuance problem is straightforward and has already been studied by other researchers [48]. However, it is more challenging to ensure that intermediate nodes along all paths are informed about the assigned flow capability in a dynamic network. To maintain up-to-date knowledge of the multi-path routing information, CapMan records the routing path for each data packet in a *Path Recording Header* (PRH). By inspecting the PRH of received packets, the responder can keep track of the routing paths currently used

by the flow. When a new route is detected, the responder will immediately notify the nodes on that path about the flow's capability. In this way, every forwarding intermediate node is guaranteed to be informed about the flow capability, which is used as the yardstick for capability enforcement.

4.3.2 Capability Enforcement

Capability enforcement requires all nodes along different routing paths to collaborate in a distributed manner on enforcing the overall capability requirement. From the capability distribution process, intermediate nodes are informed about the per-flow capability assigned by the responder. To make sure this capability is adhered to at all times, nodes on every routing path not only have to watch their local traffic, but they also need to be aware of the data rate of the same flow on all other routes. In CapMan, this is accomplished by having all nodes periodically exchange bandwidth consumption reports (i.e. capability summaries). With the information collected globally, each node can then dynamically adjust its local rate enforcement policy with regard to the flow-wide capability fulfillment. Therefore, CapMan can ensure that the overall capability is effectively enforced across multiple routing paths.

It is worth noting that a possible alternative to this solution is to split the per-flow capability among all employed routes. Admittedly, this approach could save the effort and bandwidth from conducting summary exchange. Nevertheless, considering the dynamic topology of MANETs and the imbalanced bandwidth distribution among different routes, it is almost impossible to draw an optimal traffic partition plan that can effectively utilize the capacity of each path. As such, splitting capability may adversely affect the throughput of legitimate flows while leaving available network resources underutilized. Therefore, we believe that routing decisions made by the underlying multi-path routing protocol should not be unnecessarily restrained.

4.4 Capability System Design and Implementation

Before diving into the design and implementation details of the capability distribution and the capability enforcement components, it is necessary to introduce the physical pieces that collectively constitute our capability system.

4.4.1 Packet Format

Capability Packet

First and foremost, a new type of capability packet is needed to fulfill the goal of capability distribution. This packet is created by the flow responder. It contains several key pieces of information so that an updatable capability is associated with a unique flow and can cover new paths in the case of routing updates. For the sake of flow binding, the responder derives a unique *flow identifier* (*FID*) for each incoming flow. The *FID* is composed of the initiator's ID, the responder's ID, and a unique sequence number issued by the responder. The *FID* is encoded into corresponding capability packets along with the assigned capability value. This offers two important benefits. First, for the actual data communication, the initiator only needs to include the *FID* in the data packets instead of the entire capability token, thereby lowering the overhead. Second, the *FID* can later be used to aggregate per-flow capabilities for a particular node, so as to flag attackers who attempt to gain advantage by launching a great number of flows. A capability packet also embraces a *path recording header* (described below) that dictates the routing path it should be forwarded through. This enables the flow responder to disseminate the capability token to newly discovered routes. The capability packet is signed by the responder using his group credentials so that its authenticity and integrity can be verified by other nodes.

Path Recording Header

To detect routing updates, a *path recording header* (*PRH*) is added to regular data packets. It records the routing path through which a data packet is delivered. *PRH* is incrementally

filled by each intermediate node during the forwarding process. On each hop, the processing node appends his signed ID onto the *PRH*. Therefore, when a data packet arrives, the responder would be able to inspect the *PRH* to get a complete route sequence. To reduce the transmission overhead, we employ a signature aggregation technique [110] that compresses the signatures of all intermediate nodes into one small signature field in the header of data packets. When a new routing path is found, the responder clones this header into a capability packet that is sent back to the initiator, reversing the recorded path.

Summary Packet

A new summary packet is introduced for the purpose of *capability enforcement*. A summary packet contains a list of *PRH* and rate value pairs for each flow, reporting the flow's throughput on a particular path during the previous time window. Summary packets sent by different nodes are the vehicles for cross-path collaboration on capability policing. All nodes that serve a flow are required to generate and broadcast summary packets periodically. As a result, they will be able to gain a comprehensive view of the global capability consumption.

4.4.2 Data Structures

Capability Table

Every node in CapMan maintains a capability table that archives each flow and its associated capability. This is a hash table that indexes each flow with the corresponding *FID*. The table is updated when a new capability packet is received. Old capability entries are purged at flow termination or expiration.

Summary Table

The per-node summary table is a data structure that stores summary messages reported by nodes on other routing paths. The table has a hierarchical structure that uses *FID*, *PRH*, and *node ID* to separate flows, routing paths, and reporting nodes, respectively. Each entry

in the table corresponds to one summary packet the processing node receives. An entry is updated when a new matching summary packet is received from the same node.

Leaky Bucket

Per-path throughput of any flow is computed once per time window, and the result is broadcasted via a summary packet. However, a periodical throughput check on all nodes is not enough to guarantee that the capability is abided all the time. Well-tuned bursty traffic can potentially bypass such intermittent inspections. Therefore, leaky buckets are installed on all nodes to cope with attacks exploiting bursty traffic and to provide enhanced quality of service (QoS). Each bucket is associated with one route of a flow. Conceptually, the property of rate limiting by leaky bucket is consistent with our goal of enforcing capability. In addition, it can effectively curb the degree of burstiness. Therefore, the combination of leaky bucket and regular summary exchange ensures that the capability is enforced regardless of traffic patterns.

4.4.3 Capability Distribution

To establish a transport layer flow for data communication, the initiator has to send a connection request to the responder, asking for the responder's explicit approval. The request can be either a TCP SYN packet or a special UDP packet. If the responder is willing to accept the connection, it needs to reply to the initiator with a capability packet that specifies the maximum data rate for the flow. The responder decides the capability for the flow in consideration of all necessary factors, including available bandwidth, the application behind this connection, as well as the capabilities already assigned to the same initiator. Complex algorithms can be developed to achieve a sound mapping between available resources and the assigned capabilities but are beyond the scope of this paper.

Once the capability is decided, the responder creates a capability packet and sends it back to the initiator along the reverse route of the request packet. Each intermediate node along the route extracts the capability and saves it into its local capability table under

the pertinent flow. This cached capability will serve as the passport for future packets of the same flow. Assuming the route stays unchanged within a single roundtrip time, the capability packet should eventually reach the flow initiator. After that, the initiator begins to send data packets to the responder using the capability. Each data packet only includes the *FID* of the flow instead of the entire capability token to reduce the transmission overhead. In case no capability is issued or the cached capability expires, a minimal rate limit would be enforced on the flow to only allow connection requests to proceed. When a new route is used for an existing flow, the responder will construct another capability packet and send it back to the initiator, reversing the new route.

4.4.4 Capability Enforcement

The process of enforcing the advertised capabilities is a combination of local policing and flow-wide message exchange on each node. Local policing is conducted continuously by the per-path leaky bucket. Cross-path message exchange happens periodically on pre-defined time windows with a certain degree of freedom. The purpose of undertaking such message exchange is to let all intermediate nodes be aware of the flow-wide throughput and thus adapt their bucket leaking rate.

Local Capability Policing

For each incoming data packet, the processing node inspects the packet header to extract its flow and path information and searches the local capability table for a match. If found, the data packet is inserted at the end of the corresponding leaky bucket or dropped if the bucket overflows. However, if no capability is available for this flow, the packet will be placed in the bucket for anonymous traffic with a minimal capability. Packets in different buckets are leaked at different rates to the next hop. The per-bucket leaking rate is updated when summary messages are received from other routing paths of the same flow.

After forwarding a data packet, the size of the packet is added to the per-path traffic counter, which will be reset at the start of the next window. Before the current time

window expires, the recorded traffic volume is used to compute the local throughput of the corresponding flow. The result of that computation will be included in the summary packet broadcasted to other nodes. The size of throughput calculation window should approximate a pre-defined value. However, to avoid generating too much extra traffic and thus clogging the network channel, the window should not be too small (e.g. $< 5sec$). On the other hand, the window size should not be too large (e.g. $> 30sec$). Otherwise, it may inhibit natural flow dynamics. One feasible practice is to combine the summary packets with the "Hello" messages generated by the underlying routing algorithm. Consolidating these two types of messages can lower the overhead by reducing the number of broadcasts sent by each node. In addition, using the neighbor-discovering time window for throughput calculation can help the capability enforcement protocol quickly adjust to topology changes in a cost-effective manner. However, a certain degree of randomness should be enforced on all timers to avoid global broadcast synchronization. To further reduce the chance of broadcast collision, the 802.11 CSMA/CA protocol can be adopted for simplicity, although more sophisticated algorithms are available [111, 112] for improved efficiency and fault tolerance.

Leaking Rate Determination

DoS attacks are hard to stop under the multi-path routing paradigm in that it not only boosts potential throughput for legitimate users but also widens the gate for attackers. Without a collaborative defense across all employed routes, attacks can be launched simply by approaching the assigned capability on each routing path while overwhelming the victim with the number of routes. To combat flooding attacks that exploit multi-path routing, our solution offers every cooperating node a global view that aggregates a flow's throughput on all employed paths via summary message exchange.

On receiving a summary packet, a cooperating node will break it down and inspect every ⟨Path, Throughput⟩ tuple inside. Tuples that represent paths going through the processing node will be discarded. Others will be used for updating the local summary table. Stale summary packets (whose sequence numbers are smaller than or equal to a

matched table entry) will be dropped. Each update in the summary of the capability table triggers a recalculation of the leaking rate for all pertaining leaky buckets. The algorithm for updating the leaking rate is described in Algorithm 3 and explained below.

Algorithm 3 Algorithm for updating local leaking rate.

```
for all SummaryEntry_i of F_K do
    Count NumOfPaths
for j = 1 → NumOfPaths do
    Calculate Thruput_j
    TotalThruput+ = Thruput_j
diff = CAP_{F_K} − TotalThruput
if |diff| > Threshold then
    if diff < 0 then
        LeakRate− = Threshold
    else
        LeakRate+ = Threshold
else
    LeakRate+ = diff
```

For a given flow F_K, the leaking-rate-updating algorithm first calculates the number of routing paths that are used for packet forwarding. This is done by traversing the summary entries under F_K and finding all unique *PRH*s. In addition, the algorithm computes the throughput of each route for the last time window. Ideally, for any route that the computing node (CN) is not on, CN should receive a summary message sent from each of its member nodes. When all members of a route report consistent throughput values, either the median or the mean of the reports could be selected as the final throughput value for that route. Nevertheless, under a more realistic and adverse environment, CN may receive summary messages reporting throughput values in a wide range, for some attackers may intentionally send false summary data to mislead other nodes. In that case, the median value of all received reports should be used because it is more robust to thwart anomalous values than the mean. We will show in Section 4.5.2 that unless malicious nodes can dominate the route in number, the damage will be limited if the median is used.

By aggregating per-route throughput for F_K, we can get the overall throughput from all routes of the flow. We then compare this value with the assigned capability to get the $diff$ in the algorithm. A negative $diff$ means the flow throughput already exceeds the

capability, and the leaking rate should be dropped. If the $diff$ is positive, we know that the capability is underutilized, and the throughput can be increased by as much as $diff$. However, the value of $diff$ is the flow-wide deficit or margin and should not be cut or filled in fully by nodes on any individual route. Otherwise, it would introduce recurring and detrimental fluctuations on the flow throughput in which spikes can be several times higher than the assigned capability. To avoid this undesirable consequence, we introduce a threshold $T = \frac{C_K}{P}$, where C_K is the capability for flow F_K and P represents the total number of routing paths used to transmit F_K. This threshold is designed to contain the variation range of leaking rate (LR) between consecutive time windows. When any node updates its LR, it has to make sure that $|LR_{new} - LR_{old}| \leq T$. In the worst case, we assume the overall throughput of F_K at moment A is $Thruput_A < C_K$. As soon as intermediate nodes that route packets for F_K notice this, they all immediately increase their local LR by T. At moment B when all updates are complete, we can derive that the new overall throughput $Thruput_B = Thruput_A + T \times P = Thruput_A + C_K < 2C_K$. Thus, we show that under CapMan, the throughput of any flow F_K can never exceed $2C_K$. In fact, the threshold defined by our algorithm is not the only way to confine the variance of per-flow throughput. Alternative algorithms, such as TCP congestion control routines, can also be tailored for this purpose.

4.5 Discussion

4.5.1 Improvement Over Existing Solutions

Existing capability-based mechanisms [20–22] work well under the context of uni-path routing. They use capability distribution algorithms similar to CapMan. However, the assigned per-flow capabilities are independently enforced by each route. Intermediate nodes on one route are clueless about other routes that also carry traffic for the same end-to-end flow. This is not a problem for the Internet where a single, static routing path is usually employed to deliver packets for a unicast communication. Unlike the Internet, MANETs feature more

dynamic topologies and more resource-constrained nodes. Multi-path routing is desirable and more likely, given the ad hoc nature of the network. Unfortunately, multi-path routing is not accounted for by the existing capability-based mechanisms for MANETs [48–50]. To take advantage of multi-path routing, flooding attackers can greatly boost their throughput by employing as many disjoint routes as possible, even if the throughput along each route stays under the capability limit.

CapMan addresses this threat by adopting distributed capability enforcement. As all relay nodes periodically broadcast route-specific bandwidth consumption reports for every residing flow, intermediate nodes along any route will discover the global throughput for any flow of interest. Based on this knowledge, a per-flow capability can be collaboratively enforced across multiple routes, as discussed in Section 4.4.4. The advantage of CapMan over existing solutions will be further illustrated via simulations, in Section 4.6. In addition, our capability-based mechanism works with any connected network topology and is independent of MANET routing protocols.

4.5.2 Security Analysis

This section presents an analytical evaluation of CapMan's effectiveness on mitigating insider DoS attacks. In particular, we analyze CapMan's resilience against brute-force flooding attacks and abusive attacks targeting on CapMan's internal operations. Attacks on end nodes and bottleneck intermediate nodes are both addressed.

Brute-force Flooding Attacks

To mount a flooding DoS attack on any node in the network, an attacker only needs to establish an end-to-end connection with the target and flood through the communication channel. In this case, the attacker plays the role of flow initiator, and the victim is the responder. The attack can be amplified by one or more attackers sitting on the paths between the end nodes. However, unless the attacker can take over a route completely, the attack will be blocked by the friendly nodes that realize the aggregated throughput

from the initiator is greater than the assigned capability. Indeed, in CapMan, every node is responsible for policing the traffic received from its immediate upstream node. Thus, an invisible, cross-route line of defense is formed by all friendly nodes, which can stop flooding attacks from propagating to the rest of the network. For example, if an initiator sends an excessive amount of traffic to a flow, the first hop along any route will be able to detect that violation and seize forwarding packets for the flow until the overall throughput falls below the capability. Attempts by malicious intermediate nodes to intensify the attacks will be stopped by their next hops in the same way. One exception occurs when the attacker is in the immediate neighborhood of the responder. In that case, the responder will need to block the attacking neighbors or move to a new network location to establish new neighbors.

Abusive Attacks

CapMan is robust against abusive attacks that attempt to exploit or disrupt our capability-oriented protocols in the following ways: Colluding end nodes may establish inordinate capabilities or an excessive number of flows to swamp bottleneck intermediate nodes; malicious insiders may bombard the capability request channel; and attackers may become intermediate nodes and attack established flows by abusing the internal operations of CapMan. We will discuss each potential threat and the corresponding counter-measures below.

Containing colluding end nodes Malicious initiators and responders can collude in launching DoS attacks that aim to deplete the constrained network, CPU, or energy resources of intermediate nodes along multiple paths. A malicious responder can deliberately assign a high capability that allows an immense packet rate for a flow to consume as much bandwidth as possible. This type of attack can be easily detected when intermediate nodes see a particularly high capability assigned to and used by any flow. Since every capability token encodes the identities of the associated flow initiator and responder, the colluding nodes should be quickly pinpointed and removed. Further, a default capability can be pre-configured offline by the network administrator, specifying the upper-bound of any flow capability. However, stealthier colluding nodes can set up a plethora of flows with normal

or even small capabilities to attack and avoid being detected. In such cases, per-flow capabilities need to be aggregated for particular initiators or responders for attack detection.

For automatic attack mitigation, active queue management (AQM) schemes can be employed as the foundation to ensure fairness among flows. On top of that, when congestion occurs, the packet-dropping probability derived by AQM can be further compounded by a factor set proportional to the size of corresponding per-flow and aggregated capabilities. In other words, the packets of flows or end nodes bearing considerably higher capabilities are much more likely to be dropped. As a result, colluding end nodes will be penalized for issuing excessive capabilities, and their throughput will be largely contained.

On the other hand, there might be nodes that run special or critical applications that warrant the use of high capabilities. Network administrators can make a list of such nodes and inform the entire network about these special cases.

Defense against DoC attacks From the earlier research on capability-based defense [76], we learned about a variant of DoS attacks that floods the capability request channel to deny the access of legitimate requests. Such an attack is addressed as a denial-of-capability (DoC) attack. To combat DoC attacks, CapMan ensures that only a small portion of the overall bandwidth is assigned to the capability request channel. Moreover, CapMan requires every capability request packet to be digitally signed by the corresponding initiator using its unique private key. Hence, source-based fair queuing can be adopted to ensure that capability requests from different nodes are treated fairly along each hop of delivery.

Preventing flooding attacks by malicious intermediates As discussed earlier, malicious intermediate nodes may exploit established connections to launch or assist flooding attacks. Given that the forwarding path of each packet is recorded and attested, it is difficult for intermediate nodes to impersonate the initiator by flooding with forged packets. Instead, malicious intermediate nodes can store and replay past packets in a flooding attempt. However, once the aggregate throughput of the exploited flow goes beyond the assigned capability limit, the next friendly node along each route will throttle the attack traffic

to ensure global capability compliance. The excessive packets that overflow the downstream leaky buckets will be discarded away from the target.

The only exception happens when a malicious intermediate node is an immediate neighbor of the targeted node. In that case, the attack traffic incorporating replayed packets will directly hit the victim. As discussed earlier, to stop an attack performed by immediate neighbors, the victim node will need to directly block the attacking neighbors, if possible, or move away from the attackers to establish new, friendly neighbors.

Robustness against dishonest intermediates In CapMan, friendly intermediate nodes depend on summary messages from other nodes to frame the global throughput, which is the foundation for local leaky rate computation and traffic policing. High accuracy and delivery rate of the summary reports are essential to maintaining effective cross-path collaboration. To disturb the delivery of summary packets, malicious intermediate nodes may initiate a black-hole attack by not generating or forwarding any summary packets. However, black-hole attacks are unlikely to cause much impact without having attackers dominate the network in number. Since all summary reports are broadcasted, friendly nodes will be able to disseminate and receive these packets unless they are surrounded by only attackers.

More sophisticated attackers may attempt to distort the flow-wide throughput image seen by others by advertising false summaries claiming fabricated rate values. For example, to help an attacking initiator gain higher throughput on other routes, the colluding intermediate nodes can substantially downplay their reported per-path throughput of the malicious flow. Conversely, they can drastically overstate a benign flow's throughput on their routes, with the goal of bringing down the overall throughput of the benign flow and starve its initiator.

To limit the damage that can be caused by a few dishonest nodes, all intermediate nodes on every routing path broadcast their own versions of summary reports. As a result, one should receive multiple reports for each route originated from different nodes. If all nodes of a route are friendly and honest, they should report consistent throughput values. Hence, any of the reported numbers can be used to compute the aggregate throughput of the flow.

However, given the presence of dishonest nodes, large discrepancies may arise. In this case, we use the median value of all reported throughput numbers for each route because it is robust against distorted numbers, unless attackers can dominate a route in number.

Suppose a node receives summary reports from N different nodes regarding an arbitrary route $\Re$. Among them are G good (honest) reports and B bad (dishonest) reports. Obviously, $N = G + B$. To maximize their impact, the bad reports should agree with but deviate far from the good reports. The bad reports may claim throughput values higher or lower than the good reports. In any case, dishonest nodes will cause no damage if the median value is drawn from a good report. To ensure that the median is always drawn from a good report, the following conditions must be satisfied:
If bad reports claim higher throughput values,

$$\begin{cases} \frac{N+1}{2} \leq G, & \text{if N is odd} \\ \frac{N}{2} + 1 \leq G, & \text{if N is even} \end{cases} \tag{4.1}$$

If bad reports claim lower throughput values,

$$\begin{cases} \frac{N+1}{2} > B, & \text{if N is odd} \\ \frac{N}{2} > B, & \text{if N is even} \end{cases} \tag{4.2}$$

Based on Formulas 4.1 and 4.2, we find out that to eliminate the impact of dishonest intermediate nodes that falsely report higher or lower throughput numbers, we need to make sure that

$$\begin{cases} G \geq B + 1, & \text{if N is odd} \\ G \geq B + 2, & \text{if N is even} \end{cases} \tag{4.3}$$

In other words, as long as the number of dishonest nodes is smaller than that of friendly nodes on each individual route, the global throughput perceived by individual intermediate

nodes will not be manipulated.

Undeniably, attackers can gain advantage by occupying key network positions. In particular, a dishonest node located at the joint of multiple routing paths is more dangerous because it can advertise fake throughput numbers for all these routes. On the other hand, however, friendly joint nodes are more resilient to distorted summary reports because they can obtain more first-hand throughput data than other nodes.

4.6 System Evaluation

We implemented a prototype of CapMan in the *NS2* [61] simulator and evaluated its efficacy and overhead in different scenarios. For all simulations, we use IEEE 802.11 as the MAC layer protocol and AOMDV [108] as the multi-path routing protocol. The original design of AOMDV focused on providing fault tolerance instead of load balance. Therefore, one optimal path is always used until it breaks. Alternative routes are cached and will only be activated one at a time to replace the current route when it fails. To better simulate attackers that flood the victim simultaneously along multiple routing paths, we slightly modified the NS2 implementation of AOMDV to achieve round-robin routing that employs all available paths in parallel to maximize end-to-end throughput.

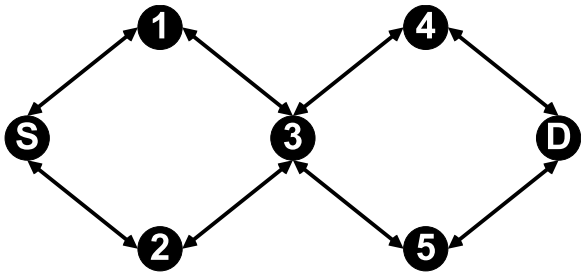


Figure 4.2: Topology for joint multi-path routing with one flow.

First of all, we assessed the effectiveness of CapMan in preventing DoS attacks launched by a flow initiator using a simple topology described in Figure 4.2. There is one malicious

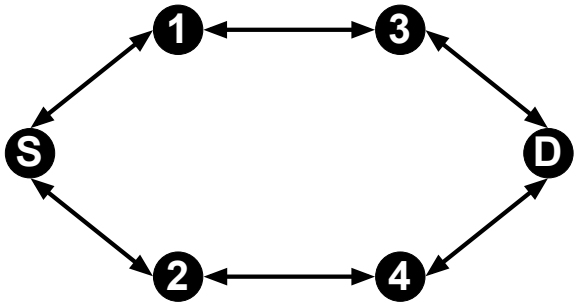


Figure 4.3: Topology for disjoint multi-path routing with one flow.

initiator (S), one responder (D), and five intermediate nodes. The packet flow between S and D is routed via four paths $1 \rightarrow 3 \rightarrow 4$, $1 \rightarrow 3 \rightarrow 5$, $2 \rightarrow 3 \rightarrow 4$, and $2 \rightarrow 3 \rightarrow 5$. D issues a capability to S that specifies a maximum data rate of $100Kbps$. However, S attempts to flood D with a constant bit rate (CBR) traffic at $900Kbps$ and a bursty (exponential on/off) traffic with a peak rate of $1{,}000Kbps$. CapMan is enforced to suppress the throughput of the attack in both cases. The results are shown in Figure 4.4 and Figure 4.5, respectively. Apparently, in both cases, CapMan is able to constrain flow throughput to the level of the assigned capability.

To demonstrate the improvement made by CapMan over existing capability mechanisms on mitigating multi-path flooding attacks, we also implemented the deny-by-default mechanism [48] in *NS2* and ran another set of simulations with the topology displayed in Figure 4.3. In this topology, two disjoint routes ($1 \rightarrow 3$, $2 \rightarrow 4$) are used to forward traffic from S to D. The flow is bound to a capability of $50Kbps$, while the malicious S attempts to flood D with a CBR traffic of $200Kbps$ and a bursty traffic with a peak rate of $200Kbps$. Figure 4.6 and Figure 4.7 exhibit the achieved throughput numbers of the CBR and bursty traffic, with and without the protection of CapMan. The differences are obvious. Like the previous simulations, CapMan once again brings down the attacker throughput to within the capability limit. In contrast, although the deny-by-default solution also reduces the throughput of the attack traffic to some extent, attackers can still score twice as high as the specified capability. In fact, the more disjoint routes that the attackers can exploit under

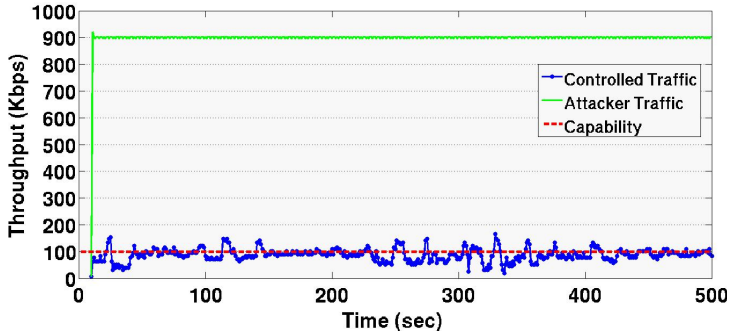


Figure 4.4: DoS attack performed by an initiator using CBR traffic.

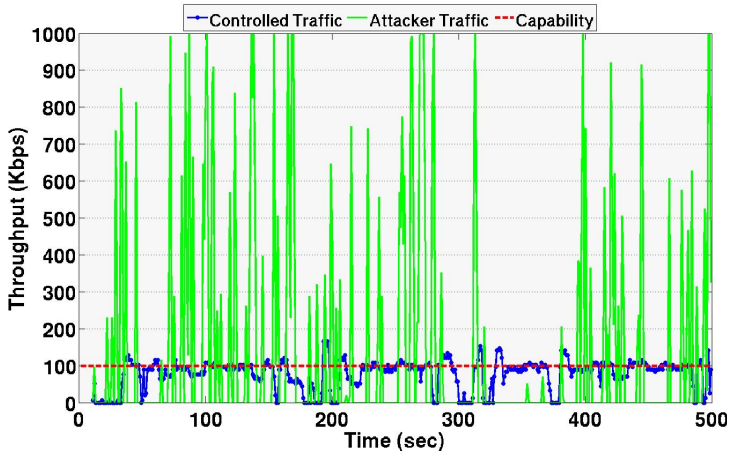


Figure 4.5: DoS attack performed by an initiator using bursty traffic.

deny-by-default, the higher throughput they will be able to achieve, given that the flow capability is only enforced independently by each route.

To evaluate CapMan under a more realistic and mobile environment, we created a large network topology with 50 nodes randomly distributed in an area of 600x600 meters. The movement of all nodes followed the *Random Waypoint* model with an average pause time of 30 seconds. We used the default radio transmission range of 250 meters and set the per-node bandwidth to $1.0Mbps$. All simulations ran for a duration of 1,000 seconds.

To further investigate the effectiveness of CapMan when facing sophisticated DoS attacks, we randomly picked 6 nodes to form three UDP flows. Flow 1 connected two benign

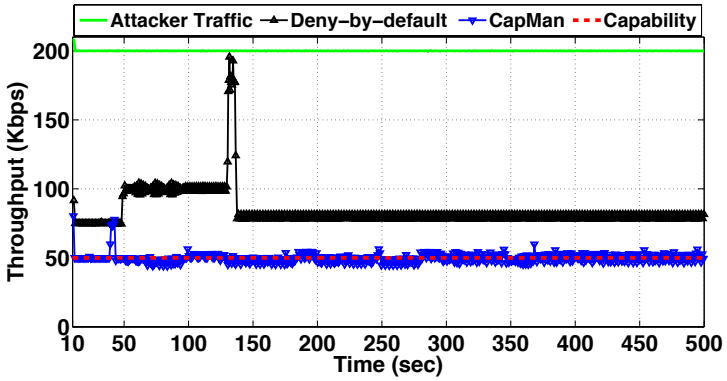


Figure 4.6: Compare CapMan with uni-path capability-based solution using CBR traffic.

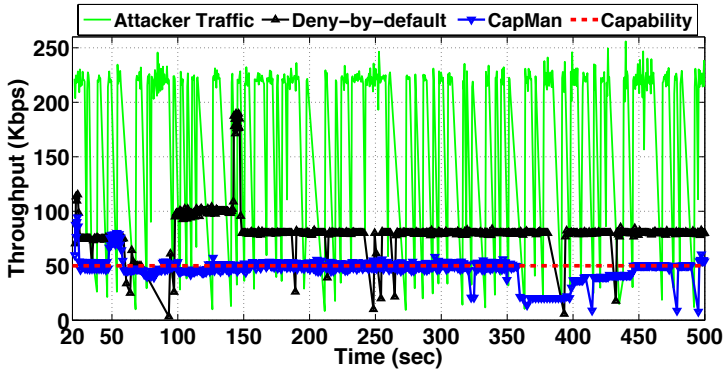


Figure 4.7: Compare CapMan with uni-path capability-based solution using bursty traffic.

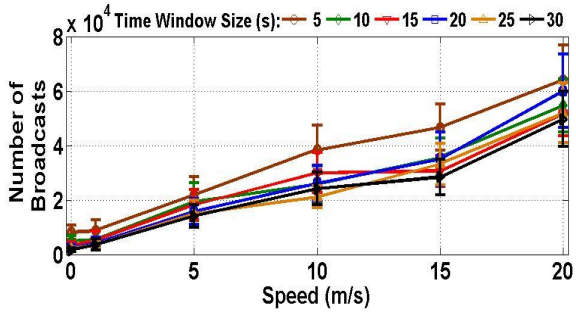
nodes. Flow 2 was started by a malicious initiator to flood a benign responder. Flow 3 was established between two colluding attackers that created a large capability to drain precious bandwidth resources from intermediate nodes. In addition, 6 other nodes were arbitrarily chosen to launch black-hole attacks against capability enforcement. With a total of 9 out of 50 nodes in the network being malicious, we intended to test: 1) the ability of CapMan to prevent flooding attacks under a dynamic topology; 2) the impact of CapMan on benign flows; 3) the robustness of CapMan against abusive nodes. In this experiment, we set the maximum node speed as $5m/s$. The capability for the first two flows was configured to be $100kbps$. The colluding attackers of the third flow deliberately set a high capability of

Table 4.1: Mitigating DoS attacks in a large, dynamic topology.

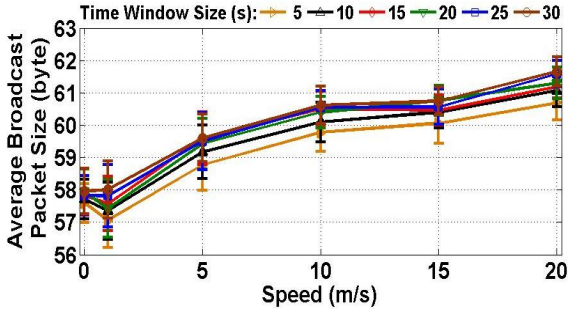
Flow ID	*Setting*	*Capability*	*No Capability*		CapMan	
			Throughput	Std	Throughput	Std
Flow 1	50Kbps	100Kbps	45.93Kbps	4.39Kbps	39.77Kbps	1.50Kbps
Flow 2	500Kbps	100Kbps	206.71Kbps	67.62Kbps	81.43Kbps	8.43Kbps
Flow 3	500Kbps	1Mbps	191.82Kbps	50.29Kbps	51.28Kbps	5.97Kbps

$1Mbps$. To combat that, we set the default capability to be $150kbps$. To show the difference with and without CapMan, we ran the same flows under each setting 10 times. The results are summarized in Table 4.1, in which the mean and standard deviation of each flow's throughput are displayed. From Table 4.1, we can see that the throughput of Flow 1 dropped approximately 13% when CapMan was in effect. In contrast to the small impact that CapMan introduced to the benign flow, the throughput of the two attacking flows was reduced by more than 60% and 73%, respectively. It is worth noting that the above results were achieved under the presence of black-hole attackers, demonstrating the reliability of broadcasting summary reports.

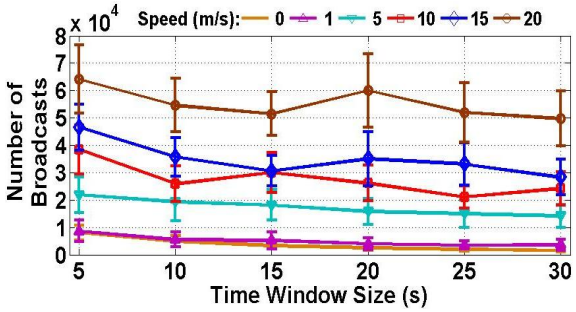
Next, we measured the overhead of CapMan using the same topology settings. A pair of end nodes were randomly selected to form one UDP flow using CBR traffic. The maximum node moving speed was set to $0.001m/s$, $1m/s$, $5m/s$, $10m/s$, $15m/s$, and $20m/s$ for a series of simulations. The packet size was configured to be $512bytes$. Packets were sent from the initiator to the responder at $50ms$ time intervals. To ensure coverage, we used a time-to-live (TTL) of 10 for all summary packets. Figure 4.8a and Figure 4.8b illustrate the variation of the overhead with increasing speeds, while Figure 4.8c and Figure 4.8d show the changes of the overhead from the perspective of growing time window sizes. For each run, we recorded the total number of broadcasts sent as well as the average sizes of all summary packets. For the convenience of implementation, in our proof-of-concept prototype of CapMan on *NS2*, a summary packet only enclosed the sender's information, some metadata (sequence number, timestamp), *PRH*, and the throughput information. No cryptographic signatures were included. Each data point in the figures represents the mean value of 30 runs with



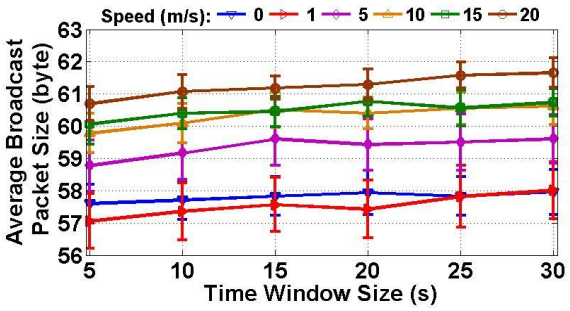
(a) Number of broadcasts regarding node speed



(b) Summary packet size regarding node speed



(c) Number of broadcasts regarding time window size



(d) Summary packet size regarding time window size

Figure 4.8: Overhead of CapMan under different node speeds and time window sizes.

90% confidence intervals plotted around it.

As the maximum node speed grows, Figure 4.8a and Figure 4.8b reveal a slower-than-linear increase in both the total number of broadcasts and the average size of the summary packets. The reason is that when all nodes in a network move faster, they are inclined to choose neighbors within shorter distances as the next hop for routing. In this case, between the same pair of end nodes, packets should travel more-but-shorter hops to reach their destinations. Nevertheless, according to Figure 4.8c and Figure 4.8d, the changes on the overhead are actually subtle when the time window size of capability enforcement changes. Ideally, the total number of broadcasts should drop considerably as per-node broadcast frequency (i.e. the inverse of time window size) decreases. However, in a mobile environment, multiple routing updates can happen within an extended time window, thus adding

to the number of broadcasting nodes. To reduce the overhead introduced by CapMan, various techniques, including [113] and [114], can be employed to improve the efficiency of the broadcast-oriented summary exchange.

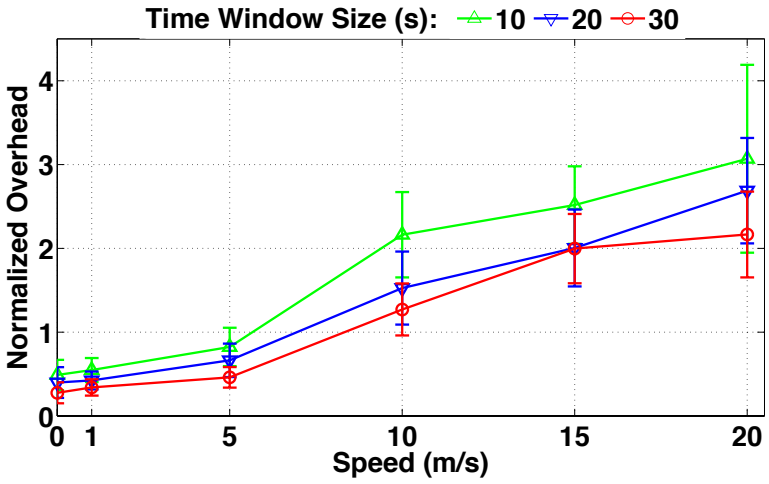


Figure 4.9: Normalized overhead of CapMan.

In addition, by computing the number of broadcasted summary packets per delivered data packet, we measured the normalized overhead of CapMan. To do that, we ran three CBR flows under the same network and mobility settings as in the previous simulations. The results obtained under different node speeds and time window sizes are plotted in Figure 4.9. As was revealed in the previous experiments, the changes on node speed have more impact on the overhead than the variances on time window size. Faster node movement not only causes more intermediate nodes to be employed for most routes, but it also results in more frequent changes on routing. Similar to the case of stretched time windows, nodes on both the switched-in routes and the switched-out routes would broadcast summary packets for the same lapsed time window. When the network is relatively stable ($speed \leq 5m/s$), the normalized overhead of CapMan stays under one summary packet per delivered data packet.

Last but not least, we compared the end-to-end packet latency as well as packet delivery ratio (PDR) of the AOMDV routing protocol with and without enforcing the CapMan defense mechanism. For each setting, we ran the same simulation 10 times and plotted 90%

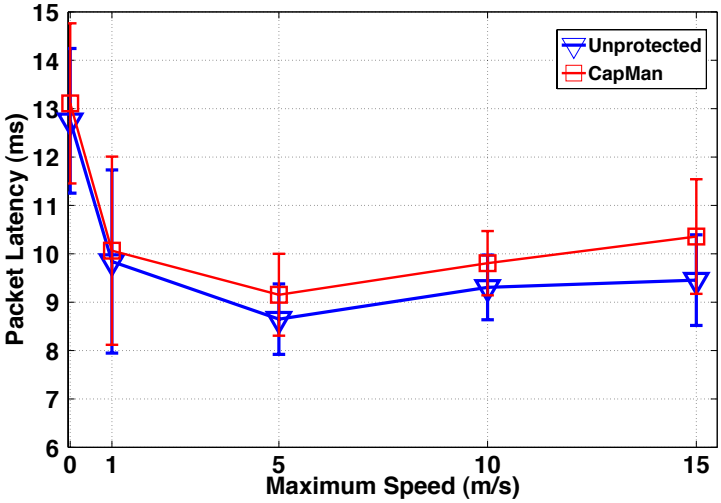


Figure 4.10: Average packet latency of AOMDV routing with and without the protection of CapMan.

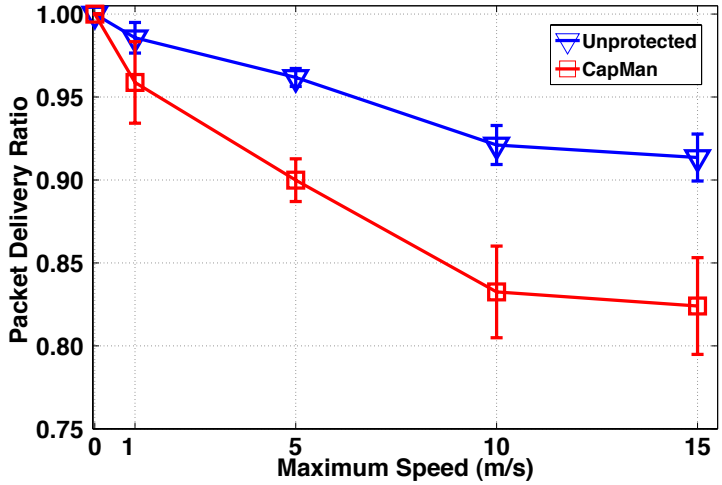


Figure 4.11: Average packet delivery ratio of AOMDV routing with and without the protection of CapMan.

confidence intervals around the means. As was stated earlier, we slightly modified the NS2 implementation of AOMDV to achieve round-robin routing using all available paths. This may introduce unpredictable end-to-end packet latency to benign flows since the shortest path is not always used.

Figure 4.10 shows that CapMan incurs only a small fraction of extra latency. The delay introduced by CapMan is negligible when the topology is relatively static and grows slowly as the maximum speed increases. When the maximum node moving speed reaches $15m/s$, the additional latency caused by CapMan is 9.5% compared to the unprotected system. A similar trend is shown in Figure 4.11 which discloses the difference in packet delivery ratios. When the maximum node moving speed reaches $15m/s$, the packet delivery ratio of CapMan is only 8.9% lower than the unprotected system.

Chapter 5: Conclusions and Future Work

5.1 Conclusions

This thesis studied the mechanisms used to mitigate flooding DDoS attacks in the contested Internet and MANET environments. The goal is to sustain the system and network availability of the protected hosts even in the presence of strong attackers. To combat DDoS attacks bombarded by powerful botnets on the Internet, we devised a shuffling-based moving target mechanism that is capable of separating benign clients from both naive and sophisticated attackers. With the help of the optimized shuffling algorithms, we can progressively quarantine the impact of a DDoS attack and restore the QoS for the affected benign clients. To implement the shuffling-based defense against network flooding attacks on critical services serving authorized and authenticated clients, we designed a MOTAG architecture [52, 53] that employs secret moving proxies as the intermediate layer to perform traffic indirection and to segregate attackers. To protect open web services from DDoS attacks targeting both network and computational resources, we constructed a cloud-enabled architecture [54] that dynamically replicates the protected servers when under attack and smartly redistributes clients as dictated by the shuffling algorithms. In addition, we created a capability-based mechanism to inhibit flooding DoS attacks exploiting multi-path communications in MANETs. Our approach enforces capability limits assigned to end-to-end flows across all employed routing paths. Our theoretical analysis and experimental evaluation demonstrate that the proposed solutions can effectively mitigate flooding DDoS attacks in different adversarial network environments.

In Chapter 3, we introduced a family of moving target mechanisms and architectures that aim to segregate DDoS attackers from benign clients on the Internet. The MOTAG architecture is established on the basis of a large pool of network proxies that can be used

to relay communications between authenticated clients and the protected servers. We keep all proxies private, only disclosing the IP addresses of the active proxies to authorized clients after successful authentication. To increase the uncertainty for attackers and also to lower the operational costs, a small yet dynamic subset of proxies are kept active to serve legitimate clients. If hit by an attack, these proxies are replaced at runtime by backup proxies at different network locations. As a result, the set of active proxies become moving targets that not only can evade naive and brute-force attackers but also help to expose and isolate advanced attackers that continue to follow.

On the basis of our research on MOTAG, we designed a more versatile, cloud-enabled architecture that endows mobility to open web servers. By deploying replicated server instances in the cloud, we leverage the network space and resource elasticity offered by the underlying cloud computing infrastructure to realize service mobility and scalability. Similar to the moving proxies in MOTAG, the attacked replica servers are dynamically replaced by newly instantiated replica servers spread in the wide space of the cloud computing network. By intelligently redirecting affected clients among the substitute servers, we are able to isolate DDoS attacks that aim to preempt server-end network and computational resources from the intended clients.

As our core effort to optimize the shuffling operations that re-allocate clients under attack to achieve attacker segregation, we developed a series of algorithms that can expectedly separate the maximum number of benign clients from attackers in each shuffle. A theoretically optimal dynamic programming algorithm and a fast greedy algorithm were proposed to compute the desired client assignment plan. A maximum-likelihood estimation (MLE) algorithm was presented and analyzed to make educated guesses on the number of persistent attackers following the constructed moving targets.

We thoroughly evaluated the effectiveness and overhead of our moving target defenses. Experiments that simulated various DDoS attacks demonstrate that the shuffling-based mechanism can successfully quarantine the constantly following attackers in a few rounds of shuffling. We also implemented proof-of-concept prototypes of the MOTAG architecture and

the cloud-enabled architecture on PlanetLab and Amazon EC2, respectively. Experiments with the prototypes show that the time overheads of redirecting clients between different proxies and different replica web servers are both low.

Compared to existing Internet DDoS defense mechanisms, our moving target solutions do not rely on dedicated infrastructures to outmuscle botnets and absorb attack traffic, nor do they require sophisticated security functionalities to be widely deployed on Internet routers. Instead, we opt for a dynamic approach that turns network proxies and replica servers into moving targets to continuously confuse and evade attackers. This approach allows us to mitigate intense DDoS attacks over time even if we own much fewer resources than the attackers do. The proposed solutions can be independently deployed by individual organizations without the support of ISPs. Modern cloud computing providers, with their abundance of resources and pay-per-use policies, are ideal for hosting our DDoS mitigation systems.

In addition to protecting Internet services from DDoS attacks, this thesis also aimed to secure hosts in MANETs against DoS attacks launched by malicious insiders. To this end, we designed CapMan, a capability-based mechanism to stop flooding attacks targeting multi-path MANET communications. CapMan binds each end-to-end flow to a unique capability issued by the flow responder and makes sure intermediate nodes on all employed routing paths are informed about the per-flow capabilities. To enforce the flow capability limit in the multi-path routing context, participating nodes across different paths exchange bandwidth consumption reports periodically to maintain a global throughput image and adjust local policing rates accordingly. Simulations using NS2 show that CapMan is able to contain the throughput of attacking flows routed through multiple paths, even in the case of colluding intermediate and end nodes. In the meantime, only a small portion of the network bandwidth will be consumed by the control packets of CapMan.

5.2 Future Work

The research presented in this thesis can lead to a number of future research directions that cover different aspects of DDoS defense in contested networking environments. We discuss a few of them in this section.

The analysis and experiments performed for this thesis show that shuffling-based moving target defense is a novel and effective strategy for mitigating Internet DDoS attacks. However, more work needs to be done before we can implement a practical defense system that realizes this idea.

First, the current threat model assumed the worst case that attackers own infinite bandwidth. Consequently, one persistent attacker that follows a moving proxy or replica server is enough to cause that service node to be attacked. In practice, however, botnets always have limited attacking power. Therefore, not all exposed service nodes will end up being attacked. Incorporating a parameter that estimates the realistic size and capability of attackers may allow us to build a more practical model of the problem, hence constructing a solution that works better against attacks of average intensity.

Second, in the cloud-enabled architecture that uses dynamic replica servers as moving targets, client redistribution operations are efficient only if no client state information needs to be migrated. This is the case for stateless services or stateful services that keep session information at the back-end storage location. In other scenarios, such as attacks targeting the back-end filesystem or database, or attacks on services without separate storage, state information needs to be transferred while clients are being re-assigned. For the purpose of swift client migration, auxiliary technologies such as lightweight virtualization are needed to encapsulate per-client sessions. So far, migrating the states of a large number of clients independently and in parallel remains an unsolved problem.

Third, the economics of the proposed moving target defense is also an interesting research problem. Compared to conventional DDoS mitigation mechanisms that are static in nature, our moving target solutions do not depend on the sheer volume of resources to dilute an attack. Instead, a relatively small number of service nodes are used to launder clients

over time. Therefore, the cost of running our defense should be multiplying the prices of employed service nodes by the time it takes to quarantine the attack. For defenders, while maximizing the number of saved benign clients, another important goal is to improve the cost-effectiveness by using resources more efficiently. From the attackers' perspective, if it becomes difficult to disable the targeted service, a different type of inflicted damage is to raise the cost of running the defense system. The strategies for defenders and attackers to achieve their respective goals warrant more study.

On combating MANET DoS attacks, future research can integrate our capability-based mechanism with dynamic routing protocols. Such an integration not only reduces the overhead of broadcasting periodical summary reports, but it also provides accurate and up-to-date multi-path routing information to the capability enforcement module. In addition, reputation-based mechanisms can be compounded with our solution on giving credibility scores to different nodes. In this way, different weights can be assigned to reports originating from intermediate nodes, enhancing the robustness of the system against dishonest and misbehaving nodes in the network.

Bibliography

Bibliography

[1] M. Handley, E. Rescorla, and IAB, "Internet Denial-of-Service Considerations," RFC 4732 (Informational), Internet Engineering Task Force, Dec. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4732.txt

[2] D. Anstee and D. Bussiere, "Worldwide infrastructure security report viii," 2012. [Online]. Available: http://www.arbornetworks.com/report

[3] T. Micro, "Russian underground 101," http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-russian-underground-101.pdf, 2012.

[4] J. Vijayan, "Update: Mastercard, visa others hit by ddos attacks over wikileaks," http://www.computerworld.com/s/article/9200521/Update_MasterCard_Visa_others_hit_by_DDoS_attacks_over_WikiLeaks?taxonomyId=82&pageNumber=1, 2010.

[5] B. McCarty, "Botnets: Big and bigger," *Security & Privacy, IEEE*, vol. 1, no. 4, pp. 87–90, 2003.

[6] J. Mirkovic and P. Reiher, "A taxonomy of ddos attack and ddos defense mechanisms," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.

[7] C. E. R. Team, "Cert advisory ca-1996-21 tcp syn flooding and ip spoofing attacks," September 1996. [Online]. Available: http://www.cert.org/advisories/CA-1996-21.html

[8] C. L. Schuba, I. V. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, and D. Zamboni, "Analysis of a denial of service attack on tcp," in *Security and Privacy, 1997. Proceedings., 1997 IEEE Symposium on*. IEEE, 1997, pp. 208–223.

[9] C. E. R. Team, "Cert advisory ca-1996-01 udp port denial-of-service attack," Februrary 1996. [Online]. Available: http://www.cert.org/advisories/CA-1996-01.html

[10] ——, "Cert advisory ca-1998-01 smurf ip denial-of-service attacks," Janurary 1998. [Online]. Available: http://www.cert.org/advisories/CA-1996-01.html

[11] ——, "Denial of service attacks using nameservers," April 2000. [Online]. Available: http://www.cert.org/advisories/CA-1996-01.html

[12] F. C. Freiling, T. Holz, and G. Wicherski, "Botnet tracking: exploring a root-cause methodology to prevent distributed denial-of-service attacks," in *Proceedings of the*

10th European conference on Research in Computer Security, ser. ESORICS'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 319–335. [Online]. Available: http://dx.doi.org/10.1007/11555827_19

[13] D. Clark, "The design philosophy of the darpa internet protocols," in *ACM SIGCOMM Computer Communication Review*, vol. 18, no. 4. ACM, 1988, pp. 106–114.

[14] V. D. Gligor, "A note on denial-of-service in operating systems," *Software Engineering, IEEE Transactions on*, no. 3, pp. 320–324, 1984.

[15] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Trans. Comput. Syst.*, vol. 2, no. 4, pp. 277–288, Nov. 1984. [Online]. Available: http://doi.acm.org/10.1145/357401.357402

[16] V. D. Gligor, "Guaranteeing access in spite of distributed service-flooding attacks," in *Security Protocols*. Springer, 2005, pp. 80–96.

[17] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, "Controlling high bandwidth aggregates in the network," *ACM Computer Communication Review*, vol. 32, pp. 62–73, 2002.

[18] P. Ferguson and D. Senie, "Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing," RFC 2827 (Best Current Practice), Internet Engineering Task Force, May 2000, updated by RFC 3704.

[19] X. Liu, X. Yang, and Y. Lu, "To filter or to authorize: network-layer dos defense against multimillion-node botnets," in *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*. New York, NY, USA: ACM, 2008, pp. 195–206.

[20] T. Anderson, T. Roscoe, and D. Wetherall, "Preventing internet denial-of-service with capabilities," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 1, pp. 39–44, 2004.

[21] A. Yaar, A. Perrig, and D. Song, "Siff: A stateless internet flow filter to mitigate ddos flooding attacks," in *IEEE Symposium on Security and Privacy*, 2004, pp. 130–143.

[22] X. Yang, D. Wetherall, and T. Anderson, "Tva: a dos-limiting network architecture," *IEEE/ACM Trans. Netw.*, vol. 16, no. 6, pp. 1267–1280, 2008.

[23] X. Liu, X. Yang, and Y. Xia, "Netfence: preventing internet denial of service from inside out," in *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 255–266. [Online]. Available: http://doi.acm.org/10.1145/1851182.1851214

[24] A. D. Keromytis, V. Misra, and D. Rubenstein, "Sos: Secure overlay services," in *Proceedings of ACM SIGCOMM*, 2002, pp. 61–72.

[25] A. Stavrou and A. D. Keromytis, "Countering dos attacks with stateless multipath overlays," in *Proceedings of the 12th ACM conference on Computer and communications security*, ser. CCS '05. New York, NY, USA: ACM, 2005, pp. 249–259. [Online]. Available: http://doi.acm.org/10.1145/1102120.1102153

[26] D. G. Andersen, "Mayday: distributed filtering for internet services," in *USITS'03: Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*. Berkeley, CA, USA: USENIX Association, 2003, pp. 3–3.

[27] R. Stone, "Centertrack: an ip overlay network for tracking dos floods," in *SSYM'00: Proceedings of the 9th conference on USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, 2000, pp. 15–15.

[28] A. Mahimkar, J. Dange, V. Shmatikov, H. Vin, and Y. Zhang, "dfence: Transparent network-based denial of service mitigation," in *NSDI*, 2007.

[29] C. Dixon, T. Anderson, and A. Krishnamurthy, "Phalanx: withstanding multimillion-node botnets," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 45–58. [Online]. Available: http://dl.acm.org/citation.cfm?id=1387589.1387593

[30] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *In Proceedings of the 13th Usenix Security Symposium*, 2004.

[31] "Akamai," http://www.akamai.com.

[32] "Limelight networks," http://www.limelight.com.

[33] B. Wu, J. Chen, J. Wu, and M. Cardei, "A Survey of Attacks and Countermeasures in Mobile Ad Hoc Networks," in *Wireless Network Security*, ser. Signals and Communication Technology, Y. Xiao, X. S. Shen, and D.-Z. Du, Eds. Boston, MA: Springer US, 2007, ch. 5, pp. 103–135. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-33112-6_5

[34] T. Karygiannis and L. Owens, "Wireless network security," *NIST special publication*, vol. 800, p. 48, 2002.

[35] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Ariadne: a secure on-demand routing protocol for ad hoc networks," *Wirel. Netw.*, vol. 11, no. 1-2, pp. 21–38, 2005.

[36] B. Awerbuch, D. Holmer, C. Nita-Rotaru, and H. Rubens, "An on-demand secure routing protocol resilient to byzantine failures," in *Proceedings of the 1st ACM workshop on Wireless security*. ACM, 2002, pp. 21–30.

[37] K. Sanzgiri, B. Dahill, B. N. Levine, C. Shields, and E. M. Belding-Royer, "A secure routing protocol for ad hoc networks," in *Network Protocols, 2002. Proceedings. 10th IEEE International Conference on*. IEEE, 2002, pp. 78–87.

[38] H. Yih-Chun and A. Perrig, "A survey of secure wireless ad hoc routing," *Security & Privacy, IEEE*, vol. 2, no. 3, pp. 28–39, 2004.

[39] P. Papadimitratos and Z. J. Haas, "Secure data transmission in mobile ad hoc networks," in *Proceedings of the 2nd ACM workshop on Wireless security*. ACM, 2003, pp. 41–50.

[40] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Packet leashes: a defense against wormhole attacks in wireless networks," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 3. IEEE, 2003, pp. 1976–1986.

[41] ——, "Rushing attacks and defense in wireless ad hoc network routing protocols," in *Proceedings of the 2nd ACM workshop on Wireless security*. ACM, 2003, pp. 30–40.

[42] S. Lu, L. Li, K.-Y. Lam, and L. Jia, "Saodv: a manet routing protocol that can withstand black hole attack," in *Computational Intelligence and Security, 2009. CIS'09. International Conference on*, vol. 2. IEEE, 2009, pp. 421–425.

[43] Y.-C. Hu, D. B. Johnson, and A. Perrig, "Sead: Secure efficient distance vector routing for mobile wireless ad hoc networks," *Ad Hoc Networks*, vol. 1, no. 1, pp. 175–192, 2003.

[44] S. Marti, T. J. Giuli, K. Lai, M. Baker *et al.*, "Mitigating routing misbehavior in mobile ad hoc networks," in *International Conference on Mobile Computing and Networking: Proceedings of the 6 th annual international conference on Mobile computing and networking*, vol. 6, no. 11, 2000, pp. 255–265.

[45] D. Sterne, P. Balasubramanyam, D. Carman, B. Wilson, R. Talpade, C. Ko, R. Balupari, C.-Y. Tseng, and T. Bowen, "A general cooperative intrusion detection architecture for manets," in *Information Assurance, 2005. Proceedings. Third IEEE International Workshop on*. IEEE, 2005, pp. 57–70.

[46] X. Wu and D. K. Yau, "Mitigating denial-of-service attacks in manet by distributed packet filtering: a game-theoretic approach," in *Proceedings of the 2nd ACM symposium on Information, computer and communications security*. ACM, 2007, pp. 365–367.

[47] X. Jin, Y. Zhang, Y. Pan, and Y. Zhou, "Zsbt: A novel algorithm for tracing dos attackers in manets," *EURASIP Journal on Wireless Communications and Networking*, vol. 2006, no. 2, pp. 82–82, 2006.

[48] M. Alicherry, A. D. Keromytis, and A. Stavrou, "Deny-by-default distributed security policy enforcement in mobile ad hoc networks," in *Proceedings of the 5th International Conference on Security and Privacy in Communication Networks*, September 2009.

[49] ——, "Evaluating a collaborative defense architecture for manets," in *IMSAA'09: Proceedings of the 3rd IEEE international conference on Internet multimedia services architecture and applications*. Piscataway, NJ, USA: IEEE Press, 2009, pp. 229–234.

[50] M. Alicherry and A. D. Keromytis, "Diploma: Distributed policy enforcement architecture for manets," in *Proceedings of the 4th International Conference on Network and System Security (NSS)*, September 2010, pp. 89–98.

[51] ——, "Securing manet multicast using diploma," in *Proceedings of the 5th International Workshop on Security (IWSEC)*, November 2010, pp. 232–250.

[52] Q. Jia, K. Sun, and A. Stavrou, "Motag: Moving target defense against internet denial of service attacks," in *Proceedings of 22nd International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2013.

[53] H. Wang, Q. Jia, D. Fleck, W. Powell, F. Li, and A. Stavrou, "A moving target ddos defense mechanism," *Computer Communications (COMCOM)*, submitted to.

[54] Q. Jia, H. Wang, D. Fleck, F. Li, and A. Stavrou, "Catch me if you can: A cloud-enabled ddos defense," *The 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2014)*, submitted to.

[55] Q. Jia, K. Sun, and A. Stavrou, "Capman: Capability-based defense against multi-path denial of service (dos) attacks in manet," in *Proceedings of the first International Workshop on Privacy, Security and Trust in Mobile and Wireless Systems*, 2011.

[56] ——, "Capability-based defenses against dos attacks in multi-path manet communications," *Wireless Personal Communications*, 2013.

[57] D. of Homeland Security, "Moving target defense," 2011. [Online]. Available: http://www.cyber.st.dhs.gov/moving-target-defense/

[58] MATLAB, *version 7.8.0 (R2009a)*. Natick, Massachusetts: The MathWorks Inc., 2009.

[59] "Planetlab," http://www.planet-lab.org/.

[60] Amazon.com, "Amazon web services," http://aws.amazon.com.

[61] "The network simulator ns-2," http://www.isi.edu/nsnam/ns/.

[62] G. Sandoval and T. Wolverton, "Leading web sites under attack," 2000. [Online]. Available: http://news.cnet.com/Leading-Web-sites-under-attack/2100-1017_3-236683.html

[63] R. Richardson, "20102011 annual CSI computer crime and security survey," Computer Security Institute, Tech. Rep., 2011.

[64] J. Mirkovic and P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 39–53, Apr. 2004. [Online]. Available: http://dx.doi.org/10.1145/997150.997156

[65] T. Peng, C. Leckie, and K. Ramamohanarao, "Survey of network-based defense mechanisms countering the dos and ddos problems," *ACM Computing Surveys (CSUR)*, vol. 39, no. 1, p. 3, 2007.

[66] M. Abliz, "Internet denial of service attacks and defense mechanisms," University of Pittsburgh, Tech. Rep. TR-11-178, Mar 2011.

[67] R. Beverly, A. Berger, Y. Hyun, and kc claffy, "Understanding the efficacy of deployed internet source address validation filtering," in *Internet Measurement Conference*, A. Feldmann and L. Mathy, Eds. ACM, 2009, pp. 356–369.

[68] X. Liu, A. Li, X. Yang, and D. Wetherall, "Passport: Secure and adoptable source authentication," in *NSDI*, J. Crowcroft and M. Dahlin, Eds. USENIX Association, 2008, pp. 365–376.

[69] S. Bellovin, M. Leech, and T. Taylor, "Internet draft: Icmp traceback messages," *http://tools.ietf.org/html/draft-ietf-itrace-04*, 2003.

[70] D. Dean, M. Franklin, and A. Stubblefield, "An algebraic approach to ip traceback," *ACM Transactions on Information and System Security (TISSEC)*, vol. 5, no. 2, pp. 119–137, 2002.

[71] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical network support for ip traceback," *ACM SIGCOMM Computer Communication Review*, vol. 30, no. 4, pp. 295–306, 2000.

[72] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer, "Hash-based ip traceback," in *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4. ACM, 2001, pp. 3–14.

[73] M. T. Goodrich, "Efficient packet marking for large-scale ip traceback," in *Proceedings of the 9th ACM conference on Computer and communications security*. ACM, 2002, pp. 117–126.

[74] J. Li, M. Sung, J. Xu, and L. Li, "Large-scale ip traceback in high-speed internet: Practical techniques and theoretical foundation," in *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*. IEEE, 2004, pp. 115–129.

[75] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y.-C. Hu, "Portcullis: Protecting connection setup from denial-of-capability attacks," in *Proceedings of the ACM SIGCOMM*, August 2007.

[76] K. Argyraki and D. R. Cheriton, "Network capabilities: The good, the bad and the ugly," in *ACM HotNets-IV*, 2005.

[77] L. Overlier and P. Syverson, "Locating hidden servers," in *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, ser. SP '06, Washington, DC, USA, 2006, pp. 100–114.

[78] V. Kambhampati, C. Papadopoulos, and D. Massey, "Epiphany: A location hiding architecture for protecting critical services from ddos attacks," in *The 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, 2012.

[79] X. Wang and M. K. Reiter, "Wraps: Denial-of-service defense through web referrals," in *25th IEEE Symposium on Reliable Distributed Systems*. IEEE Computer Society, 2006, pp. 51–60.

[80] X. Wu and D. K. Y. Yau, "Mitigating denial-of-service attacks in manet by distributed packet filtering: a game-theoretic approach," in *ASIACCS '07: Proceedings of the 2nd ACM symposium on Information, computer and communications security*. New York, NY, USA: ACM, 2007, pp. 365–367.

[81] R. Akbani, T. Korkmaz, and G. Raju, "Heap: A packet authentication scheme for mobile ad hoc networks," *Ad Hoc Networks*, vol. 6, no. 7, pp. 1134–1150, 2008.

[82] H. Krawczyk, R. Canetti, and M. Bellare, "Hmac: Keyed-hashing for message authentication," 1997, updated by RFC 6151. [Online]. Available: http://tools.ietf.org/html/rfc2104

[83] J.-P. Hubaux, L. Buttyán, and S. Capkun, "The quest for security in mobile ad hoc networks," in *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, ser. MobiHoc '01. New York, NY, USA: ACM, 2001, pp. 146–155. [Online]. Available: http://doi.acm.org/10.1145/501436.501437

[84] L. Zhou, F. B. Schneider, and R. Van Renesse, "Coca: A secure distributed online certification authority," *ACM Transactions on Computer Systems (TOCS)*, vol. 20, no. 4, pp. 329–368, 2002.

[85] S. Yi and R. Kravets, "Moca : Mobile certificate authority for wireless ad hoc networks," *Ad Hoc Networks*, vol. 51, p. 65, 2003. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.11.1281&rep=rep1&type=pdf#page=71

[86] C. Zouridaki, B. L. Mark, K. Gaj, and R. K. Thomas, "Distributed CA-based PKI for mobile ad hoc networks using elliptic curve cryptography," in *Public Key Infrastructure. First European PKI Workshop: Research and Applications, EuroPKI 2004. Proceedings.* Springer-Verlag, 2004, pp. 232–45 BN – 3 540 22 216 2+.

[87] Y. Zhang and W. Lee, "Intrusion detection in wireless ad-hoc networks," in *Proceedings of the 6th annual international conference on Mobile computing and networking.* ACM, 2000, pp. 275–283.

[88] H. Yang, J. Shu, X. Meng, and S. Lu, "Scan: self-organized network-layer security in mobile ad hoc networks," *Selected Areas in Communications, IEEE Journal on*, vol. 24, no. 2, pp. 261–273, 2006.

[89] L. Buttyan and J.-P. Hubaux. (2001) Nuglets: a virtual currency to stimulate cooperation in self-organized mobile ad hoc networks. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.23.8953

[90] S. Buchegger and J.-Y. Le Boudec, "Nodes bearing grudges: Towards routing security, fairness, and robustness in mobile ad hoc networks," in *Parallel, Distributed and Network-based Processing, 2002. Proceedings. 10th Euromicro Workshop on.* IEEE, 2002, pp. 403–410.

[91] S. Antonatos, P. Akritidis, E. P. Markatos, and K. G. Anagnostakis, "Defending against hitlist worms using network address space randomization," in *Proceedings of the 2005 ACM workshop on Rapid malcode*, ser. WORM '05. New York, NY, USA: ACM, 2005, pp. 30–40. [Online]. Available: http://doi.acm.org/10.1145/1103626.1103633

[92] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling, "Measuring and detecting fast-flux service networks," in *NDSS.* The Internet Society, 2008.

[93] A. Stavrou, A. D. Keromytis, J. Nieh, V. Misra, and D. Rubenstein, "Move: An end-to-end solution to network denial of service," in *Network and Distributed System Security Symposium, NDSS 2005*. The Internet Society, 2005.

[94] T. Aura, P. Nikander, and J. Leiwo, "Dos-resistant authentication with client puzzles," in *Security Protocols Workshop*, 2000, pp. 170–177.

[95] D. Dean and A. Stubblefield, "Using client puzzles to protect tls," in *Proceedings of the 10th conference on USENIX Security Symposium - Volume 10*, ser. SSYM'01. Berkeley, CA, USA: USENIX Association, 2001, pp. 1–1. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251327.1251328

[96] B. Waters, A. Juels, J. A. Halderman, and E. W. Felten, "New client puzzle outsourcing techniques for dos resistance," in *Proceedings of the 11th ACM conference on Computer and communications security*, ser. CCS '04. New York, NY, USA: ACM, 2004, pp. 246–256. [Online]. Available: http://doi.acm.org/10.1145/1030083.1030117

[97] T. M. Gil and M. Poletto, "Multops: a data-structure for bandwidth attack detection," in *Proceedings of the 10th conference on USENIX Security Symposium - Volume 10*, ser. SSYM'01. Berkeley, CA, USA: USENIX Association, 2001, pp. 3–3. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267612.1267615

[98] A. Hussain, J. Heidemann, and C. Papadopoulos, "A framework for classifying denial of service attacks," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM '03. New York, NY, USA: ACM, 2003, pp. 99–110. [Online]. Available: http://doi.acm.org/10.1145/863955.863968

[99] T. Brisco, "DNS Support for Load Balancing," RFC 1794 (Informational), Internet Engineering Task Force, Apr. 1995. [Online]. Available: http://www.ietf.org/rfc/rfc1794.txt

[100] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616 (Draft Standard), Internet Engineering Task Force, Jun. 1999, updated by RFCs 2817, 5785, 6266, 6585. [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt

[101] F5.com, "Big-ip system," http://www.f5.com/pdf/products/big-ip-platforms-datasheet.pdf.

[102] N. Johnson and S. Kotz, *Urn Models and Their Applications: An Approach to Modern Discrete Probability Theory*. New York: Wiley, 1977, ch. 1.3.2.

[103] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, pp. 3–30, Jan. 1998. [Online]. Available: http://doi.acm.org/10.1145/272991.272995

[104] "Iperf," http://iperf.sourceforge.net.

[105] "Node.js," http://www.nodejs.org.

[106] I. Fette and A. Melnikov, "The WebSocket Protocol," RFC 6455 (Informational), Internet Engineering Task Force, Dec. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6455.txt

[107] S. Mueller, R. Tsang, and D. Ghosal, "Multipath routing in mobile ad hoc networks: Issues and challenges," in *Performance Tools and Applications to Networked Systems, volume 2965 of LNCS*. Springer-Verlag, 2004, pp. 209–234.

[108] M. K. Marina and S. R. Das, "On-demand multipath distance vector routing in ad hoc networks," in *Proceedings of IEEE International Conference on Network Protocols (ICNP*, 2001, pp. 14–23.

[109] W. chang Feng, D. D. Kandlur, D. Saha, and K. G. Shin, "Stochastic fair blue: A queue management algorithm for enforcing fairness," in *Proceedings of IEEE INFOCOM*, 2001, pp. 1520–1529.

[110] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *EUROCRYPT*, 2003, pp. 416–432.

[111] X. Zhang and K. G. Shin, "Chorus: collision resolution for efficient wireless broadcast," in *Proceedings of the 29th conference on Information communications*, ser. INFOCOM'10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 1747–1755. [Online]. Available: http://portal.acm.org/citation.cfm?id=1833515.1833756

[112] C.-Y. Koo, V. Bhandari, J. Katz, and N. H. Vaidya, "Reliable broadcast in radio networks: the bounded collision case," in *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, ser. PODC '06. New York, NY, USA: ACM, 2006, pp. 258–264. [Online]. Available: http://doi.acm.org/10.1145/1146381.1146420

[113] B. Williams and T. Camp, "Comparison of broadcasting techniques for mobile ad hoc networks," in *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, ser. MobiHoc '02. New York, NY, USA: ACM, 2002, pp. 194–205. [Online]. Available: http://doi.acm.org/10.1145/513800.513825

[114] S. Pleisch, M. Balakrishnan, K. Birman, and R. van Renesse, "Mistral: efficient flooding in mobile ad-hoc networks," in *Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*, ser. MobiHoc '06. New York, NY, USA: ACM, 2006, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/1132905.1132907

Curriculum Vitae

Quan Jia received his Bachelor of Science in Software Engineering from South China University of Technology in 2006. He received his Master of Science in Information Security and Assurance from George Mason University in 2012. His research interests include network and system availability, operating system security, and virtualization.