A DEFENSE FRAMEWORK AGAINST DENIAL-OF-SERVICE IN COMPUTER NETWORKS

by

Sherif Khattab

M.Sc. Computer Science, University of Pittsburgh, USA, 2004

B.Sc. Computer Engineering, Cairo University, Egypt, 1998

Submitted to the Graduate Faculty of

the Department of Computer Science in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2008

UNIVERSITY OF PITTSBURGH

DEPARTMENT OF COMPUTER SCIENCE

This dissertation was presented

by

Sherif Khattab

It was defended on

June 25th, 2008

and approved by

Prof. Rami Melhem, Department of Computer Science

Prof. Daniel Mossé, Department of Computer Science

Prof. Taieb Znati, Department of Computer Science

Prof. Prashant Krishnamurthy, School of Information Sciences

Dissertation Advisors: Prof. Rami Melhem, Department of Computer Science,

Prof. Daniel Mossé, Department of Computer Science

A DEFENSE FRAMEWORK AGAINST DENIAL-OF-SERVICE IN COMPUTER NETWORKS

Sherif Khattab, PhD

University of Pittsburgh, 2008

Denial-of-Service (DoS) is a computer security problem that poses a serious challenge to trustworthiness of services deployed over computer networks. The aim of DoS attacks is to make services unavailable to legitimate users, and current network architectures allow easy-to-launch, hard-to-stop DoS attacks. Particularly challenging are the *service-level DoS attacks*, whereby the victim service is flooded with legitimate-like requests, and the *jamming attack*, in which wireless communication is blocked by malicious radio interference. These attacks are overwhelming even for massively-resourced services, and effective and efficient defenses are highly needed.

This work contributes a novel defense framework, which I call *dodging*, against service-level DoS and wireless jamming. Dodging has two components: (1) the careful assignment of servers to clients to achieve accurate and quick identification of service-level DoS attackers and (2) the continuous and unpredictable-to-attackers reconfiguration of the client-server assignment and the radio-channel mapping to withstand service-level and jamming DoS attacks. Dodging creates hard-to-evade baits, or traps, and dilutes the attack "fire power". The traps identify the attackers when they violate the mapping function and even when they attack while correctly following the mapping function. Moreover, dodging keeps attackers "in the dark", trying to follow the unpredictably changing mapping. They may hit a few times but lose "precious" time before they are identified and stopped.

Three dodging-based DoS defense algorithms are developed in this work. They are more resource-efficient than state-of-the-art DoS detection and mitigation techniques. **Honey-**

bees combines channel hopping and error-correcting codes to achieve bandwidth-efficient and energy-efficient mitigation of jamming in multi-radio networks. In **roaming honeypots**, dodging enables the camouflaging of honeypots, or trap machines, as real servers, making it hard for attackers to locate and avoid the traps. Furthermore, shuffling requests over servers opens up windows of opportunity, during which legitimate requests are serviced. **Live baiting**, efficiently identifies service-level DoS attackers by employing results from the group-testing theory, discovering defective members in a population using the minimum number of tests. The cost and benefit of the dodging algorithms are analyzed theoretically, in simulation, and using prototype experiments.

Keywords Computer Networks, Network Security, Denial-of-Service Attack, Dodging, Honeypots, Group Testing, Wireless Jamming, Sensor Networks, Multi-radio.

PREFACE

With my PhD journey reaching its end, I would like to thank my advisors, Rami Melhem and Daniel Mossé, for their support. I learned a lot from their guidance, advice, and wisdom through the hard and the happy moments of this journey. Dr. Rami and Dr. Daniel, I hope I can be as good an advisor to my future students as you are to me. Thank you.

I would like to thank my PhD committee members, Taieb Znati and Prashant Krishnamurthy, for their insightful and constructive advice and feedback.

I would like to thank Mahmoud Elhaddad for the insightful discussions and the pleasant company. Sameh Gobriel and I had a fruitful collaboration that resulted in good research and, more importantly, a portfolio of nice memories of hard and cheerful moments.

I would like to thank Chatree Sangpachatanaruk for conducting the simulation study of the proactive server roaming and roaming honeypots schemes. Michael Gualtieri and Miguel Abele contributed to building the FreeBSD prototype and participated in useful discussions.

I would like to acknowledge the support from NSF and from Andrew Mellon Fellowship.

The staff of the Computer Science Department at Pitt have made every administrative issue a smooth breath. Thank you.

I would like to thank Mohamed Noamany, Mohamed Sharaf, and Rany Tawfik for supporting me in the toughest times during my PhD journey.

I dedicate this dissertation to my parents, my father Essam Khattab and my mother Laila Abdul-Qader. Iman, my wife, your love and support made me stay on track. Tassneem and Sondos, my precious daughters, you make my life pretty.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

1.0 INTRODUCTION

Computer networks, such as the Internet and wireless networks, have undoubtedly become part of today's infrastructure. For instance, thirty-six million American households used Internet banking in 2004 [86], Internet shoppers spent $30.1 billion during the 2005 holiday season [51], and physical infrastructure systems, such as the power grid, use the Internet for remote access and coordination. On the wireless side, sensor networks have emerged as a powerful networking paradigm with many civilian and military applications, such as emergency response, surveillance, health applications, and habitat monitoring [9].

However, services deployed over computer networks are still far from being trustworthy. Indeed, fighting Internet viruses is an everyday practice of computer system administrators. A computer virus is not only a nuisance, but also a way to recruit an army of "zombies", virus-infected computers commandeered by the virus's creator. Zombies are then used for various types of malicious activity. In the *Denial-of-Service (DoS)* attack, the focus of this work, they jointly bombard a target Internet site with a high volume of network traffic, such that legitimate users' traffic cannot go through, and the victim site's services are thus denied. The victim site can be a competitor's website, an emergency command and control center, or even (as has happened) `www.whitehouse.gov` [134]. The synchronized participation of many attacking nodes can achieve high illegitimate traffic rate in a Distributed Denial-of-Service (DDoS) attack [37]. Although the mechanisms behind DoS attacks are relatively simple, current technology falls short of a bulletproof defense.

In the wireless arena the situation is not much different. Wireless networks should not be trusted in life- and mission-critical deployments until their security vulnerabilities are adequately handled [66]. One such vulnerability is wireless jamming, in which attackers induce radio interference in an attempt to block wireless communication [156]. Jamming is

a form of Denial-of-Service against wireless networks [151].

The main facilitator of DoS is that network-borne services have finite ability to process traffic and little control over which traffic to receive [8]. A computer network, like any common facility, needs to be regulated. However, regulation of open networks is difficult to achieve. For instance, the Internet is not owned by a single authority; it is the joint forces of many entities that maintain the Internet. Moreover, the Internet is a very large and complex system connecting millions of machines worldwide. Similarly, the wireless medium is shared and open to both benign interference (e.g., from wireless phones, microwaves, etc.) and malicious jamming.

DoS is an attack on system's resources. Network services are provided using a combination of resources at the hosts providing the services, at the clients requesting and receiving the service, and at the network carrying service traffic. Examples of shared resources are link bandwidth, router buffers, server memory, CPU cycles, and operating system structures. DoS attackers inject maliciously-designed packets into the network to deplete some or all of these resources: either the resource gets exhausted (by driving utilization to reach the resource capacity), destroyed (by forcing its capacity to zero), or captured (by acquiring control over the resource). When a resource is captured, it can be used to launch attacks against other resources or caused to behave unexpectedly in a stealthy way that is hard to detect [91].

With the above considerations, a DoS defense system is mainly challenged with the need to accurately and efficiently distinguish between legitimate and illegitimate traffic so that only illegitimate traffic is regulated. Attacks which employ legitimate-like traffic, such as reflector [111], and service-level [49] attacks, complicate accurate detection and limit the efficacy of filtering rules that are based on simple traffic features. Resource-intensive defenses, such as those based on stateful monitoring and heavy cryptography, are more complex and more easily overwhelmed with illegitimate traffic, and thus, become DoS attack targets themselves.

1.1 DENIAL OF SERVICE (DOS)

Denial-of-Service (DoS) is a major security problem in computer systems and networks [47, 48]. The DoS threat started to materialize in the Internet with the massive attack against the University of Minnesota in 1999 [37]. The access links of the university were flooded by packets launched from many machines, the links were completely hogged up by the attack packets, and legitimate (non-attack) packets were dropped. Since then, many DoS attacks have been and continue to be launched [98].

Gligor defined DoS as follows [161]: "a group of otherwise-authorized users of a specific service is said to deny service to another group of authorized users if the former group makes the specified service unavailable to the latter group for a period of time which exceeds the intended (and advertised) waiting time." Some instances of service denial are enabled by unauthorized information disclosure (confidentiality problem), unauthorized deletion and modification of information objects (integrity problem), and benign faults (fault-tolerance problem) [161]. For example, some DoS attacks are launched from compromised machines [36] (confidentiality breach), some use spoofed (forged) source addresses [20] (integrity breach), and some result from failure of system components (fault-tolerance problem). DoS instances that involve no violation of confidentiality or integrity nor traditional faults represent the heart of the DoS problem and are fundamentally harder to defend against because they require non-traditional approaches. In this work, I focus on these DoS instances, illustrated by the shaded area in Fig. 1.

In particular, I consider two DoS attack types, both falling in the resource-exhaustion category [91]: *service-level* DoS attacks, which target server resources by issuing legitimate-like service requests at a high rate to overwhelm the victim servers, and *link-level* DoS attacks, particularly the radio jamming attack. Both attack types are challenging: Service-level DoS attackers are hard to identify because they send seemingly legitimate traffic [49, 65], and jamming is hard to mitigate, especially with limited node resources [151].

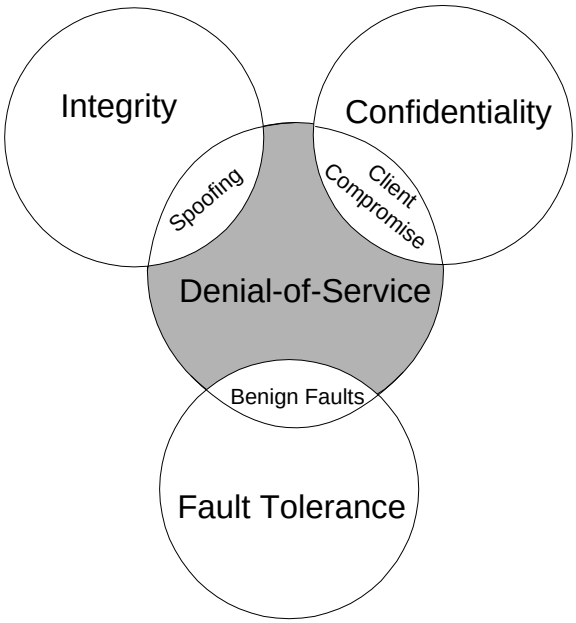


Figure 1: Denial-of-Service vs. integrity, confidentiality, and fault tolerance. This work focuses on the shaded area of the figure, not on the overlap with other fields.

1.2 DOS DEFENSE CHALLENGES

In general, four fundamental challenges contribute to the difficulty of the DoS problem.

- **Hard prevention.** Software bugs, misconfiguration, and human's inherent vulnerability to social engineering [96] all enable a remote attacker to cause a resource to be unavailable to legitimate users. For instance, an attacker may exploit a bug in a packet queue implementation that enables a specially crafted packet to set `tail = head`, effectively rendering queue capacity to zero.
- **Weak deterrents.** Sending attack packets is almost free, user authentication in order to identify attack sources is hard, and it is easy for the real attack perpetrators to hide behind an army of unwitting zombies that launch the attack on their behalf.
- **Difficult resource control.** Effective resource management and protection ensures that entities (e.g., users and processes) be allocated their resource shares and no entity can

gain more than its share. A challenge here is the definition of entities. Entities can be processes, sockets, IP addresses, user tokens, etc. The granularity of an entity determines how much control is available and how much overhead is incurred. For instance, enforcing fair resource allocation between open TCP sockets consumes less resources than enforcing fair-sharing between IP addresses, because the number of open sockets is usually much less than the IP address space. Controlling at the socket level, however, is vulnerable to an attacker with a single IP address gaining all socket resources and leaving nothing for other users. IP spoofing [46] and application-layer proxies [29] reduce the effectiveness of using the IP address as an identifier. A finer-grain control (e.g., based on application-level cookies) leads to a huge state overhead, especially in large services.

- **Difficult attacker identification.** It is hard to come up with a silver-bullet definition of malicious entities to block their access to attacked resources. Many attempts have been made to define characteristics of a malicious entity: sends one-way traffic [93], sends more traffic than normal [90], sends more "heavy" requests than normal [120], appears on system logs only during periods of attack [112], and is controlled by *bots* (non-human agents controlling compromised machines) [64]. These attempts are either specific to particular services or can be bypassed by new attacks. For instance, defense systems that detect non-human bots are only suitable for Web services and other services with "direct" human users. Also, defense systems that detect network-level anomalies are bypassed by service-level attacks that target server resources without causing congestion at the network level [49].

To overcome these challenges, the current techniques for DoS protection fall into three main approaches: *prevention, mitigation*, and *detection and recovery*. Although successful against some instances of the DoS problem, these approaches have limitations. DoS prevention is generally impossible unless all users comply with a *user agreement* that defines acceptable user behavior [161]. For instance, the notorious SYN flooding attack [126] resulted from violating the user agreement (as established in the TCP protocol [139]) to release a resource shortly after using it. When all users are trusted, the user agreement can be safely assumed, but when some users are not trusted (e.g., in an open service deployed over the Internet), the user agreement needs to be enforced by an external mechanism [50].

Nevertheless, existing prevention mechanisms add a layer of complexity that is sometimes ineffective or limits the scope of the service being protected. For instance, researchers have proposed that service clients should exert "effort", in terms of CPU cycles, memory accesses, or network bandwidth, in order to get serviced [62,99,145,147,148,149,154]. This mechanism provides only weak guarantees on service response-time [49]. Another DoS prevention example is the CAPTCHA puzzles (e.g., [99]), which distinguish humans from bots but restrict the protected service's clients to human users.

DoS mitigation mechanisms (e.g., replication [30] and overlay-based systems [69,99,135]) allow services to sustain acceptable performance under attack. However, they require resource over-provisioning and consequently suffer from high overhead. DoS detection is complicated by the fact that attackers tend to hide their traffic in such ways that make it look legitimate, making it extremely hard to come up with accurate and "efficient" (low filtering overhead) attack *signatures*. Identifying service-level DoS attackers, who send attack requests that are indistinguishable from legitimate requests, is a challenging task [120]. Another example is the SYN flooding attack, in which it is hard to distinguish between an attacker that intentionally keeps an entry in the TCP connection queue for a long time and a legitimate user that intermittently experiences long network delays before its ACK packets are received [126].

1.3 DODGING

As mentioned previously, the focus of this dissertation is on resource exhaustion attacks. Such attacks can be mitigated by *replication* to increase the total resource capacity beyond the attack resource consumption [69]. Resource-exhaustion attacks can also be detected and *evaded*, whereby the system is reconfigured to remove the resource under attack from the service path [41,136]. For instance, when a router queue is exhausted and legitimate packets start getting dropped, moving the service to a different location that avoids traversing the attacked router may lead to the attack being dodged.

This dissertation addresses the question whether replication and evasion can be combined

in a synergistic way that brings about benefits not attainable using any of the schemes by itself. The basic challenges in such a combination are how often should evasion be used, how much replication should be used, and what benefits can be achieved by the combination. The thesis of this dissertation provides a positive answer to this question: the replication-evasion synergy can be achieved using a careful combination of the three DoS defense approaches, namely prevention, mitigation, and detection-and-recovery.

This work puts forward the following thesis: **a hybrid of the three DoS defense approaches provides better protection than the individual approaches**. The thesis is supported by developing a novel hybrid DoS defense framework, which I call *dodging*. At a glance, dodging is achieved by continuous, carefully designed system reconfigurations with the two-fold goal of quickly identifying attackers and mitigating attacks. As will be shown in this dissertation, dodging-based defense mechanisms are more resources-efficient than pure mitigation mechanisms. Also, dodging leverages user agreements to allow for attacker identification mechanisms that are more accurate and more efficient than pure detection mechanisms.

Dodging is a hybrid of the three DoS defense approaches. First, dodging borrows the notion of user agreement from DoS prevention, but with the goal of identifying violators rather than enforcing the agreement. Second, dodging enables a novel approach for identifying DoS attackers, which I call Primary-Effect-based Detection (PED). PED is inspired by the observation that detecting DoS *attacks* is easier than identifying the attackers. Indeed, DoS attacks result (by definition [48]) in an increased response time, which can be easily detected. Meanwhile, identifying the attackers is challenging in general, especially when the attack traffic closely resembles legitimate traffic, which is the case in service-level DoS attacks, and when the attack is launched against a large service with huge client population [63]. Finally, dodging allows for mitigating the attack and sustaining service performance using a careful combination of replication and attack evasion.

An interesting analogy of dodging in the physical world is the boxing strategy of Muhammad Ali, which he refers to as:

Float like a butterfly, sting like a bee.—Muhammad Ali

Ali used to move continuously, very quickly, and unpredictably to his opponent on the boxing ring, causing his opponent to either (1) attack blindly and lose his strength chasing Ali or (2) focus on predicting Ali's moving pattern, miss and attack the wrong places, and thus, become easy to knock off-guard.

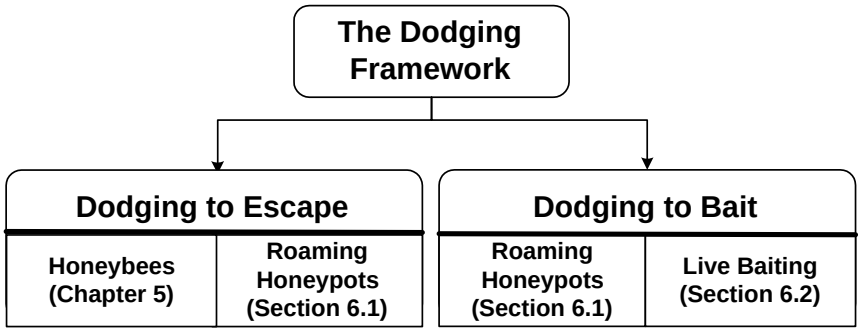


Figure 2: The components of the dodging framework.

Analogous to Ali's techniques, dodging has two components (Fig. 2): dodging to *escape* and dodging to *bait*. First, dodging causes attackers to dilute their "fire-power" trying to follow the system reconfigurations, hitting a few times, but losing "precious" time attempting to follow making it more probable that they are identified and filtered out. Second, dodging creates hard-to-evade baits, or traps, for attackers. Particularly, when attackers access the wrong configurations and even when they successfully follow the service reconfigurations, they are identified. Dodging incurs an overhead, analogous to the physical effort exerted by Ali in his continuous movement. However, this overhead is strategic, and it pays off by mitigating the DoS attack.

The proposed dodging framework raises the following challenges:

- How to use dodging to create hard-to-evade traps, and how to use these traps to identify attackers accurately, quickly, and efficiently?
- How to design a dodging strategy that has the best cost-benefit trade-off. This involves the questions of how frequently to dodge and how many traps to use.

- How to make dodging applicable to public services, such as google.com, which can be accessed by any client, and there is no a priori difference between legitimate clients and attackers. This is different from private services, in which legitimate clients are known a priori.

1.4 CONTRIBUTIONS

The two key contributions of this work are **the formulation and analysis of dodging as a practical and effective hybrid framework for mitigating denial-of-service** and **the formulation of the Primary-Effect-based (PED) detection approach**. Additionally, three dodging-based algorithms are developed and evaluated. The application of dodging through these algorithms yielded other contributions, which are described next.

1.4.1 Dodging Algorithms

Dodging is a dynamic mapping between clients and resources in an unpredictable way that allows for attack mitigation and attacker identification. Three dodging algorithms are developed and analyzed. As illustrated in Fig. 2, the three algorithms together achieve the two components of dodging, namely dodging to escape and dodging to bait. The first dodging algorithm, namely **honeybees** [74, 73] (Chapter 5), applies dodging in the wireless networks arena, whereby dodging allows for energy-efficient and bandwidth-efficient mitigation of radio jamming in multi-radio wireless networks [15], where each node is equipped with multiple radio interfaces. Honeybees efficiently augments the redundancy of radio transceivers in multi-radio devices with a layer of evasion by *channel hopping* [54, 100, 153, 155]. The honeybees algorithm combines the software-based channel-hopping technique with Error-Correcting Codes (ECC) (e.g., [118]). First, the best channel-hopping strategy (reactive or proactive) is determined by formulating the jamming problem as a max-min game between defense and attack, and through simulation the dependency of the game outcome on the payoff function is illustrated. Reactive channel-hopping was found to provide better jam-

ming tolerance when considering communication availability. However, both reactive and proactive defenses had almost the same performance when energy consumption was considered. Reactive channel-hopping is further studied through theoretical modeling to aid in understanding the inherent trade-offs between throughput and jamming-tolerance.

The contributions of the honeybees algorithm can be summarized as follows:

- The problem of maximizing goodput under jamming in multi-radio networks using a combination of channel-hopping and ECC is formalized.
- Models for different combinations of defense and attack hopping strategies are developed and validated using simulation. The models incorporate the effect of ECC redundancy on communication availability.

The second algorithm, which I name **roaming honeypots** [77] (§6.1), significantly improves the usage of honeypots [116] against directed (vs. random) DoS attacks by camouflaging honeypots as servers [71, 72, 76, 77, 123, 124]. The algorithm moves the protected service among replicated servers continuously and unpredictably to attackers. Secure, distributed, randomized, and adaptive algorithms for triggering the roaming and determining the active servers are developed. Also, modifications to the state recovery process of existing TCP connection-migration schemes [129, 141] to better suit roaming are proposed. Results from simulation and prototype experiments show that the overhead, in terms of response time, of roaming is small in the absence of attacks. Further, during an attack, roaming significantly improves the response time. Shuffling requests over servers opens windows of opportunity for legitimate connections to get serviced, allowing the service to sustain performance under attack. Also, a good trade-off between costs and benefits of roaming is reached by carefully setting both the frequency of roaming and number of active servers. The algorithm is presented and analyzed in §6.1.

The third algorithm, namely **live baiting** [70] (§6.2), identifies service-level DoS attackers and scales to large services with millions of clients [70]. Live baiting requires low state overhead, enabled by a novel leverage of the group-testing theory [42, 43, 57, 67, 102, 138], which aims at detecting "defective" members in a population using the minimum number of tests.

The merits of the roaming honeypots and live baiting dodging algorithms are as follows.

- They quickly and efficiently identify DoS attackers and protect large services with large client populations. In live baiting, identification of DoS attackers is done with manageable memory and computational overheads.
- They protect open services whose legitimate clients are not known a priori, and they are not restricted to services with human clients who can solve puzzles, such as CAPTCHA [49, 144].
- Live baiting is the first work to apply group-testing theory to the DoS attack problem. Moreover, live baiting can detect the actual number of attackers and adaptively adjust its parameters. This adaptability is a novel contribution to group-testing theory.
- Both algorithms require no modifications inside the network and only require minor server and client modifications, thus reducing the introduction or addition of vulnerabilities.

According to the two dodging components and the type of attackers, I divide the DoS problem space into four parts shown in Fig. 3. Attacker compliance refers to whether or not attackers follow the system reconfigurations introduced by the dodging algorithms (more details in §3.4). The defense algorithms presented in this work differ in the dodging component they provide and the attack types they cover. As I will show throughout this dissertation, the honeybees algorithm (Chapter 5) mitigates non-compliant attacks, and the roaming honeypots algorithm (§6.1) provides attack mitigation against both compliant and non-compliant attackers. Roaming honeypots also identifies non-compliant attackers, and live baiting (§6.2) identifies compliant attackers.

1.4.2 Primary-Effect-based Detection (PED)

Dodging enables a new approach for attacker identification, which advocates a departure from the three main detection approaches, namely anomaly-based (e.g., PacketScore [7, 14] and DDoS Shield [120]), misuse-based (e.g., the Snort Intrusion Detection System [4]), and specification-based (e.g., RED-PD [89]). Although successful in many instances of the DoS problem, the current detection approaches have limitations, which I discuss in §2.1.2.

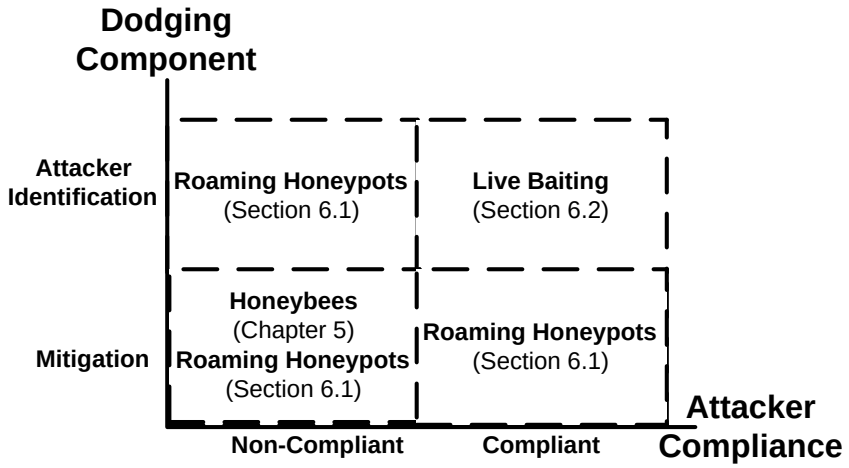


Figure 3: The DoS problem space is divided into four parts along the attacker-compliance and dodging-component dimensions.

In the novel attacker detection approach enabled by dodging, which I name Primary-Effect-based Detection (PED), DoS attackers are identified based merely on the unavailability of the attacked resources, without the need to develop models of legitimate behavior, keep track of attack signatures, or tightly specify legitimate behavior. The simplicity of the signals used to identify attackers enables PED to overcome the limitations of current detection approaches.

1.5 ROADMAP

The rest of the dissertation is organized as follows. In the next chapter, background material as well as state-of-the-art in DoS defense are presented. Chapter 3 describes system and attack models. In chapter 4, the dodging approach, its components, and its properties are presented. The three dodging algorithms, honeybees, roaming honeypots, and live baiting are presented in Chapter 5, §6.1, and §6.2, respectively. The simulation study to determine the best channel-hopping strategy (reactive or proactive) in the honeybees algorithm is presented

in §5.1. §5.1 also presents a formulation of the jamming problem as max-min games between defense and attack and an analysis of the game outcome using simulation. Models of reactive channel-hopping are presented in §6.2.3, whereby the adaptive honeybees algorithm is also described. Results from simulation and prototype experiments of the roaming honeypots algorithm are presented in §6.1.4. Finally, the live baiting algorithms is analyzed theoretically and through simulation in §6.2.5. Chapter 7 summarizes the contributions and describes their foreseen impact.

2.0 BACKGROUND AND RELATED WORK

In this chapter, dodging is qualitatively compared to the state-of-the-art in the three main DoS protection approaches, namely prevention (§2.1.1), detection-and-recovery (§2.1.2), and mitigation (§2.1.3). Dodging, as a hybrid of the three approaches, qualitatively outperforms the detection and mitigation approaches individually. Then, a review of the background material behind the dodging algorithms is presented. The roaming honeypots algorithm employs TCP-connection migration (§2.2.1) and improves the usage of honeypots (§2.2.2) against DoS attacks. Live baiting leverages results from group-testing theory (§2.3). Finally, the honeybees algorithm defends multi-radio networks (§2.4.4) against radio jamming (§2.4.1) by combining channel-hopping (§2.4.2) and error-correcting codes (§2.4.3). Results from game theory [106], max-min games in particular(§2.4.5), are used to model the honeybees strategies.

2.1 STATE-OF-THE-ART IN DOS DEFENSE

Gligor [47] defined the DoS problem and described its relationship with other security and fault-tolerance problems. DoS defense approaches can be classified into prevention, detection-and-recovery, and mitigation. Mirkovic *et al.* presented an extensive taxonomy of DoS attack and defense techniques [94]. The relationship between dodging and state-of-the-art DoS defense mechanisms, classified based on the network-stack layer they defend, is summarized in Table 1, which indicates that dodging provides mitigation against link-layer DoS and both detection and mitigation against service-level DoS. Although dodging borrows the user-agreement concept from the prevention approach, dodging does not prevent

attacks; as discussed earlier, prevention is impossible unless the user agreement can be enforced. Dodging does not attempt to enforce the agreement [1], but define it in a way that allows easy identification of violators.

2.1.1 Prevention

Prevention mechanisms (if successful) provide strong guarantees on the service and network availability even under attack. It has been shown that prevention is impossible in general unless a user agreement that defines acceptable user behavior is defined and externally enforced [161]. Recently, some prevention mechanisms achieved strong guarantees that depend on attack parameters, in particular the number of attack machines [109]. Examples of the user agreements include exerting an "effort" before getting serviced [62,99,145,147,148,149,154], making a reservation prior to accessing the service [110,49], and paying for the service [24]. Some mechanisms have focused on preventing particular attack methods, such as anti-spoofing mechanisms [23,46,68,88,93,108,148,154] and physical-layer anti-jamming [103,114], by enforcing rules that make these attack methods impossible or at least hard to launch.

Client puzzles [62,99,145,147,148,149,154] and ticket-based systems (e.g., [110] and the rate-control service [49]) are indeed effective against many DoS instances but fall short of preventing service-level DoS attacks in general. For instance, the client puzzles approach assumes that an attacker does not have enough resources to solve hard puzzles at a high rate or is not willing to perform a hard computational problem on a compromised machine in order to remain undetected to the machine's user. However, a legitimate client is also required to perform the same heavy computation, and users of compromised machines may not easily distinguish between high CPU utilization due to legitimate requests and due to attack requests launched from their machines. Furthermore, some of these techniques require human intervention in sending the requests (e.g., solving CAPTCHA puzzles in [99]) or assume trusted hardware (e.g., [49]).

Wide deployment of spoofing-prevention mechanisms (e.g., [23,46,68,88,93,108,148,154]) does not prevent DoS attacks at the service level and does not stop non-spoofing attacks.

[1] Enforcing user agreements can be very hard in open systems, such as the Internet, where some users are not trusted.

Table 1: Dodging and state-of-the-art in DoS defense

	Prevention	**Detection and Recovery**	**Mitigation**
link-level DoS (jamming)	Directional (sectored) antennas [103]; Spread-spectrum [114];	Channel Surfing [155]; Spatial Retreats [157]; JAM [152]; Data blurting and schedule switching [82, 10]; **Dodging**	Worm-holes [26]; Proactive Channel-Hopping [54, 100, 153]; Agility [104, 87]; **Dodging**
network-level DoS	Network-level Puzzles [145] [149, 154]	PacketScore [7]; RED-PD [89]; Heavy-hitter detection [44]; DCAP [34]; Pushback [90]; MOVE [136]; Capabilities [13, 158]; IP Hopping [61]	Replication [30]; Overlay-based [12, 69, 99, 135]
service-level DoS	Application-level Puzzles [62] [99, 147, 148]; Reservation-based [49, 110]	DDoS Shield [120]; Shadow Honeypots [11]; Kill-Bots [64]; **Dodging**	Replication [30]; **Dodging**

(Non-spoofing attacks at the service-level are addressed by this dissertation.) For instance, Liu *et al.* propose to append a secure source identifier, which they call passport, to the IP packet header. Packet passports are hard to forge and are checked by intermediate routers

to drop invalid ones [88]. D-WARD [93] and ingress filtering [46] are designed to filter out one-way, spoofing, flooding attacks.

Brustoloni proposed the VIPnets architecture, which uses QoS mechanisms to protect traffic from paying users of e-business websites [24]. These mechanisms are designed to be effective against particular attack methods or in particular service models, but their effectiveness against other attack methods and service models is limited.

2.1.2 Detection and Recovery

The three main approaches for identifying attack packets are anomaly-based, misuse-based, and specification-based [21]. Anomaly-based mechanisms identify attackers when their behavior deviates from normal behaviors that are learned at peace (no-attack) time. Examples are the DDoS Shield [120] and PacketScore [7] systems. However, the anomaly-based approach is highly susceptible to errors if its learning phase gets manipulated by attackers to learn false behavior [101]. Also, attacks with behavior similar to legitimate behavior pose a problem to the anomaly-based approach. Misuse-based mechanisms (e.g., the Snort system [4]) identify attackers when their behavior matches pre-known attack patterns. However, this approach is not effective against attackers that mimic legitimate behavior and against attack mechanisms that are previously unknown. Finally, specification-based mechanisms (e.g., RED-PD [89]) identify attackers when their behavior deviates from a specification of legitimate behavior. Again, this approach cannot identify attackers with legitimate-like behavior. Moreover, in many cases attackers can cause damage while complying with specifications, because of the difficulty of developing "tight" specification of legitimate behavior. In what follows I discuss examples of each detection approach that are most relevant to dodging.

Anomaly-based detection. Ranjan *et al.* propose a scheme to mitigate service-level DoS attacks in an open Web service, making it most relevant to the work presented in this dissertation. They enumerate a set of abnormal request behaviors that DDoS attackers use and propose a session scheduling algorithm based on suspicion level that needs to keep state for each session [120]. The live baiting algorithm presented in §6.2 scales to the number of

sessions by using an amount of state proportional only to the number of attackers, and it does not require models of legitimate sessions.

Misuse-based detection. Kandula *et al.* has improved CAPTCHA-based defenses by identifying attackers that keep sending wrong solutions at a high rate [64]. Hussain *et al.* developed a detection algorithm based on spectral analysis to detect repeated attacks, e.g., attacks from the same set of attackers [56]. Jin *et al.* proposed the hop-count filter to detect and block packets with spoofed source addresses [60]. Estan and Varghese proposed an elegant algorithm that requires low memory overhead to detect "heavy-hitter" network flows, making it suitable for in-router implementation [44]. Their sample-and-hold technique is proposed for network-level detection and requires operators to set a threshold to define heavy-hitters. For instance, a heavy hitter may be defined as any flow that uses 10% or more of link capacity. In the live baiting algorithm (§6.2), a heavy hitter corresponds to a service-level DoS attacker, and although a threshold, in terms of number of request drops, is required in live baiting, this threshold is easier to design than sample-and-hold thresholds in the context of DoS detection. This relative ease is because in a well-provisioned server, there are typically very few request drops under no attack, so the live baiting threshold can be set to a small number once and for many services, while a sample-and-hold threshold would require knowledge about legitimate client behavior for each service (more details in §6.2).

Specification-based detection. Chuah *et al.* [34] proposed an $\sqrt{N}$ distributed algorithm for detecting flows that deviate from traffic specifications. The RED-PD mechanism [89] identifies misbehaving flows when their bandwidth utilization exceeds the bandwidth obtained by a *reference* TCP flow. Preferential packet dropping is then applied to the identified flows.

The location of the detection system varies. For instance, Sekar *et al.* developed a triggered, multi-staged automatic DDoS detection system for a large ISP to accurately identify customers under attack [127].

It has been shown that in multi-source DoS attacks, the attack rate, as observed at the victim, exhibit a ramp-up behavior, that may take from 200 ms to 14 seconds [55]. This behavior is mainly caused by the staggered starting times of attack zombies and makes roaming honeypots and live baiting very suitable to effectively detect and filter out zombies

and throttle the attack traffic before reaching its peak rate.

Recovery schemes depend on the accuracy of attack detection. For instance, in-network filtering schemes (e.g., Pushback [90] and Max-min-fair throttling [160]) do not distinguish normal-rate DDoS attackers from legitimate clients and thus propagate filters to block traffic from both [90].

In capability-based schemes, such as [158, 13], attack victims recover by revoking access from detected attackers. For instance, Kreibich *et al.* proposed a scheme in which a victim server can implicitly block traffic from detected attackers by simply not replying to the attack traffic; in their scheme, in-network filters, which enforce packet symmetry of passing flows, will drop the attack traffic [79].

Migrating the service to a new location has also been proposed as a recovery mechanism for private services from DoS attacks [40, 136]. However, service migration provides limited protection if the attacker can quickly discover the new service location.

Another approach to recover from spoofing attacks is to trace the detected attack packets back to their sources (e.g., [125, 19, 133, 107, 38, 159, 132, 85, 140, 25, 122, 105, 90]). Traceback schemes can be classified into packet marking schemes and hop-by-hop schemes.

Packet marking schemes (e.g., [125, 133, 107, 38, 159, 105]) construct attack paths locally at the victim by collecting markings stamped into packets by intermediate routers. However, these schemes are vulnerable to compromised routers, which can inject forged markings to increase the number of false positives [133]. Authentication schemes have been proposed to solve this problem [133], but they require high computational overhead at the routers and high storage overhead at the victims to maintain router keys. The Path Identifier (Pi) is a deterministic packet marking scheme that approximately identifies the path took by a packet. The attack victim uses the Pi mark to filter out malicious packets on a per-packet basis.

Hop-by-hop traceback: The basic idea of hop-by-hop schemes is that traceback starts at the router next to the attack victim, where neighboring routers upstream on the attack paths are identified using a signature of attack packets. This process continues until the attack machines are reached. CenterTrack uses an overlay network to determine ingress points of attack traffic into a network cloud (an AS for instance) [140], and controlled flooding injects

packet floods into the network and detects attack paths based on traffic perturbations [25].

Pushback automates the hop-by-hop traceback process, whereby aggregate-based, rate-limiting filters are propagated upstream from congested routers [59]. Both detection of misbehaving aggregates and assignment of rate limits are done using the Aggregate-based Congestion Control (ACC) mechanism [90]. The honeypot back-propagation scheme [72] can be viewed as a realization of this feature; when a server takes the role of a honeypot, the server's destination address defines the malicious aggregate.

Hop-by-hop schemes are less vulnerable to false positives caused by compromised routers than packet-marking schemes; if a compromised router maliciously drives the traceback process into an upstream router not on the attack path, traceback will stop at that router because the attack signature will not be matched by any packets. Hop-by-hop schemes, however, require the identification of an accurate attack signature, that is, one that leads only to attack sources and that remains unchanged until the attack sources are reached. An exception is the single-packet traceback scheme [132,85], which can use a single attack packet as the signature. However, it requires high storage overhead at routers or high bandwidth overhead. Hop-by-hop schemes require cooperation of many ISPs as well. Honeypot back-propagation is an efficient hop-by-hop scheme that extracts accurate attack signatures and provides deployment incentives for ISPs [72].

Most traceback schemes take a long time to collect the number of packets needed to accurately construct each attack path. Consequently, they produce a large number of false positives if a large number of low-rate attackers launch an on-off attack with short bursts (e.g., the shrew attack [81]). To address this limitation, progressive honeypot back-propagation can trace back to low-rate attackers within a reasonable time and with a low false positive rate [72].

IP hopping is another recovery mechanism that can protect a public server, whose clients use DNS to look up its IP address, against specific DoS attacks that use hard-coded IP addresses to locate their targets [61]. IP hopping can be triggered reactively, whereby the server changes its IP address after detecting that it is under attack, or proactively in anticipation of attacks. In either case, the service does not change its physical location. All packets destined to the old IP address are filtered at the network perimeter by a firewall.

To avoid continuous server reconfiguration, a NAT (Network Address Translation) gateway can be used. One limitation of this mechanism is that during the period of time in which the DNS entry of the old IP address is cached, all legitimate client requests using that entry are filtered out. Also, IP hopping does not block a persistent attacker that looks up the new IP address using DNS. Moreover, illegitimate packets will still go to the same network after the address is changed, potentially congesting upstream routers. Finally, the service is still vulnerable to malicious state entries possibly implanted in the server during the attack. By physically moving the service from one server to another and cleaning the state of the old servers, the roaming honeypots algorithm, presented in §6.1, avoids the limitations of logical roaming schemes, such as IP hopping.

2.1.3 Mitigation

Mitigation-oriented defense mechanisms aim at maintaining acceptable system performance under attack. Redundancy is a common mitigation strategy against DoS attacks. Two ways to use redundancy are the addition of extra service replicas and standby sparing. Whereas service replication can distribute the DoS attack load over the replicas, its inability to distinguish between legitimate and illegitimate requests makes it prone to failure under heavy-load attacks that target all replicas simultaneously. Even with a smaller number of packets, some DoS attacks can bring down all replicas almost simultaneously [31]. Spare servers have been suggested as a mechanism to mitigate DoS attacks [30], whereby a spare takes over the functionalities of an active server when the latter comes under attack. However, the effectiveness of this scheme is strongly limited against fast-building, heavy-load attacks that leave very little time to transition successfully to the spare server. It is also the case that the spare itself can come under attack.

Another form of redundancy is the overlay-based systems, such as SOS [69, 99, 135], Mayday [12], and DAM [35], which are effective in protecting private services, with pre-known clients, and services that require human intervention, such as the Web. Vasudevan *et al.* has proposed a framework for automatic, on-line evaluation of the effects of DDoS defense mechanisms, such as black-holing (directing attack traffic away from the paths of legitimate

traffic through routing updates) and traffic scrubbing (deep inspection of suspicious network flows to identify attacking flows) [143].

Kargl *et al.* proposed a defense scheme based on class-based queuing and traffic monitoring to protect Web servers from DDoS attacks [65]. Their results (along with results from other researchers) highlight the challenges introduced by service-level attacks, which target the CPU, and are harder to detect because they do not trigger any network-traffic alarms.

TDMA-based protocols and multi-frequency link-layer protocols, both static [162] and dynamic [16], provide jamming mitigation as long as the TDMA schedule is secure. To mitigate schedule compromise, data blurting with schedule switching [82] and data exfiltration (by time-multiplexing redundant data over multiple channels) [10] have been proposed.

2.2 ROAMING HONEYPOTS BACKGROUND

2.2.1 Connection Migration

Several schemes have been proposed to support TCP connection migration. Most of these schemes are designed to operate in a wireless, mobile environment and focus primarily on supporting mobility [130] or achieving high tolerance to faults and attacks [129, 142].

Two well-known TCP-connection migration mechanisms are the TCP-Migrate [129], developed at MIT, and the Migratory-TCP [142], developed at Rutgers University. Both attempts provide the framework for moving one end point of a live TCP connection from one location and reincarnating it at another location having a different IP address and/or a different port number. Both mechanisms deal with four issues in a slightly different way: (1) how the TCP connection is continued between the new end points; (2) impact on the network stack and application layer in both the server and the client sides; (3) how to recover both TCP and application states; and (4) when to trigger the migration mechanism. The last two issues are considered independent of the actual migration framework and are presented as examples of possible usage of the mechanism.

In MIT's TCP-Migrate, during connection establishment, the migration feature is re-

quested through a TCP option (Migrate-Enabled). By the means of a handshaking protocol, a shared key is established between the two connection end points. As per a migration request from one end point, represented by another TCP option (Migrate), the TCP control block at the fixed end point is updated to reflect the new location of its peer. To protect against connection hijacking, the secret key agreed upon during the connection establishment should accompany the migration request. As an application of the TCP-Migrate mechanism in a fine-grained fail-over scheme [129], state recovery in the new server is achieved via periodic state updates from the old server to the server pool. A widget implementation is responsible for extracting HTTP state from TCP packets. The migration request is issued by a new server and triggered by an overload at the old server, detected by a health monitor. Implementations at both the transport, whereby the TCP-layer in both the server and the client needs to be changed, and session layers are available. However, no application layer updates are necessary, although the widget implementation is already application-dependent.

During connection establishment between two Migratory-TCP-enabled peers, a list of available servers, along with a certificate for each server, are passed from the server to the client. A migration request, also implemented as a TCP option, consists of both the certificate of the new server and the connection identifier (client IP address, client port, old server IP address, old server port) of the migrating connection. However, no security measures are implemented to protect the migration process. State recovery at the new server is achieved either on-demand, that is, when the client sends the migration request to the new server, or through periodic state updates. Triggered by an internal QoS monitor in its kernel, a client can issue a migration request to any server in the server list which the client receives in the connection establishment phase. Both the server and the client TCP layers should be changed and the server application layer should also be modified to allow for application-layer state snapshots and for state recovery at the new server. It should be noted that the limitation of the on-demand state update in the case of old server's failure or crash due to an attack was mentioned briefly in [142]. As an alternative, it was proposed to send state check-points to the client and use this client-stored state to recover the connection at the new server. An approach similar to [141,131] has been adopted in the roaming honeypots algorithm (§6.1).

Client and server transparency, whereby client and server software does not need to be changed, is a favorable property to ease deployment burden. Although the techniques proposed in [6] achieve client and server transparency, they require that both the primary and backup servers share a broadcast medium. As such, these techniques are not adequate to handle connection migration in wide-area network environments.

2.2.2 Honeypots

Baiting, or setting traps for malicious users, has been successfully used as a detection mechanism against specific types of DoS attackers [11,72,77,80,83,116,117,150]. The more hidden the traps, the harder they can be evaded by attackers.

Honeypots [116], or decoy systems deployed in unused IP address ranges, have been successfully used as traps for many types of malicious traffic, such as random-scanning worms [150, 83, 117, 80]. Physical honeypots run full versions of services, and "toned down" versions are typically run by virtual honeypots (e.g., Honeyd [117]). The effectiveness of honeypots against DoS attack traffic is limited, however, because they can be easily detected and consequently avoided by DoS attackers. This ease of detection stems from typical honeypot deployment at fixed addresses, particularly unused IP address ranges, and on machines that act exclusively as honeypots. In a sense, honeypots represent "dead baits" that are easy to detect by attackers. The roaming honeypots algorithm (§6.1) camouflages the honeypots as servers to increase their ability to trap DoS attackers.

In shadow honeypots, suspected requests are sent to specially instrumented servers to detect attacker attempts to exploit vulnerabilities [11]. This helps reduce the false alarm rate of intrusion detection systems. Finally, in the live baiting algorithm (§6.2), the service itself is the trap, or a "live bait", and, thus, the trap is more attractive to attackers and hard to evade.

2.3 GROUP TESTING THEORY

The first application of group testing [138, 42] was during WWI; instead of testing every blood sample individually, groups of samples were pooled together and tested collectively [138], whereby if the outcome of the group test is negative, all samples in the group are considered to be good (disease-free). Although group testing has been used in many security and networking applications, such as data forensics [52], cryptography [33], multiple-access channels [22], and broadcast security against jamming [39], this work is the first to apply this powerful theory to the DoS attack problem.

Group testing aims mainly at identifying the defective (special) members of a population with as few tests as possible. There are two classes of group-testing mechanisms [102]: *Non-adaptive*, or single-stage, techniques conduct all tests simultaneously with no feedback from previous test results, whereas *adaptive* (multi-stage) techniques use previous test results to determine subsequent tests. This work uses a simple non-adaptive group-testing scheme as a proof-of-concept, but adaptive schemes can be used as well.

Non-adaptive group testing uses a $n_t \times n_c$ matrix to specify tests [102], where n_c is the total number of members and n_t the number of tests. The matrix rows represent tests and columns represent group members. When a matrix element (i, j) is set to 1, this means that member j participates in test i. An example of a group testing matrix for a population of 10 members with 4 tests is shown in Fig. 4.

	Members (defectives underlined)										*Test Results*
	1	2	3	4	5	6	7	8	9	10	
Tests	0	1	0	0	0	0	1	1	1	0	1
	1	0	1	0	0	0	1	0	1	1	0
	0	0	0	1	0	1	0	1	0	1	1
	1	0	0	1	1	0	1	0	0	0	0

Figure 4: An example of a group-testing matrix.

Test results are represented as a vector with an element for each test. For simplicity, binary test results are assumed in this dissertation. So, a test result is set to 1 if the

corresponding test returns positive, that is, if the test was applied to a group with at least one defective member. In the example shown in Fig. 4, the $\underline{2}nd$, $\underline{6}th$, and $\underline{8}th$ members are defective, and consequently, only the first and third tests have positive results.

Detection of defective members. The detection algorithm discovers the defective members using the result vector and the matrix. The algorithm adopted in this work starts with all members in a *suspect list* and excludes members from the list if they participate in a "large enough" number of tests that have negative results. For instance, consider that a member has to participate in only *one* negative test to be excluded, then in the above example, members 1, 3, 7, 9, and 10 are excluded because they participated in the second tests, which has a negative result. Similarly, members 4 and 5 are excluded because of their participation in the fourth test. Thus, only the (defective) members 2, 6, and 8 are left as suspects. In this specific example, all defective elements are detected, and all non-defective members are cleared.

Matrix construction. Many matrix construction algorithms have been proposed, such as the deterministic construction by Hwang and Sós and the randomized construction by Dyachkov and Sebo [102, 43]. The constant-weight matrices, d-disjunct matrices, and superimposed codes are all examples of matrices that can be used in the non-adaptive group testing [43]. For simplicity, this work employs a randomized construction method [52, 102], in which each bit in the matrix is set to 1 with a fixed probability p. As will be discussed later, the value of p is derived to maximize the detection accuracy.

False Positive and False Negative Probabilities. A false positive is when a non-defective member gets falsely identified as defective, whereas a false negative is when a defective member ends up not detected. In the example in Fig. 4, both the false positive probability and the false negative probability are 0. In general, Goodrich *et al* [52] showed that the simple detection algorithm discussed above detects all defective members with the false positive probability $FP = [1 - p(1-p)^d]^{n_t}$, where d is the number of defective members in the group and T is the number of tests used to detect defective members. By differentiating the above equation with respect to p, the optimal value of p, the value that yields the minimum false positive probability, is $\frac{1}{d+1}$ [52]. Thus, the minimum false positive

probability for a given number of tests n_t is:

$$FP = [1 - \frac{1}{d+1}(1 - \frac{1}{d+1})^d]^{n_t} \tag{2.1}$$

Finally, from Eqn. 2.1 the number of tests, n_t^{fp}, required to achieve a target false positive probability fp can be derived as:

$$n_t^{fp} = \frac{log(fp)}{log(1 - \frac{1}{d+1}(1 - \frac{1}{d+1})^d)} \tag{2.2}$$

As will be shown in §6.2, n_t^{fp} is $O(d)$ for a given target fp, that is, the number of tests is in the order of number of defective members (attackers) not the total number of members for a given target false positive probability. This observation is key to the scalability of the live baiting algorithm.

The above equations assume that an estimate of the number of defective members (d) is known a priori. It is also assumed that if a defective member participates in a test, the test result is always negative, that is, test results are reliable and a defective member cannot arbitrarily alter the result of a test. These assumptions may not hold in DoS attacker detection. First, it may be hard to estimate a priori the number of attackers. Second, DoS attackers want to avoid detection and may maliciously alter the results of tests (the meaning of tests in the DoS attack domain and the mapping from group-testing domain to DoS attack domain are presented in §6.2.1) to obscure their presence. Both these assumptions are relaxed in §6.2.3 and 6.2.4, respectively.

2.4 JAMMING MITIGATION IN WIRELESS NETWORKS

2.4.1 Radio Jamming

Radio jamming is a DoS attack targeting physical and link layers of wireless networks [151, 156]. Radio jamming aims at preventing sender nodes from accessing the shared wireless medium by keeping the medium busy or from successful reception by causing high radio

interference at the receiver. Many anti-jamming techniques have been proposed, spanning many layers in the network stack.

Physical-layer defenses: Physical-layer anti-jamming techniques, such as directional (sectored) antennas [103] and spread-spectrum [114], create hard-to-jam "virtual channels" or "wormholes" within the shared wireless medium [26, 53, 146]. Sectored antennas are potentially effective but not widely deployed. Nodes equipped with spread-spectrum radio chips [32, 84, 115] are still vulnerable to jamming from nodes with similar radios [153]. Moreover, the recent 802.11a and g standards have replaced frequency hopping, which tolerates jamming but has low bandwidth, with high-bandwidth channel coding schemes (OFDM), which are vulnerable to jamming [100].

Link-layer defenses: At the link layer, channel hopping improves jamming resiliency in both 802.11 [54, 100] and sensor networks [155, 153]. Channel hopping will be discussed in detail in the next section. TDMA-based protocols and multi-frequency link-layer protocols, both static [162] and dynamic [16], mitigate low-power, selective jamming as long as the TDMA and frequency-switching schedules are secure. To mitigate schedule compromise, data blurting with schedule switching [82] and data exfiltration (by time-multiplexing redundant data over multiple channels) [10] have been proposed.

Network-layer defenses: The Jammed-Area Mapping (JAM) scheme [152] identifies regions of jammed sensors to be avoided by routing protocols. Jammed sensors "sleep" to outlast jammers. However, intelligent jammers can detect the communication silence and adjust their power consumption accordingly. In spatial retreats, jammed mobile nodes change their physical locations away from jammed areas [157]. This work differs in that the goal is to allow jammed nodes to communicate *while* the jamming attack is on.

All previous work studies jamming in the context of single-radio networks. This work investigates jamming mitigation in the multi-radio context as well, whereby multiple radios cooperate to deliver the data and multiple jammers conspire as well to cause maximum damage.

2.4.2 Channel Hopping

Channel hopping, whereby channel switching is controlled at the software-level, has been proposed to mitigate jamming in single-interface wireless sensor networks (WSNs) and 802.11 networks [54, 100, 153, 155]. Channel hopping utilizes the fact that there is a number of orthogonal radio channels in many of today's wireless standards. The 802.11a standard has been reported to have 12 orthogonal channels [100], 802.15.4 (e.g., CC2420 radio in MICAz motes) has 16 channels [153], and even the older CC1000 radio in Mica2 motes has been reported to allow up to 32 orthogonal channels in the 900MHz band [155]. I have experimented with 4 orthogonal channels on the Mica2 motes in the 433MHz band, 3 channels on 802.11b/g, and 13 channels on 802.11a. In all these standards, a radio cannot transmit or receive while switching channels.

Different values of the channel-residence time between hoppings have been used to serve different purposes. Whether channel hopping is implemented at the driver or user levels has an effect on the attainable granularity of the channel-hopping frequency. Channel hopping has been implemented to occur every few microseconds [153], enough to send a small packet fragment, every few milli-seconds [16, 54], every hundred milli-seconds [100], and every few seconds [95]. Wood *et al.* show that the packet fragment time should be small (in the order of the delay needed to switch channels) to prevent a fast-switching attacker from disrupting communication on all channels [153]. This work follows their recommendation and uses short packets (in the same order as channel-switching delay).

Two variations of channel hopping have been proposed. Proactive channel-hopping has been shown to improve resiliency to jamming in both 802.11 [54, 100] and sensor networks [153]. Against static and scanning attackers, channel hopping was enough to improve throughput in 802.11a and 802.11b networks [54, 100]. However, against fast-switching attackers, channel hopping was not enough; packet fragmentation and redundant encoding were needed to defend against this type of jamming [153]. Channel hopping is coordinated synchronously [54, 100, 153] assuming loose clock synchronization. Asynchronous channel hopping has also been proposed but only for low-bandwidth message delivery [26]. Reactive channel-hopping, or channel surfing, occurs after radio jamming is detected and causes the

entire network or only the jammed region to switch to a different radio channel [155].

Again, previous channel-hopping research has only considered channel hopping in single-radio networks. This work generalizes the channel-hopping technique to multi-radio networks and studies its interaction with data redundancy using error-correcting codes.

2.4.3 Error-correcting Codes (ECC)

Error-correcting codes (ECC) and cryptographic bit interleaving have been proposed to mitigate low-power jamming attacks against data networks [104]. Alone, these techniques are not effective against high power or link-layer jammers. They also incur unnecessarily high communication and processing overhead if ECC parameters are not carefully selected. This work introduces the problem of optimizing communication *goodput* by combining ECC and channel-hopping.

In an ECC scheme, the piece of data to be transmitted reliably is augmented with carefully-designed redundancy. The augmented data piece is then divided into a number, n, of smaller pieces, which then get transmitted over the unreliable communication channel. Finally, the original piece of data can be recovered from any combination of m out of the n pieces. The ECC is usually described as a tuple (n, m). The goodput of the communication channel is reduced by the amount of redundancy. In particular, in the Information Dispersal Algorithm (IDA), goodput is $\frac{m}{n}$ of the channel throughput [118].

2.4.4 Multi-Radio Wireless Networks

Multi-radio wireless networks have been proposed to increase overall network capacity by exploiting channel diversity [15]. For instance, multi-radio 802.11 mesh nodes [2] are equipped with multiple radio interfaces operating at orthogonal channels to decrease interference between parallel streams of data at different radio channels. Also, sensors equipped with multiple radio chips [3] are used to increase throughput and/or reliability.

Table 2: An example payoff function for a max-min game.

	y_1	y_2	y_3
x_1	4	7	2
x_2	5	6	8

2.4.5 Max-min Games

A max-min game (also known as a zero-sum game) [106] involves two players with each player's gain maximized when the gain of the other player is minimum. Each player has a set of actions, and a payoff function is defined over the action pairs. The payoff of one player can be viewed as the negative of the other's. An example of a payoff function is shown in Table 2. Player 1 has two actions: x_1 and x_2. Player 2 has three actions: y_1, y_2, and y_3. The values shown in the table are those of player 1 (payoff for player 2 is the negative of these values).

The steady state of a max-min game is when each player cannot get a higher payoff by acting unilaterally. The *Nash equilibrium* [106] represents this state. An action pair (x^*, y^*) is a Nash equilibrium if and only if each of the actions x^* and y^* is a *maxminimizer* [106]. That is,

$$\min_{y \in A_2} u_1(x^*, y) \geq \min_{y \in A_2} u_1(x, y) \text{ for all } x \in A_1, \text{ and}$$

$$\min_{x \in A_1} u_2(x, y^*) \geq \min_{x \in A_1} u_2(x, y) \text{ for all } y \in A_2.$$

where $u_i()$ and A_i are the payoff function and the set of actions for player i.

In other words, a maxminimizer is an action that maximizes the payoff that the player can guarantee, taking into consideration that the other player wants to cause the maximum damage [106]. The maxminimizer of player 1 solves the problem $\max_x \min_y u_1(x, y)$. The maxminimizer of player 2 solves the problem $\max_y \min_x u_2(x, y)$.

In the game in Table 2, the Nash equilibrium is the action pair (x_2, y_1). Both players

cannot get higher payoff by choosing a different action alone. For instance, if player 1 chooses action x_1 instead, he will get a lower payoff (4 instead of 5). Although the example has only one solution, there may be, in general, more than one Nash equilibrium.

3.0 SYSTEM AND ATTACK MODELS

This chapter describes and models the service-level DoS attack against Internet services (§3.1) and the radio-jamming DoS attack in multi-radio wireless networks (§3.2). An abstraction of system and attack models encompassing both the Internet and the wireless networks domains is then presented in §3.3 and §3.4 and used to define dodging in the next chapter.

3.1 THE SERVICE-LEVEL DOS ATTACK IN THE INTERNET

The Internet service (e.g., Web and FTP) to be protected is an open, public service, where any host can access the service at any time from anywhere in the Internet. The service has a request-processing capacity of ρ requests per second, provided by a server cluster of n_s homogeneous servers spread over multiple mirror sites, as illustrated in Fig. 5, which depicts a Web service as an example. The edge router of each site has a packet-filtering firewall at each of its input interfaces to block requests from detected attacker addresses. A legitimate client sends requests at an average rate of r requests per second. The service provides each client with a **unique, un-spoofable** identifier. For instance, this identifier can be stored in a "cookie" in Web applications. In general, the client identifier is provided to the client in an initial handshake.

A number, n_x, of attackers issue legitimate-like service requests at a high rate, pushing the aggregate request arrival rate beyond the service capacity and overwhelming the servers. In other words, attackers are *misbehaving* clients that send requests at a high rate of $r_{attacker}$

requests per second, which is higher than the legitimate rate of r requests per second[1].

Three service-level DoS attack classes are considered. In the **massive** attack, attackers flood all the servers continuously and simultaneously. In the **follow** attack [76], attackers try to target only the *active* servers in the current configuration. Attackers can discover the active servers (e.g., by monitoring server load or snooping on legitimate client traffic) but take some time to detect server activity and to redirect attack traffic every time the set of active servers changes. This time is called the *follow delay*. Finally, in the **compliant** attack, attackers attack only active servers with zero follow delay.

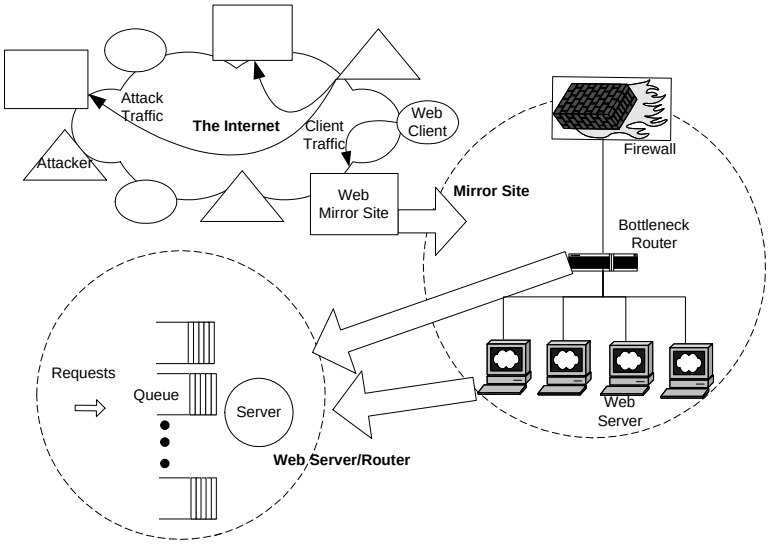


Figure 5: Internet Service Model. A Web service is presented as an example.

3.2 THE JAMMING ATTACK IN MULTI-RADIO WIRELESS NETWORKS

In multi-radio wireless networks, nodes are equipped with a number, n_f, of identical radio interfaces, each being able to transmit on any of n_h different channels. Each piece of data to

[1]It should be noted that knowing the exact values of r and $r_{attacker}$ is not necessary for the operation of the algorithms presented in this work.

be transmitted is encoded using an (n_f, m) IDA ECC (any combination of at least m pieces can be used to reconstruct the ECC-encoded data) [118]; because the number of encoded data pieces is the same as the number of radios, each piece of the encoded data is then transmitted over a different radio interface. A data piece is lost if more than $n_f - m$ radios are jammed, because there would not be enough non-faulty pieces to recover the data.

Fig. 6 depicts the architecture of an example multi-radio wireless sensor network. In this example, multi-radio sensors have 5 radios each. Radios in each sensor send encoded data pieces to the base-station over orthogonal channels. As described in §2.4.2, many wireless standards support multiple orthogonal channels. A number, n_h, of orthogonal channels is assumed. The base-station has as many wireless transceivers as the orthogonal channels. Medium access is scheduled between sensors using TDMA, for instance. For simplicity of presentation, in what follows only one sensor is considered. Radios hop among the channels according to the defense strategy in effect. Channel hopping takes a delay of τ time units, which is in the range of tens of microseconds ($80\mu s$ in [16]).

The jamming attack is launched by a number of attack radios, n_x, and each attack radio can jam one channel at a time. If a channel is jammed, no data can be communicated on that channel. Jamming radios override the MAC-protocol and send packets continuously (low-power attack methods are also feasible [156]). I assume that Spread Spectrum (SS) [114] by itself cannot prevent this jamming attack, as the jammers may use the same SS hardware as the attacked radios. As a result of the jamming attack, communication is blocked on some channels. Let P_b represent the jamming-induced **blocking probability**, or the probability that more than $n_f - m$ radios are jammed at the same time, resulting in data loss.

Two main attack strategies are considered, namely **scanning** and **sweeping** attacks, which span a wide spectrum of possible strategies. In both strategies, attackers hop between channels so that the set of jammed channels change over time. The two attack strategies differ in the activity-sensing capability of attackers as follow.

Scanning attackers sense legitimate channel activity to determine if the channel they jam is being used or not. Each attack radio keeps hopping until it finds a channel that has legitimate activity, and it stays there until it detects lack of activity. It takes an attack radio a *channel-sensing time* to determine channel activity or lack thereof. For simplicity of

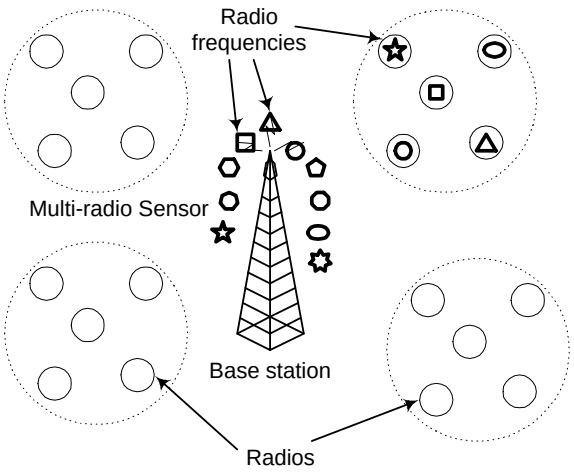


Figure 6: An example multi-radio wireless network. Radios in multi-radio sensors communicate with the base-station over orthogonal channels. The figure depicts four sensors with five radios in each sensor ($n_f = 5$) and a total of ten channels ($n_h = 10$). The base-station is equipped with ten wireless transceivers to cover the full spectrum.

analysis, the channel-sensing time is assumed to be a multiple of the channel-hopping delay, that is, $\delta_x \tau$. This delay depends on the legitimate traffic rate and on how frequently the attack radio stops jamming to sense the medium activity. After this delay, the jammer radio selects the next channel to jam uniformly at random.

It should be noted that the attack radio has two possibilities in selecting the next channel. First, the jamming radio selects its next channel from all channels. Second, the jamming radio keeps history of the channels it visits and chooses its next channel from only the channels it has not visited since its last encounter with an active channel. The second option is more effective if the communication radio would stay put at its channel until being "caught" by the jammer. This is guaranteed to happen only if the defense strategy is reactive and there is only one jamming radio. Therefore, scanning attackers in this work choose their next channels out of all the channels, because even if there is only one jamming radio, the defense strategy may be unknown. This assumption of unknown defense strategy is relaxed in the analysis in §5.1.1. In either option, scanning attackers make sure that a channel is

not jammed by more than one jamming radio.

Sweeping attackers periodically switch channels all at once irrespective of channel activity, and they jam their channels until the next period. For simplicity of analysis, the sweeping attack period is assumed to be a multiple of the channel-hopping delay, that is, $s_x\tau$. In other words, each radio in the sweeping attack jams a channel for $(s_x-1)\tau$ time units and changes the jammed channel every $s_x\tau$ time units. It selects the next channel uniformly at random from the n channels. Two variations are considered: the *fast-sweeping* attack, where attackers switch every *packet-time*[2], and *slow-sweeping*, where the period length is a multiple of the packet-time.

3.3 MAPPING CLIENTS TO RESOURCES

In this section and the next, I develop an abstraction of the system and attack models presented in the previous two sections. This abstraction will be used to define dodging in the next chapter. Generally, service provision involves assigning system resources to resource consumers, or clients, and a DoS attack aims at exhausting or destroying some or all system resources [91]. A set, C, of n_c clients are *authorized* to access the service under consideration, whereby the set C includes legitimate clients and attackers, and a set, R, of n_r homogeneous (identical) resources, such as servers, provide the service with a total capacity of ρ "requests" per second, where a request can be simply a packet or a complete service request. A legitimate client uses its assigned resources at an average rate of r "requests" per second. Each client request is sent to one of its assigned resources.

The set of resources may include physical or *virtual* resources. *Virtualization* allows a single physical resource to appear as many virtual ones. The virtual resources are isolated from each other (i.e., an attack on one virtual resource has no effect on the other virtual resources). Virtualization technology has enabled efficient multiplexing of physical resources among many virtual machines with an adequate level of resource isolation (see for instance

[2] The packet-time is the time to transmit one packet. If this is the case, in order for defenses to mitigate jamming, the packet-time has to be close to the channel-hopping delay [153].

the Xen virtual machine monitor [18]). This work uses a toned-down version of virtualization, wherein the server queue is divided into a number of isolated sub-queues (called *buckets* in §6.2) that are scheduled using a fair queuing paradigm (e.g., Deficit Round Robin (DRR) [128]).

Clients are assigned system resources according to different criteria, such as resource load and resource and client locations. To enforce resource assignment in the proposed dodging framework, each client is assigned a set of hard-to-guess *tokens* containing IDs of its assigned resources. The client then attaches a token to each request, which in turn is directed to the assigned resource based on the attached token.

To model client-resource assignment, a mapping function, μ, is assumed to be defined by the service or the underlying communications network. In what follows, time is discretized into equal-length time slots, for which the mapping function is constant. In other words, the mapping function is time-dependent and is defined as follows.

$$\mu : C \times TimeSlot \rightarrow 2^R$$

where C is the set of clients, R the set of resources, and 2^R the power set of R, that is, the set of all subsets of R. Each client c is assigned the set of resources $\mu(c,t)$ during time slot t. The set of resources to which at least one client is mapped during time slot t is called the set of *active* resources at t, or $R_A^t = \bigcup_{c \in C} \mu(c,t)$, and the set $R_I^t = R - R_A^t$ is called the set of idle resources at t. Note that we imply no order of priority among resources in the set R_A^t.

To clarify, consider the following examples of the mapping function. As a first example, regular DNS lookup in the Internet can be modeled as if every client is mapped to all servers of the looked-up service ($\forall c_1, c_2, \ \mu(c_1,t) = \mu(c_2,t) = R_A^t = R$). In this example, μ is time-independent, that is, $\forall t_1, t_2, c \in C, \ \mu(c,t_1) = \mu(c,t_2)$. A second example is content distribution networks (e.g., Akamai [1]), where clients are mapped to servers with some selection criteria such as geographic location or server load. In this example, the function μ is both client- and time-dependent, because the assigned server depends on the client's location and varies with time as the server loads change. A third example is the token-based service-access model (e.g., Kerberos [137]), in which clients acquire tokens before they

access servers. In this example, the mapping function μ is the composition of two mapping functions: from clients to tokens and from tokens to servers. In the first function, the set of possible token values serves as the set of resources, R. If the token is of length b_t bits, then the set of tokens is the set of binary numbers with length b_t and its cardinality is 2^{b_t}. The active resources are the valid tokens. The client-token mapping is one-to-one and time-dependent, because each client has its own unique token that changes with time to improve security, and the token-server mapping is many-to-many and time-independent, because any valid token is accepted by all servers.

3.4 MAPPING FROM ATTACKERS TO RESOURCES

The DoS attack is launched from a set $X \subset C$ of $n_x < n_c$ attackers. Attackers aim at exhausting system resources by creating a request load that exceeds the system capacity. Each attacker uses its assigned resources at a rate $r_{attacker} > r$, that is, each attacker sends requests at a rate higher than normal.

A mapping function, μ_X, that maps attackers into resources to attack is defined as follows:

$$\mu_X : X \times TimeSlot \rightarrow 2^R$$

where X is the set of attackers. Each attacker x sends attack requests to the resources $\mu_X(x,t)$ on time slot t. The set of resources assigned to attackers on time slot t is referred to as the set of *attacked* resources at t or R_X^t, that is, $R_X^t = \bigcup_{x \in X} \mu_X(x,t)$. Although the function μ_X may be unknown to the system, it is used for reasoning only to help define the attack types as will be presented shortly.

Whereas the client-resource mapping functions μ is defined by the defense system, the function μ_X is defined by the attackers. Consequently, these functions may differ, that is, $\mu_X \neq \mu$. An attacker x is *compliant* with the client-resource mapping function defined by the system, μ, if and only if the set of resources assigned by μ_X to the attacker is the same as

the set assigned by μ for all time slots. That is, x is compliant $\iff \forall t \;\; \mu_X(x,t) = \mu(x,t)$. The attacker is *non-compliant* otherwise.

The system and attack mapping functions are depicted in Fig. 7. Clients assigned to the resources at the intersection between active and attacked resources are assumed to receive no service from the attacked resources. The aim of the attackers is to focus their attack always on active resources. That is, to make the intersection between the active and attacked sets as large as possible and to make the intersection between the idle and attacked sets as small as possible.

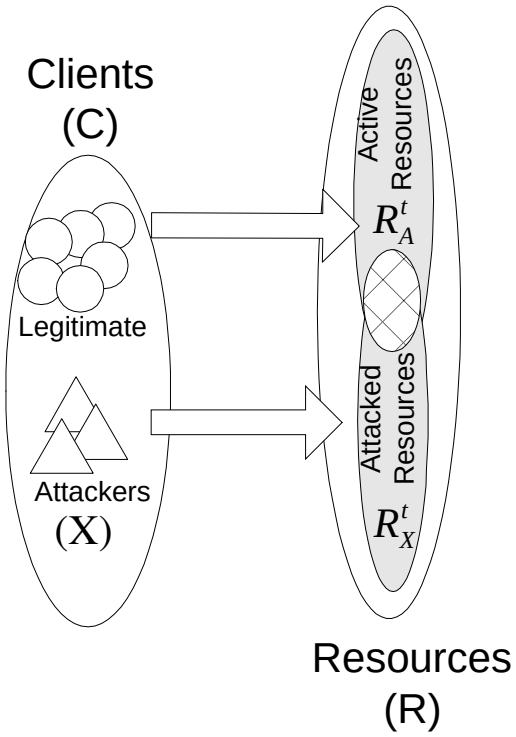


Figure 7: The system and attack mapping functions during one time slot. The sets of active and attacked resources change over time.

3.5 ATTACK TYPES

This section redefines the attack types, which are presented in §3.1 and §3.2, in terms of the client-resource (μ) and attacker-resource (μ_X) mapping functions. The follow and scanning attacks are merged into one type because scanning attack is an example of follow attack.

Follow Attack. An interesting case of non-compliant attack arises when attackers manage to track the active resources but with a delay caused by the time they need to detect resource activity. In other words, the attack mapping function *follows* the system mapping function with a delay δ_X, that is, $R_X^t = R_A^{t-\delta_X}$. An example of the follow attack is the **scanning attack**, in which attackers have the capability of sensing resource activity (e.g., by sensing radio activity on a wireless channel), and they search for active resources simply by iterating through resources and checking for signs of activity.

Massive Attack. Another case of non-compliant attack is when all the resources are attacked simultaneously and continuously. That is, $R_X^t = R$. This powerful attack renders a defense based on replication alone or on evasion alone ineffective.

Sweeping Attack. The sweeping attack is a non-compliant attack in which attackers focus their attack on a subset of the resources, and they periodically change the set of attacked resources regardless of resource activity. That is, $R_X^t \neq R_X^{t+s_x\tau}$, where $s_x\tau$ is the sweeping period and both R_X^t and $R_X^{t+s_x\tau}$ are strict subsets of R.

Compliant Attack. In the compliant attack, each attacker x attacks only the resources assigned to it by the system-assigned mapping function μ. That is, $\mu_X(x,t) = \mu(x,t)$. Consequently, the set of attacked resources is a subset of active resources for all time slots. That is, $\forall t \;\; R_X^t \subseteq R_A^t$.

As summarized in Fig. 8, the follow attack is addressed by the Honeybees (§5) and the roaming honeypots (§6.1) algorithms, whereby the attack impact is mitigated. Massive service-level attacks are mitigated and their attackers identified by the roaming honeypots algorithm. This dissertation does not address massive nor compliant wireless jamming attacks, in which all possible (active) channels are jammed, because such attacks are impossible to defend against. Instead, the sweeping attack, a toned down version of the massive attack, is studied. The roaming honeypots algorithm mitigates compliant attacks, and finally, the

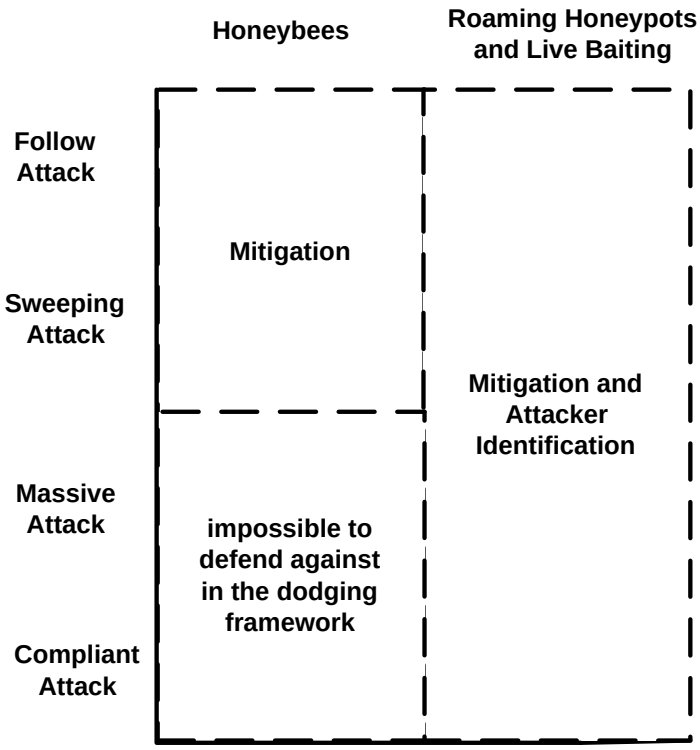


Figure 8: Attack types and defense algorithms.

live baiting algorithm (§6.2) is designed only to identify compliant attackers.

4.0 THE DODGING FRAMEWORK

The client-resource mapping function, introduced in the previous chapter, abstracts the relationship between resources (e.g., servers and radio channels) and clients (e.g., service users and radio transceivers) of a service. This chapter defines the dodging framework, which addresses the problem of how to define the client-resource mapping function to identify DoS attackers and to mitigate their attack.

Two components contribute to build the dodging framework. The first component is the careful combination of replication and evasion to achieve elusiveness of the protected service and allow for attack mitigation. The second component, namely the Primary-Effect-based Detection (PED), is a novel approach for identifying DoS attackers using simple detection signals.

The two components of the dodging framework, namely attack mitigation and attacker identification, are defined in terms of properties of the mapping function. These properties are first introduced in §4.1 followed by the definition of the attack mitigation (§4.2) and attacker identification (§4.3) components.

4.1 DODGING PROPERTIES

First, I present a set of properties of the mapping function to be used in the definition of the dodging components.

- **Non-blocking**. The mapping function μ is *non-blocking* when client requests are *always* assigned to resources. That is, $\forall t, c \in C \;\; \mu(c,t) \neq \phi$.

- **Uniform**. The mapping function μ is *uniform* when it distributes the request load uniformly over the active resources. A resource η gets a fraction of the requests sent by each client c assigned to it, which depends on how many other resources are assigned to c. The mapping function is uniform if for any two resources η_1 and η_2 on any time slot t, the total client shares assigned to each of the two resources is roughly the same. Let $C_1(t) = \{c_1 | \mu(c_1, t) = \eta_1\}$ denote the set of all clients assigned to resource η_1 at time t. $C_2(t)$ is similarly defined. Also, assume that clients issue requests at the same rate and distribute their requests uniformly over their assigned resources. Then, the mapping function is uniform if and only if $|\sum_{c_1 \in C_1(t)} \frac{1}{|\mu(c_1,t)|} - \sum_{c_2 \in C_2(t)} \frac{1}{|\mu(c_2,t)|}| < \epsilon$.
- **Dynamic**. The mapping function μ is *dynamic* when the set of active resources changes over time. The change frequency is a design parameter, and its effect is studied in §6.1.
- **Loose**. The mapping function is *loose* when the set of idle resources is never empty on any time slot, that is, $\forall t \;\; R_I^t \neq \phi$, and the set of active resources contain more than one element, that is, $\forall t \;\; |R_A^t| > 1$.
- **Unpredictable**. The mapping function is *unpredictable* when it is difficult for a non-compliant attacker x to keep attacking *only* active resources for a long time period, T, with increasing level of difficulty as the length of T increases. That is, for any non-compliant attacker x and any period T of length > 0, the probability that at some point of time in T the attacker will hit an idle resource, $Pr\{\exists t \in T \text{ s.t. } \mu_X(x,t) \bigcap R_I^t \neq \phi\}$, is > 0. Moreover, this probability increases as the length of T increases.
- **d-unique**. The client-resource mapping function μ is *d-unique* if the membership of any client in any client group of cardinality at most d can be discovered based on the sets of resources assigned to the client and to the group on any time slot. More specifically,

$$\forall c, C_1 \;\; c \in C \text{ and } |C_1| \leq d \text{ and } c \notin C_1 \Rightarrow \forall t \;\; \mu(c,t) \neq \bigcup_{c_1 \in C_1} \mu(c_1, t).$$

  The parameter d should be an estimate of the number of attackers. This property is borrowed from d-disjunct codes, which are used in building non-adaptive group-testing matrices [43]. More on this property will be discussed in the live baiting algorithm in §6.2.

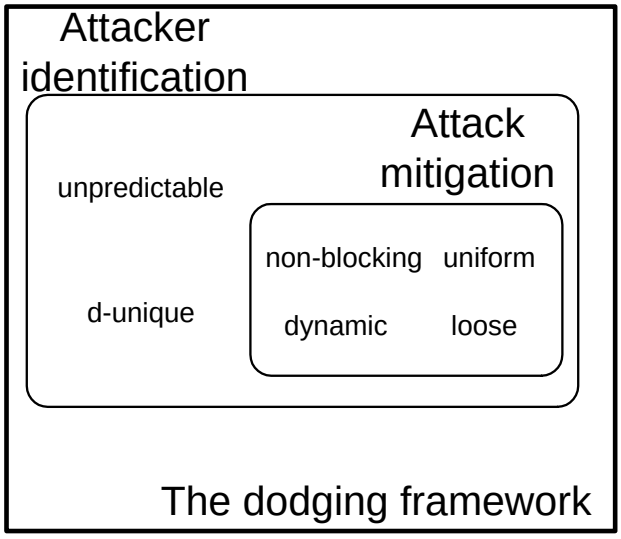


Figure 9: The components of the dodging framework and properties of the client-resource mapping function.

As previously mentioned, the above properties are used to define the dodging framework and its components. Fig. 9 depicts the relationship between the properties and the components of the dodging framework. It shows that the non-blocking, uniform, dynamic, and loose properties form the core of the dodging framework. These properties ensure that dodging has a low impact on the performance of the service being protected and create the controlled dynamic nature that allows dodging to efficiently achieve attack mitigation and attacker identification, as will be described in the next two sections.

4.2 ATTACK MITIGATION BY COMBINING REPLICATION AND EVASION

The first dodging component achieves attack mitigation by a careful combination of replication and evasion. This component enables the protected service to sustain acceptable performance under attack. Attack mitigation is particularly useful when it is hard to filter

out attacks, which is the case in the radio jamming attack in wireless networks.

The dodging framework adopts the approach of combining replication and evasion to achieve attack mitigation. The set of active resources contains multiple, replicated resources, which evade attacks by switching to the never-empty set of idle resources. As depicted in Fig. 9, the mitigation component utilizes the looseness and dynamic properties of the client-resource mapping function. These properties allow requests from legitimate clients to get serviced even under attack, because when resources switch from idle to active, they start "fresh", that is, with empty service queues, and hence attackers take some time to discover and exhaust the newly-activated resources. During these transient intervals, which I call **windows of opportunity**, the legitimate requests have a chance to get serviced. In particular, the looseness and dynamic properties guarantee that these windows of opportunity are created at every reconfiguration of the set of active (idle) resources.

To further illustrate the usefulness of the dynamic and loose properties, consider two mitigation schemes, namely *plain replication* [30] and *plain evasion* [54, 136, 155], that lack these two properties, respectively. The following qualitatively shows that dodging outperforms these two schemes. Moreover, a quantitative comparison is presented in §5.1.2 (against plain evasion) and §6.1.4 (against plain replication).

Dodging vs. plain replication. In the plain replication scheme, all resources are always active, that is, $R_A^t = R$. At its extreme, a unique resource (physical or virtual) is assigned to each client, and each resource is isolated from other resources, in which case attacks can be mitigated because the attack impact is limited to only the resources assigned to the attackers. Plain replication, however, has two drawbacks: First, it ungracefully degrades to massive attacks that overwhelm all the resources. Second, plain replication is too expensive in terms of energy (if active resources consume more energy than idle ones, which is typically the case in wireless environments) or in terms of state and control overhead to maintain the active resources (even virtualization comes at the cost of state and control overhead to maintain the virtual resources). Dodging allows an efficient usage of the replicated resources by enabling resources to evade attacks and to create the windows of opportunity mentioned above, resulting in more graceful degradation to attack intensity (see, for instance, Fig. 25 and Fig. 37).

Dodging vs. plain evasion. In the plain evasion scheme, only one resource is always active (i.e., $|R_A^t| = 1$), and the active resource changes within the resource pool to escape from attacks (e.g., by relocating the service away from attacked servers). Clients are then directed to the new active resource, for instance by purging old DNS records and pushing records with the new server addresses instead. Application-layer redirection (e.g., HTTP redirection) can also be used as a vehicle to achieve seamless migration to the new server. Evasion can be triggered reactively (in response to attack detection) [136, 155] or proactively [54]. Plain evasion, however, has limited effectiveness when attackers can discover the newly activated resources. Because in dodging there are more than one active resource, it takes attackers a longer time to discover all their locations. In the honeybees algorithm (Chapter 5), both reactive and proactive evasion strategies are studied and their relative performance is compared under different system and attack scenarios.

The following definition formulates the mitigation component in terms of properties of the client-resource mapping function.

Mitigation Component. *The mitigation component defines the client-resource mapping function to be non-blocking, uniform, dynamic, and loose.*

4.3 ATTACKER IDENTIFICATION BY PRIMARY-EFFECT-BASED DETECTION (PED)

The second component of dodging identifies attackers within a short time interval, with high accuracy, and with small overhead. The ability of dodging to achieve low overhead is particularly important in scenarios that require scalability of the detection system, such as when the system is to be deployed at an ISP to protect many of its customers at once. Although each customer can keep the state of and monitor its own clients, the total number

of clients over all customers may be huge and beyond the capacity of the ISP to monitor. Therefore, the per-client monitoring approach (e.g., [120]), which requires state that grows linearly with the total number of clients, can be prohibitively expensive, although it may provide the most accurate DoS detection with the smallest detection time. As will be shown in the live baiting algorithm in §6.2, dodging enables a detection algorithm that maintains $O(n_x)$ state, that is, state in the order of the number of attackers not the number of clients. This is a big saving in state because service clients may be in the order of millions, whereas botnet sizes have been recently reported to be in the order of thousands of zombies [119].

Attacker identification is addressed for two attack cases separately: identification of compliant attackers and identification of non-compliant attackers. As depicted in Fig. 9, the attacker identification component utilizes the unpredictability and d-uniqueness properties of the client-resource mapping function. Unpredictability guarantees that any **non-compliant** attacker attacks an idle resource with a probability that increases as the length of the attack period increases. Once an attacker attacks an idle resource, it can be identified, because legitimate clients do not access idle resources by design of the mapping function. In §6.1, this property is used to identify non-compliant service-level attackers.

The d-uniqueness property allows dodging to identify **compliant** attackers (as long as their number is at most d) merely using the set of attacked resources[1]. This is because the d-uniqueness property guarantees that any client, c, outside the attacker group, C_1, is assigned to at least one resource that is not attacked (see the definition of d-uniqueness in §4.1), and thus, all legitimate clients can be excluded using a detection algorithm similar to the one used in group testing (see §2.3 for more details). In §6.2, the d-uniqueness property is approximated using a randomized technique, and thus, false positives and false negatives may occur.

The above discussion shows that dodging can identify attackers using basic signals, the set of attacked (idle and active) resources, without requiring any complex models of normal or attack traffic. I call this novel approach of attacker identification the Primary-Effect-based Detection (PED), which is a departure from traditional detection approaches, namely

[1]Note that the attacked resources are active resources in this case, because the attackers are compliant, and thus, they never access idle resources.

anomaly-based (e.g., PacketScore [7, 14] and DDoS Shield [120]), misuse-based (e.g., the Snort Intrusion Detection System [4]), and specification-based (e.g., RED-PD [89]). As will be shown in §6.2, PED is also scalable, and its scalability stems from its reliance on the most basic signals to identify attackers.

To appreciate the PED approach, the following discussion contrasts its underlying philosophy to the common philosophy behind traditional detection approaches. The latter is two-fold. First, in many security problems, detecting the attacker actions is easier than detecting the security violation or the damage caused by the attack. This relative ease is true in confidentiality and integrity, for instance, whereby attackers, if left undetected, can cover their tracks and make it extremely hard to detect that a piece of data has been read or altered by them [21]. Second, waiting until security is violated is undesirable in cases that involve damage that cannot be easily recovered (e.g., stolen data), and in such cases it is crucial to detect and stop the attack before it materializes as a violation.

Denial of service is different. The attack damage can always be detected [47], and in many cases detection is easy. For instance, the flooding DoS attack results in a high, sustained packet drop rate at bottleneck routers [90], which can be easily detected. Moreover, the damage is in most cases readily recoverable once the attack is stopped.

In PED, DoS attackers are detected based merely on the unavailability of the attacked resources, without the need to develop models of legitimate behavior, keep track of attack signatures, or tightly specify legitimate behavior[2]. Consequently, the dodging-enabled detection overcomes the limitations of current detection approaches. The attacker identification problem in the PED approach is defined as follows.

Definition 4.1 (The PED problem). *The PED problem is that of identifying the attackers given (1) a function $\Delta(\sigma, \tau)$, which indicates whether resource σ is under attack (i.e., its response time is longer than predicted or contracted [47]) at any time instant τ; (2) the list of clients assigned to resource σ at τ, that is, $\{c|\mu(c, \tau) = \sigma\}$; and (3) the list of clients accessing σ at τ.*

[2]Although this work assumes that service-level attackers have a higher than normal request rate (§3.1), this assumption is only for concreteness; any other attack model can be identified by the schemes presented in this work as long as attackers attack all the resources assigned to them.

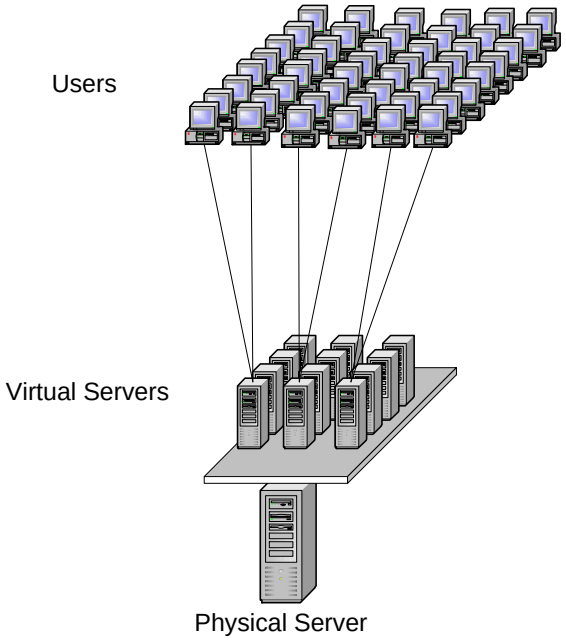


Figure 10: Two-to-one mapping between clients and virtual servers.

The function Δ in the above definition can be thought of as an oracle that answers *attacked or not attacked* questions for any resource at any time instant. Chapter 6 presents approximations of Δ that provide partial (yet effective) information.

One possible solution of the PED problem is a one-to-one mapping between clients and virtual resources, whereby each client is restricted to access its assigned resource only, and the virtual resources are isolated from each other (i.e., an attack on one resource has no effect on the other resources). In this case, it is straightforward to identify the attackers using the set of attacked resources; the attackers are the clients whose assigned resources are attacked. Another solution is to assign a virtual resource for each client *group*. An example of this *many-to-one* mapping between clients and virtual resources is depicted in Fig. 10. Consequently, when some resources become under attack, the attackers must be among the clients assigned to the attacked virtual resources.

Along this line, the live baiting algorithm (§6.2) uses results from the theory of group testing [42, 43, 57, 67, 102, 138] to design the client-resource mapping to achieve a good trade-

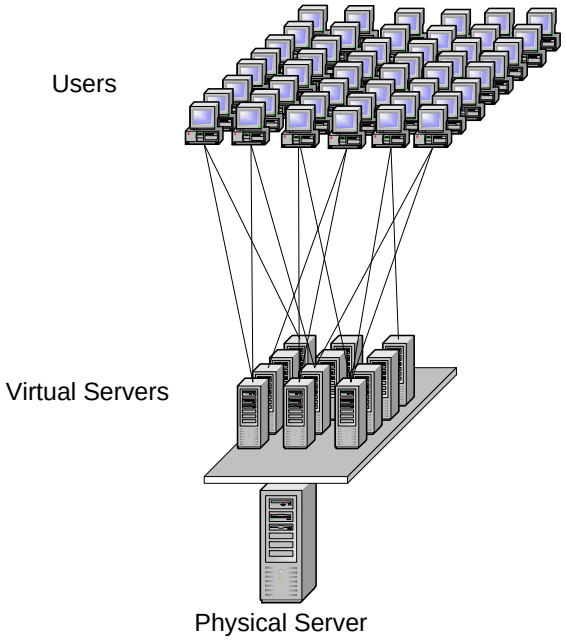


Figure 11: Many-to-many mapping between clients and virtual servers.

off between the number of virtual resources (the system state), the false positive probability, and the false negative probability [70]. Live baiting uses a many-to-many mapping between clients and virtual resources as depicted in Fig. 11 [70], whereby each client is assigned a number of resources, and each resource is assigned to more than one client.

The following two definitions formalize the attacker identification component against both compliant and non-compliant attackers in terms of properties of the client-resource mapping function.

Non-compliant Identification Component. *The non-compliant attacker identification component defines the client-resource mapping function to be non-blocking, uniform, dynamic, loose, and unpredictable.*

Compliant Identification Component. *The compliant attacker identification component defines the client-resource mapping function to be non-blocking, uniform, dynamic, loose, unpredictable, and d-unique.*

As I will show throughout the rest of the dissertation, the roaming honeypots algorithm (§6.1) and the honeybees algorithm (Chapter 5) provide attack mitigation against both compliant and non-compliant attackers. Roaming honeypots also identifies non-compliant attackers, and live baiting (§6.2) identifies compliant attackers.

5.0 MITIGATION OF JAMMING ATTACKS IN MULTI-RADIO WIRELESS NETWORKS

In the previous chapter, the two components of dodging were introduced: to mitigate attacks and to identify attackers. In this chapter, the mitigation component is presented and evaluated against the wireless jamming attack. As illustrated in Fig. 12, the honeybees algorithm presented in this chapter is an energy-efficient, bandwidth-efficient defense that achieves attack mitigation against non-compliant radio jammers [73, 74].

Advances in radio technology have enabled the paradigm of *multi-radio* wireless networks, which have been proposed to increase overall network capacity by exploiting channel diversity [15]. For instance, multi-radio 802.11 nodes are equipped with multiple radio interfaces operating at orthogonal channels. Honeybees protects single-hop communication between multi-radio nodes by combining two jamming countermeasures that have been proposed elsewhere, namely *software-based channel hopping* [54,100,153,155] and *error-correcting codes*

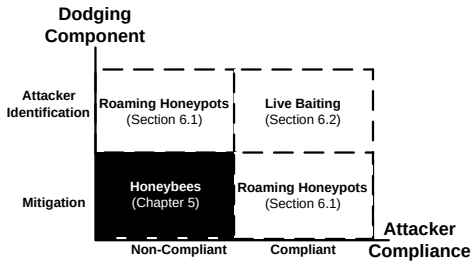


Figure 12: The part of the DoS problem space addressed by the honeybees algorithm. Honeybees provides mitigation against non-compliant attackers.

(e.g., [118]). Channel hopping, whereby the used radio channels are switched at the software-level, is an effective mechanism to mitigate jamming in wireless sensor networks and 802.11 networks. It has been proposed and evaluated in the context of single-radio networks, where each wireless node is equipped with a single radio interface [54, 100, 153, 155]. In multi-radio networks, the multiple radio interfaces can be used to increase communication reliability against benign errors and malicious radio interference by sending redundant data: either same data on different channels or encoding data and sending it in parallel through all channels. Clearly, reliability comes at the expense of reduced goodput. Understanding the trade-off between goodput and reliability is essential to make optimal utilization of the multiple radios. This chapter addresses the problem of maximizing network goodput under jamming attacks using a combination of channel hopping and error-correction coding.

In §**5.1**, the solution space of channel-hopping strategies is first analyzed by presenting two channel-hopping strategies, namely reactive and proactive, and comparing the two against different attack strategies. The question "*which of the two channel-hopping strategies provides better jamming resiliency than the other?*" is addressed in the context of the typical single-radio wireless devices and in the context of multi-radio devices equipped with multiple radio interfaces. The reactive and proactive defense strategies are studied against two attack strategies: *scanning* and *sweeping* attacks defined in §3.2. Fig. 13 illustrates the analysis components. In the single-radio context, theoretical models are developed to analyze the blocking probability under each combination of defense and attack strategies. In the multi-radio setting, due to intractability of the theoretical analysis, simulation is used instead to address the interaction between the defense and attack strategies. The jamming problem is then formulated as a max-min game between defense and attack, and through simulation the game outcome is shown to depend on the payoff function. Both analytical and simulation results show that the reactive defense provides better jamming tolerance than proactive when considering communication availability. Simulation results also show that the scanning attack causes more damage than the sweeping attack. Therefore, the reactive defense against scanning attack combination is further studied in §5.2.

In §**5.2**, the integration of ECC and reactive channel-hopping is analyzed, and its effectiveness in defending against scanning attack is evaluated. Specifically, the reactive defense

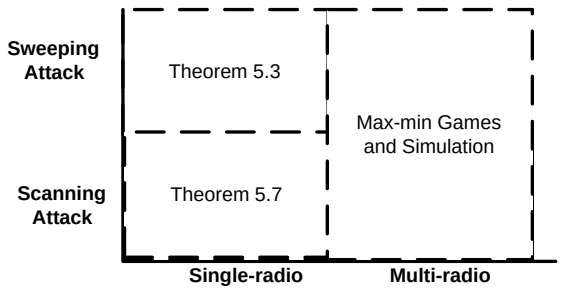


Figure 13: Overview of the analysis of channel-hopping in single- and multi-radio networks.

and the scanning attack are modeled by a Markov chain that incorporates the ECC parameters. The model is then validated using simulation experiments. Finally, an adaptive algorithm, namely *honeybees*, is devised based on the model and is shown to improve the resiliency of multi-radio networks against jamming.

In honeybees, channel-hopping defines the mapping function μ between the radio transceivers and the radio channels, whereby dodging is achieved by switching the radio transceivers among radio channels. The mapping function in honeybees satisfies the properties of the mitigation component (§4.2): (**non-blocking**) all radio transceivers are always assigned active radio channels, (**uniform**) radios are distributed evenly over the active channels, (**dynamic**) the set of active channels changes over time, (**loose**) there are always some unused channels, and (**unpredictable**) the set of active channels is hard to predict by attackers.

The honeybees scheme achieves attack mitigation because with each reconfiguration, jammers incur a delay until they find the newly activated radio channels. During these opportunity time-windows, communication is not blocked. These opportunity windows happen when radio channels switch from idle into active and start "afresh".

5.1 CHANNEL-HOPPING STRATEGIES

Current research in jamming defense in single-radio networks has a bias towards proactive channel-hopping as opposed to reactive channel-hopping. This is due in part to the implementation simplicity of the proactive channel-hopping and the overhead and difficulty of jamming detection [54, 100, 153, 156]. However, these factors are closely related to the wireless technology and may be affected by new generations of wireless networks. One of the emerging wireless network paradigms is the multi-radio networks. An important question arises in the context of jamming mitigation in both single- and multi-radio networks: *which channel-hopping strategy, reactive or proactive, achieves the best jamming resiliency?*

This section addresses this question by studying how channel hopping is affected by (a) jamming strategies and (b) system parameters. More specifically, two jamming strategies are studied: *scanning* and *sweeping*, whereby scanning jammers have and use the capability of sensing channel activity, whereas sweeping attackers lack (or do not use) this capability and resort to jamming each channel for a fixed amount of time before moving on to other channels (see §3.2 for more details). The effect of the system parameters, such as the number of radios per node, the number of channels, and the delay of jamming detection, is also considered in all four defense-attack combinations.

The reactive-or-proactive question is decomposed into two subquestions: *(1) under a given system and attack scenario which of the two defense strategies provides better jamming resiliency?* and, because the attack strategy may be unknown, *(2) which attack strategy is more reasonable for attackers to choose and which defense strategy achieves better protection against the best attack strategy?*

To address the first subquestion, models of defense and jamming strategies are developed and the interaction between the four combinations of defense and attack strategies is analyzed: theoretically for the single radio case and through simulation for the multiple radio case due to intractability of theoretical analysis in the multi-radio setting. Communication blocking probability and energy efficiency are the performance metrics in this study.

To address the second subquestion, jamming is modeled as a max-min game between defense and attack. Specifically, two games are defined, and they differ in the optimization

metric, or the payoff function. The *availability* game considers availability of the communication channel, whereby the defense aims at maximizing availability, and the jamming attack aims at minimizing it. In the *efficiency* game, energy consumption is considered by both defense and attack. The availability game is relevant in scenarios where jammers want to disable network functioning for a focused, short period of time, such as in emergency-alarm networks. The efficiency game, on the other hand, can be used to model scenarios where jamming for extended periods aims at depleting the network's energy resources, and attackers are energy-limited as well. The outcomes of the two games are studied through simulation.

The main results can be summarized as follows:

- In single-radio networks, reactive channel-hopping is provably better than proactive against the fast-sweeping attack. Against other attacks, formulas are derived to decide which defense strategy is better under different system and attack scenarios.
- In multi-radio networks, reactive channel-hopping achieves better jamming tolerance than proactive hopping under the availability game. Under the efficiency game, however, both strategies perform almost the same. Therefore, it can be concluded that reactive hopping is a more plausible strategy than proactive in multi-radio networks.

As mentioned previously, channel-hopping defines a dynamic mapping between radios and channels. This mapping is changed either periodically (proactive hopping) or triggered by attacks (reactive hopping). In both the reactive and proactive strategies, a radio cannot transmit or receive while switching channels. However, the two strategies differ based on the jamming detection capabilities in the wireless nodes as follows.

Reactive channel-hopping. In the reactive channel-hopping strategy, each radio stays at its current radio channel as long as no jamming is detected. Once it detects jamming, it switches to a different channel. Jamming may keep the wireless medium busy, resulting in a long waiting-time to access the channel, or may corrupt packets by causing high interference at the receiver, resulting in excessive retransmissions. Only the sender has to detect jamming and decide to switch channels. The receiver (or the base-station in §3.2) does not need to be informed explicitly with the channel-switching decisions as it is already listening on all

the channels. A simple jamming detection algorithm is used: if the waiting-time for a free channel or for a successful transmission exceeds a threshold, $\delta_d\tau$, jamming is assumed and the radio hops to a different channel selected uniformly at random using a securely seeded random-number generator.

It should be noted that the jamming-detection threshold is usually much longer than the channel-hopping delay, that is, the time taken by the radio to switch channels. Channel-switching times of tens of micro-seconds have been reported (e.g., [16, 153]), whereas the jamming detection can take up to seconds [156]. Based on this large gap, the hopping overhead is assumed to be zero in the models presented in §5.2.2, and the jamming-detection and attack channel-sensing delays are set to one time-slot each. In the simulation study in §5.2.3, more realistic delay values are used.

Proactive channel-hopping. In the proactive defense strategy, radios switch channels periodically and simultaneously according to a pseudo-random schedule. For simplicity of analysis, the channel-hopping period is a multiple of the channel-hopping delay, τ. Every $s_d\tau$ time units, all radios switch channels, and they reside at their channels for $(s_d-1)\tau$ time units. The proactive strategy is oblivious to jamming status, so unjammed radios may be triggered to switch and jammed ones may be kept. Clock-synchronization is a requirement of the proactive strategy. However, loose clock synchronization is not difficult to achieve among radios on the same device (synchronization between sender and receiver is not needed because the base-station has enough transceivers to cover all channels).

5.1.1 Channel-hopping in Single-radio Networks

In single-radio networks, channel hopping has been studied with a focus on proactive (periodic) channel-hopping [54, 100, 153, 155]. In this section, jamming in single-radio networks is analyzed by comparing proactive and reactive channel-hopping under the two attack strategies, namely scanning and sweeping. The goal is to answer the following question: under a given system and attack scenario which defense strategy provides better jamming resiliency? In what follows, it is assumed for ease of presentation that time variables are measured in units of τ, the channel-hopping delay.

5.1.1.1 Sweeping Attack

First, the blocking probability (defined in §3.2) of both reactive and proactive defense is studied against the sweeping attack.

Lemma 5.1. *In single-radio networks, the blocking probability of reactive channel-hopping against sweeping attack is*

$$P_b^{reactive-sweeping} = \begin{cases} \frac{1+\delta_d}{n_h s_x} & when\ (s_x - 1) \geq \delta_d; \\ \frac{s_x - 1}{n_h s_x} & otherwise. \end{cases}$$

where δ_d is the jamming detection delay, n_h the number of channels, and s_x the period of the sweeping jammer.

Proof. Every sweeping period, the sweeping attacker selects a channel uniformly at random, and thus, it hits the channel used by the communication radio with a probability $\frac{1}{n_h}$, where n_h is the total number of channels. Consider two cases.

(1) the attack channel-residence time is longer than the jamming detection delay, or $(s_x - 1) \geq \delta_d$: The radio detects jamming and hops to another channel. Therefore, the radio stays blocked for $\delta_d + 1$ if hit by the jammer, where δ_d is the time it takes the radio to detect jamming and 1 is the channel-hopping delay during which the radio cannot communicate as well. Hence, the expected blockage time in each attack period is $\frac{1+\delta_d}{n_h}$, resulting in a blocking probability of $\frac{1+\delta_d}{n_h s_x}$.

(2) $(s_x - 1) < \delta_d$: The radio cannot detect that it is being jammed and will stay blocked for the whole attack residence time, $s_x - 1$, if hit by the jammer, resulting in a blocking probability of $\frac{s_x - 1}{n_h s_x}$. □

The blocking probability of **proactive defense against sweeping attack** is then derived. To give an intuition, the example illustrated in Fig. 14 is used. In the figure, the defense period is 3 time units, the attack period is 2 time units, and the least common multiple (hyperperiod) of the two periods $LCM = 6$ time units. Note that the radio is readily blocked for $2 = \frac{LCM}{3}$ time units, during which it is hopping channels. Also, the radio is free for $2 = \frac{6}{2} - 1$ time units due to the attacker hopping channels and consequently not jamming (the attacker hops in 3 time units but in one of them the radio is blocked because it is hopping channels itself). In the remaining 2 time units (out of the 6 time units of the

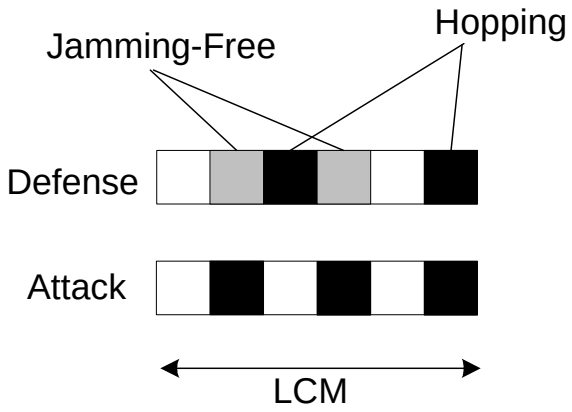


Figure 14: The hyperperiod of length LCM of the periods of proactive defense and sweeping attack.

hyperperiod length), both the radio and the jammer are not hopping and are residing on some channel. Again, because both defense and attack radios select channels uniformly at random, the probability that both the radio and the jammer reside on the same channel is $\frac{1}{n_h}$, and thus, the radio is blocked for $\frac{2}{n_h}$ time units on average. So, there are $2+\frac{2}{n_h}$ blocked time units on average out of the $LCM = 6$ time units, resulting in a blocking probability of $\frac{2+\frac{2}{n_h}}{6}$.

Lemma 5.2. *In single-radio networks, the blocking probability of proactive channel-hopping against sweeping attack is*

$$\frac{1}{s_d} + \frac{1 - \frac{1}{s_d} - \frac{1}{s_x}}{n_h} \leq P_b^{proactive-sweeping} \leq$$

$$\frac{1}{s_d} + \frac{1 - \frac{1}{s_d} - \frac{1}{s_x} + \frac{1}{LCM}}{n_h}$$

where s_d is the channel-hopping period of the proactive defense, s_x the period of the sweeping jammer, and LCM the least common multiple of s_d and s_x, or the hyperperiod.

Proof. Let LCM be the hyperperiod length, that is, the least common multiple of defense and attack periods. Let $LCM = ps_d = qs_x$, for some integers $p > 0$ and $q > 0$. During any

hyperperiod, there are p time units when the communication radio is hopping and q time units when the attacker is hopping. Depending on the values of the defense and attack periods and their phase shift, the channel-hopping times of the attack and the defense overlap (i.e., there exist time slots when both attack and defense radios hop channels simultaneously), or the defense and attack hopping instances never intersect.

Consider first the case that there is intersection between the attack and defense hopping instances and without loss of generality consider the hyperperiod that ends at one of the intersections. By definition of the hyperperiod, exactly one of the q attack hopping instances (specifically, the last hopping instance in the hyperperiod) coincides with a hopping instance of the communication radio. Therefore, out of the LCM time units of a hyperperiod, the radio is hopping in p time units and the jammer is hopping for q time units, but both are hopping together for one time unit. Hence, both the communication radio and the attacker are not hopping and are residing on some channel for $(LCM - p - q + 1)$ time units.

The radio is blocked in the p hoppings. It is also blocked for $\frac{1}{n_h} \cdot (LCM - p - q + 1)$ time units on average, where again $\frac{1}{n_h}$ is the probability that the jammer hits the channel used by the radio. Hence, the expected number of blocked time units during each hyperperiod is $(p + \frac{LCM-p-q+1}{n_h})$, resulting in a blocking probability of $\frac{p+\frac{LCM-p-q+1}{n_h}}{LCM}$.

Now, consider the case that the hopping instances of the defense and attack never intersect. There are still p blocked time units in which the radio is hopping channels. However, the number of time slots during which the radio and the jammer are not hopping is $(LCM-p-q)$ time units, one less than in the previous case, because there is no intersection between attack and defense hopping instances. This results in an expected number of blocked time units of $(p + \frac{LCM-p-q}{n_h})$ during each of the hyperperiods, and the blocking probability is $\frac{p+\frac{LCM-p-q}{n_h}}{LCM}$. □

The proof of the next theorem follows directly from comparing the blocking probabilities in Lemma 5.1 and Lemma 5.2.

Theorem 5.3. *Against sweeping attack in single-radio networks, (a) when $\delta_d > s_x - 1$, reactive channel-hopping achieves less or the same blocking probability as proactive channel-hopping and (b) when $\delta_d \leq s_x - 1$, reactive channel-hopping achieves less blocking probability*

if $\delta_d < s_x - 1 + ((n_h - 1)\frac{s_x}{s_d} - 1)$ *and proactive channel-hopping achieves less blocking probability if* $\delta_d > s_x - 1 - (1 - \frac{s_x}{LCM} - (n_h - 1)\frac{s_x}{s_d})$.

Theorem 5.3 supports the intuition that the best defense against a jammer that sweeps the channels very fast ($(s_x - 1) < \delta_d$) is to stay put. Indeed, the stay-put radio can be viewed as a reactive radio with jamming detection delay (δ_d) longer than attack residence-time ($s_x - 1$) (the reactive radio would not detect jamming and would not switch channels). Part (a) of the theorem states that the reactive radio achieves less or the same performance as the proactive radio[1].

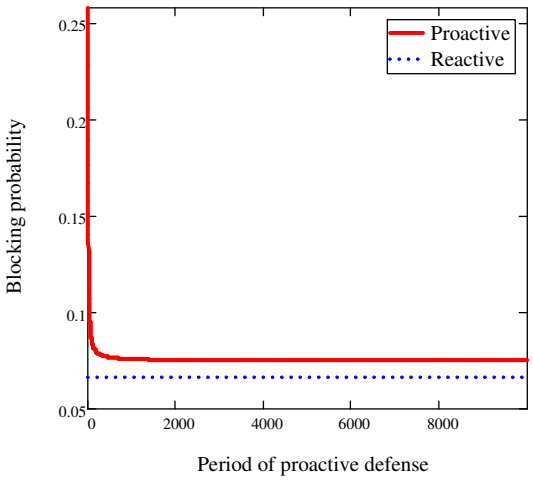
Against a slow sweeping attacker, reactive channel-hopping is better than proactive as long as the radio has some jamming-free time after it detects jamming and moves to a different channel. The next corollary formalizes this condition.

Corollary 5.4. *Against sweeping attack in single-radio networks, reactive channel-hopping achieves less blocking probability than proactive if* $\delta_d \leq s_x - 1$ *and* $(s_x - 1) - \delta_d > 1$.

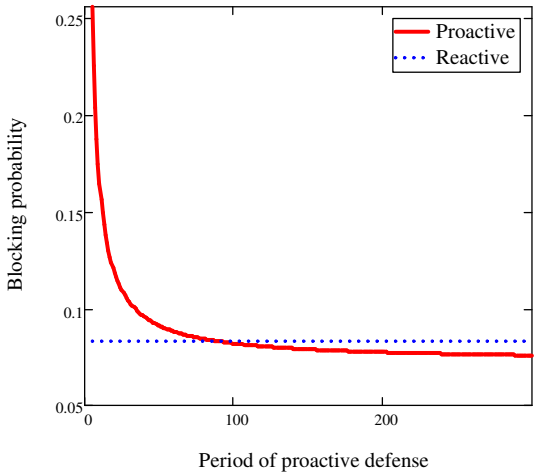
The corollary follows from part (b) in Theorem 5.3, where reactive achieves less blocking probability when $\delta_d < s_x - 1 + ((n - 1)\frac{s_x}{s_d} - 1)$, or $(s_x - 1) - \delta_d > (1 - (n - 1)\frac{s_x}{s_d})$. Noting that $(n - 1)\frac{s_x}{s_d} \geq 0$, the condition is true if $(s_x - 1) - \delta_d > 1$. The following example illustrates the above corollary. If $s_x = 10$ time units, $\delta_d = 7$ time units (as in 802.11 retransmission threshold), and $n_h = 12$ channels (as in 802.11a), the blocking probability of the reactive defense will be $\frac{1+7}{12 \cdot 10}$ (the first case in Lemma 5.1), and there is no value of s_d that makes proactive defense achieves less blocking probability (Fig. 15(a)); even when $s_d = \infty$, the minimum blocking probability of the proactive defense is $\frac{9}{12 \cdot 10}$ (the lower bound in Lemma 5.2). However, if $s_x = 8$, the blocking probability of the reactive defense becomes $\frac{1+7}{12 \cdot 8} = \frac{1}{12}$, and the proactive defense achieves less blocking probability for $s_d > 88$ (Fig. 15(b)).

5.1.1.2 Scanning Attack The blocking probability under the scanning attack is now considered. In what follows, for simplicity, time is assumed to be slotted, and the slot time is assumed to be equal to the time it takes the attacker to detect lack of channel activity and hop. That is, the slot time is $\delta_x + 1$. Let α denote the ratio between the channel-hopping

[1]Same performance is achieved only when the proactive radio has a period $s_d = \infty$, that is, stay-put.



(a) period of sweeping attack $s_x = 10$ time units



(b) period of sweeping attack $s_x = 8$ time units

Figure 15: The blocking probability of the reactive and proactive channel-hopping strategies against the sweeping attack in single-radio networks. 12 channels, jamming detection delay $\delta_d = 7$ time units.

delay and the slot time, that is, $\alpha = \frac{1}{\delta_x+1}$. For simplicity, it is also assumed that the time it takes the radio to detect jamming and hop channels, that is, $\delta_d + 1$, is a multiple of the time slot length [100].

Expressions for the blocking probability of the **reactive defense against the scanning attack** are then derived. To this end, the Markov model depicted in Fig. 16 is used, whereby the state space is divided into two classes: in the B states the radio is blocked (or jammed) and in the F states the radio is free from jamming. State B_i represents that the radio has been jammed for i time slots, and state F_i represents that the jammer is still scanning for the radio and has i channels yet to visit.

Once the radio is in the first B state (B_1), it stays blocked for $(\delta_d + 1)$ time units, which is the time to detect jamming and hop to another channel. This time interval corresponds to $\frac{\delta_d+1}{\delta_x+1}$ states, because of our assumption that the time is slotted into $(\delta_x + 1)$-sized time slots.

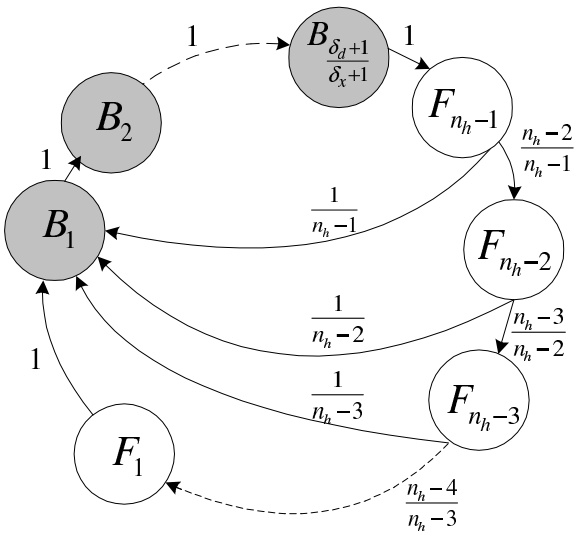


Figure 16: Markovian Model for reactive defense against scanning attack.

Also, the transition probability from each state B_i to B_{i+1} is 1 to represent the deterministic jamming-detection and hopping delays.

After the radio hops to a different channel, it stays free from jamming for at least one time slot (state F_{n_h-1}) while the jammer senses lack of channel activity and hops channels. The jammer selects its next channel uniformly at random from $n_h - 1$ channels. The probability that the jammer hits the channel used by the radio and drives the radio back into the first blocked state is $\frac{1}{n_h-1}$, which is the transition probability from state F_{n_h-1} to state B_1. The transition probability from state F_{n_h-1} to F_{n_h-2} is the probability that the jammer misses, which is $1 - \frac{1}{n_h-1} = \frac{n_h-2}{n_h-1}$. Similarly, at state F_{n_h-i}, the jammer chooses its next channel out of $n_h - i$ channels, hitting the used channel with probability $\frac{1}{n_h-i}$ and missing with probability $\frac{n_h-i-1}{n_h-i}$. This process may continue until the jammer scans all the channels but one (state F_1), in which case it hits the channel used by the radio with probability 1 in the next time slot.

Lemma 5.5. *In single-radio networks, the blocking probability of reactive channel-hopping*

against a scanning attack is

$$P_b^{reactive-scanning} = \frac{1}{1+\frac{n_h(\delta_x+1)}{2(\delta_d+1)}}$$

Proof. In the Markov model in Fig. 16, the transition probabilities yield the following equations: $\pi_{B_1} = \pi_{B_1} = \cdots = \pi_{B_{\frac{\delta_d+1}{\delta_x+1}}} = \pi_{F_{n_h-1}}$, where the π's are the steady-state probabilities of the Markov model. Also, $\pi_{F_{n_h-2}} = \frac{n_h-2}{n_h-1}\pi_{F_{n_h-1}}$ and $\pi_{F_{n_h-3}} = \frac{n_h-3}{n_h-2}\pi_{F_{n_h-2}} = \frac{n_h-3}{n_h-2}.\frac{n_h-2}{n_h-1}\pi_{F_{n_h-1}} = \frac{n_h-3}{n_h-1}\pi_{F_{n_h-1}} = \frac{n_h-3}{n_h-1}\pi_{B_1}$. In general, $\pi_{F_{n_h-i}} = \frac{n_h-i}{n_h-1}\pi_{B_1}$. Also, the sum of the steady-state probabilities of all states is 1, that is, $\sum_{j=1}^{\frac{\delta_d+1}{\delta_x+1}} \pi_{B_j} + \sum_{i=1}^{n_h-1} \pi_{F_i} = 1$. Substituting in this equation to solve for π_{B_1} yields: $\frac{\delta_d+1}{\delta_x+1}\pi_{B_1} + n_h\pi_{B_1} - \frac{n_h}{2}\pi_{B_1} = 1$. Thus, $\pi_{B_1} = \frac{1}{\frac{\delta_d+1}{\delta_x+1}+\frac{n_h}{2}}$. Noting that the blocking probability is the summation of the steady-state probabilities of the B states, that is, $\sum_{j=1}^{\frac{\delta_d+1}{\delta_x+1}} \pi_{B_j} = \frac{\delta_d+1}{\delta_x+1}\pi_{B_1} = \frac{1}{1+\frac{n_h}{2\alpha(\delta_d+1)}}$. □

Lemma 5.6. *In single-radio networks, the blocking probability of proactive channel-hopping against a scanning attack is:*

$$P_b^{proactive-scanning} = \frac{\frac{2n_h}{\delta_x+1} + s_d(s_d-1)}{2n_h(s_d-1+\frac{1}{\delta_x+1})}$$

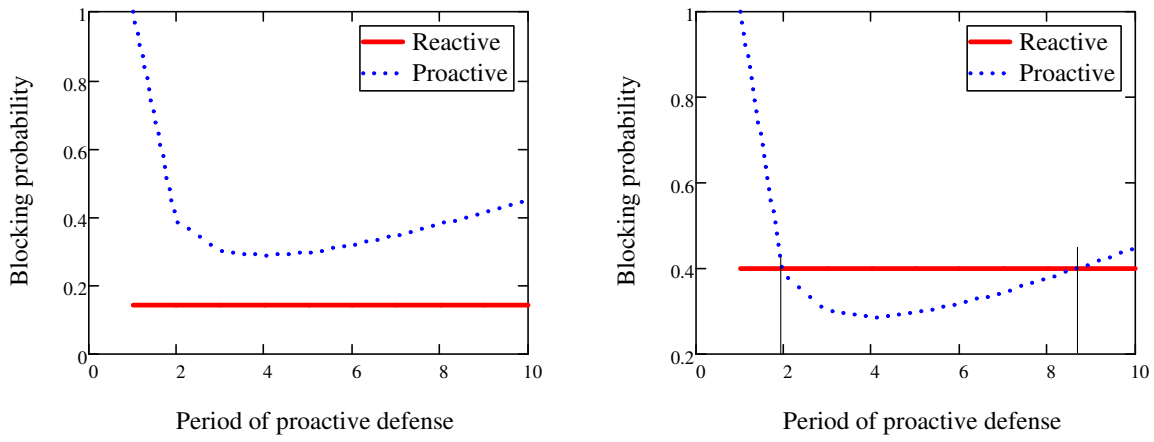
Proof. In Eqn. 2 of [100], which uses the same assumptions as in Lemma 5.5, the throughput of the proactive defense against the scanning attack is derived as $\frac{2n_h(s_d-1)-s_d(s_d-1)}{2n_h(s_d-1+\alpha)}$, where $\alpha = \frac{1}{\delta_x+1}$. The throughput is defined as the fraction of time the communication is not jammed. By definition, the blocking probability is 1 minus the throughput, resulting in the formula presented in the lemma. □

The proof of the next theorem follows directly from comparing the blocking probability in Lemma 5.5 and Lemma 5.6.

Theorem 5.7. *In single-radio networks, reactive channel-hopping achieves less blocking probability than proactive channel-hopping against scanning attack if and only if*

$$\frac{1}{1+\frac{n_h(\delta_x+1)}{2(\delta_d+1)}} < \frac{\frac{2n_h}{\delta_x+1} + s_d(s_d-1)}{2n_h(s_d-1+\frac{1}{\delta_x+1})}$$

For example, when $\delta_x = \delta_d = 1$ and $n_h = 12$ (the number of channels in 802.11a), Theorem 5.7 reduces to that reactive achieves less blocking probability if and only if $7s_d^2 - 31s_d + 96 > 0$ (Fig. 17(a)). This quadratic expression is > 0 for all values of s_d, meaning that reactive achieves less blocking probability than proactive for these parameter values. However, the window over which proactive defense achieves better performance increases with slower jamming detection. For instance, with the same example but with $\delta_d = 7$, proactive defense achieves less blocking probability than reactive for $1.9 < s_d < 8.66$ time units. The bounds are the roots of the quadratic equation $5s_d^2 - 53s_d + 84 = 0$, which is graphed in Fig. 17(b).



(a) jamming detection delay $\delta_d = 1$ time unit

(b) jamming detection delay $\delta_d = 7$ time units

Figure 17: The blocking probability of the reactive and proactive channel-hopping strategies against the scanning attack in single-radio networks. 12 channels; attack channel-sensing time $\delta_x = 1$ time unit.

In summary, reactive channel-hopping achieves better jamming resilience than proactive channel-hopping for most of the spectrum of attack and system parameters against both scanning and sweeping attacks in single-radio networks. The next subsection studies the relative performance of reactive and proactive strategies in multi-radio networks. As will be shown, the reactive strategy continues to outperform the proactive strategy in multi-radio

networks.

5.1.2 Channel-hopping in Multi-radio Networks

This subsection analyzes the jamming problem in multi-radio networks. Due to intractability of theoretical analysis in this case, simulation is used to compare proactive and reactive channel-hopping against the two attack strategies. The goal is also to determine the best defense strategy under a given system and attack scenario.

Table 3: Simulation parameters for comparing reactive and proactive channel-hopping strategies (Bold face represents default values)

Parameter	**Values**
Number of communication radios	[1-9],**3**
Number of attack radios	[3-11],**3**
Number of channels	[4-24], **12**
Channel-hopping delay	0,**1**,2,3,4,5
Period for proactive defense	10
Period for sweeping attack	1 (fast),10 (slow)
Jamming-detection threshold	1,2,3,4,5,6,**7**,8,9,10
Attacker channel-sensing time	1
communication power	40mW
jamming power	40mW
Channel-hopping power	[0-40], **0**mW

To this end, a simulation-based study was conducted to analyze the interaction between defense and attack strategies in multi-radio networks. Simulation time is divided into slots, where each time slot represents the time to transmit one packet or packet fragment, or what I call the packet-time.

The simulator models the multi-radio channel under jamming as a time sequence of decisions (by jammers and defenders) regarding which channels to operate their radio interfaces.

Assume that there are n_h channels so that the defense and attack decision vectors are modeled as n_h-bit vectors, each bit corresponding to whether the corresponding channel is used by defense radios or jammers, respectively. Because the number of used channels on any time slot cannot exceed the number of radio interfaces, the number of 1-bits in each vector is at most the number of communication radios for defense and the number of jammer radios for attack.

The space of defense and attack hopping strategies consists of all the feasible time sequences of decision vectors. Some sequences are not feasible, particularly those when a radio switches channels in a time less than the hopping delay. Depending on the overlap of jammed and communication channels, and the coding scheme used by the defenders, the probability of the communication being blocked can be calculated. In the following experiments, the blocking probability is the percentage of time **all** radios are jammed.

Table 3 summarizes the parameters used. The number of channels was varied from 4 as in the 433MHz band of CC1000 radio up to 24 with a default value of 12 as in 802.11a. The number of communication radios was varied from 1 to 9, and the number of communication and attack radios was set to be the same unless otherwise specified.

The packet time was used as the unit for time-based parameters. The channel-hopping delay was varied from 0 up to 5 packet-time with a default value of 1 packet-time. The period of the proactive hopping strategy was set at 10 (corresponding to 10% hopping cycle). The jamming-detection threshold was set to 7 packet-time (as in 802.11 retransmission threshold) by default and varied in an experiment between 1 and 10 packet-time. The attack channel-sensing time was set to 1 packet-time as a worst-case of a highly effective attacker. The energy-related parameters will be discussed in §5.1.4.

In each experiment, the performance of all six combinations of attack (scanning, fast-sweeping, and slow-sweeping) and defense (reactive and proactive) strategies was compared. Each experiment run lasted for about one million packet-time and the average of 10 runs is reported. The 90% confidence intervals were smaller than 2% of the average reported at each data point and are not shown to improve presentation.

In the first set of experiments, the number of communication radios was varied, and the number of attack radios matched the number of communication radios. As shown in

Fig. 18, as more radios are used, the blocking probability decreases, except for scanning attack against both proactive and reactive defenses. To explain, recall that both defense and attack radios are increased simultaneously in this experiment and note that when the number of radios exceeds half the total number of channels (a total of 12 channels is used in the simulations), there are always ($2\times$ number of radios $-$ number of channels) radios that have no room to escape if they get jammed. However, only the scanning attackers can make full usage of this situation; sweeping attackers are limited because they do not sense channel activity and may hop away from an active channel. It can also be observed that the blocking probability for proactive defense is always at least 10%. This is expected due to the hopping cycle of 10%; every 10 packet-time, one packet-time is blocked during channel hopping.

It was observed that with a single radio both proactive and reactive defense strategies performed almost the same against the scanning attack. This observation may explain the use of proactive defense in previous channel-hopping research (e.g., [54, 100]); proactive defense is much simpler to implement and achieves the same performance as reactive. As more radios are used, the reactive defense strategy achieved strictly less blocking probability than the proactive strategy against all attack strategies. The simulation results (at $n_f = 1$) matched the models presented in the previous subsection, except for the scanning attack. This difference is intentional and expected. Whereas the model assumes that the scanning attacker keeps history of its visited channels and avoids them when selecting its next channel, in the simulation it is assumed that the attacker selects its next channel without keeping history (see §3.2 for more details).

In the second experiment, the total number of channels was varied while fixing the number of radios and attackers at three. Fig. 19 shows that, as expected, with more channels the blocking probability decreased. The reactive strategy achieved better performance than the proactive strategy against all attack types.

Other experiments were conducted where the number of attack radios (with fixed number of defense radios), the channel-hopping overhead, the jamming-detection threshold, the attack channel-sensing delay, and the period of proactive defense were varied. A similar interaction was observed between defense and attack strategies, that is, reactive channel-hopping

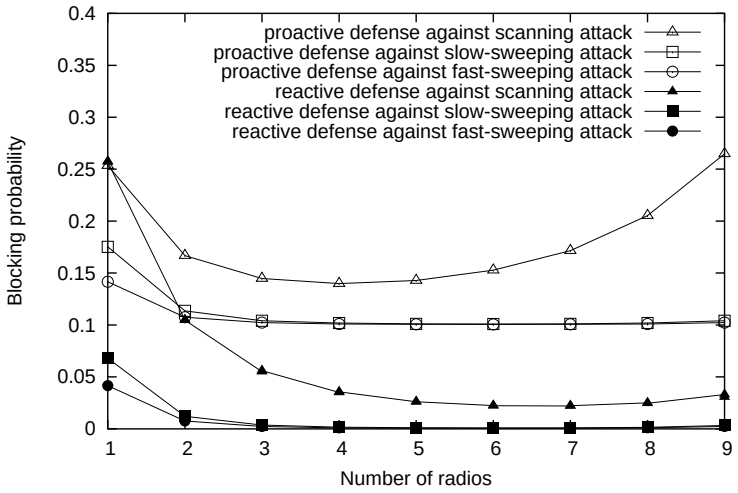


Figure 18: Effect of number of radios of both defense and attack on the communication blocking probability; 12 channels.

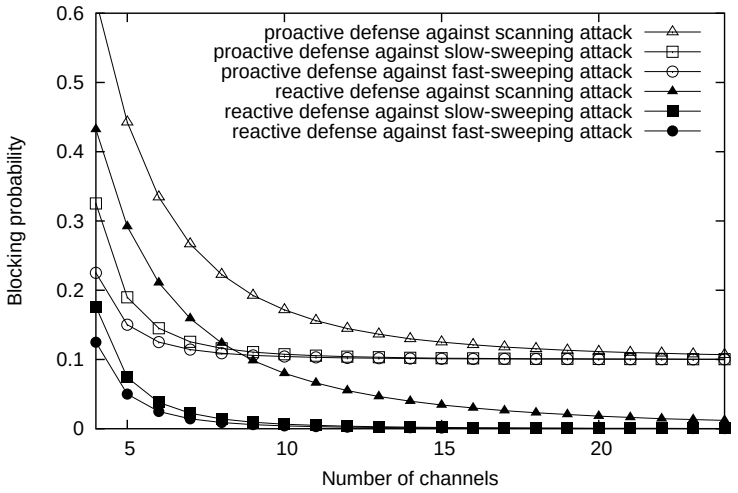


Figure 19: Effect of number of channels on the communication blocking probability; 3 radios.

achieved less or the same blocking probability compared to proactive channel-hopping.

5.1.3 The Defense-Attack Availability Game

The previous discussion addressed the question of which defense strategy is better given that the attack strategy is known. This subsection addresses the case when the attack strategy is not known. To this end, the jamming defense problem is modeled as a max-min game [106] (refer to §2.4.5 for a brief background on max-min games) between defense and attack. Two payoff functions are considered, communication availability (analyzed in this section) and energy efficiency (analyzed in the next section). This subsection defines the availability game and analyzes its outcome using simulation.

In the availability game, the main goal of the network is to deliver data. The attack on the other hand aims at blocking communication for as much time as possible. The blocking probability, $P_b(d, a, S)$, in this game is the percentage of time communication is blocked, whereby communication is blocked only when *all* radios are jammed. The blocking probability depends on the defense strategy (d), attack strategy (a), and system parameters (S), such as the number of radios and the number of channels. The communication availability is $1.0 - P_b(d, a, S)$.

The *maxminimizer* defense strategy is the one that guarantees the maximum availability knowing that the attack aims at minimizing it. It solves the problem:

$$\arg\max_{d \in \mathcal{D}} \min_{a \in \mathcal{A}} P_b(d, a, S)$$

where $\mathcal{D} = \{\text{proactive, reactive}\}$ are the defense strategies and $\mathcal{A} = \{\text{scanning, fast-sweeping, slow-sweeping}\}$ are the attack strategies.

To determine the solution of the availability game defined above, that is, to find the maxminimizer defense strategy, recall that in Fig. 18 the reactive defense achieved better performance than proactive except for the single-radio setting, whereby they both achieved almost the same performance. Also, the scanning attack strategy achieved more damage (higher blocking probability) than the sweeping attack (the bottom two curves in Fig. 18) against reactive defense. It can be concluded that the best attack strategy in the availability game is the scanning strategy, leading to the following observation under the studied system parameters: **The Nash equilibrium in the availability game is** $\langle reactive, scanning \rangle$

in multi-radio networks. In single-radio networks, there are two Nash equilibria $\langle reactive, scanning \rangle$ **and** $\langle proactive, scanning \rangle$**.**

5.1.4 Using Energy Efficiency as Performance Metric

Through the previous discussion, the focus was on communication availability. However, wireless networks are usually limited in their energy resources, and thus, taking the energy consumption into consideration is crucial to determine the best defense strategies. This subsection analyzes the jamming problem as an energy-efficiency problem. In this context, the question of which defense strategy is better is first answered. Then, a max-min game is defined with energy efficiency as the payoff function, and its outcome is analyzed using simulation.

5.1.4.1 Energy Model A metric that emphasizes energy efficiency is used. Specifically, the used metric is the Jamming Defense Power Efficiency (JDPE), which represents the communication availability achieved per unit energy, and the energy is defined relative to the attack energy consumption.

$$JDPE(d, a, S) = \frac{1.0 - P_b(d, a, S)}{\frac{\text{defense power consumption}}{\text{attack power consumption}}} \tag{5.1}$$

In order to define the power consumption of defense and attack in Eqn. 5.1, an energy model is incorporated into the simulator. A radio is either in stationary or channel-hopping states. While in stationary state, a communication radio tries to send and receive data whereas an attacker jams. Let the average power consumed by a radio while in stationary state be PS_d (PS_a for attack), and the power consumed in channel-hopping state PC_d (PC_a for attack). The average power consumed by the defense is a weighted sum of stationary power and channel-hopping power: $ws_d \cdot PS_d + wm_d \cdot PC_d$, where the weights ($ws_d$ and wm_d) are the average over time of the number of defense radios in the stationary and hopping states, respectively. Similarly, the average attack power is: $ws_a \cdot PS_a + wm_a \cdot PC_a$, where the weights ($ws_a$ and wm_a) are similarly defined.

The attack and defense stationary power was set to 40mW [115]. Equal values of stationary power were selected for both defense and attack based on the feasibility of low-power jamming attacks [82,104,153,156]. The power consumed while hopping was varied from 0mW (i.e., channel hopping consumes negligible power compared to transmit/receive power [32]) to 40mW (similar to stationary power).

5.1.4.2 Simulation Results In the first experiment, the number of radios was varied with a matching number of attackers. When a single radio is used, both proactive and reactive strategies performed almost the same (Fig. 20), again explaining the adoption of proactive defense in single-radio networks [54, 100]. With more radios, the JDPE improved except for scanning attack against proactive defense (when number of radios exceeded half the channels). This trend is similar to the proactive-scanning curve in Fig 18, which was explained previously. It was also observed that the reactive strategy performed better (higher JDPE) than proactive except against fast-sweeping attack, where they both achieved exactly the same JDPE (the bottom two lines in Fig. 20). Whereas reactive defense achieved less blocking probability against fast-sweeping attack (Fig. 18), this advantage was neutralized as it also consumed more power because of spending more time in communication.

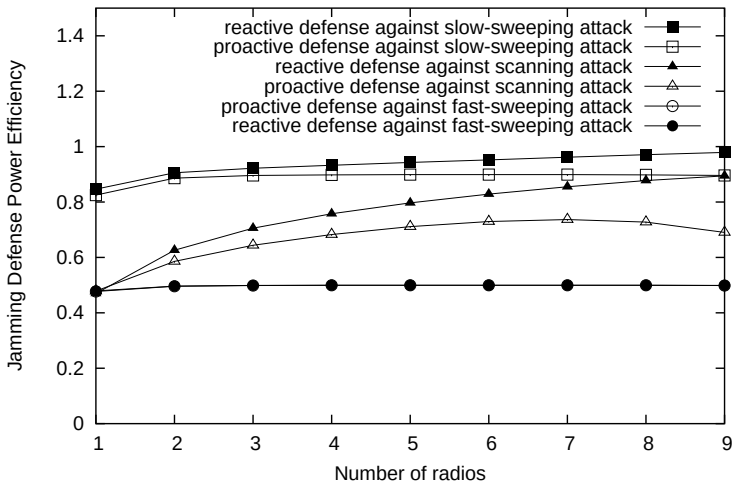


Figure 20: Effect of number of radios on JDPE; 12 channels.

In the next experiment, the total number of channels was varied while fixing the number

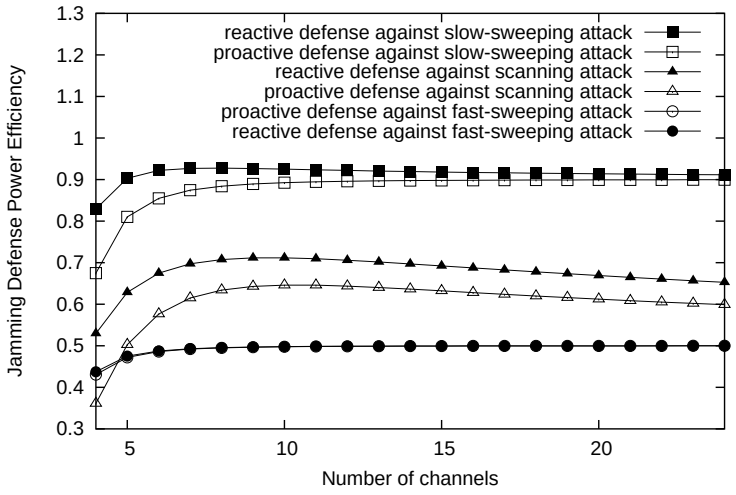


Figure 21: Effect of number of channels on JDPE; 3 radios.

of radios (for both defense and attack) at three. As shown in Fig. 21, against the sweeping attack (both slow and fast), the JDPE increased at first and then saturated at six channels for both reactive and proactive strategies. To explain, note that the relative energy consumption of proactive defense vs. fast-sweeping attack is independent of the number of channels; in the fast-sweeping attack, no matter how many channels there are, defenders (attackers) in this combination transmitted (jammed) for 9 (1) packet-time and hopped for 1 packet-time. Noting that the transmission and jamming energy consumption is the same and channel-hopping energy is 0, the proactive defense consumed $\frac{9}{5}$ more energy than fast-sweeping attack. Thus, the saturation of JDPE is mainly due to the almost constant blocking probability after 6 channels (in Fig. 19).

The saturation of JDPE in the fast-sweeping vs. reactive case has a different reason. Because the blocking probability kept on decreasing after 6 channels (bottom curve in Fig. 19), radios spent more time transmitting and receiving. Consuming more power, the relative defense-to-power consumption increased resulting in almost constant JDPE (Eqn. 5.1).

The JDPE in proactive defense vs. slow-sweeping attack followed the trend of the blocking probability in Fig. 19: improving until around 10 channels then staying constant. A slight decrease in JDPE after 6 channels in the reactive vs. slow-sweeping scenario was also

observed. This is because radios spent more time unjammed and thus consumed more power in transmission/reception, while the sweeping attackers have constant power consumption.

Against the scanning attack, both reactive and proactive defenses failed to maintain the improvement in JDPE after around 10 channels. With more channels, the radios have more chance to escape, and the scanning attackers hop more often, because a scanning attacker stays one packet-time at an idle channel before it hops to another, whereas it stays eight packet-time at an active channel before the jammed radio detects jamming and hops away. Thus, they spend more time in the zero-power channel-hopping state.

In the next experiment, the channel-hopping delay was varied. As channel-hopping delay increased, the JDPE worsened for all defense-attack combinations as expected (Fig. 22). Reactive defense was more efficient than proactive except against fast-sweeping attack, whereby both achieved the same performance.

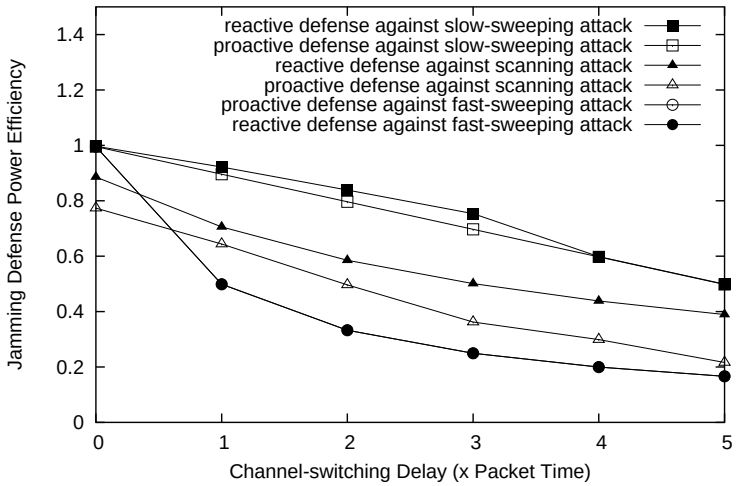


Figure 22: Effect of channel-hopping delay on JDPE. 3 radios and 12 channels.

As can be observed from Fig. 23, increasing the channel-switching power (PC_d and PC_a in §5.1.4.1) resulted in improving JDPE in all defense-attack combinations except for proactive defense against slow-sweeping attack. The fast-sweeping attackers consume half of their time hopping channels, and thus, they suffered the most from increasing the hopping energy cost, resulting in the steepest increase in JDPE. On the other hand, slow-sweeping attackers consume only 10% of their time hopping channels, and thus, are the least impacted by the

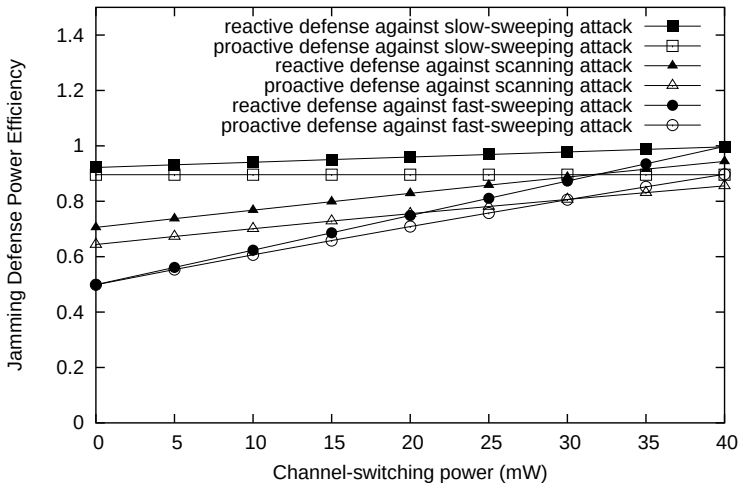


Figure 23: Effect of channel-switching power on JDPE. 3 radios and 12 channels.

increasing hopping power. Because the hopping cycle is the same for slow-sweeping attackers and proactive radios, the JDPE in their combination was not affected by the hopping power. When the channel-switching power reached the communication (jamming) power of 40mW, the effect of energy consumption on JDPE was neutralized, and JDPE was only affected by the blocking probability; the relative order among the defense-attack combinations at $PC_d = PC_a = 40$mW is the reverse of their order in Fig. 18 at $n_f = n_x = 3$;

The jamming-detection threshold affected only the reactive defense strategy as expected (Fig. 24). Although it may be expected that faster reaction to jamming would result in better performance for the reactive defense, its JDPE worsened at short jamming-detection thresholds against scanning attack. The blocking probability indeed improved at shorter thresholds. However, radios spent more time unjammed and consequently spent more power in transmission/reception, and attackers spent more time channel-hopping, resulting in a small JDPE. Other than this point in the curve, reactive was more or same energy-efficient as proactive.

In the last experiment, the number of attack radios was varied while fixing the number of defense radios (Fig. 25). Increasing the number of attackers had two effects: increased blocking probability and increased attack power consumption. At first, blocking probability

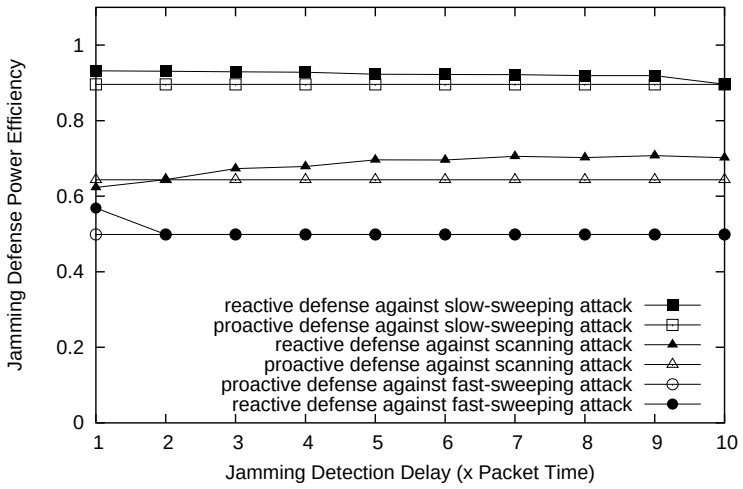


Figure 24: Effect of jamming-detection threshold on JDPE. 3 radios and 12 channels.

did not increase as fast as the attack power consumption resulting in improving JDPE. However, the blocking probability kept on increasing until its effect took over and resulted in a worsening JDPE. The turning point was different in different defense-attack combinations. In all scenarios, reactive defense achieved better or same performance as proactive defense.

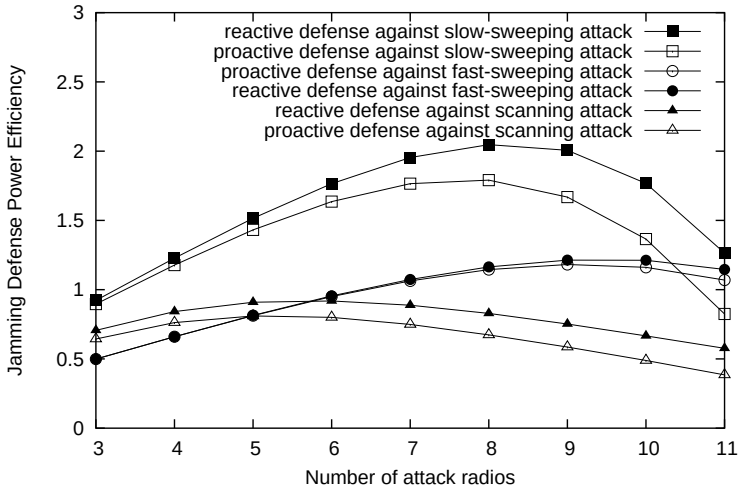


Figure 25: Effect of number of attack radios on JDPE. 3 communication radios.

In summary, the reactive hopping strategy achieved the same or higher energy-efficiency

than proactive hopping against the three attack strategies except against the scanning attack and only when the jamming-detection threshold is very small.

5.1.4.3 Efficiency Game A max-min game between defense and attack is defined to determine the most "reasonable" attack strategy to deplete network energy, in case the attack strategy is not specified a priori. The efficiency game takes energy consumption of both defense and attack into consideration. In this game, the network aims at delivering as much data as possible while extending its lifetime. Attackers on the other hand aim at incurring as much damage as possible with as low energy as possible.

Again, the *maxminimizer* defense strategy in this game is the one that solves the problem:

$$\arg \max_{d \in \mathcal{D}} \min_{a \in \mathcal{A}} JDPE(d, a, S)$$

The previous simulation analysis is used to determine the solution of the efficiency game. From Fig. 20, it is best for attackers to use the fast-sweeping attack, to cause the least JDPE, and in this case both reactive and proactive defenses perform the same.

This is confirmed at different numbers of channels. The fast-sweeping attack (bottom two lines in Fig. 21) is still the best attack strategy causing the worst JDPE. Even with 4 channels, although scanning attackers caused less JDPE when proactive defense was used, fast-sweeping attack achieved less JDPE when the best defense (reactive) was used.

From Fig. 22, when channel-hopping overhead is negligible (compared to packet-time), the best attack strategy is scanning and the best defense strategy is reactive. For the other delay values, fast-sweeping attack is best for attackers, and both reactive and proactive defenses perform the same.

An important observation is that there is a turnover in the best attack strategy as the number of jamming radios increased, as observed in Fig. 25. As the number of attack radios is below 6 (half the channels), the best attack strategy is fast-sweeping; it is scanning afterwards. The reason is that once attackers exceed half the channels, some of them do not need to move and waste time in channel-hopping.

From the previous analysis, the following conclusion is reached: **Under the studied system parameters, if the channel-hopping overhead is negligible or the number**

of attack radios far exceeds the number of communication radios, the Nash equilibria in the efficiency game in multi-radio networks is $\langle reactive, scanning \rangle$. Otherwise, there are two Nash equilibria: $\langle reactive, fast\text{-}sweeping \rangle$ and $\langle proactive, fast\text{-}sweeping \rangle$.

5.1.5 Summary

This section generalized the software-based channel-hopping jamming defense into multi-radio wireless networks. Two defense strategies were compared, namely proactive and reactive, against three attack strategies, namely scanning, slow-sweeping and fast-sweeping. The jamming defense problem was modeled as a max-min game between defense and attack. Two variations of this game differing in the payoff function were presented.

The results presented in this section show that the proactive defense achieves the same or a little less jamming resiliency than the reactive defense in single-radio networks. The proactive defense is also easier to implement, and thus, it is a more attractive option than reactive in single-radio networks. This conclusion coincides with and supports the current research's focus on proactive channel-hopping [54,100,153]. However, for multi-radio networks, reactive defense is more resilient to jamming. These conclusions are based on theoretical analysis for single-radio networks and on simulation experiments for multi-radio networks, whereby all defense-attack combinations were compared under varying system parameters.

The next section presents approximate models developed for the reactive defense and scanning attack strategies, and use these models to adapt system behavior on-line.

5.2 HONEYBEES

This section considers the problem of maximizing network goodput under jamming attacks in multi-radio networks by combining channel-hopping and (n, m) error-correcting codes (ECC) (§2.4.3), such as the IDA algorithm [118]. Two factors affect goodput. First, high redundancy in ECC reduces goodput. Second, jamming may result in data loss if the number

of clear (unjammed) radios is smaller than the number necessary to recover transmitted data. However, these two factors are interestingly interdependent. Increasing ECC redundancy results in increased overhead but also in increased jamming resiliency. This interlocking suggests the existence of optimal ECC redundancy that achieves maximum goodput. Noting that such optimal redundancy depends as well on system and attack parameters and that the attack strategy is not always known beforehand, an adaptive mechanism is needed to discover attack parameters and tune the ECC accordingly.

A first step in devising such an adaptive mechanism is modeling defense and attack strategies under different ECC parameters. Such model can be used to detect attack strategies given the known system and defense parameters. This section develops and validates models for reactive hopping strategies against scanning attack strategies, taking into consideration that the data is encoded to correct errors.

Two variations of the reactive defense are modeled, namely *straightforward* and *deceptive*. In the straightforward reactive defense, jammed radios select the next target channels randomly from the set of unused channels, whereas in the deceptive reactive defense, jammed radios select their next channels randomly from a set containing all channels (including currently used channels).

Also, two variations of the scanning attack are modeled, namely *exploratory* and *conservative*. In the exploratory-scanning attack, jammers at unused channels select the next target channels randomly from the set of unjammed channels, whereas in the conservative-scanning attack, jammers at unused channels select their next targets randomly from a set containing all channels (including the currently jammed channels) in anticipation of the deceptive defense.

5.2.1 Maximizing Network Goodput

As mentioned in §3.2, and (n_f, m) ECC is used, where n_f is the number of radios. This ECC reduces the goodput by the code rate $\frac{m}{n_f}$ (§2.4.3). Furthermore, jamming reduces the goodput of an error-correction-encoded channel by blocking communication at more than $n_f - m$ radios. The goodput (as a fraction of the maximum throughput achievable in

absence of jamming) of the multi-radio channel can then be formulated in terms of the ECC parameters and the jamming-induced blocking probability (P_b) as:

$$goodput = \frac{m}{n_f}(1 - P_b)$$

There is a relation between m, representing the amount of redundancy of the ECC, and the blocking probability, P_b. As m decreases, and correspondingly, the amount of redundancy required in each encoded piece increases, it becomes harder for jammers to cause data loss, and hence, the blocking probability decreases. The amount of decrease of the blocking probability and the resulting net effect on goodput depend on the hopping and jamming strategies as well as the number of channels.

Considering the problem of maximizing goodput, the solution space encompasses the selection of the coding parameter m and the hopping strategy. Because the goodput also depends on the adversarial attack strategy, which may not be always known beforehand, a mechanism is needed to discover the attack parameters, particularly the attack strategy and the number of attackers, and adjust the coding parameters and hopping strategy accordingly. A main building block of such mechanism is a model that captures the interaction between defense, attack, and system parameters. Building this model is the focus of the next subsection.

5.2.2 Markovian Models

This subsection presents models to derive the blocking probability given defense, attack, and system parameters. The main envisioned usage of this model is to drive an adaptive mechanism that infers the otherwise unknown attack parameters and adjusts the defense parameters to maximize goodput. To this end, Markov Chains are used to model reactive defense and scanning attack in multi-radio networks. A Markov chain is represented by a set of states and transition probabilities, p_{ij}, between these states. In these models, each state represents the number of jammed radios, ranging from 0 to n_f. Therefore, there are $n_f + 1$ states.

From the steady-state probabilities $\pi_i, i = 0, 1, \cdots, n_f$, the blocking probability can

be derived given the parameters of the ECC in effect. For an (n_f, m) ECC, the blocking probability is the sum of the steady-state probabilities of states in which more than $n_f - m$ radios are jammed. That is, $P_b = \sum_{i=n_f-m+1}^{n_f} \pi_i$.

To solve the Markov model, the transition-probability matrix, $[p_{ij}]$, is derived. The steady state probabilities can then be derived using the standard matrix equations:

$$[\pi_i][p_{ij}] = [\pi_i] \quad \text{and} \quad \sum_{i=0}^{n_f} \pi_i = 1$$

The models assume instantaneous channel hopping, that is, the delay of switching channels is 0, and that the delay of detecting jamming (for defense) and sensing channel activity (for attack) is one time slot. These assumptions are needed to keep the model analysis simple; without these assumptions, the state definition of the Markov chain has to include state for each communication and attack radio, such as the number of remaining time slots for a hopping radio to reach its next channel, the number of remaining time slots for a jammed communication radio to detect jamming, and the number of remaining time slots for an attack radio to detect channel inactivity.

The *drawing without replacement* formula is extensively used in the models, and hence a shortcut is used:

$$DWR(a, b, c, d) = \frac{\begin{pmatrix} b \\ d \end{pmatrix} \begin{pmatrix} a-b \\ c-d \end{pmatrix}}{\begin{pmatrix} a \\ c \end{pmatrix}}$$

$DWR(a, b, c, d)$ is the probability that c drawings (without replacing the drawn members) from a population of size a, of them b are distinguished, yield exactly d distinguished members. $\begin{pmatrix} x \\ y \end{pmatrix}$ is the binomial coefficient of x and y, also called "x choose y".

In what follows, state i represents the state where i of the n_f radios are jammed as depicted in Fig. 26. All the jammed i radios hop channels in the next time-step. In the straightforward defense, the i radios randomly select the next target channels from among the unused $n_h - n_f$ channels, whereas in the deceptive defense, the i radios select their next

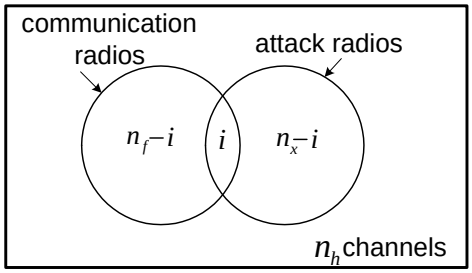


Figure 26: Communication and attack radios at state i (i jammed radios).

channels from both the unused channels as well as their current channels. It should be noted, however, that in straightforward defense, if i is larger than $n_h - n_f$, then there are not enough unused channels for all the i jammed radios to move to, and hence, $\ell = i - (n_h - n_f)$ (randomly selected) of the i jammed radios are forced not to hop, while the remaining $i - \ell$ hop to new channels.

Similarly, when transitioning out of state i, all the $n_x - i$ attack radios that are not successfully jamming active channels will move to new channels. In the exploratory attack, the $n_x - i$ attack radios randomly select the next target channels from among the unjammed $n_h - n_x$ channels, while in the conservative attack, the $n_x - i$ attack radios select their next channels from both the unjammed channels as well as their current channels. It should be noted, however, that in exploratory attack, if $n_x - i$ is larger than $n_h - n_x$, then there are not enough unjammed channels for all the $n_x - i$ attack radios to move to, and hence, $\ell_x = (n_x - i) - (n_h - n_x)$ (randomly selected) of the $n_x - i$ attack radios are forced not to hop, while the remaining $n_x - i - \ell_x$ hop to new targets.

The following four theorems present formula for the transition probabilities p_{ij} for the four attack-defense combinations.

Theorem 5.8. *For a* straightforward reactive defense vs. conservative scanning attack *with no channel-hopping overhead, the transition probability p_{ij} is:*

$$p_{ij} = DWR(n_h - i, n_f - \ell, n_x - i, j - \ell)$$

where n_x is the number of attackers, n_f the number of communication radios, n_h the number of channels, and $\ell = max(0, i - (n_h - n_f))$.

Proof. At state i, all the jammed i radios hop channels in the next time-step. However, $\ell = max(0, i - (n_h - n_f))$ radios have to stay put in their current channels if there are not enough new channels to hop to, where ℓ is the difference between the i radios that need to hop and the $n_h - n_f$ available unused channels, that is, $\ell = max(0, i - (n_h - n_f))$. Meanwhile, $n_x - i$ attackers detect that their channels are unused and select new channels out of $n_h - n_x$ unjammed channels plus the previously-jammed $n_x - i$ for a total of $n_h - i$ channels in the next time-step.

In the next time-step, ℓ radios are already jammed because they have to stay put in the jammed channels. Thus, to jam a total of j radios, the $n_x - i$ hopping attackers have to jam exactly $j - \ell$ radios. The probability of this event is derived using drawings-without-replacement:

$$DWR(n_h - i, n_f - \ell, n_x - i, j - \ell)$$

where the hopping attackers draw $n_x - i$ channels out of $n_h - i$ channels with $n_f - \ell$ of them used by communication radios. □

Intuitively, the first component of the next formula represents deceptive-reactive communication radios selecting their next channels from the ones they currently use, and thus, stay jammed. The second component represents the probability that attack radios jam enough radios to cause exactly j jammed radios in the next time slot.

Theorem 5.9. *For a* deceptive reactive defense vs. conservative scanning attack *with no channel-hopping overhead, the transition probability p_{ij} is:*

$$p_{ij} = \sum_{o=o_{min}}^{o_{max}} DWR(n_h - (n_f - i), i, i, o) \cdot$$
$$DWR(n_h - i, n_f - o, n_x - i, j - o)$$

where $o_{min} = max(0, j - (n_x - i))$, and $o_{max} = min(i, j)$.

Proof. At state i, all the jammed i radios hop channels in the next time-step and select from the $n_h - n_f$ unused channels plus their current i channels. Let o radios hop to channels from the previously-used i channels. The probability of this event is:

$$p_o = DWR(n_h - (n_f - i), i, i, o)$$

where radios draw i channels out of $n_h - n_f + i$ channels, i of them active in the previous time-step.

Meanwhile, $n_x - i$ attackers detect that their channels are unused and hop to channels selected out of the $n_h - i$ unjammed channels.

In the next time-step, o radios are their jammed because they stay at already-jammed channels. Thus, to jam a total of j radios, the $n_x - i$ hopping attackers have to jam exactly $j - o$ radios. The probability of this event is:

$$p = DWR(n_h - i, n_f - o, n_x - i, j - o)$$

where the hopping attackers draw $n_x - i$ channels out of $n_h - i$ channels with $n_f - o$ of them used by communication radios.

To compute the overall transition probability, note that the sample space is partitioned based on o, the number of radios falling into the i channels. Therefore, the transition probability is computed as follows: $p_{ij} = \sum_o [p_o \cdot p]$. Substituting in this equation yields the formula presented in the theorem. The summation limits o_{min} and o_{max} are computed by solving the constraints for each combination x choose y: that both x and y are ≥ 0 and that $x \geq y$. □

Intuitively, the first and second components of the next formula represent exploratory-reactive communication radios escaping from their jammed channels but ending up selecting their next channels where attack radios stay because they do not have enough unjammed channels to move to. The third component represents the probability that attack radios end up jamming j radios in the next time slot.

Theorem 5.10. *For a* straightforward reactive defense vs. exploratory scanning attack *with no channel hopping overhead, the transition probability* p_{ij} *is:*

$$p_{ij} = \sum_{k=k_{min}}^{k_{max}} DWR(n_h - n_f, n_x - i, i, k) \cdot$$
$$\sum_{m=m_{min}}^{m_{max}} DWR(n_x - i, \ell_x, k, m) \cdot$$
$$DWR(n_h - n_x, n_f - \ell - k, n_x - i - \ell_x, j - \ell - m),$$

where $\ell_x = max(0, n_x - i - (n_h - n_x))$, $k_{min} = max(0, n_f + n_x - n_h - \ell)$, $k_{max} = min(n_x - i, i - \ell)$, $m_{min} = max(0, i - \ell - n_h + n_f + \ell_x, j - n_f + k, i - n_x + \ell_x + j - \ell, k - n_x + i + \ell_x)$, *and* $m_{max} = min(k, \ell_x, j - \ell, n_h - 2n_x - n_f + k + i + j + \ell_x)$.

Proof. All the jammed i radios at state i hop channels in the next time-step. They select new channels out of the unused $n_h - n_f$ channels. However, it may be the case that there are not enough new channels to accommodate all hopping radios, and ℓ of them have to stay put in their current channels.

Also, some radios hop to channels in the $n_x - i$ channels that were occupied by the rest of the attackers. The probability that k radios fall on these $n_x - i$ channels is: $p_k = DWR(n_h - n_f, n_x - i, i, k)$, where radios draw i channels out of $n_h - n_f$ channels with $n_x - i$ jammed in the previous time-step.

Meanwhile, $n_x - i$ attackers detect that their channels are unused and hop to new channels (out of the $n_h - n_x$ unjammed channels) in the next time-step. But, it may be the case that there are not enough new channels to accommodate all hopping attackers. The number of attackers that have to stay put is ℓ_x, where ℓ_x is the difference between the $n_x - i$ attackers that need to hop and the $n_h - n_x$ available unjammed channels, that is, $\ell_x = max(0, n_x - i - (n_h - n_x))$.

In the next time-step, ℓ radios are already jammed because they have to stay put in the jammed channels, and some of the k radios that hopped to the $n_x - i$ channels previously occupied by attackers may be jammed as well if they coincide with the ℓ_x attackers that have to stay put. Let the number of radios that fall into the ℓ_x channels be denoted as m.

The probability that m radios fall on the ℓ_x staying attackers given that k radios hop to the previously-jammed $n_x - i$ channels is: $p_m = DWR(n_x - i, \ell_x, k, m)$, where radios draw k channels out of $n_x - i$ channels with ℓ_x jammed by staying attackers. The rest of the k

radios are not jammed, because there are no other attackers in the $n_x - i$ channels except the ℓ_x attackers.

On the attack side, there are $n_x - i - \ell_x$ attackers that hopped their channels in search of used channels out of $n_h - n_x$ channels. Because $\ell + m$ radios are already jammed, all that these attackers have to do now in order to jam a total of j radios is to jam $j - \ell - m$ radios from the rest of the radios $(n_f - \ell - k)$. The probability that the hopping attackers jam *exactly* $j - \ell - m$ radios is: $p_{km} = DWR(n_h - n_x, n_f - \ell - k, n_x - i - \ell_x, j - \ell - m)$, where the hopping attackers draw $n_x - i - \ell_x$ channels out of $n_h - n_x$ channels with $n_f - \ell - k$ used by communication radios.

The sample space is partitioned based on k, the number of radios falling into the $n_x - i$ channels. The space is partitioned further by m, the number of radios falling into the staying attackers. Therefore, the transition probability is computed as follow: $p_{ij} = \sum_k [p_k \cdot \sum_m [p_m \cdot p_{km}]]$. Substituting in this equation yields the formula presented in the theorem. Again, the summation limits k_{min}, k_{max}, m_{min}, and m_{max} are computed by solving the constraints for each choose combination. □

Intuitively, the first component of the next formula represents jammed radios that stay jammed because they end up staying at their current channels. The second and third components represent jammed radios that stay jammed because they escape to channels jammed by attack radios that do not have enough channels to move to. The fourth component represents the probability that attack radios jam enough radios to cause exactly j jammed radios in the next time slot.

Theorem 5.11. Deceptive reactive defense vs. exploratory scanning attack. *The transition probability p_{ij} for deceptive reactive defense and exploratory scanning attack with no channel hopping overhead is:*

$$p_{ij} = \sum_{o=o_{min}}^{o_{max}} DWR(n_h - (n_f - i), i, i, o) \cdot$$

$$\sum_{k=k_{min}}^{k_{max}} DWR(n_h - n_f, n_x - i, i - o, k) \cdot$$

$$\sum_{m=m_{min}}^{m_{max}} DWR(n_x - i, \ell_x, k, m) \cdot$$

$$DWR(n_h - n_x, n_f - o - k, n_x - i - \ell_x, j - o - m)$$

where $o_{min} = max(0, i - (n_h - n_f), n_f - n_h)$, $o_{max} = i$, $k_{min} = max(0, n_f - n_h + n_x - o)$, $k_{max} = min(n_x - i, i - o, n_f - o)$, $m_{min} = max(0, j - n_f + k, k - n_x + i + \ell_x, j - o + n_x + i + \ell_x)$, *and* $m_{max} = min(k, \ell_x, j - o, n_h - 2n_x - n_f + i + \ell_x + j + k)$.

Proof. At state i, all the jammed i radios hop channels in the next time-step. Instead of selecting all new channels, they hop to channels out of the unused $n_h - n_f$ channels plus their current i channels. Let o radios select their channels from the i channels previously used. As previously discussed, the probability of this event is: $p_o = DWR(n_h - (n_f - i), i, i, o)$.

Also, let k of the remaining $i - o$ radios, which now select their channels out of only the unused $n_h - n_f$ channels, hop to channels from the $n_x - i$ channels occupied by the rest of the attackers. The probability of this event is: $p'_k = DWR(n_h - n_f, n_x - i, i - o, k)$, where radios draw $i - o$ channels out of $n_h - n_f$ channels with $n_x - i$ jammed in the previous time-step. On the other side, $n_x - i$ attackers detect that their channels are unused and hop to new channels (out of the $n_h - n_x$ unjammed channels) in the next time-step. Also, ℓ_x attackers stay put because there is not enough new unjammed channels.

In the next time step, o radios are already jammed because they stay put at the jammed i channels. Let m radios, out of the k that hop to the $n_x - i$ channels previously occupied by attackers, coincide with the ℓ_x attackers that have to stay put. These m radios will be jammed as well. As previously discussed, the probability of this event is: $p'_m = DWR(n_x - i, \ell_x, k, m)$. The probability that the $n_x - i - \ell_x$ hopping attackers jam *exactly* $j - o - m$ radios is: $p_{okm} = DWR(n_h - n_x, n_f - o - k, n_x - i - \ell_x, j - o - m)$, where the hopping attackers draw $n_x - i - \ell_x$ channels out of $n_h - n_x$ channels with $n_f - o - k$ used by communication radios.

The sample space is partitioned based on o, the number of radios falling into the i channels. The space is partitioned further by k, the number of radios falling into the previously-jammed $n_x - i$ channels, and further by m, the number of radios falling into the ℓ_x staying attackers. Therefore, the transition probability is computed as follows: $p_{ij} = \sum_o [p_o [\sum_k [p'_k \cdot \sum_m [p'_m \cdot p_{okm}]]]]$. Substituting in this equation yields the formula pre-

sented in the theorem. The summation limits o_{min}, o_{max}, k_{min}, k_{max}, m_{min}, and m_{max} are computed simply by solving the constraints for each choose combination. □

5.2.3 Simulation Analysis and Model Validation

Table 4: Simulation parameters for validating the models of reactive channel-hopping against scanning attack (Bold face represents default values)

Parameter	Values
Number of pieces needed to correct errors (m)	[1-3], **1** (full replication)
Number of channels	[4-20], **12**
Number of radios	[1-11],**3**
Number of attackers	[1-11],**3**
Channel-hopping delay	**1** packet-time
Jamming-detection threshold	**10** packet-time
Attacker channel-sensing time	**10** packet-time

A simulation study was conducted to validate the models presented in the previous subsection. The simulator models the reactive defense and scanning attack strategies with the same assumptions used in deriving the theoretical results except the negligible hopping delay assumption. For simulator validation, the simulations were also run with the same values as the model, and the simulation results exactly matched the model results; curves are not shown because they would be redundant and not contribute to the presentation. Simulation time is divided into slots, where each time slot represents the time to transmit one piece of ECC-encoded packets, or the packet-time. The default ECC used in the simulation is $(n_f, 1)$, that is, any piece of encoded data is enough to recover the original data, and correspondingly, the blocking probability is the percentage of time *all* radios are jammed.

Table 4 summarizes the parameters used. The number of channels was varied from 4 up to 20 with a default value of 12 as in 802.11a. The number of radios was varied from 1 to

11 with a default value of 3, and the number of attack radios was set to be the same as the number of defense radios unless otherwise specified. To examine the effect of the assumption of negligible hopping delay, the channel-hopping delay was set to 1 packet-time (instead of 0 packet-time in the model), and both the jamming-detection threshold and the attack channel-sensing time were set to 10 packet-time (instead of 1 packet-time in the model). Each experiment run lasted for about one million packet-time, and the average of 10 runs is reported. The 90% confidence intervals were smaller than 2% of the average reported at each data point.

For all the tests, as can be seen from the figures, the model matched the simulation results almost exactly for most of the studied parameter range.

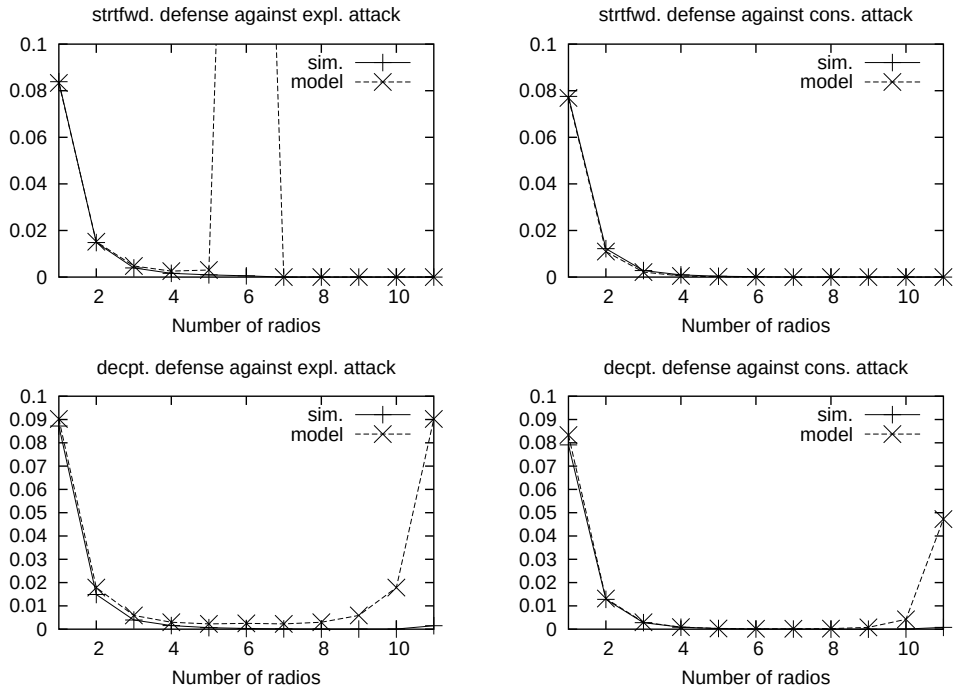


Figure 27: Effect of number of communication and attack radios on blocking probability; 12 channels.

Effect of number of communication and attack radios. In the first set of experiments, the number of radios per communication node was varied, and the number of attack radios was changed to match the number of communication radios. As shown in Fig. 27, as more radios per node are used, the blocking probability decreases. A notable difference

between simulation results and the model occurred in the deceptive defense strategy against both attack strategies (the bottom two graphs in Fig. 27) when the number of radios is very close to the number of channels (12 channels in this experiment). I hypothesize that this discrepancy occurs as channel hopping occurs more frequently (many radios with less room to escape) and the effect of the non-zero hopping delay (compared to 0 delay in the model) becomes more pronounced. In particular, the non-zero hopping delay results in *de-synchronization* of the attack radios, so that they do not jam at the same time, and thus, cannot block communication (i.e., the blocking probability is zero).

Another point where simulation results differed from the model prediction is the bump in straightforward defense against exploratory attack (top-left graph in Fig. 27). In the straightforward-exploratory combination, the blocking probability is predicted by the model to increase to 0.5 at 6 radios[2]. This bump occurs at a number of radios exactly half of the number of channels. The reason is that at this number the system alternates between all radios being jammed in one time slot followed by all radios free in the next and so on. This alternation happens because on each time slot, the only option for communication (attack) radios is to hop to the other half of channels. I hypothesize that this bump did not occur in the simulations because the non-zero hopping delay breaks this synchronized alternation. This alternation is not predicted in conservative attack (and deceptive defense) because conservative attack (deceptive defense) radios have the option of choosing their next channels from the ones they currently use.

Effect of number of channels. In the second experiment, the total number of channels was varied while fixing the number of radios and attackers at three. Fig. 28 shows that, as expected, with more channels the blocking probability decreased except for the bump in the straightforward defense (in the model only as explained above) at a number of channels twice the number of radios. The straightforward defense exhibits superior performance at low number of channels (except for the critical number of twice the number of radios).

Effect of number of attack radios. In the third experiment, the number of attack radios was varied while fixing the number of communication radios at three and the number

[2]A similar bump can also be observed in Fig. 28 (top-left graph) at 6 channels; the number of radios per node was 3 in that experiment.

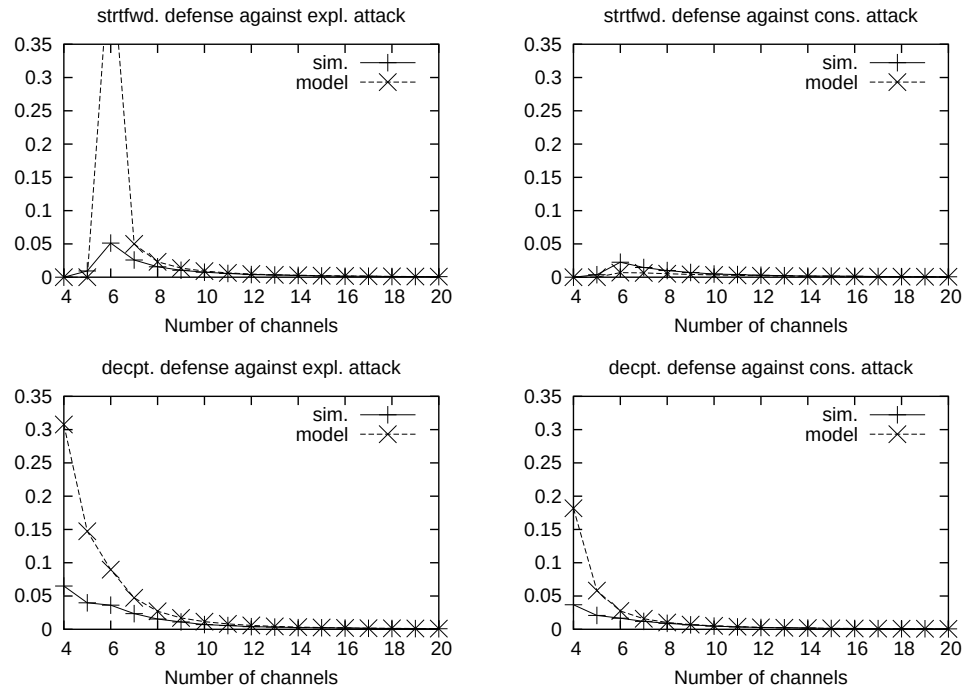


Figure 28: Effect of number of channels on blocking probability; 3 radios.

of channels at 12. As expected, the blocking probability increases with increasing number of attack radios. The straightforward defense achieves slightly better performance at high number of attackers.

Simulation results differed significantly from model prediction at high number of attackers in the deceptive defense. At high number of attackers, deceptive radios hop more frequently, emphasizing the effect of the non-zero hopping delay. This discrepancy was not as pronounced in the straightforward defense; straightforward radios escape from jamming slightly better, and, thus, they experience less hopping. The model is otherwise a little more conservative than the simulation results, predicting higher blocking probability in most cases.

In summary, the exploratory-scanning attack causes more damage in all cases, and the straightforward-reactive defense is more effective than the deceptive defense.

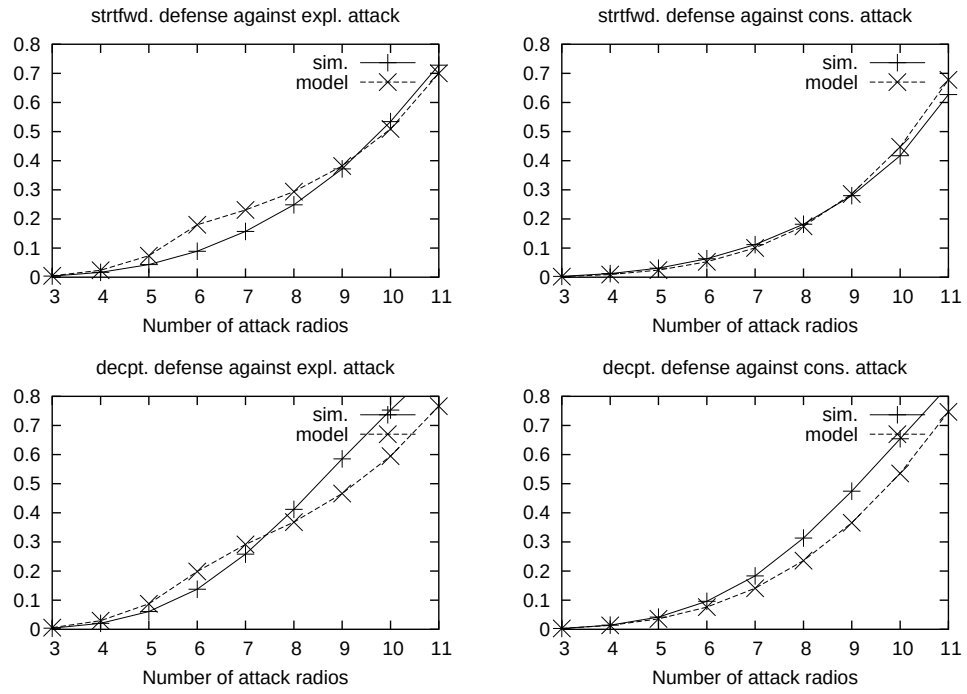


Figure 29: Effect of number of attack radios on blocking probability; 3 communication radios and 12 channels.

5.2.4 The Honeybees Adaptive Algorithm

The number of attack radios cannot always be known beforehand. The honeybees algorithm uses the models developed in §5.2.2 to discover the number of attack radios and adjust the ECC parameters accordingly to achieve the best goodput. Periodically, the goodput is measured and fed into the models to predict the number of attack radios. This can be visualized as a horizontal line at the measured goodput value. The x-value of the intersection of this line with the model curve of the currently used ECC parameters is used as an estimate of the actual number of attack radios, which is then used in a feedback loop to adjust the ECC parameters optimally.

To illustrate the operation of the algorithm, an experiment was conducted, whereby the ECC parameters were varied, in particular the number of encoded data pieces required to recover from errors. The straightforward-reactive defense and the exploratory-scanning

attack were simulated, because the simulation results above indicate that these strategies are superior for defense and attack, respectively.

Fig. 30 shows goodput results for 3 communication radios, 12 channels. The number of attack radios was varied to examine the effect of attack parameters on the optimal ECC. It can be observed that the best ECC depends on the number of attackers. This dependency is predicted by the models in §5.2.2 and confirmed by simulation. Up to 6 attack radios, $(3, 2)$ ECC achieved the best goodput (as a ratio of the maximum throughput in absence of jamming). With more than 6 radios, full-replication, or $(3, 1)$ ECC, achieved the highest goodput.

Consider that the ECC used is $(3, 2)$, and the goodput is measured as 0.4 of the maximum achievable throughput. Then, from the $(3, 2)$ ECC (model) curve in Fig. 30, the number of attack radios can be estimated as 5. Because the $(3, 2)$ ECC is still optimal at 5 attack radios, the algorithm causes no change in ECC parameters. However, if the goodput is measured as < 0.2, the number of attack radios is estimated as > 7, and the algorithm changes the ECC to $(3, 1)$.

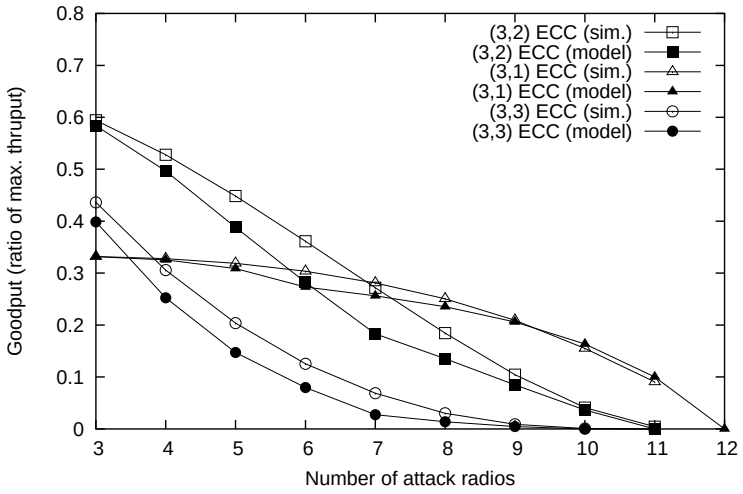


Figure 30: The best ECC redundancy depends on the number of attack radios. 3 communication radios, 12 channels, straightforward defense against exploratory attack.

5.2.5 Summary

This section presented the honeybees algorithm, which combines the software-based channel hopping with error-correcting codes (ECC). The problem of maximizing network goodput is defined, and the inter-dependency between ECC redundancy and jamming-induced blocking probability is illustrated, suggesting the existence of an optimal ECC redundancy. The optimal ECC depends not only on system parameters, such as the number of radio interfaces and radio channels, but also on the defense hopping strategy, number of attackers, and attack hopping strategy.

Models were developed for reactive defense strategies against scanning attack strategies under varying ECC parameters. These models compute the blocking probability given the attack and defense hopping strategies, the number of channels, communication radios, and attack radios. The models were validated using simulation experiments. The honeybees algorithm uses these models to detect attack parameters, if unknown, from measured blocking probability and known defense and system parameters and adaptively adjust the ECC parameters on the fly as attack parameters change. The implementation and evaluation of honeybees are subjects of future work.

5.3 CONCLUSIONS

This chapter presents the honeybees dodging algorithm, which integrates channel-hopping and error-correcting codes (ECC) to mitigate the wireless jamming attack in multi-radio networks.Two channel-hopping strategies, namely proactive and reactive, were compared: analytically in the context of single-radio networks and using a combination of simulation and game-theoretic analysis in the context of multi-radio networks. Results show that the reactive strategy achieves better jamming resiliency for most of the space of system and attack parameters. Results also show that the scanning attack causes more damage than the sweeping attack. Therefore, the combination of reactive defense against scanning attack is studied further, and an adaptive algorithm that is based on a Markov-Chain model is

devised to detect the number of jammers and successfully adjust the redundancy of the ECC to achieve efficient operation.

6.0 IDENTIFICATION OF SERVICE-LEVEL DOS ATTACKERS IN THE INTERNET

The dodging framework proposed in this dissertation provides tools and techniques to mitigate DoS attacks and to identify DoS attackers. In the previous chapter, the mitigation component has been presented and studied in the context of wireless networks. In many situations, mitigation is the only solution against a radio-interference DoS attack. Even if attackers are identified, it is hard to prevent them from sending interfering radio signals into the shared medium. On the other hand, Internet sites have more control over traffic that enters their networks. For instance, they can employ packet filtering to stop traffic from identified attackers. For this reason, attacker identification (the second component of dodging) plays a more important role in DoS defense in the Internet than its role in wireless networks. I thus devise and study attacker identification in the context of Internet services.

This chapter presents two algorithms to identify non-compliant and compliant attackers. In §6.1, the roaming honeypots algorithm for identifying non-compliant attackers is presented and evaluated. Roaming honeypots defines the mapping between clients and *physical* servers. Roaming honeypots provides attack mitigation as well. As was described in §4.3, identification of compliant attackers needs much more resources than can be physically provisioned, and thus, roaming honeypots is limited in this regard. §6.2 presents the live baiting algorithm, which combines virtual resources and group-testing theory (§2.3) to identify compliant attackers.

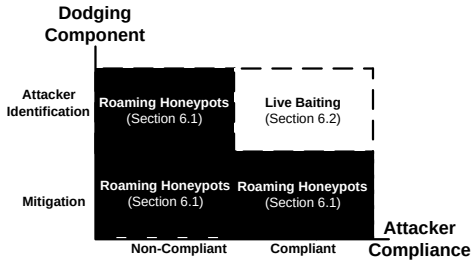


Figure 31: The part of the DoS problem space addressed by the roaming honeypots algorithm. Roaming honeypots provides identification of non-compliant attackers as well as mitigation against compliant and non-compliant attackers.

6.1 IDENTIFICATION OF NON-COMPLIANT DOS ATTACKERS

As illustrated in Fig. 31, the roaming honeypots algorithm achieves identification of non-compliant attackers and attack mitigation. The roaming honeypots algorithm defines the mapping function μ between service users and servers such that the set of active servers continuously changes in a pseudo-random way. More specifically, time is divided into epochs, and during each epoch the algorithm ensures that some servers are active and some idle. The active and idle servers at each time epoch are determined using a *hash chain* seeded by a shared secret between servers and clients as will be described in more detail in §6.1.2.

The mapping function, μ, in roaming honeypots satisfies the properties of the non-compliant identification component described in §4.3: (**non-blocking**) all client (except non-compliant attackers) requests are always sent to the active servers, (**uniform**) clients (except non-compliant attackers) distribute their request load evenly over the active servers, (**dynamic**) the set of active servers changes from epoch to epoch, (**loose**) there are idle servers at each epoch, and (**unpredictable**) the set of active servers is hard to predict by non-compliant attackers.

The roaming honeypots algorithm provides an approximation of the function Δ presented in Definition 4.1. In roaming honeypots, $\Delta(\sigma, \tau)$ returns *under attack* if resource σ is idle (i.e., $\sigma \in R_I^\tau$) and is being accessed at time instant τ. Otherwise, $\Delta(\sigma, \tau)$ returns *not attacked*.

6.1.1 Roaming Honeypots Overview

The roaming honeypots algorithm enables the camouflaging of honeypots (see §2.2.2 for more details about honeypots) as real servers by designating pseudo-randomly-selected, time-varying subsets of servers to act as honeypots at different time epochs. Honeypots are camouflaged in two ways. First, the pseudo-random selection makes their addresses highly unpredictable to non-compliant attackers. Second, the idle servers, which act as honeypots, respond to requests in a similar way as active servers, to increase the delay that an attacker would incur to discover them.

Clients (except non-compliant attackers) direct their requests only to active servers away from the honeypots. Therefore, any client that sends requests to a honeypot is considered an attacker, and its address is added into a *black-list*. Requests from identified attackers are filtered out during subsequent epochs.

Attack mitigation is achieved because roaming opens up windows of opportunity for requests from clients to get serviced even under attack. These opportunity windows happen at each roaming event when some servers switch from idle to active. These servers start "fresh", that is, with empty service queues, and hence attackers take some time to discover and exhaust the resources of the newly-activated servers. During these transient intervals, the compliant requests have a chance to get serviced.

In what follows, the components of the roaming honeypots algorithm are described. The notation is summarized in Table 5.

6.1.2 Distributed honeypot selection

The first component of the roaming honeypots algorithm is the selection of the honeypots. Honeypot selection is distributed, that is, it is run at both servers and clients. As noted earlier, time is divided into epochs with each epoch containing one or more time slots, and roaming occurs only at epoch boundaries. During each time epoch E_i, k_s servers out of a total of n_s servers are pseudo-randomly selected to be active, and the remaining servers are idle and act as honeypots. To increase unpredictability, epoch lengths are computed pseudo-randomly as well. For each time epoch E_i, the algorithm uses hash chains to select

Table 5: Notation of the Roaming Honeypots Algorithm

Symbol	Description
S	Set of servers
n_s	$\lvert S\rvert$
E_i	Time epoch i
r_i	Length of E_i in time slots
$m, m+u$	Lower and upper bounds on r_i
H_i	Set of idle servers (honeypots) during E_i
A_i	Set of active servers in E_i
k_s	$\lvert A_i\rvert$
$\mathcal{P}_{k_s}(S)$	Set of all subsets of S of size k_s
$n_p = \binom{n_s}{k_s}$	$\lvert\mathcal{P}_{k_s}(S)\rvert$
$\mathrm{MSB}_x(y)$	The x most-significant bits of y
G, G', G''	One-way hash functions

the set of honeypots, H_i, the set of active servers, A_i, and epoch length, r_i. A hash chain is a sequence of pseudo-random numbers, whereby the first number is randomly selected, and each subsequent number is generated from the previous one using a one-way hash function, such as SHA-2 [45]. In the algorithm, a long hash chain is generated and used in reverse, that is, the last number used first, similar to PayWord [121] and TESLA [113].

On the server-side of the algorithm, the first number in the hash chain (last number to be used), K_ℓ, is randomly generated and used to compute a set of *roaming keys*, $K_i = G^{\ell-i}(K_\ell), (i < \ell)$, where $G^{\ell-i}$ is the application of the public one-way hash function G $(\ell - i)$ times. Then, the set of active servers, A_i, is selected as the subset with index $\mathrm{MSB}_{\lfloor \lg n_p \rfloor}(G'(K_i))$ in $\mathcal{P}_{k_s}(S)$, where $\mathcal{P}_{k_s}(S)$ is the set of all subsets of size k_s of servers,

```
input : S, the server pool; k_s, number of active servers
input : Server ID, s
input : One-way hash functions, G, G', G''
input : A lower bound, m, and upper bound, m + u, on epoch length
input : Check-pointing period, T_checkpoint
Compute P_{k_s}(S), set of all subsets of S with size k_s;
Generate a one-way hash chain, K_ℓ, K_{ℓ-1}, ···, K_1 using the one-way hash function G ;
for i = 1, 2, 3, ··· do
    r_i ← m + MSB_{⌊lg u⌋}(G''(K_i));
    A_i ← P_{k_s}(S)[MSB_{⌊lg n_p⌋}(G'(K_i))];
    (active servers) H_i ← S − A_i;
    (idle servers (honeypots)) while in time epoch E_i do
        if s ∈ H_i then
            foreach received request Req_att do
                Insert into Blacklist the source address of Req_att;
                Send to all servers the source address of Req_att;
                Service Req_att;
        else
            foreach received request Req_received do
                if source of Req_received ∈ Blacklist then
                    Drop Req_received;
                else
                    Service Req_received;
            Every T_checkpoint seconds do
                foreach active request Req_active do
                    Encapsulate request state in state-update message;
                    Send state-update message to the client of Req_active;
        foreach attacker source address received from other servers do
            Insert the address into Blacklist ;
            if s ∈ A_i then purge all active requests from the received address;
```

Algorithm 1: Roaming Honeypots Algorithm at Servers

input : S, the server pool; k_s, number of active servers
input : Service expiration epoch, t
input : One-way hash functions, $\mathtt{G}, \mathtt{G}', \mathtt{G}''$
input : K_t, the key assigned to the client from the server hash chain
input : Start time of epoch $p < t$, $start_p$
input : A lower bound, m, and upper bound, $m + u$, on epoch length.

Compute $\mathcal{P}_{k_s}(S)$, set of all subsets of S with size k_s;
Generate the hash chain, $K_t, K_{t-1}, \cdots, K_1$ using the assigned key K_t and the one-way hash function $\mathtt{G}$;
if *client has no active requests* **then**
  if *new request* Req_{new} *to be sent* **then**
    $\mathsf{CurrentEpoch} \leftarrow arg\ min_e\ \{start_p + \sum_{i=p}^{e}(m + \mathrm{MSB}_{\lfloor \lg u \rfloor}(\mathtt{G}''(K_i))) > current\_time\}$;
    if $\mathsf{CurrentEpoch} > t$ **then**
      Service subscription expired;
    else
      $e \leftarrow \mathsf{CurrentEpoch}$;
      $r_e \leftarrow m + \mathrm{MSB}_{\lfloor \lg u \rfloor}(\mathtt{G}''(K_e))$;
      $A_e \leftarrow \mathcal{P}_{k_s}(S)[\mathrm{MSB}_{\lfloor \lg n_p \rfloor}(\mathtt{G}'(K_e))]$;
      $H_e \leftarrow S - A_e$;
      Send Req_{new} to a randomly-selected server in A_e;
      $\mathsf{RequestStateTable}\ [Req_{new}] \leftarrow Req_{new}$;
while *client has active requests* **do**
  for $i = \mathsf{CurrentEpoch}, \mathsf{CurrentEpoch} + 1, \cdots, t$ **do**
    $r_i \leftarrow m + \mathrm{MSB}_{\lfloor \lg u \rfloor}(\mathtt{G}''(K_i))$;
    $A_i \leftarrow \mathcal{P}_{k_s}(S)[\mathrm{MSB}_{\lfloor \lg n_p \rfloor}(\mathtt{G}'(K_i))]$;
    $H_i \leftarrow S - A_i$;
    foreach *active request* Req_{active} *with a server* $s \in H_i$ **do**
      Close Req_{active};
      Send $\mathsf{RequestStateTable}\ [Req_{active}]$ to a randomly-selected server in A_i;
    while *in time epoch* E_i **do**
      foreach *new request* Req_{new} **do**
        Send Req_{new} to a randomly-selected server in A_i;
        $\mathsf{RequestStateTable}\ [Req_{new}] \leftarrow Req_{new}$;
      foreach *received* state-update *message for an active request* Req_{active} **do**
        $\mathsf{RequestStateTable}\ [Req_{active}] \leftarrow$ received message;

Algorithm 2: Roaming Honeypots Algorithm at Clients

n_p the cardinality of this set, that is, $n_p = |\mathcal{P}_{k_s}(S)| = \binom{n_s}{k_s}$, G' another public one-way hash function, and $\text{MSB}_x(y)$ the x most significant bits of y. The epoch length is selected uniformly in the interval $[m, m+u]$ as follows: $r_i = m + \text{MSB}_{\lfloor \lg u \rfloor}(G''(K_i))$, where G'' is also a one-way hash function, and m and $m+u$ are lower and upper bounds on the epoch length. These steps correspond to lines 1–6 in Algorithm 1.

On the client-side, clients send their requests only to active servers. To keep track of active servers, each client is given a roaming key, K_t, from the hash chain and the start time, $start_p$, of an epoch $p < t$. When the client wants to send a request, it first determines the current epoch as the minimum e that satisfies $(start_p + \sum_{i=p}^{e} r_i) > current\_time$[1], where each r_i is computed using K_i, which is computed in turn from K_t. The client then uses K_e to compute H_e, A_e, and r_e for the current epoch as described in the server-side algorithm. These steps correspond to lines 1–19 in Algorithm 2.

In a complex distributed system whereby clients vary in their trust level, it may be desired that a more trusted client controls and delegates service access of less trusted clients. This trust hierarchy can be easily enforced through the distribution of roaming keys. As can be noted from the previous discussion, the index of the roaming key assigned to each client (i.e., t in K_t) determines the client's *subscription duration*, because it represents the last epoch up to which the client can correctly calculate the set of active servers and epoch lengths. To implement a trust hierarchy, a more trusted client gets a roaming key with a large key index (say K_{large}), effectively granting it a long subscription duration. The trusted client can then grant access to clients down in the trust hierarchy simply by sending them roaming keys from its chain ($K_1 \cdots K_{big}$), effectively assigning them shorter durations. The assignment of trust levels is beyond the scope of this work.

6.1.3 Request migration and filtering

When a server switches from active to idle (honeypot), requests from compliant clients are migrated and resumed on other active servers. The states (e.g., TCP and application states)

[1]The running time of this procedure is very small compared to the epoch length, so it is possible to safely assume that the procedure is instantaneous.

of these requests are migrated from the old to the new server as well. Although more costly, periodic state updates are preferred over lazy, on-demand updates (refer to §2.2.1 for more details). This is because of two reasons. First, when the end of the epoch arrives, there is no guarantee that the old server is alive and available to process state transfer (the server may be under attack). Second, servers may not know which of their requests are from legitimate clients, and hence may end up migrating state of requests from non-compliant attackers. This allows the servers switching from idle to active to start with empty service queues, creating time-windows for compliant requests to get some service before queue is filled up with attack requests. Per-request state is pushed to clients using periodic (every $T_{checkpoint}$ seconds) *state-update* messages (lines 19–22 in Algorithm 1 and lines 27–28 in Algorithm 2), and clients in turn send the state to the new servers (lines 20–22 in Algorithm 2).

Request filtering. Because clients (except non-compliant attackers) direct their requests only to active servers, a request received by a honeypot is considered malicious. The requester's IP address is added into a black-list (lines 8–12 in Algorithm 1) made available to all servers so that no more requests are accepted from that source (lines 14–16 and lines 23–25 in Algorithm 1).

As common with many distributed systems, loose clock synchronization between clients and servers is assumed [92]. To accommodate this loose clock synchronization and to ensure proper handling of in-transit compliant requests, switching from idle to active and vice versa is not done instantly but over a transitional period instead. In other words, each server switching from idle to active starts accepting requests a little earlier than scheduled. Also, each server switching from active to idle, starts assuming the role of a honeypot a little later than scheduled to accommodate in-transit requests as well. During the transitional period, all requests are serviced (except from previously blacklisted attackers).

6.1.4 Roaming Honeypots Evaluation

I have studied the performance and overhead of the roaming honeypots algorithm using both a prototype implementation on a FreeBSD cluster and NS-2 [5] simulation[2]. The purpose

[2]The simulation study was conducted mainly by Chatree Sangpachatanaruk.

of the study is to determine the efficacy of roaming honeypots in identifying non-compliant attackers and mitigating attacks. The following describes the main results.

6.1.4.1 Results from Prototype System The first set of results is from a prototype of a roaming file-transfer service that has been developed and tested in a FreeBSD network. Fig. 32 depicts the topology used in the experiment reported below. Six PII machines running FreeBSD 4.5 were used, and two of them were configured as routers. The server machines (S_1 and S_2) ran a simplified version of Algorithm 1 with fixed-length epochs and without request filtering. When an active server receives a roaming trigger, it drops all its connections. A server does not accept incoming connections unless it is the current active server. On the client machine C, client processes were launched, each running a simplified version of Algorithm 2 also with fixed-length epochs. Each client process holds the number of bytes received so far, and upon receiving a roaming trigger, each client drops current connections and establishes new ones with the new server to resume the transfer.

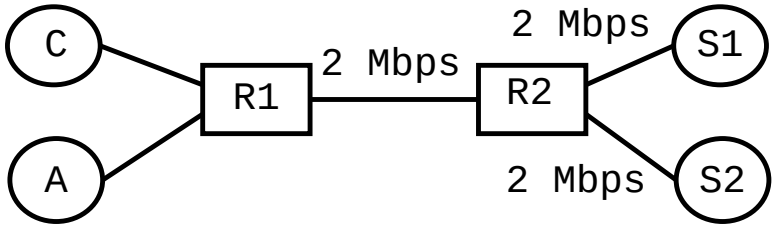


Figure 32: Network topology of an experiment using a roaming honeypots prototype.

The experiment compared plain replication, whereby request load is balanced over two servers, with roaming honeypots with one active server and without request filtering. The active server roamed every 60 seconds between two servers. The goal was to assess the level of mitigation achieved by the algorithm and to show that roaming honeypots achieves better attack mitigation than plain replication even without the request filtering component. The performance metric was the average client response time, or the average time needed to download a file. The overhead at peace (no-attack) time was also studied. Machine C issued 20 file requests with exponential inter-arrival times with a mean of 8 seconds. Each request was for a file of size 1 MBytes, taking about 4 seconds in an unloaded network (the bottleneck

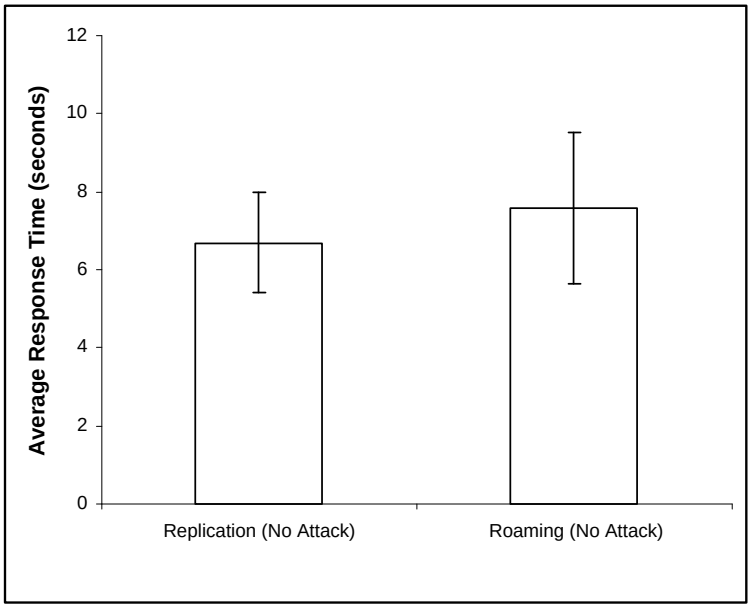


Figure 33: Average response time for replication and roaming without attack.

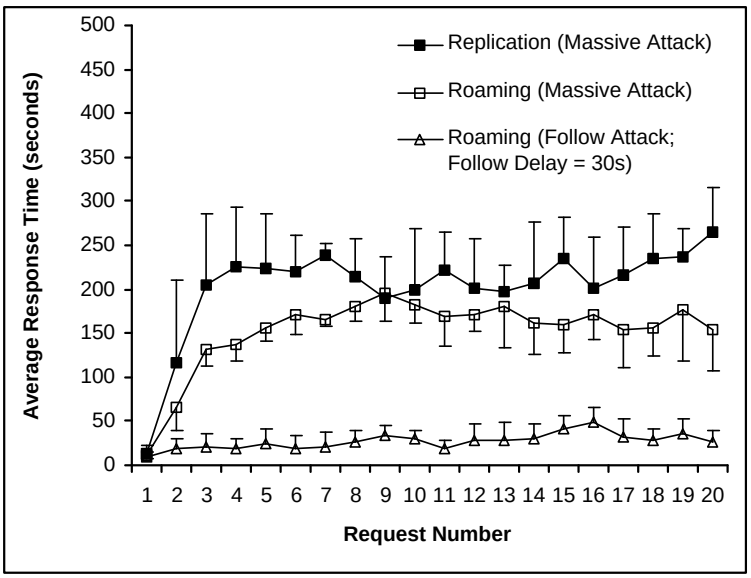


Figure 34: Average response time of individual requests against massive and follow attacks.

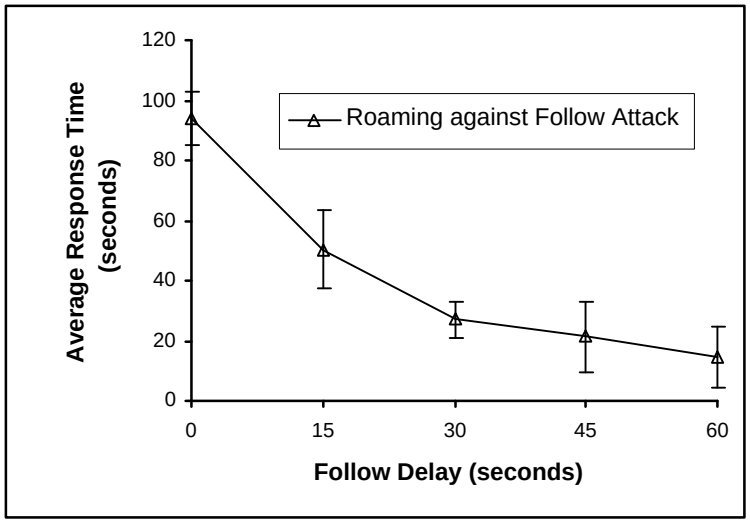


Figure 35: Effect of follow delay on average response time for roaming against follow attack.

is the 2Mbps link between routers R1 and R2). Thus, the client load was about 50%. Machine A, the attacker, issued file requests with exponential inter-arrival times with a mean of 1 second for a duration of 5 minutes. The attacker launched massive and follow attacks (see Chapter 3) with the follow delay varied from 0 to 60 seconds. Because the active server changed every 60 seconds, the 60-second follow delay resulted in that the attacker always attacked the wrong server, whereas the zero follow delay represented a compliant attacker (the attacker always attacked the active server).

Under no attack, roaming caused an increase of about 14% in average response time over replication (Fig. 33). In Fig. 34, the average response time of individual requests is plotted for replication and roaming against massive attack and for roaming against follow attack. It shows that roaming managed to lower the average response time under attack, as compared to plain replication. However, the massive attack caused more damaged than the follow attack. Under follow attack (Fig. 35), roaming achieved an average response time that ranged from about 50% (at zero follow delay) to about 10% (at 60-second follow delay) of the average response time with replication under massive attack. That is, roaming outperformed plain replication even with zero follow delay. Error bars in the figures represent

90% confidence intervals.

Attack mitigation results from the fact that attack requests have a delay with each roaming event until they congested the new server's request-processing queue. During these opportunity time-windows, compliant client requests got a higher share of the server compared to load-balanced connections, which were stuck at congested servers. The length of these time windows increases with increasing follow delay.

6.1.4.2 Simulation Results A simulation study using the NS-2 [5] simulator has been conducted to assess the performance and overhead of the roaming honeypots algorithm [77, 78]. Three schemes were compared: Roaming Honeypots (RH), Server Roaming (SR), and plain load-balancing (Replication). The SR scheme is the same as RH but without attacker identification, and as such it shows the mitigation effectiveness of roaming honeypots. In what follows, epoch length is fixed, and two honeypots ($n_s - k_s = 2$) out of five ($n_s = 5$) total servers are designated at each epoch.

Fig. 36 depicts the simulated network topology, which is generated using the GT-ITM topology generator [27] and is composed of five servers, each on a separate stub network, thirty-four legitimate clients, and ten attackers. The authenticator is responsible for sending roaming keys to clients. The links are assigned the following capacities: 1 Mb/s for intra-stub links (e.g., links incident on each server) and 10 Mb/s for all inter-stub links. The 1 and 10 Mb/s link speeds were chosen to model the relative speed of access and backbone links.

In all experiments, clients and attackers request files of size 1 Mbits each. For each request, a server is picked uniformly at random out of the five servers to receive the request. The request inter-arrival times are exponentially distributed. Both client and attack loads are measured in requested bits per second. For example, a load of 2.4 Mbps corresponds to requesting 2.4 files per second on average. Attackers launch a massive attack (see §3.1) by flooding all the five servers.

Effect of Attack Load. Fig. 37 shows that RH kept the Average client Response Time (ART) stable with increasing Attack Loads (AL) for a fixed number of attack nodes (ten in this experiment). This is because once the attack nodes were detected, their requests were

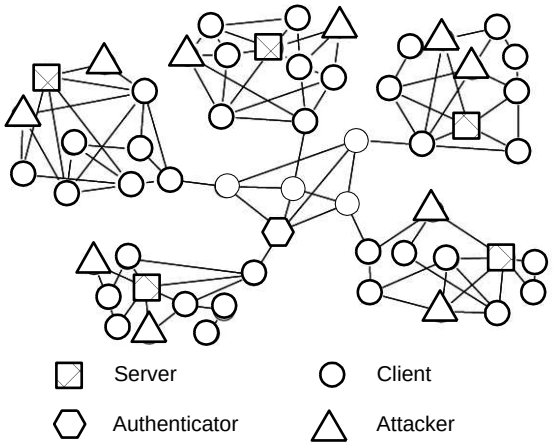


Figure 36: Simulation topology for the roaming honeypots algorithm.

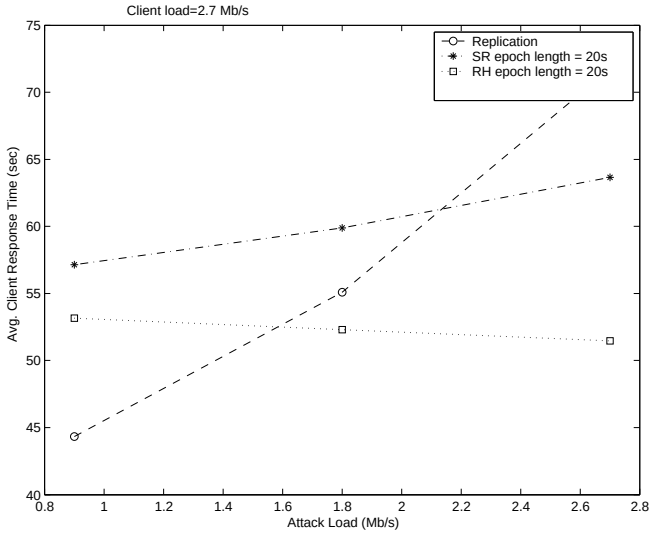


Figure 37: Effect of attack load on average response time of the roaming honeypots algorithm; two honeypots out of five servers and ten attackers.

filtered out, and the attack had no impact. The ART even slightly decreased with increasing attack load because of faster attacker identification: the higher the load (with fixed number

of attackers), the faster attack requests are issued and received by idle servers (honeypots), which add the request sources to the blacklist. For Replication, distributing the load on all the servers helped in the case of low attack loads, but ART steeply increased with increasing attack load. For SR, ART increased with increasing attack load, because the length of the opportunity windows that roaming created decreased with increasing attack load.

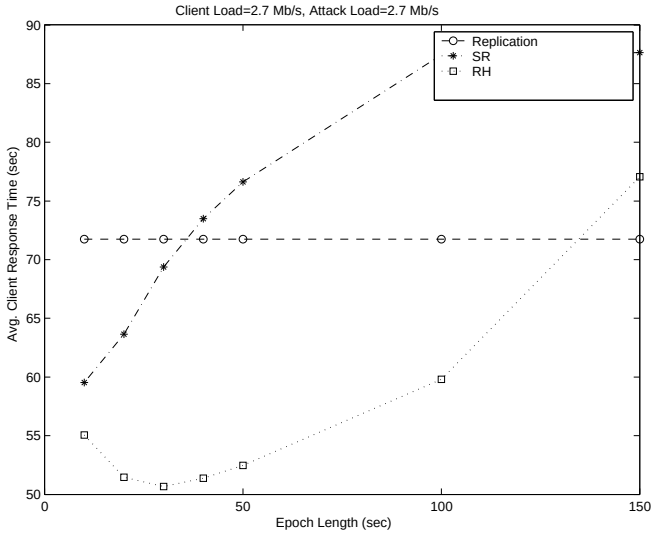


Figure 38: Effect of epoch length on average response time of the roaming honeypots algorithm; two honeypots out of five servers and ten attackers.

Effect of Epoch Length. As Fig. 38 shows, there is a critical value of epoch length that struck a good balance between roaming benefit and overhead (30 seconds for RH, at AL=2.7 Mb/s). For epoch lengths below the critical value, roaming overhead was dominant; as the epoch length increased while still below the critical value, the frequency of connection re-establishment and slow start decreased, resulting in a decreasing ART. As the epoch length increased beyond the critical value, the ART increased. The reason is two-fold: First, as roaming happened less frequently, less opportunity windows were opened. Second, the larger the epoch length, the more client requests that were received during each epoch and the more client requests that migrated into new servers, diluting the effect of opportunity windows among them. Finally, the optimal value of epoch length depends on a combination

of parameters, such as attack load, client load, and average service time.

6.1.5 Conclusions

Honeypots, proposed elsewhere, are deployed to trap attackers. However, honeypots can be avoided by DoS attackers because of their fixed and detectable locations, and because they are typically deployed on machines with no production value. Roaming honeypots is a novel algorithm for mitigating DoS attacks and identifying the attackers, whereby honeypots are disguised within the protected server pool. In a sense, the roaming honeypots algorithm integrates plain service replication and honeypots in a defense mechanism that is more effective than each of its components alone. At any point of time, a subset of servers is active and provides service while the rest are acting as honeypots to capture attack requests, record and distribute their source addresses, and filter attack requests issued from these sources. A distributed, randomized algorithm has been developed for changing the active servers and allowing only compliant clients to follow them.

Through NS-2 simulations and a prototype implementation, the effectiveness and feasibility of roaming honeypots against service-level DoS attacks have been demonstrated. As compared to plain replication, roaming honeypots shows a performance gain under the more realistic scenarios of high attack loads. In another piece of work, I have extended roaming honeypots to handle attacks with spoofed source addresses [71, 72].

A mechanism that adaptively changes the number of active servers depending on attack and client loads is a fertile subject of future work. Also mechanisms to smooth out client request arrivals on current active servers to reduce connection migrating and request flocking into new servers are worth further investigation.

6.2 IDENTIFICATION OF COMPLIANT DOS ATTACKERS

The second scheme that achieves attacker identification within the dodging framework is referred to as live baiting, which accurately and efficiently identifies service-level DoS at-

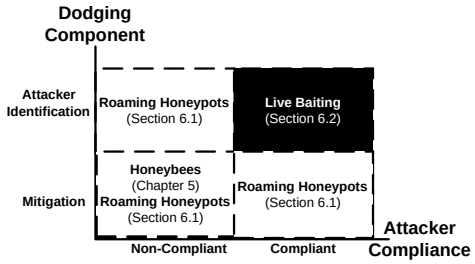


Figure 39: The part of the DoS problem space addressed by the live baiting algorithm. Live baiting provides identification of compliant attackers.

tackers among clients of a public service. Live baiting divides the service capacity into virtual servers, called *buckets*, whereby each virtual server is isolated and individually monitored. The algorithm employs group testing (refer to §2.3 for more details) to carefully design client assignment to buckets to enable efficient pinpointing of DoS attackers with low false positive and almost zero false negative probabilities.

As illustrated in Fig. 39, the live baiting algorithm achieves identification of compliant attackers. The live baiting algorithm defines the mapping function μ between service users and buckets, such that the set of active buckets continuously changes in a pseudo-random way, and users are assigned to buckets in such a way to facilitate attacker identification. The mapping function in live baiting satisfies the properties of the compliant identification component (§4.3)[3]: (**non-blocking**) all client requests are always sent to active buckets, (**uniform**) clients distribute their request load evenly over the active buckets, (**dynamic**) the set of active buckets changes from epoch to epoch, (**loose**) there are always some idle buckets at each epoch, (**unpredictable**) the set of active buckets is hard to predict by non-compliant attackers, and (*d*-**unique**) the group-testing theory is used to achieve the d-uniqueness property as will be described shortly.

The live baiting algorithm provides an approximation of the function Δ presented in

[3]Against service-level attacks, the live baiting algorithm can be used instead of the roaming honeypots algorithm (§6.1); live baiting identifies both compliant and non-compliant attackers and mitigates their attacks. However, against attacks that bypass the "bucket-layer" and attack the physical servers directly (e.g., link-level packet-flooding attacks studied in [72]), roaming honeypots provides protection that is not available using live baiting alone.

Definition 4.1. In live baiting, $\Delta(\sigma, \tau)$ returns *under attack* if resource σ is active (i.e., $\sigma \in R_A^\tau$) and its incoming request load exceeds its request-processing capacity at τ[4]. Otherwise, $\Delta(\sigma, \tau)$ returns *not attacked*.

6.2.1 Live Baiting Overview

As mentioned earlier, the service capacity is divided into a number of virtual servers, or buckets. Each client receives from the service a set of tokens that are valid to access only a set of buckets assigned to the client. The client then sends its requests to the buckets assigned to it. To prevent attackers from accessing buckets other than those assigned to them, the tokens are securely generated and checked at the servers so that forged tokens/requests are dropped.

The theory of group testing is used to decide the set of buckets assigned to each client with the goal of identifying the DoS attackers amongst the service clients. The following illustrates the mapping from the group-testing theory domain to the DoS-attacker detection domain.

Service clients correspond to the population and DoS attackers are the defective members that we seek to detect. As described in §2.3, a group testing design is represented as a binary matrix with rows corresponding to tests and columns to population members. Buckets are mapped to the tests or the matrix rows, and clients are mapped to matrix columns. Each client is given tokens for each *1-bit* in its column, allowing it to send requests to the corresponding buckets. Clients piggyback a token on each request they send, so that requests are credited to the corresponding buckets.

Periodically, the number of requests sent to each bucket is counted. If that number exceeds a High Water Mark (HWM), the test result corresponding to the overflowed bucket is labeled as positive, meaning that the clients assigned to the overflowed bucket contain at least one DoS attacker. Another alternative is to count the number of requests *dropped* in each bucket. This alternative is further explored in §6.2.5.2. The detection algorithm (in §2.3) is then used to identify the attackers.

[4]This information is only available at the boundaries of the detection periods (see §6.2.1).

Table 6: Live Baiting Terminology

Symbol	Description
FP	false positive probability
FN	false negative probability
n_t	number of buckets
HWM	high water mark of buckets
P	length of detection interval
d	initial estimate of the number of attackers
n_x	actual number of attackers
$B_{attackers}$	number of buckets attacked

6.2.2 Live Baiting at Servers and Clients

The building blocks of the live baiting algorithm at the servers and clients are described.

Buckets and Tokens. The algorithm virtually divides the service capacity, ρ, into n_t buckets, where each bucket is implemented as a virtual server (e.g., a sub-queue in a fair queuing system [128]). Given an upper bound, n_c, on the total number of clients and an initial estimate of the number of attackers, d, a $T \times N$ binary matrix is constructed in which an entry is 1 with probability $\frac{1}{d+1}$ and 0 otherwise. The matrix depth, T, is determined based on the required false positive probability according to Eqn. 2.2.

Each client is assigned one column in the matrix and is given one token for each *1-bit* in its column. A token is a tuple $\langle i, version, hash(i, version, K_{server})\rangle$ that contains the row index i, the matrix version (to allow for matrix updates as will be described in §6.2.3), and a secure hash of length at least 160-bits computed using a secure hash function, such as SHA-2 [45]. The input to the hash function is the row number, the matrix version, and a server secret, K_{server}, of length at least 80-bits, which is changed periodically (e.g., every day) with the matrix version incremented correspondingly.

```
   input   : An upper bound, n_c, on total number of clients
   input   : An initial estimate, d, of number of attackers
   input   : The required false positive probability, FP
   input   : The server secret, K_server
   input   : Detection period, P
 1 Compute the number of tests, n_t, needed to achieve FP using Eqn. 2.2 with p = 1/(d+1);
 2 Generate a n_t × n_c binary matrix, M, with Pr{M_ij = 1} = 1/(d+1);
 3 Compute the High Water Mark for each bucket, HWM, using Eqn. 6.1.
 4 foreach matrix row i from 1 to n_t do
 5     BucketCounter [i] ← 0;
 6 foreach matrix column j from 1 to N do
 7     ClientAddress [j] ← nil;
 8 MatrixVersion ← 0;
 9 foreach token-request message from client address c do
10     if c ∈ ClientAddress then
11         Find j such that ClientAddress [j]= c;
12     else
13         Find the first j s.t. ClientAddress [j]= nil;
14         ClientAddress [j] ← c;
15     foreach 1-bit M_ij in column j do
16         Token ← ⟨i, MatrixVersion,SecureHash(i, MatrixVersion, K_server) ⟩;
17         Send Token to client;
18 foreach service-request from client c with token ⟨i, version, hash_i⟩ do
19     if c ∈ Block-list then
20         Drop request;
21         Continue;
22     if version ≠ MatrixVersion then
23         Send a token-request message on behalf of client c;
24         Continue;
25     if SecureHash(i, version, K_server) ≠ hash_i then
26         Insert c into Block-list ;
27         Continue;
28     BucketCounter [i]++;
29     if BucketCounter [i] > HWM then
30         Service the request with low priority;
31     else
32         Service the request with high priority;
33 Every P seconds do
34     Set SuspectList ← all clients in ClientAddress ;
35     foreach row i s.t. BucketCounter [i] ≤ HWM do
36         foreach column j s.t. M_ij = 1 do
37             Remove ClientAddress [j] from SuspectList ;
38     Insert SuspectList into Block-list ;
39     foreach matrix row i from 1 to n_t do
40         BucketCounter [i] ← 0;
```

Algorithm 3: Live Baiting at Servers

The secure token generation described above prevents attackers from generating valid tokens on their own. If they were to achieve that, they would have been able to attack buckets other than the ones assigned to them, and the detection scheme would have failed to detect them. The probability of successfully guessing a valid token value is $\frac{n_t}{2^{80}}$, where n_t is the number of valid tokens, and 80 is the server secret length. This probability is very low; even in a matrix designed to identify one million attackers, the number of buckets is about 20 million $\simeq 2^{25}$, yielding a successful bucket guessing probability of $\frac{1}{2^{55}} \simeq 10^{-16}$.

Although tokens are available to all clients that request them (including attackers), each client is limited to a single column in the matrix. Enforcing this limit can be done using Bloom filters as in the Kill-Bots system [64] and the sample-and-hold algorithm [44]. Note that these algorithms cannot be readily used to enforce usage limits in a general service, which requires more complex models of legitimate or anomalous behavior [120].

High Water Mark (HWM). To decide whether or not a bucket is under attack, a threshold on the number of received requests is used. This threshold is called the High Water Mark (HWM) of a bucket. A bucket is assumed under attack if the number of received requests exceeds its HWM. The HWM value is set to the number of requests each bucket is expected to receive in a detection interval of length P seconds, assuming that all its clients are legitimate. Each legitimate client is expected to send $r \cdot P$ requests every P seconds, distributed uniformly over its assigned buckets (on average, $\frac{n_t}{d+1}$ buckets per client). Also, clients are distributed uniformly over the buckets; each bucket has on average $\frac{n_c}{d+1}$ clients assigned to it. Thus, buckets are expected to receive the same number of requests every P seconds. This number can be computed as follows: $\frac{n_c}{d+1} \cdot r \cdot P \cdot \frac{d+1}{n_t} = \frac{n_c \cdot r \cdot P}{n_t} = \frac{\rho \cdot P}{n_t}$ requests. Thus,

$$HWM = \frac{\rho \cdot P}{n_t} \tag{6.1}$$

For each bucket, only HWM requests out of the received requests are serviced at a high priority, and the rest are serviced at a lower priority. Low priority requests are dropped first when the servers are overloaded. Note that, in the worst case, all the high-priority HWM requests are attack requests.

Detection. Every P seconds, the following detection algorithm is run. If a bucket has

received more than HWM requests in the previous P seconds, the corresponding test is marked as positive. The detection algorithm starts with all clients in a *suspects list*, and for each negative test, the clients assigned to the corresponding bucket are removed from the list. The order in which the buckets (tests) are checked does not make a difference in the algorithm outcome. According to the attack model, if these clients were attackers, they would have caused *all* their assigned buckets to exceed their HWM (in §6.2.4 this assumption is relaxed). The rest of the suspects list are labeled as attackers and are added to a *block-list* to drop their future requests.

Note that an orthogonal mechanism is needed to remove clients from the suspects list due to false positives (if any) or due to node recovery from intrusion. However, such a mechanism is orthogonal to this work and is not investigated further.

Request Filtering. When a request arrives at a server with a token $\langle i, version, hash_i \rangle$, the server drops the request if it is from an attacker already detected (in the block-list). The server also checks whether the token has expired by checking its version against the current matrix version. If the token has expired, the server drops the request and sends to the client a new set of tokens. Otherwise, if the token is current, the server checks its validity by comparing the secure hash of $(i, version, K_{server})$ with the received $hash_i$. If they differ, the client sending the token is inserted into the block-list. Clearly, checking the token validity is a quick operation and can be made even quicker by reducing the size of the secret key, but with a higher probability of illegal token generation. Request filtering at the servers is depicted in Fig. 40.

Mapping Buckets into Physical Servers. Each physical server maintains counters for all the buckets, and it increments the corresponding counters for each request it processes. These local counters contain partial counts because requests of a particular bucket may be serviced by more than one server. Each server securely sends its bucket counters to a central detection server at the ISP, which in turn runs the detection algorithm and updates the block-list. Requests are mapped to the physical servers using a variety of mapping functions, such as load-balancing and session-based routing (requests in the same session are routed to the same server to maintain session integrity). Live baiting is orthogonal to the request-mapping strategy, because its operation relies only on maintaining the bucket counters. Live baiting

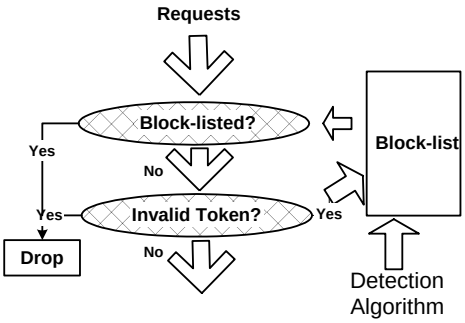


Figure 40: Request Filtering at the Servers in the Live Baiting Algorithm.

is also independent of the number of servers.

Live Baiting at Clients. Each client receives a set of tokens and piggybacks one token on each request it sends to the server. The client uses its tokens in a round-robin fashion. Attackers also use this algorithm to overflow all their assigned buckets; in §6.2.4, other attack strategies are considered.

6.2.3 Adaptive Live Baiting

Live baiting relies on an estimate, d, of the number of attackers, which may be different than the actual number of attackers, n_x. Overestimation yields better FP but causes unnecessarily high storage and computational overhead (see §6.2.5.1) when the actual attacker number is much less than estimated. On the other hand, an underestimate causes FP to increase beyond its target value. The live baiting algorithm can adapt to a number of attackers different than the estimated one by detecting the discrepancy and changing its parameters accordingly as follows.

When $n_x \neq d$, the false positive probability from Eqn. 2.1 on Page 27 becomes:

$$FP = [1 - \frac{1}{d+1}(1 - \frac{1}{d+1})^{n_x}]^{n_t} \tag{6.2}$$

where n_t is the number of buckets. Eqn. 6.2 is plotted in Fig. 41(a), with the number of buckets set using Eqn. 2.2 so that $FP = 10^{-3}$. The actual number of attackers, n_x,

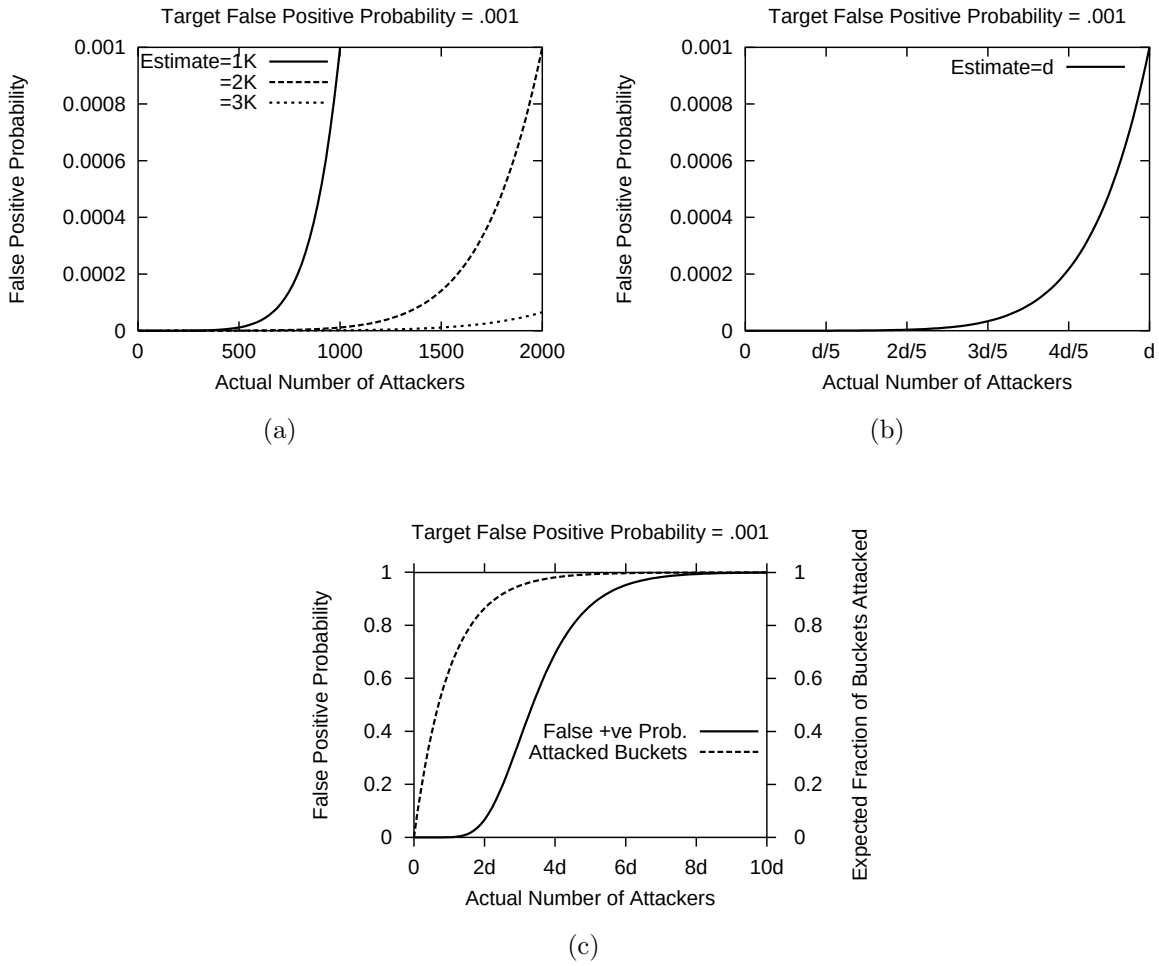


Figure 41: Effect of underestimating and overestimating the number of attackers in the live baiting algorithm. (a) As the estimate of number of attackers increases, false positive probability decreases. (b) An overestimate of the number of attackers improves the false positive probability. However, an unnecessarily large estimate results in unnecessarily high overhead. (c) An underestimate of the number of attackers is detected when a high percentage of buckets get attacked. d is the estimate of the # attackers used in generating the matrix.

is varied for different values for the estimate d. For a fixed actual number of attackers, corresponding to a vertical line in the figure, the false positive probability decreases as the

estimate increases. This trend can also be seen in Fig. 41(b), which plots Eqn. 6.2 for values of n_x less than d. On the other hand, Fig. 41(c) (left y-axis) plots the same equation with values of n_x greater than d, in which case the target FP of 10^{-3} is no longer maintained.

Detection of inaccurate estimate. An important observation is that the number of attacked buckets reflects the discrepancy between actual and estimated number of attackers. Indeed, the average number of buckets assigned to n_x attackers is:

$$\begin{aligned} B_{attackers} &= n_t(1-(1-\frac{1}{d+1})^{n_x}) \\ &\simeq n_t(1-\frac{1}{e}) \text{ as } d = n_x \to \infty \\ &\simeq 0.63 n_t \end{aligned} \tag{6.3}$$

In the above equation $(1-\frac{1}{d+1})^{n_x}$ is the probability that a particular bucket has no attackers assigned to it. Thus, $1-(1-\frac{1}{d+1})^{n_x}$ is the fraction of buckets to which at least one attacker is assigned, and thus, the bucket is "acquired" by attackers.

The mechanism that live baiting uses to adaptively set the estimate d as close to the actual number of attackers n_x as possible starts with an initial value for d. When an attack is detected, the system calculates an estimate of the actual number of attackers from the observed percentage of attacked buckets (tests with positive result), $B_{attackers}$. Fortunately, $B_{attackers}$ can be used to detect the case $n_x > d$. To illustrate, Eqn. 6.3 is used to plot the expected percentage of attacked buckets (right y-axis in Fig. 41(c)) against the actual number of attackers. As derived in the equation, when $n_x = d$, this percentage is 63%.

However, as n_x increases beyond d, the fraction of attacked buckets increases. This is because now $\frac{1}{d+1}$ is constant relative to n_x, the number $1-\frac{1}{d+1}$ is < 1, and as n_x increases, $(1-\frac{1}{d+1})^{n_x}$ decreases and $1-(1-\frac{1}{d+1})^{n_x}$ increases.

This increase triggers the system to increase the estimate of the number of attackers to $\hat{d}$:

$$\hat{d} = \frac{log(1-\frac{B_{attackers}}{n_t})}{log(1-\frac{1}{d+1})}, B_{attackers} < n_t \tag{6.4}$$

where $B_{attackers}$ is the observed number of attacked buckets. As shown in Fig. 41(c), $B_{attackers} = n_t$ when $n_x \geq 5d$. Thus, $\hat{d}$ is set to $5d$ when all buckets are attacked. The above formula is derived from Eqn. 6.3 by taking the logarithm of both sides and solving for n_x.

Matrix regeneration. Once a new estimate, $\hat{d}$, is calculated, if it is larger or much smaller ($d - \hat{d}$ >threshold[5]) than the current estimate (d), a new matrix is generated with the new estimate, and the number of buckets is updated according to Eqn. 2.2 to maintain the target FP. The *matrix version* is incremented every time the matrix is regenerated. Tokens with versions older than the current version are dropped, and clients are assigned new tokens as described in §6.2.2.

Extension to the general group-testing theory. This adaptive mechanism is readily useful in the general group-testing domain, particularly when an estimate of the number of defective members is not known. This can be illustrated by the following example. Consider a population with $10,000$ defective members. One can start with a *probing* set of tests to discover the number of defective members. Let's start with a matrix that is designed based on an estimate of $1,000$ defective members. Using Eqn. 2.2 with a target false positive probability of 0.001, the number of tests needed for probing is $20,000$ roughly. From Eqn. 6.3, all the tests will yield a negative result. This can be also observed in Fig. 41(c) at $x = 10d$. Another set of probing tests is then conducted, but with an estimate of $5,000$ defective members. The number of tests in this second set is roughly $100,000$, and about 87% of the tests produce negative results. Using Eqn. 6.4 yields the actual number of defective members.

6.2.4 Extensions to the Simple Service and Attack Models

In this subsection, extensions to the live baiting algorithm are presented to handle multiple service classes and more intelligent attackers.

Handling Different Types of Services. In the previous discussion, the focus was on a server providing one service class. The system capacity in handling this service, ρ, is

[5]Setting the matrix regeneration threshold is a trade-off between state overhead and matrix regeneration overhead. Investigation of this trade-off is subject of future work.

divided into virtual buckets which were assigned to clients based on the group testing theory. The simple model of one service can be easily generalized to the case when the system is supporting different types of services (e.g., HTML, CGI, video download, etc.) each with a different service time. For each class i of service provided, the service has a certain aggregate request-processing capacity, ρ_i requests per second. Each ρ_i is then divided into virtual buckets and these are assigned to clients based on the client-token assignment matrix for group testing.

In other words, to generalize the case of one service to multiple service classes, simply repeat the same procedure for each service class. This can be simply implemented as follows. (1) Use the live baiting algorithm (see §6.2.2) to assign to each client a list of tokens, the same list of tokens that would have been assigned to that client for a single-class service. (2) When a request arrives at the server, based on the service type, the *color* of the token piggybacked with the request is determined (i.e., the server colors the request on the fly). (3) Enqueue the request to the right virtual bucket based on the token and the token color. By implementing this coloring scheme, attacks on the different service classes are isolated.

Relaxing the Maximum-Damage Attack Assumption. In the previous discussion, a massive attack was assumed, that is, attackers attack **all** buckets assigned to them. Although this attack scheme causes the *maximum damage* to the service (i.e., maximum number of dropped requests), a more intelligent attacker might choose to attack only a subset of the assigned buckets. In this case the attacker will be cleared (removed from the list of suspects) for some of the buckets, and hence, evades detection (recall that the detection algorithm removes a client from the suspects list when at least one bucket assigned to the client is not attacked). To model this intelligent attack, assume that each attacker attacks each assigned bucket with an *attack probability* (ρ_{attack}). The false negative probability is no longer zero in this case:

$$FN = 1 - [1 - \frac{1}{d+1}(1 - \rho_{attack})(1 - \frac{1}{d+1} \cdot \rho_{attack})^{d-1}]^{n_t}$$

In the above equation, $\frac{1}{d+1}(1.0 - \rho_{attack})(1 - \frac{1}{d+1} \cdot \rho_{attack})^{d-1}$ is the probability that a particular bucket is assigned to a particular attacker and ends up attack free. Note that using the above equation, $FN = 0$ when $\rho_{attack} = 1$ as expected. On the other hand, the

number of attacked buckets, and hence, the damage caused by the attack, is reduced to:

$$B_{attackers} = n_t(1 - (1 - \frac{1}{d+1} \cdot \rho_{attack})^d) \tag{6.5}$$

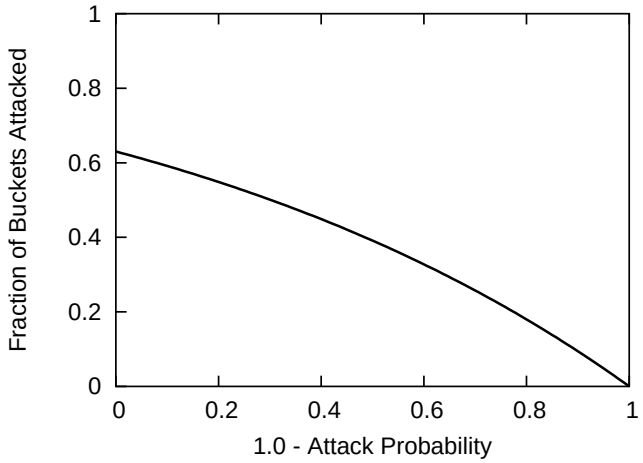
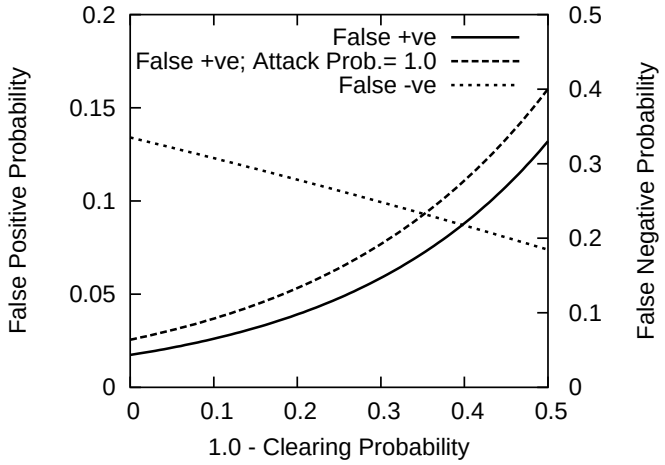


Figure 42: Effect of attack probability (ρ_{attack}) on the damage caused by the attack in live baiting. $d = 100$.

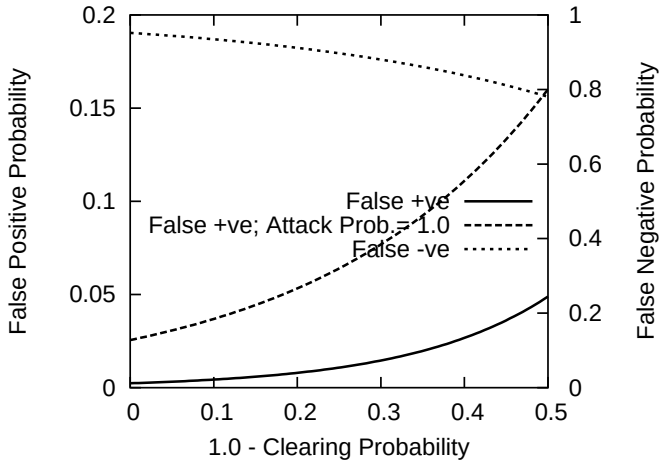
Fig. 42 plots the fraction of attacked buckets ($\frac{B_{attackers}}{n_t}$) at $d = 100$. It shows that this fraction decreases, and hence, the amount of damage caused by the attack is reduced, with decreasing attack probability.

This attack is feasible because attackers control which buckets receive their requests by attaching particular tokens to requests. If the assignment of requests to buckets is controlled solely by the servers, this attack is not possible. However, removing the client control requires the servers to look up the matrix for each request to assign it to one of the requesting client's buckets. Although the matrix size may be manageable, it is still proportional to the number of clients. Investigation of how to remove client control over bucket assignment is a subject of future research.

Meanwhile, live baiting deals with this attack by clearing the client with a *clearing probability* (ρ_{algo}) for each of the client's assigned buckets that is found to be not attacked. In other words, the algorithm excludes a client from the suspects list if "enough" buckets assigned to the client are not attacked, instead of just one bucket. The false positive (FP)



(a) Attack Prob. = 0.9



(b) Attack Prob. = 0.5

Figure 43: Effect of clearing probability (ρ_{algo}) on the false positive and false negative probabilities in live baiting. $d = 100$.

and false negative (FN) probabilities in this case are as follows.

$$FP = [1 - \frac{1}{d+1}(1 - \frac{1}{d+1} \cdot \rho_{attack})^d \cdot \rho_{algo}]^{n_t} \tag{6.6}$$

$$FN = 1.0 - [1 - \frac{1}{d+1}(1.0 - \rho_{attack}) \cdot (1 - \frac{1}{d+1} \cdot \rho_{attack})^{d-1} \cdot \rho_{algo}]^{n_t} \quad (6.7)$$

The algorithm sets the clearing probability to a fixed value that achieves acceptable false positive and false negative probabilities assuming a lower bound on ρ_{attack}. The lower bound on ρ_{attack} is determined based on the amount of damage (number of attacked buckets) that can be mitigated. Fig. 43 (note the two y-axes) shows the effect of changing ρ_{algo} when $\rho_{attack} = 0.9$ and 0.5, respectively. For comparison, the false positive rate at attack probability of 1.0 is also plotted.

6.2.5 Live Baiting Evaluation

Theoretical and simulation analyses have been conducted to evaluate three aspects of live baiting: (a) coverage; (b) effectiveness: detection time, false positive and false negative probabilities; and (c) efficiency: memory, message, and computational complexities. The theoretical results were compared with NS-2 simulations that are based on real Web traces (details in §6.2.5.2).

6.2.5.1 Theoretical Evaluation *Summary of Theoretical Results.* A summary of the theoretical results is first presented followed by the detailed analysis. First, the number of buckets needed to achieve a specific false positive probability is linear in the number of attackers. Second, the length of the detection interval is also linear in the number of attackers. Third, the memory overhead of the matrix is $O(n_c)$. However, the matrix is accessed only every P seconds and not for each service request. Fourth, per-request processing has negligible ($O(log(d))$) overhead, and the detection algorithm is $O(n_c)$ but is invoked only every P seconds. Finally, the message overhead caused by the tokens is small, and tokens can be piggybacked on service requests, such as HTTP GET.

Effectiveness Analysis: *False Positive and False Negative Probabilities.* As discussed in §2.3, the detection algorithm used in live baiting has a theoretical zero false negative

probability. Its false positive probability is represented by Eqn. 2.1. The algorithm is scalable in terms of the required number of tests or buckets, n_t, as modeled by Eqn. 2.2. Fig. 44 shows that the relationship between n_t and d, the number of attackers, is linear, and the slope increases with decreasing target false positive probabilities. For instance, the number of buckets needed to detect a million attackers with 10^{-4} false positive probability is about 25 million, which can be implemented using less than 10 MB of memory.

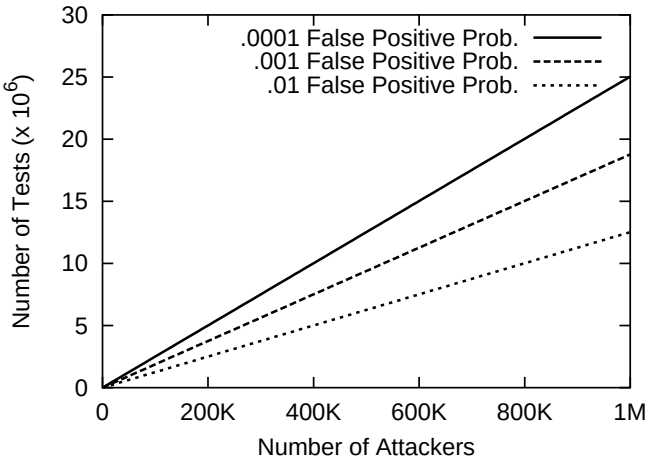


Figure 44: Scalability of live baiting to number of clients. The number of buckets required to get a target false positive probability is linear in the number of attackers and independent of the number of clients.

However, in reality, extra false positives and false negatives may occur due to burstiness in request arrivals and irregularities in service time distributions. This effect is studied in the simulation analysis in the next subsection.

Detection Time. The detection time is the length of the detection period P plus the running time of the detection algorithm. The detection time is set so that the HWM is a positive integer. That is,

$$HWM \in \{1, 2, \cdots\}$$

Or,

$$\frac{\rho P}{n_t} \in \{1, 2, \cdots\}$$

Equivalently,

$$P \in \{\frac{n_t}{\rho}, 2\frac{n_t}{\rho}, \cdots\} \tag{6.8}$$

Because n_t is linear in the number of attackers, d, as shown in Fig. 44, and ρ is constant with respect to the number of attackers, then P, the detection time, is linear in the number of attackers.

Efficiency Analysis: *Storage overhead.* The storage overhead of the matrix is $O(n_t \cdot n_c) = O(d \cdot n_c)$, because n_t is linear in d. However, this overhead can be greatly reduced by storing only the *1-bits* of this sparse matrix. The number of *1-bits* is $O(\frac{dn_c}{d+1}) = O(n_c)$. The servers also keep n_t bucket counters each with size $O(log(HWM))$. Because $HWM = \frac{\rho P}{n_t}$ is independent of the number of clients and attackers, the storage overhead of the bucket counters is $O(n_t) = O(d)$.

Computational overhead. The server looks up the source address of each incoming request in the block-list of size $O(d)$. The lookup is thus an $O(log(d))$ operation. The server also applies a light-weight hash operation to check the validity of the request's token and increments the corresponding bucket counter. Both are $O(1)$ operations. The detection algorithm, which runs every P seconds, has a running time $O(n_c)$, because it has to check each *1-bit* in the matrix to determine which clients to clear for each negative test. This relatively high computation time can be mitigated by running the detection algorithm only when the number of attacked buckets exceed a threshold, flagging a strong attack.

Message overhead. For a large matrix with millions of rows, the size of each token is 24 bytes, 4 bytes for row index and 20 bytes for the hash. This small-sized token can easily be piggybacked in a service request, such as the HTTP GET message. Thus, no new messages are introduced, and the extra overhead is only in message size. This overhead is $O(log(\text{number of matrix rows})) = O(log(d))$.

Coverage Analysis. Two conditions are necessary to achieve zero FN and the FP of Eqn. 2.1 using the group-testing detection algorithm described in §2.3. These conditions are:

C1. For all buckets to which zero attackers are assigned, the bucket counter every detection

period is less than or equal to the HWM, that is, the corresponding test result is negative.

C2. For all buckets to which at least one attacker is assigned, the bucket counter every detection period is more than the HWM, that is, the corresponding test result is positive.

C1 holds with high probability, noting that the number of clients assigned to each bucket is a Binomial random variable, and, thus, its values are around the mean with high probability [97]. Since the HWM is set according to the mean number of clients assigned to each bucket, each bucket receives $\leq HWM$ requests every detection period with high probability, if no attackers are assigned to the bucket.

To cause denial of service each attacker has to send requests at a rate higher than a legitimate client's, that is, $r_{attacker} > r$. Because the HWM is set assuming only legitimate clients (sending at rate r) are assigned to buckets, and because each attacker distributes its attack rate over its assigned buckets equally, each bucket that has an attacker assigned to it will receive more requests than its HWM. As a result, condition C2 holds as well.

6.2.5.2 Simulation Results The purpose of the simulation study is to examine the extent to which the above theoretical results apply in more complex system and attack scenarios. In particular, the effect of burstiness in request arrivals and irregularities in request processing times of real services is studied. The simulation results described below show that live baiting is indeed effective under real service models.

To this end, a simulation model of live baiting was developed in NS-2 [5]. The model extends the PackMime HTTP traffic generation module [28], which generates HTTP 1.1 sessions with session inter-arrival times, client waiting times, request sizes, and response sizes derived from empirical traces. The PackMime module was modified to include separate client and attacker distributions.

Instead of generating server delays independently from system load, the server CPU was modeled as a link, called "CPU-link", with a bandwidth equivalent to the service capacity. The link is attached to a round-robin queue with sub-queues corresponding to the buckets. The queue buffer is shared fairly among the buckets, so that each bucket has a guaranteed share equal to total buffer size divided by number of buckets. Servicing a request is modeled by transmitting a packet in the CPU-link with size equal to the response size. When the

packet is received at the end of the CPU-link, a packet is sent with 0 delay back to the server, who immediately sends the response to the client. If a packet is dropped from the CPU-link queue, the client of the dropped request is notified, and in turn attempts a maximum of three retransmissions with one second delay between attempts.

Attack sessions have similar request size, response size, and client waiting time distributions as legitimate sessions. The detection algorithm was slightly modified to count request drops (more specifically, packet drops in the CPU-link's buffer) in each bucket instead of the number of requests. The number of request drops was found to be a better metric than request count in this setting, because it is a more direct indicator of server overload; using the number of requests would result in more false positives due to request burstiness of legitimate clients. A sensitivity analysis of the effect of HWM was conducted, and a good performance was reached with a HWM value of 10 dropped requests.

To evaluate live baiting, six metrics were used: average response time of legitimate requests, throughput of legitimate requests (legitimate requests completed per second) and legitimate sessions (legitimate sessions completed per second), false positive rate, false negative rate, and legitimate utilization; the latter is defined as the fraction of time the server's CPU was processing legitimate requests. Two performance baselines were considered, namely *no attack* and *no defense*.

A total of $10,000$ legitimate clients was simulated with aggregate rate of 20 sessions per second. The CPU-link was 100Kbps, zero propagation delay, and 100Kbit total buffer size in one direction. The reverse direction was 100Mbps with zero propagation delay, provisioned to introduce negligible delays and no drops. The simulation topology consisted of a client node modeling a "client cloud" and a server node, connected by a 100Mbps link with 10ms propagation delay. The simulation time of each run was $1,000$ seconds, the attack started at 250s, and ended at 750s. The warm-up period was 100 seconds, after which statistics were collected. The detection intervals started after the attack started, and the request-drop counters were reset every detection interval.

Fig. 45 illustrates the effectiveness of live baiting in detecting attackers and reducing the attack effect on response time. In this figure, requests were grouped every 10 seconds based on their start time, and the average response time of each group was plotted. The

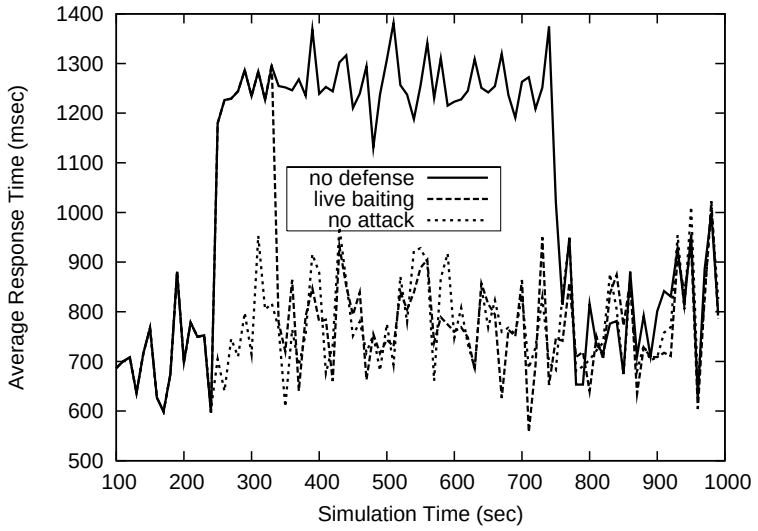


Figure 45: Average response time for no attack, no defense, and live baiting vs. simulation time. Attack time = [250s, 750s]; detection interval = 90s.

aggregate attack rate was 100 sessions per second, number of buckets $n_t = 1,000$, number of attackers $n_x = 100$, and the estimate $d = 100$ as well. The detection interval was 90 seconds. With no defense, the response time almost doubled after the attack started. With live baiting, response time spiked after attack started, but quickly (at 340s, after the first detection interval) went down, after the attackers were detected and added to the block-list. After the attack, the response time with live baiting was slightly less than with no attack, because a few legitimate clients were blocked (false positives), and consequently, the load on the system was slightly reduced.

The first experiment explores the effect of the detection interval length (P) on false positives and false negatives. The detection interval length was varied from 10s to 200s. Error bars represent 90% confidence intervals.

As Fig. 46(top) shows, the false positive rate increased with increasing values of P. This effect is due to burstiness in request arrivals and irregularities in request processing times, which caused some legitimate clients to accumulate drops in their buckets. Increasing the detection interval allowed some of these accumulated drops to exceed the HWM. This

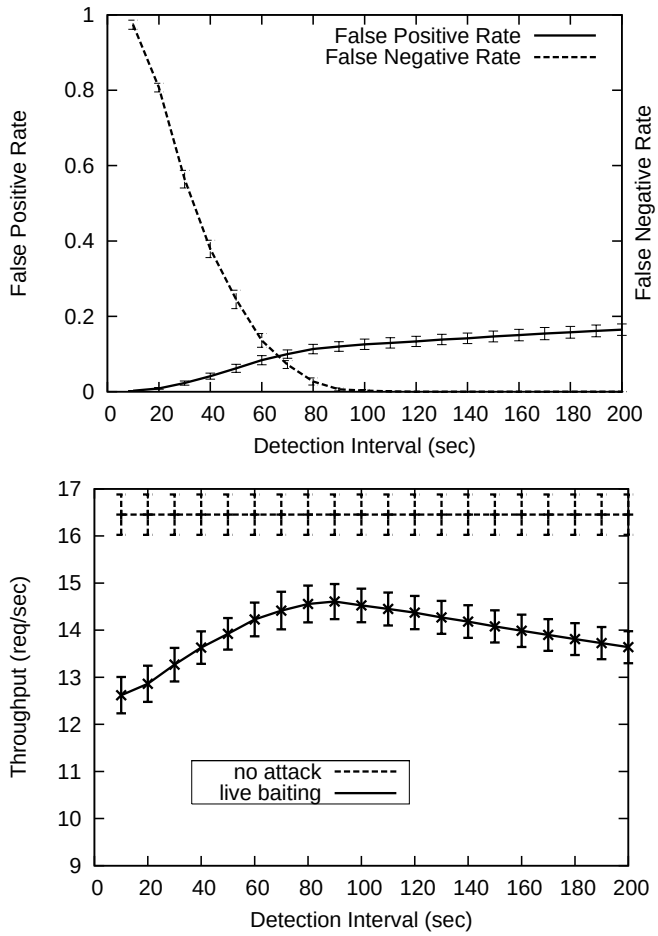


Figure 46: Effect of detection interval length (P) on the performance of live baiting.

effect of P on false positives did not show up in the theoretical results, which ignored request burstiness of legitimate clients.

The same figure also shows that the false negative rate decreased and reached almost zero at $P = 90$s, because increasing P gave more time for each attacker to overflow its assigned buckets and get detected.

Fig. 46(bottom) shows that the throughput peaked at $P = 90$s at a value of about 88% of the throughput under no attack. For detection interval lengths beyond this point, the increase in false positives caused throughput to decrease.

The second experiment studied the false positive and false negative rates. First, to

validate Eqn. 2.1, the number of attackers was increased with accurate estimation, that is, $d = n_x$.

As Fig. 47 shows, live baiting with 90s detection interval and $1,000$ buckets was successful in detecting up to 50 attackers with a false positive probability of less than 0.005 and almost zero false negatives. Simulation results matched Eqn. 2.1 when the detection interval was 30 seconds (Fig. 47 (top)). However, 90s interval introduced more false positives than predicted by the model. This difference was due to bucket overflows resulting from bursty legitimate request arrivals. Although the model correctly predicted the trend of the false positive probability increase with increasing d and fixed number of buckets, it needs further enhancement to take the effect of the detection interval into consideration.

The false negative rate with P = 90s is almost zero (Fig. 47 (middle)), because 90 seconds is enough time for each attacker in this scenario to overflow all its assigned buckets. On the other hand, the false negative rate with 30s interval was close to one and decreased with increasing number of attackers. Although the rate per individual attacker decreased, resulting in more time needed for the attacker to overflow its buckets, the number of these buckets ($= \frac{n_t=1000}{d+1}$) decreased as well.

The effect of the number of attackers on the legitimate utilization depended on the detection interval length, as shown in Fig. 47 (bottom). When the interval was short, the decrease in false negatives and the small increase in false positives had two effects: the legitimate utilization was much smaller than at large intervals, and the utilization slightly increased. Conversely, when the interval was large, legitimate utilization slightly decreased due to the increase in false positive rate while the false negative rate was constant.

6.2.5.3 Analysis of Adaptive Live Baiting To analyze the adaptive live baiting algorithm, a simulation experiment was conducted where the number of attackers varied while fixing the estimate $d = 100$. The purpose is to study the effect of underestimating and overestimating the number of attackers and to validate Eqn. 6.2 and Eqn. 6.3. As Fig. 48 (top) shows, the false positive rate increased with increasing number of attackers. The 90s setting has more false positives than predicted by Eqn. 6.2 because of the burstiness reason previously mentioned. Moreover, the false negative rate was almost zero until the number

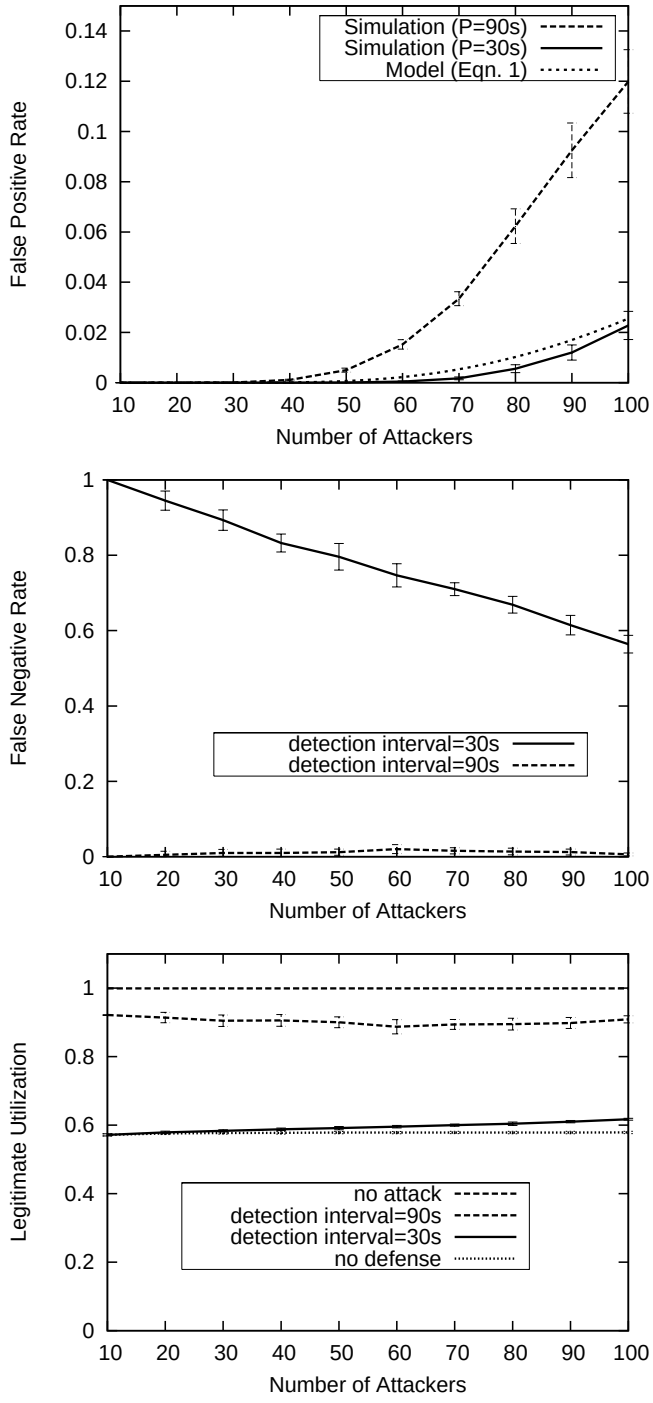


Figure 47: Effect of number of attackers on live baiting performance with accurate estimation (i.e., assuming $d = n_x$).

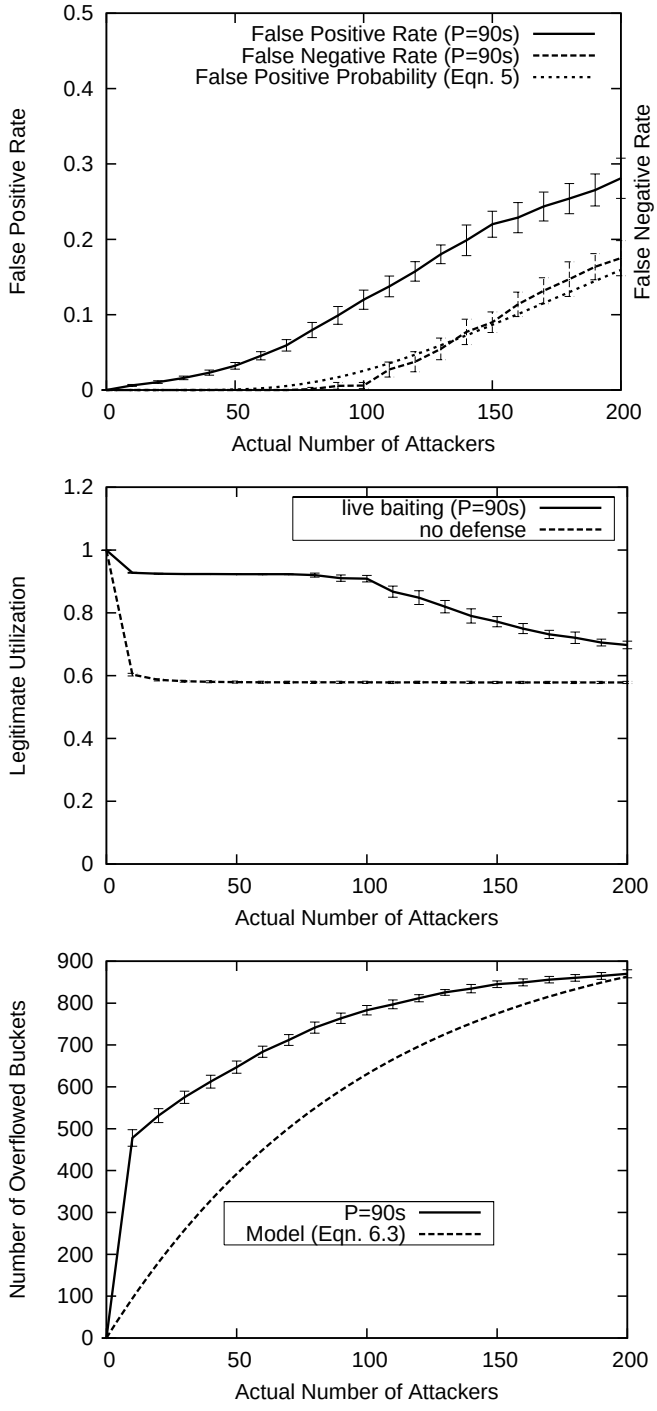


Figure 48: Effect of under-estimation and over-estimation of the number of attackers in the live baiting algorithm. The number of buckets under attack can be used to detect both under-estimation and over-estimation situations.

of attackers exceeded the estimate (Fig. 48 (top)). The false negative rate then increased because the rate per attacker decreased (fixed aggregate attack rate) and number of buckets per attacker remained fixed (because d was fixed), so the time that each attacker needed to overflow its buckets exceeded the 90s detection interval. The legitimate utilization (Fig. 48 (middle)) slightly dropped to about 0.93 until number of attackers exceeded the estimate, then dropped steeply due to increasing false negatives and approached the no defense baseline.

The maximum number of attacked buckets (buckets with more drops than HWM) was measured over all detection intervals within each run. This number is plotted in Fig. 48 (bottom) together with Eqn. 6.3, which models the number of attacked buckets. With 90s detection interval and $HWM = 10$, the number of attacked buckets exceeded the model estimation. Thus, using Eqn. 6.3 to estimate the number of attackers would yield a larger number than the actual number of attackers. For example, at 630 attacked buckets, the model estimates the number of attackers as about 100, but the actual number of attackers would be about 60 attackers (this can be shown by drawing a horizontal line at $y = 630$ in Fig. 48 (bottom)).

Effectively, using Eqn. 6.3 would make the system over-estimate the number of attackers, which would cause slightly more state but results in fewer false positives. For instance, at an estimate of $d = 100$, the actual number of attackers would be about 60 (this can also be shown by drawing a horizontal line at $y = 630$ in Fig. 48 (bottom) intersecting the two curves at around $n_x = 60$ and $n_x = 100$, respectively). The false positive rate from Fig. 48 (top) would be 0.04 (at $n_x = 60$) instead of 0.12 if the estimate was accurate ($n_x = d = 100$).

6.2.6 Conclusions

Live baiting is a novel algorithm that efficiently identifies service-level DoS attackers. It is based on a novel application of group testing theory. Live baiting is scalable and can efficiently and quickly identify DoS attackers in a large service with millions of clients even when attackers send service requests that are indistinguishable from legitimate requests. Theoretical results using simple service and attack models confirm the algorithm effectiveness, in

terms of false positive and false negative probabilities, and efficiency, in terms of memory, message, and computational complexity. Moreover, adaptive live baiting contributes to the general group-testing theory domain the mechanism that discovers the actual number of attackers starting from an initial estimate. The effectiveness of live baiting to more complex service and attack models was validated using NS-2 simulations, whereby techniques to extend live baiting to these models were presented.

The investigation of how to tune the algorithm parameters, such as the detection interval, the high water mark, and the clearing probability, is left for future work. Although live baiting was presented in the context of attacks against servers (service-level attacks), it can be applied to protecting network-level resources as well. The investigation of this application is a subject of future work as well.

7.0 CONCLUSIONS

The last decade has witnessed growing interaction between physical and cyber worlds. Of the main thrusts behind the cyber-physical convergence is the pervasiveness of computer networks, which range from small-scale networks of home appliances to large-scale "eco-systems", such as the Internet and large-scale sensor networks that monitor physical infrastructure systems. Although computer networking has enabled a huge multitude of services with many potentially useful services yet to be unleashed, full-scale adoption of these services highly depends on the trustworthiness of the networking infrastructure. To be trusted, a computer network has to be available, dependable, reliable, privacy-aware, and usable. Many factors have contributed to a shaky image of network availability and reliability. For instance, a denial-of-service (DoS) attack in 2002 caused a major Internet "blackout". In the wireless arena, radio jamming, malicious or benign, poses a major hurdle to the deployment of mission- and life-critical wireless services.

The work in this dissertation aims at making today's computer networks more trustworthy by putting forward the following thesis: *a hybrid of the three DoS protection approaches, namely prevention, mitigation, and detection-and-recovery, provides better protection than the individual approaches.* The thesis is supported by presenting the *dodging* framework, which relies on dynamic, unpredictable reconfigurations of the client-resource mapping to allow a service under DoS attack to provide service to legitimate clients despite the attack and to create hard-to-evade baits, or traps, to identify the attackers.

The contributions of this work can be summarized as follow.

- In Chapter 4, the dodging framework is formalized and its two components, namely attack mitigation and attacker identification, are defined in terms of the mapping function

between resources and clients. Each of the two components relies on a set of properties of the mapping function. These properties are formally defined, and their impact on achieving the dodging components is described.

- §4.3 presents the Primary-Effect-based (PED) detection problem, which represents a departure from traditional attacker identification approaches. In PED, DoS attackers are identified based merely on the unavailability of the attacked resources, without the need to develop models of legitimate behavior, keep track of attack signatures, or tightly specify legitimate behavior.
- §5.1 presents an analytical study of the blocking probability for different attack and defense combinations in single-radio networks. Closed-form equations are derived and used to decide the best defense strategy (reactive or proactive) given the attack strategy (scanning or sweeping) and values of system parameters.
- In §5.1, the jamming problem in multi-radio networks is formulated as two max-min games (the availability game and the efficiency games) between defense and attack, and through simulation the Nash equilibria of both games are determined.
- §5.2 formulates the problem of maximizing goodput under jamming in multi-radio networks using a combination of channel-hopping and error-correcting codes (ECC). It also presents Markov-Chain-based models of the reactive channel-hopping against the scanning attack.
- §5.2 also introduces the honeybees adaptive algorithm, which achieves energy-efficient and bandwidth-efficient mitigation of jamming in multi-radio networks. The algorithm used the Markov models to detect the number of jammers and adjust the ECC parameters accordingly.
- §6.1 presents the roaming honeypots algorithm, which significantly improves the usage of honeypots to identify service-level DoS attackers. The algorithm is evaluated via simulation and experiments using a prototype.
- §6.2 presents the live baiting algorithm, which identifies service-level DoS attackers and scales to large services with millions of clients. Live baiting requires low state overhead, enabled by a novel leverage of the group-testing theory.

- §6.2 also presents a novel mechanism to detect the actual number of attackers and adaptively adjust defense parameters, which is a contribution to group-testing theory.

7.1 GAINED INSIGHTS

In pursuing the work of this dissertation, I have gained a number of insights about DoS defense and network security in general.

Predict future threats. Security is often described as an arms race between defense and attack. New threats emerge as a response to deployed defense systems. To maintain an edge, security research has to always anticipate future attacks and prepare effective defenses. Along this line, in [77] I predicted the impact of **service-level** DoS attacks, or what is now known as HTTP CyberSlam attacks, which emerged as a response to wide deployment of ingress filtering and source-end monitoring of one-way network flows. Service-level DoS is particularly challenging because it is easy to launch, hard to stop by current network-level defense approaches, and overwhelming even for massively-resourced services and networks. The roaming honeypots and live baiting defense systems provide efficient, accurate, and fast identification and filtering of service-level DoS attackers.

New threats also emerge by exploiting new technologies that oversight security in their design process. In [75], I was one of the first researchers to point to the vulnerability of sensor networks and even the recent 802.11a and 802.11g wifi standards to radio jamming. The jamming-tolerant frequency-hopping physical layer has been replaced by cheaper and broader bandwidth physical layers that are vulnerable to jamming. Later papers have confirmed this vulnerability and devised defense schemes in single-radio networks [54, 100, 153, 155]. I have proposed and evaluated software-based channel-hopping at the link layer coupled with error-correcting coding over multiple radios.

Use simple detection signals. Many state-of-the-art defense systems go a long way in tailoring very effective defense techniques against specific attack tools and vectors. These solutions are unfortunately temporary and simply result in attackers shifting gears into different tactics. On the contrary, my defense systems assume the least about how attackers

perpetrate and execute their attacks. In both the roaming honeypots and live baiting systems, attacker identification and filtering relies merely on the fact that some resources (e.g., server queues) are overloaded by the attack. Attackers will have a hard time evading this detection method; without resource overloading, there is actually no attack!

Borrow ideas from relevant fields. The network security field has its own characteristics, but it can benefit from other fields. For instance, the problem of identifying "defective" members within a large population with the minimum number of tests has been extensively studied in combinatorics, particularly in the **group-testing** theory. I have adopted techniques from group-testing to design the live baiting defense system. The borrowed ideas enable live baiting to identify—with low memory overhead—DoS attackers within the clients of a large service. Moreover, the concept of a "test tube" has lead to the use of virtualization to create isolated, individually monitored virtual servers, whereby attacker identification relies only on overload events in these virtual servers. **Game theory** has been used to understand and analyze anarchy situations. I have used game theory to analyze jamming and defense strategies and determine the most reasonable combinations. To this end, I have defined utility functions for both defense and attack and modeled jamming as a max-min game.

Resource awareness. The defense systems that I have developed make efficient utilization of resources, so as not to become attack targets themselves. For instance, as previously mentioned, live baiting uses group testing to minimize the amount of state used to identify DoS attackers. The algorithm utilizes state that grows with the number of attackers, as compared to state-of-the-art systems that need to keep state in the order of the number of users. Also, in the honeybees defense system, which defends wireless networks against jamming, energy efficiency is used as a utility function in the game-theoretic analysis. One thing to note is that efficiency does not imply using few resources. Indeed, using a single radio interface to defend against jamming is sub-optimal when considering either communication availability or energy efficiency as performance metrics.

7.2 FORESEEN IMPACT AND FUTURE WORK

The dodging strategy is general and can be applied to both wired and wireless networks. This work is thus poised as a starting point for a wider deployment of defense techniques that use dodging as the underlying concept.

This work also introduced a new DoS detection paradigm, namely the Primary-Effect-based Detection (PED), for identifying DoS attackers among service users. This paradigm would invite new research in defining the detection function for other DoS attack instances, such as those based on exploiting software and protocol bugs, and for the detection of attackers in other security problems. The fact that the new detection approach does not require thorough monitoring of individual client behavior makes it suitable in scenarios in which client privacy is of particular importance, such as when the DoS defense is done by an entity other than the protected service (e.g, the Internet Service Provider). This approach would pioneer the concept of privacy preservation in on-line DoS defense systems.

Scalable identification of DoS attackers. With expected growth of network services, client populations will keep on increasing in size and diversity. Additionally, with increasing awareness and deployment of virus and worm defense mechanisms, the number of attack hosts will decrease. Guided by these factors, I project that one of the main challenges in DoS defense will be the scalable identification of attackers among service clients. I plan to extend my live baiting defense, which addresses the scalability issue by using monitoring state that depends on the number of attackers not the total number of clients. I will implement live baiting as a service-independent middle-box that sits in front of the load balancer in a server cluster. I will also investigate a methodology to monitor the virtual servers across server boundaries. A technique based on distributed resource containers [17] is a good candidate for this task.

Secure wireless communication in emerging wireless technologies. In my research to defend against radio jamming, I have identified a number of research challenges in applying channel-hopping to multi-radio wireless networks. I will investigate techniques to assess the jamming threat and adaptively adjust data redundancy level and channel-hopping strategy and frequency based on the threat level. I will also investigate the application of

the channel-hopping defense in the newly-emerging cognitive radios [58], whereby channel bandwidth and radio capabilities can be heterogeneous. Finally, I will investigate techniques to coordinate channel-hopping across multiple network hops.

Virus and worm detection. I will apply the group-testing approach to detecting viruses and worms. Current worm-detection techniques incur high overhead in monitoring processes and threads to identify anomalous behavior. The premise of group testing, as demonstrated in my live baiting algorithm, is to minimize the monitoring overhead by combining tested elements into isolated "test tubes". I will investigate techniques to employ the virtual machine technology to test processes and threads. The main challenge will be how to maintain program integrity across virtual machines.

Privacy-preserving DoS protection. A perfectly-secured network, if such a thing really exists, will not be trusted if user privacy is not maintained. A basic requirement for a DoS defense system to preserve privacy is to avoid monitoring of individual client traffic. In my DoS defense systems, client traffic is not tapped into unless it is known to be attack traffic (except for a few false positives). Moreover, because attacker identification depends on signals over aggregate traffic, the defense system can be *outsourced* to specialized defense entities. The design and research challenges to achieve DoS defense outsourcing are in my long-term research plans.

DoS on the telephone network. The public telephone network has not suffered from DoS attacks due in part to the difficulty and high-cost of deploying automated, remotely-controlled dialers. I project that this situation is going to change soon with the wide deployment of VoIP clients. Being a piece of software, a VoIP client is vulnerable to compromise and, thus, can act as a "zombie" to launch DoS attacks on the telephone network. This threat is aggravated by the fact that many embedded systems used to monitor and control critical infrastructure facilities are networked and controlled through dial-in interfaces.

BIBLIOGRAPHY

[1] Akamai Corporation. http://www.akamai.com.

[2] Quad-radio wi-fi access point. www.hopling.nl.

[3] Sensor board with four attached radio boards. www.atific.fi.

[4] Snort. http://www.snort.com.

[5] The Network Simulator - ns-2. http://www.isi.edu/nsnam/ns/.

[6] *The 2003 International Conference on Dependable Systems and Networks (DSN'03)*, San Francisco, California, October 2003.

[7] PacketScore: A Statistics-Based Packet Filtering Scheme against Distributed Denial-of-Service Attacks. *IEEE Trans. Dependable Secur. Comput.*, 3(2):141–155, 2006. Yoohwan Kim and Wing Cheong Lau and Mooi Choo Chuah and H. Jonathan Chao.

[8] Daniel Adkins, Karthik Lakshminarayanan, Adrian Perrig, and Ion Stoica. Taming IP Packet Flooding Attacks. In *Proceedings of Workshop on Hot Topics in Networks (HotNets-II)*, November 2003.

[9] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A Survey on Sensor Networks. *IEEE Communications Magazine*, 40(8):102–116, August 2002.

[10] Ghada Alnifie and Robert Simon. A multi-channel defense against jamming attacks in wireless sensor networks. In *Q2SWinet '07: Proceedings of the 3rd ACM workshop on QoS and security for wireless and mobile networks*, pages 95–104, New York, NY, USA, 2007. ACM.

[11] K. G. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A. D. Keromytis. Detecting targeted attacks using shadow honeypots. In *Proceedings of the 14th USENIX Security Symposium*, 2005.

[12] David G. Andersen. Mayday: Distributed Filtering for Internet Services. In *4th Usenix Symposium on Internet Technologies and Systems*, Seattle, WA, March 2003.

[13] Katerina Argyraki and David Cheriton. Network capabilities: The good, the bad and the ugly. In *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets-IV)*, 2005.

[14] P.E. Ayres, H. Sun, H.J. Chao, and W.C. Lau. ALPi: A DDoS Defense System for High-Speed Networks. *IEEE Journal on Selected Areas in Communications*, 24(10):1864–1876, Oct 2006.

[15] Paramvir Bahl, Atul Adya, Jitendra Padhye, and Alec Walman. Reconsidering wireless systems with multiple radios. *SIGCOMM Comput. Commun. Rev.*, 34(5):39–46, 2004.

[16] Paramvir Bahl, Ranveer Chandra, and John Dunagan. SSCH: slotted seeded channel hopping for capacity improvement in IEEE 802.11 ad-hoc wireless networks. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 216–230, New York, NY, USA, 2004. ACM.

[17] Gaurav Banga, Peter Druschel, and Jeffrey C. Mogul. Resource containers: a new facility for resource management in server systems. In *OSDI '99: Proceedings of the third symposium on Operating systems design and implementation*, pages 45–58, Berkeley, CA, USA, 1999. USENIX Association.

[18] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, 2003.

[19] Steven M. Bellovin, Marcus Leech, and Tom Taylor. ICMP Traceback Messages. In *draft-ietf-itrace-01.txt, internet-draft, October 2001. Expired draft.*

[20] Robert Beverly and Steven Bauer. The spoofer project: Inferring the extent of source address filtering on the internet. In *Proceedings of USENIX Steps to Reducing Unwanted Traffic on the Internet (SRUTI) Workshop*, pages 53–59, July 2005.

[21] Matt Bishop. *Computer Security: Art and Science*. Addison Wesley Professional, 2003.

[22] Annalisa De Bonis and Ugo Vaccaro. Constructions of generalized superimposed codes with applications to group testing and conflict resolution in multiple access channels. *Theor. Comput. Sci.*, 306(1-3):223–243, 2003.

[23] Anat Bremler-Barr and Hanoch Levy. Spoofing Prevention Method. In *Proceedings of IEEE Infocom*, March 2005.

[24] J. Brustoloni. Protecting Electronic Commerce From Distributed Denial-of-Service Attacks. In *Proceedings of the eleventh International World Wide Web conference*, May 2002.

[25] Hal Burch and Bill Cheswisk. Tracing Anonymous Packets to Their Approximate Source. In *14th Systems Administration Conference, LISA 2000.*

[26] Mario Cagalj, Srdjan Capkun, and Jean-Pierre Hubaux. Wormhole-based antijamming techniques in sensor networks. *IEEE Transactions on Mobile Computing*, 6(1):100–114, 2007.

[27] Ken Calvert, Matt Doar, and Ellen W. Zegura. Modeling Internet topology. *IEEE Communications Magazine*, 35:160–163, June 1997.

[28] J. Cao, W.S. Cleveland, Y. Gao, K. Jeffay, F.D. Smith, and M.C. Weigle. Stochastic models for generating synthetic http source traffic. In *IEEE Infocom 2004*, volume 3, pages 1546–1557, March 2004.

[29] Martin Casado and Michael J. Freedman. Peering through the shroud: The effect of edge opacity on IP-based client identification. In *Proc. 4th Symposium on Networked Systems Design and Implementation (NSDI 07)*, Cambridge, MA, Apr 2007.

[30] CERT. Denial of Service Attacks. www.cert.org/tech_tips/denial_of_service.html, 1997.

[31] CERT. MS-SQL Server Worm, Advisory CA-2003-04. http://www.cert.org/advisories/CA-2003-04.html, January 2003.

[32] Chipcon AS. CC2420 2.4GHz IEEE 802.15.4 compliant RF Transceiver. http://www.chipcon.com, November 2003.

[33] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In *CRYPTO '94: Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology*, pages 257–270. Springer-Verlag, 1994.

[34] Chen-Nee Chuah, Lakshminarayanan Subramanian, and Randy H. Katz. Dcap: detecting misbehaving flows via collaborative aggregate policing. *SIGCOMM Comput. Commun. Rev.*, 33(5):5–18, 2003.

[35] Byung-Gon Chun, Puneet Mehra, and Rodrigo Fonseca. DAM: a DoS Attack Mitigation Infrastructure. *Class Paper CS261: Computer Security, UC Berkeley,*, January 2003.

[36] Evan Cooke, Farnam Jahanian, and Danny McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Proceedings of USENIX Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI)*, pages 39–44, July 2005.

[37] Paul J. Criscuolo. Distributed Denial of Service Trin00, Tribe Flood Network, Tribe Flood Network 2000, And Stacheldraht. Technical Report CIAC-2319, Department of Energy, Computer Incident Advisory Capability (CIAC), Lawrence Livermore National Laboratory, February 2000.

[38] Drew Dean, Matt Franklin, and Adam Stubblefield. An algebraic approach to IP traceback. *ACM Trans. Inf. Syst. Secur.*, 5(2):119–137, 2002.

[39] Yvo Desmedt, Rei Safavi-Naini, Huaxiong Wang, Lynn Batten, Chris Charnes, and Josef Pieprzyk. Broadcast anti-jamming systems. *Comput. Networks*, 35(2-3):223–236, 2001.

[40] Prashant Dewan, Parth Dasgupta, and Vijay Karamcheti. Defending against Denial of Service Attacks using Secure Name Resolution. Poster session of ICDCS 2003.

[41] Prashant Dewan, Partha Dasgupta, and Vijay Karamcheti. Defending against Denial of Service attacks using Secure Name resolution. In *Proceedings of the 2003 International Conference on Security and Management (SAM 2003)*, Las Vegas, NV, June 2003.

[42] Robert Dorfman. The detection of defective members of large populations. *The Annals of Mathematical Statistics*, 14(4):436–440, Dec. 1943.

[43] D.Z. Du and F.K. Hwang. *Combinatorial Group Testing and its Applications*. World Scientific, Singapore, 2nd edition, 2000.

[44] Cristian Estan and George Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Trans. Comput. Syst.*, 21(3):270–313, 2003.

[45] Federal Information Processing Standards Publication 180-2. Secure hash standard, August 2002.

[46] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. In *RFC 2827*, May 2001.

[47] Virgil D. Gligor. A note on denial-of-service in operating systems. *IEEE Trans. Software Eng.*, 10(3):320–324, 1984.

[48] Virgil D. Gligor. On denial-of-service in computer networks. In *Proceedings of the Second International Conference on Data Engineering*, pages 608–617, Washington, DC, USA, 1986. IEEE Computer Society.

[49] Virgil D. Gligor. Guaranteeing access in spite of distributed service-flooding attacks. In *Security Protocols Workshop*, pages 80–96, 2003.

[50] Virgil D. Gligor, Matt Blaze, and John Ioannidis. Denial of service - panel discussion. In *Security Protocols Workshop*, pages 194–203, 2000.

[51] Sachs & Co. Goldman, Nielsen//NetRatings, and Harris Interactive. espending report, 2005.

[52] M. T. Goodrich, M. J. Atallah, and R. Tamassia. Indexing information for data forensics. In J. Ioannidis, A. Keromytis, and M. Yung, editors, *Proceedings of the Third International Conference on Applied Cryptography and Network Security (ACNS)*, pages 206–221, 2005.

[53] T.A. Gulliver and E.B. Felstead. Anti-jam by Fast FH NCFSK - Myths and Realities. In *Conf. Rec. IEEE Military Commun. Conf.*, pages 187–191, 1993.

[54] Ramakrishna Gummadi, David Wetherall, Ben Greenstein, and Srinivasan Seshan. Understanding and mitigating the impact of rf interference on 802.11 networks. In *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 385–396, New York, NY, USA, 2007. ACM.

[55] Alefiya Hussain, John Heidemann, and Christos Papadopoulos. A Framework for Classifying Denial of Service Attacks. In *ACM SIGCOMM 2003*.

[56] Alefiya Hussain, John Heidemann, and Christos Papadopoulos. Identification of repeated denial of service attacks. In *Proceedings of the IEEE Infocom*, Barcelona, Spain, April 2006. IEEE.

[57] F. K. Hwang. A method for detecting all defective members in a population by group testing. *Journal of the American Statistical Association*, 67(339):605–608, Sep. 1972.

[58] J. Mitola III and Jr. G. Q. Maguire. Cognitive radio: Making software radios more personal. *IEEE Personal Communications*, 6:13–18, 1999.

[59] John Ioannidis and Steven M. Bellovin. Implementing Pushback: Router-Based Defense Against DDoS Attacks. In *Proceedings of Network and Distributed System Security Symposium*. The Internet Society, February 2002.

[60] Cheng Jin, Haining Wang, and Kang G. Shin. Hop-count filtering: an effective defense against spoofed DDoS traffic. In *Proceedings of the 10th ACM conference on Computer and communication security*, pages 30–41, 2003.

[61] Jim Jones. Distributed Denial of Service Attacks: Defenses, A Special Publication. Technical report, Global Integrity, 2000.

[62] A. Juels and J. Brainard. Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks. In *Proceedings of NDSS (Networks and Distributed Security Systems)*, 1999.

[63] Jaeyeon Jung, Balachander Krishnamurthy, and Michael Rabinovich. Flash crowds and denial of service attacks: characterization and implications for cdns and web sites. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 293–304, New York, NY, USA, 2002. ACM Press.

[64] Srikanth Kandula, Dina Katabi, Matthias Jacob, and Arthur W. Berger. Botz-4-Sale: Surviving Organized DDoS Attacks That Mimic Flash Crowds. In *2nd Symposium on Networked Systems Design and Implementation (NSDI)*, Boston, MA, May 2005.

[65] Frank Kargl, Joern Maier, and Michael Weber. Protecting web servers from distributed denial of service attacks. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 514–524, New York, NY, USA, 2001. ACM Press.

[66] Chris Karlof and David Wagner. Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures. *Elsevier's AdHoc Networks Journal, Special Issue on Sensor Network Applications and Protocols*, 1(2–3):293–315, September 2003.

[67] W.H. Kautz and R.R. Singleton. Nonrandom binary superimposed codes. *IEEE Trans. Inform. Theory*, pages 363–377, 1964.

[68] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. In *IETF, RFC 2401*, November 1998.

[69] A. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure Overlay Services. In *ACM SIGCOMM*, 2002.

[70] Sherif Khattab, Sameh Gobriel, Rami Melhem, and Daniel Mossé. Live baiting for service-level dos attackers. In *IEEE INFOCOM'08*.

[71] Sherif Khattab, Rami Melhem, Daniel Mossé, and Taieb Znati. Honeypot Backpropagation for Mitigating Spoofing Distributed Denial-of-Service Attacks. In *Proceedings of the 2nd International Workshop on Security in Systems and Networks (SSN'06)*, 2006.

[72] Sherif Khattab, Rami Melhem, Daniel Mossé, and Taieb Znati. Honeypot backpropagation for mitigating spoofing distributed denial-of-service attacks. *Journal of Parallel and Distributed Computing (JPDC) Special Issue on Security in Grid and Distributed Systems*, 66(9):1152–1164, Sept 2006.

[73] Sherif Khattab, Daniel Mossé, and Rami Melhem. Jamming mitigation in multi-radio wireless networks: Reactive or proactive? In *under submission*, 2008.

[74] Sherif Khattab, Daniel Mossé, and Rami Melhem. Modeling of the channel-hopping anti-jamming defense in multi-radio wireless networks. In *MOBIQUITOUS'08 (to appear)*, 2008.

[75] Sherif M. Khattab, Daniel Mossé, and Rami G. Melhem. Honeybees: combining replication and evasion for mitigating base-station jamming in sensor networks. In *IPDPS*, 2006.

[76] Sherif M. Khattab, Chatree Sangpachatanaruk, Rami Melhem, Daniel Mossé, and Taieb Znati. Proactive Server Roaming for Mitigating Denial-of-Service Attacks. In *ITRE'03*.

[77] Sherif M. Khattab, Chatree Sangpachatanaruk, Daniel Mossé, Rami Melhem, and Taieb Znati. Roaming Honeypots for Mitigating Service-level Denial-of-Service Attacks. In *ICDCS 2004*.

[78] Sherif M. Khattab, Chatree Sangpachatanaruk, Daniel Mossé, Rami Melhem, and Taieb Znati. Roaming Honeypots for Mitigating Service-level Denial-of-Service Attacks (under submission).

[79] Christian Kreibich, Andrew Warfield, Jon Crowcroft, Steven Hand, and Ian Pratt. Using packet symmetry to curtail malicious traffic. In *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets-IV)*, 2005.

[80] Balachander Krishnamurthy. Mohonk: mobile honeypots to trace unwanted traffic early. In *NetT '04: Proceedings of the ACM SIGCOMM workshop on Network troubleshooting*, pages 277–282, New York, NY, USA, 2004. ACM Press.

[81] Aleksandar Kuzmanovic and Edward W. Knightly. Low-Rate TCP-Targeted Denial of Service Attacks. (The Shrew vs. the Mice and Elephants). In *ACM SIGCOMM 2003*.

[82] Yee Wei Law, Lodewijk van Hoesel, Jeroen Doumen, Pieter Hartel, and Paul Havinga. Energy-efficient link-layer jamming attacks against wireless sensor network mac protocols. In *SASN '05: Proceedings of the 3rd ACM workshop on Security of ad hoc and sensor networks*, pages 76–88, New York, NY, USA, 2005. ACM.

[83] John Levine, Richard LaBella, Henry Owen, Didier Contis, and Brian Culver. The Use of Honeynets to Detect Exploited Systems Across Large Enterprise Networks. In *Proceedings of the 2002 IEEE Workshop on Information Assurance and Security*.

[84] Philip Levis et al. The Emergence of Networking Abstractions and Techniques in TinyOS. In *First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[85] Jun Li, Minho Sung, Jun Xu, and Li Li. Large-Scale IP Traceback in High-Speed Internet: Practical Techniques and Theoretical Foundation. In *Proc. of IEEE Symposium on Security and Privacy*, May 2004.

[86] eMarketer Lisa Phillips. Online banking: Remote channels, remote relationships? http://www.emarketer.com/Reports/All/Banking_on_jun06.aspx, May 2006.

[87] Xin Liu, Guevara Noubir, Ravi Sundaram, and San Tan. SPREAD: Foiling Smart Jammers Using Multi-Layer Agility. In *INFOCOM*, pages 2536–2540, 2007.

[88] Xin Liu, Xiaowei Yang, David Wetherall, and Thomas Anderson. Efficient and secure source authentication with packet passports. In *Proceedings of the 2nd Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI '06)*, 2006.

[89] R. Mahajan, S. Floyd, and D. Wetherall. Controlling high-bandwidth flows at the congested router. In *Proceedings of the Ninth International Conference on Network Protocols (ICNP)*, pages 192–201, 2001.

[90] Ratul Mahajan, Steven M. Bellovin, Sally Floyd, John Ioannidis, Vern Paxson, and Scott Shenker. Controlling high bandwidth aggregates in the network. 32(3):62–73, 2002.

[91] J. Millen. A Resource Allocation Model for DoS. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, pages 137–147, 1992.

[92] David L. Mills. Internet time synchronization: the network time protocol. *IEEE Trans. Communications*, 39(10):1482–1493, 1991.

[93] J. Mirkovic, G. Prier, and P. Reiher. Attacking DDoS at the Source. In *Proceedings of ICNP 2002*.

[94] Jelena Mirkovic, Janice Martin, and Peter Reiher. A Taxonomy of DDoS Attacks and DDoS Defense Mechanisms. Technical Report 020018, Computer Science Department, University of California, Los Angeles, 2002.

[95] Arunesh Mishra, Vivek Shrivastava, Dheeraj Agrawal, Suman Banerjee, and Samrat Ganguly. Distributed channel management in uncoordinated wireless environments. In *MobiCom '06: Proceedings of the 12th annual international conference on Mobile computing and networking*, pages 170–181, New York, NY, USA, 2006. ACM.

[96] Kevin Mitnick and William Simon. *The Art of Deception: Controlling the Human Element of Security*. John Wiley and Sons, 2002.

[97] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[98] David Moore, Colleen Shannon, Douglas J. Brown, Geoffrey M. Voelker, and Stefan Savage. Inferring internet denial-of-service activity. *ACM Trans. Comput. Syst.*, 24(2):115–139, 2006.

[99] William G. Morein, Angelos Stavrou, Debra L. Cook, Angelos D. Keromytis, Vishal Misra, and Dan Rubenstein. Using graphic turing tests to counter automated ddos attacks against web servers. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 8–19, 2003.

[100] Vishnu Navda, Aniruddha Bohra, Samrat Ganguly, and Dan Rubenstein. Using channel hopping to increase 802.11 resilience to jamming attacks. In *INFOCOM*, pages 2526–2530, 2007.

[101] James Newsome, Brad Karp, and Dawn Xiaodong Song. Paragraph: Thwarting signature learning by training maliciously. In *RAID*, pages 81–105, 2006.

[102] Hung Q. Ngo and Ding-Zhu Du. A survey on combinatorial group testing algorithms with applications to dna library screening. *Discrete mathematical problems with medical applications (New Brunswick, NJ) DIMACS Ser. Discrete Math. Theoret. Comput. Sci., Amer. Math. Soc.*, 55:171–182, 2000.

[103] G. Noubir. On connectivity in ad hoc network under jamming using directional antennas and mobility. In *International Conference on Wired /Wireless Internet Communications, Lecture Notes in Computer Science, Springer-Verlag*, 2004.

[104] G. Noubir and G. Lin. Low Power DoS Attacks in Data Wireless LANs and Countermeasures. *ACM SIGMOBILE Mobile Computing and Communications Review (MC2R)*, 7(3), 2003.

[105] Masafumi Oe, Youki Kadobayashi, and Suguru Yamaguchi. An implementation of a hierarchical IP traceback architecture. In *SAINT 2003 Workshop on IPv6 and applications*, Jan 2003.

[106] Martin J. Osborne and Ariel Rubinstein. *A course in game theory.* MIT Press, 1994.

[107] K. Park and H. Lee. On the effectiveness of probabilistic packet marking for IP traceback under denial of service attack. In *IEEE INFOCOM*, pages 338–347, 2001.

[108] Kihong Park and Heejo Lee. On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets. In *ACM SIGCOMM 2001.*

[109] Bryan Parno, Dan Wendlandt, Elaine Shi, Adrian Perrig, Bruce Maggs, and Yih-Chun Hu. Portcullis: Protecting connection setup from denial-of-capability attacks. In *Proceedings of the ACM SIGCOMM*, August 2007.

[110] Bhrat Patel and Jon Crowcroft. Ticket based service access for the mobile user. In *MobiCom '97: Proceedings of the 3rd annual ACM/IEEE international conference on Mobile computing and networking*, pages 223–233, 1997.

[111] Vern Paxson. An Analysis of Using Reflectors for Distributed Denial-of-Service Attacks. *ACM Computer Communications Review (CCR)*, 31(3), July 2001.

[112] Tao Peng, C. Leckie, and K. Ramamohanarao. Protection from distributed denial of service attacks using history-based ip filtering. volume 1, pages 482–486 vol.1, 11-15 May 2003.

[113] A. Perrig, R. Szewczyk, V. Wen andD. Cullar, and J. D. Tygar. SPINS: Security protocols for sensor networks. In *Proceedings of MOBICOM*, 2001.

[114] R. L. Pickholtz, D. L. Schilling, and L. B. Milstein. Theory of spread spectrum communications—a tutorial. 20:855–884, May 1982.

[115] Joseph Polastre, Robert Szewczyk, Cory Sharp, and David Culler. The Mote Revolution: Low Power Wireless Sensor Network Devices. In *Hot Chips 16: A Symposium on High Performance Chips*, 2004.

[116] The Honeynet Project. *Know Your Enemy.* Addison-Wisley, Indianapolis, IN, 2002.

[117] Niels Provos. A virtual honeypot framework. In *13th USENIX Security Symposium*, August 2004.

[118] Michael O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM*, 36(2):335–348, 1989.

[119] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. My botnet is bigger than yours (maybe, better than yours): Why size estimates remain challenging. In *Proceedings of USENIX/HotBots*, April 2007.

[120] S. Ranjan, R. Swaminathan, M. Uysal, and E. Knightly. Ddos-resilient scheduling to counter application layer attacks under imperfect detection. In *Proceedings of the IEEE Infocom*, Barcelona, Spain, April 2006. IEEE.

[121] Ronald L. Rivest and Adi Shamir. PayWord and MicroMint–Two Simple Micropayment Schemes. In Mark Lomas, editor, *Proceedings of 1996 International Workshop on Security Protocols*, number 1189 in Lecture Notes in Computer Science, pages 69–87. Springer, 1996.

[122] G. Sager. Security fun with OCxmon and cflowd. Internet2 working group meeting, November 1998.

[123] C. Sangpachatanaruk, S. M. Khattab, T. Znati, R. Melhem, and D. Mossé. A Simulation Study of the Proactive Server Roaming for Mitigating Denial of Service Attacks. In *Proceedings of ANSS'03*.

[124] C. Sangpachatanaruk, S. M. Khattab, T. Znati, R. Melhem, and D. Mossé. Design and analysis of a replicated elusive server scheme for mitigating denial of service attacks. *Journal of Systems and Software*, 73(1):15–29, Sept 2004.

[125] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Practical network support for IP traceback. In *ACM SIGCOMM 2000*.

[126] Christoph L. Schuba, Ivan V. Krsul, Markus G. Kuhn, Eugene H. Spafford, Aurobindo Sundaram, and Diego Zamboni. Analysis of a Denial of Service Attack on TCP. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 208–223, May 1997.

[127] Vyas Sekar, Nick Duffield, Oliver Spatscheck, Jacobus van der Merwe, and Hui Zhang. Lads: Large-scale automated ddos detection system. In *Proceedings of the 2006 USENIX Annual Technical Conference*, 2006.

[128] M. Shreedhar and George Varghese. Efficient fair queueing using deficit round robin. In *SIGCOMM '95: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 231–242, New York, NY, USA, 1995. ACM.

[129] Alex C. Snoeren, David G. Andersen, and Hari Balakrishnan. Fine-Grained Failover Using Connection Migration. In *Proc. of 3rd USENIX Symposium on Internet Technologies and Systems (USITS)*, 2001.

[130] Alex C. Snoeren and Hari Balakrishnan. An end-to-end approach to host mobility. In *Proceedings of the sixth annual international conference on Mobile computing and networking*, pages 155–166, 2000.

[131] Alex C. Snoeren, Hari Balakrishnan, and M. Frans Kaashoek. The Migrate Approach to Internet Mobility. In *Proc. of the Oxygen Student Workshop*, July 2001.

[132] Alex C. Snoeren, Craig Partridge, Luis A. Sanchez, Christine E. Jones, Fabrice Tchakountio, Beverly Schwartz, Stephen T. Kent, and W. Timothy Strayer. Single-packet IP traceback. 10(6):721–734, December 2002.

[133] Dawn X. Song and Adrian Perrig. Advanced and Authenticated Marking Schemes for IP Traceback. In *Proceedings of IEEE Infocom*, 2001.

[134] S. Staniford, V. Paxson, and N. Weaver. How to 0wn the Internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium*, August 2002.

[135] Angelos Stavrou and Angelos D. Keromytis. Countering dos attacks with stateless multipath overlays. In *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*, pages 249–259, 2005.

[136] Angelos Stavrou, Angelos D. Keromytis, Jason Nieh, Vishal Misra, and Dan Rubenstein. MOVE: An End-to-End Solution to Network Denial of Service. In *The 12th Annual Network and Distributed System Security Symposium (NDSS)*, 2005.

[137] J. G. Steiner, B. Clifford Neuman, and J.I. Schiller. Kerberos: An Authentication Service for Open Network Systems. In *Proceedings of the Winter 1988 Usenix Conference*, February 1988.

[138] Andrew Sterrett. On the detection of defective members of large populations. *The Annals of Mathematical Statistics*, 28(4):1033–1036, Dec. 1957.

[139] Richard W. Stevens. *TCP/IP Illustrated, Volume 1 (The Protocols)*. Addison-Wesley, 1994.

[140] Robert Stone. CenterTrack: An IP Overlay Network for Tracking DoS Floods. In *Proceedings of the 9th USENIX Security Symposium*, August 2000.

[141] Florin Sultan, Kiran Srinivasan, Deepa Iyer, and Liviu Iftode. Migratory TCP: Connection Migration for Service Continuity in the Internet. In *Proceedings of ICDCS 2002*.

[142] Florin Sultan, Kiran Srinivasan, Deepa Iyer, and Liviu Iftode. Migratory TCP: Highly Available Internet Services Using Connection Migration. Technical Report DCS-TR-462, Rutgers University, December 2001.

[143] Rangarajan Vasudevan, Z. Morley Mao, Oliver Spatscheck, and Jacobus van der Merwe. Reval: A tool for real-time evaluation of ddos mitigation strategies. In *Proceedings of the 2006 USENIX Annual Technical Conference*, 2006.

[144] Luis von Ahn, Manuel Blum, and John Langford. Telling humans and computers apart automatically. *Commun. ACM*, 47(2):56–60, 2004.

[145] Michael Walfish, Hari Balakrishnan, David Karger, and Scott Shenker. Dos: Fighting fire with fire. In *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets-IV)*, 2005.

[146] Q. Wang, T.A. Gulliver, V.K. Bhargava, and E.B. Felstead. Performance of Fast Frequency Hopped Noncoherent MFSK with a Fixed Hop Rate Under Worst Case Jamming. 38:1786–1798, 1990.

[147] X. Wang and M. K. Reiter. Defending against denial-of-service attacks with puzzle auctions. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003.

[148] X.F. Wang and M. Reiter. Mitigating Bandwidth-Exhaustion Attacks using Congestion Puzzles. In *ACM CCS 2004*.

[149] Brent Waters, Ari Juels, J. Alex Halderman, and Edward W. Felten. New client puzzle outsourcing techniques for dos resistance. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 246–256, 2004.

[150] N. Weiler. Honeypots for distributed denial-of-service attacks. In *Proceedings of WET ICE 2002*.

[151] Anthony D. Wood and John A. Stankovic. Denial of Service in Sensor Networks. *IEEE Computer*, 35(10):54–62, 2002.

[152] Anthony D. Wood, John A. Stankovic, and Sang H. Son. JAM: A Jammed-Area Mapping Service for Sensor Networks. In *RTSS*, 2003.

[153] Anthony D. Wood, John A. Stankovic, and Gang Zhou. DEEJAM: Defeating Energy-Efficient Jamming in IEEE 802.15.4-based Wireless Networks. In *The 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, San Diego, CA, June 2007.

[154] Wu-chang Feng. The case for TCP/IP puzzles. In *ACM SIGCOMM workshop on Future directions in network architecture*, pages 322–327, 2003.

[155] Wenyuan Xu, Wade Trappe, and Yanyong Zhang. Channel surfing: defending wireless sensor networks from interference. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 499–508, New York, NY, USA, 2007. ACM.

[156] Wenyuan Xu, Wade Trappe, Yanyong Zhang, and Timothy Wood. The feasibility of launching and detecting jamming attacks in wireless networks. In *ACM MobiHoc*, 2005.

[157] Wenyuan Xu, Timothy Wood, Wade Trappe, and Yanyong Zhang. Channel surfing and spatial retreats: defenses against wireless denial of service. In *Proceedings of the 2004 ACM workshop on Wireless security (WiSe)*, pages 80–89, 2004.

[158] Avi Yaar, Adrian Perrig, and Dawn Song. An endhost capability mechanism to mitigate DDoS flooding attacks. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2004.

[159] Avi Yaar, Adrian Perrig, and Dawn Song. FIT: Fast Internet traceback. In *Proceedings of IEEE Infocom*, March 2005.

[160] David K. Y. Yau, John C. S. Lui, and Feng Liang. Defending Against Distributed Denial-of-service Attacks with Max-min Fair Server-centric Router Throttles. In *Proceedings of the IEEE International Workshop on Quality of Service (IWQoS)*, May 2002.

[161] Che-Fn Yu and Virgil D. Gligor. A specification and verification method for preventing denial of service. *IEEE Trans. Software Eng.*, 16(6):581–592, 1990.

[162] Gang Zhou, Chengdu Huang, Ting Yan, Tian He, John A. Stankovic, and Tarek F. Abdelzaher. MMSN: Multi-Frequency Media Access Control for Wireless Sensor Networks. In *INFOCOM*, 2006.