An Overlay Architecture for End-to-End Service Availability

Angelos Stavrou

Submitted in partial fulfillment of the
Requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2007

ABSTRACT

An Overlay Architecture for End-to-End Service Availability

Angelos Stavrou

Perhaps one of the most compelling problems of the modern Internet is the lack of a comprehensive and unifying approach to dealing with service security and resilience. Although many such individual reliability mechanisms exist, no general set of policies or standards exists for how these mechanisms can be combined to achieve an overall robust state of security for the network. In short, no ”security and availability” architecture exists. This thesis introduces and analyzes mechanisms that boost the security, resilience, and performance of network systems in a manner that is transparent to both the existing infrastructure and the end-users.

In this dissertation, we discuss our work on defending against distributed denial of service (DDoS) attacks. Such attacks involve large numbers of compromised hosts (bots) that send unsolicited traffic toward a target, thereby congesting the network links close to it, rendering its services unusable. To frustrate these types of attacks, we propose and evaluate practical mechanisms that can protect a wide range of services while maintaining or even improving their performance characteristics. Our approach is focused on network-level faults and attacks: we do not focus our attention on application-level service vulnerabilities. We do, however, offer protection against malicious or unexpected increases in network-bound service requests. Our end goal is to provide a practical end-to-end framework that significantly improves service availability and connectivity without incurring a prohibitive deployment or performance cost. Ideally, the protection system should be able to scale to millions of users and accommodate any applications' requirements including network latency and throughput. We developed a number of systems (PROOFS, WebSOS, MOVE, and packet spreading via multi-path overlays) that illustrate a progression toward the aforementioned goals.

Because our solutions depend on large scale overlay networks, we present a novel mechanism for protecting a wide class of these networks against insider attacks. For overlay networks that exhibit well-defined properties (due to their topology or structure), we demonstrate how to defend such networks against non-conforming (i.e., abnormal) behavior of participating nodes. In particular, we can defend against DoS attacks from within the overlay itself. We use a lightweight distributed detection mechanism that exploits inherent structural invariants of Distributed Hash Tables (DHTs) to ferret out anomalous flow behavior. Upon detection, we invoke a Pushback-like protocol to notify and prompt into action (e.g., throttle the traffic) the predecessor node: the node from which the offending traffic arrives. In addition, we demonstrate how to remain TCP-friendly by using packet spreading and replication techniques with regular TCP connections in addition to our UDP-based techniques. Our experiments show that our system can take advantage of the underlying multi-path link capacity without starving other flows over shared links. For TCP flows, we show that there is no significant throughput or latency degradation when using regular TCP connections.

To demonstrate the applicability of our system for real-time and interactive applications, we introduce Access Assured Mobile desktop computing (A^2M), a secure and attack-resilient remote desktop computing hosting infrastructure. A^2M combines a stateless and secure communication protocol, a single-hop Indirection-based network (IBN) and a remote display architecture to provide mobile users with continuous access to their desktop computing sessions. Our architecture protects both the hosting infrastructure and the client's connections against a wide range of service disruption attacks. Unlike any other DoS protection system, A^2M takes advantage of its low-latency remote display mechanisms and asymmetric traffic characteristics by using multi-path routing to send a small number of of each packet transmitted from client to server. This multi-path packet replication diversifies the client-server communication, boosts system resilience, and reduces end-to-end latency. Through deployment on a planet-lab, a distributed network, we show that A^2M significantly

increases the hosting infrastructure's attack resilience. Using current ISP bandwidth data, we can protect against attacks involving millions of bots while providing good performance for multimedia and web applications and basic GUI interactions even when up to 30% and 50%, respectively, of indirection nodes become completely unresponsive.

Contents

List of Figures

List of Tables

Acknowledgements

Chapter 1

Introduction

> *The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible.*
>
> *J.H. Saltzer, D.P. Reed and D.D. Clark*
>
> END-TO-END ARGUMENTS IN SYSTEM DESIGN

The current Internet infrastructure is clearly successful at providing connectivity for a wealth of ubiquitous and diverse services that have increasingly become part of our everyday life. Perhaps one of the most compelling problems of the Internet today is the lack of an overarching approach to dealing with online service security and resilience: there exist a lot of mechanisms but no ”security and availability architecture” – no set of policies or standards for how these mechanisms will be combined to achieve overall good security [56]. Modern networking systems still lack a comprehensive and unifying approach to guarantee service resiliency against faults and attacks both in the network and application layer.

One of the of most prominent and disruptive threats against online network services is the class of network attacks that aim to deny users access to the service. Such attacks are called Denial of Service (DoS) attacks [75]. DoS attacks become even more devastating

and difficult to detect and mitigate when they involve attackers that are distributed in the network. Current forms of DoS attack implicate multiple groups of Internet machines that have been taken over and controlled by an attacker. In these Distributed Denial of Service (DDoS) attacks [122, 121, 74, 110], network machines (called bots) are manipulated by the attacker to produce an excessive surge of traffic toward a target server or victim [166, 74, 42]. The surge of malicious traffic effectively blends in with legitimate or normal traffic, making it difficult to weed out and causing starvation for the unwitting clients. Unfortunately, DDoS attacks can only worsen. Despite the current frequent and rapid increases in network and processing speeds, botnet sizes and attack capabilities of real-world botnets also increase; furthermore, botnets also benefit from increases in network and processing capacity. Moreover, attackers devise sophisticated software to infect and subsequently control thousands of infected machines while remaining stealthy [74, 166].

Solving the network denial of service (DoS) problem [121, 72] is extremely hard given the fundamentally open nature of the Internet and the apparent reluctance of router vendors and network operators to deploy and operate new, potentially complex mechanisms [75]. Overlay-based approaches such as SOS [91] and MayDay [9] offer an attractive alternative, as they do not require changes to protocols and routers, and need only minimal collaboration from Internet Service Providers (ISPs). Such systems use an Internet-wide network of nodes that act as first-level firewalls that discriminate between legitimate traffic and potentially malicious traffic based on some form of user or end-host authentication. Their distributed nature requires an extremely well provisioned adversary to suppress their functionality, since attack traffic must be split among all the nodes to disrupt protected communications.

Moreover, Internet routing lacks robustness and path selection optimality: Internet Service Providers (ISPs) select their routing policies enforcing packet routes based on their peering agreements rather than minimum hop count or link capacity. There are research studies [8, 165] indicating that in many cases, the use of alternative routing can significantly improve the end-to-end path characteristics. Empowering the end user and applications with

the ability to select the routing of their own packets can be achieved using overlay networks [161, 49, 156, 7, 6, 154, 164]. Such networks are installed on top of the existing network infrastructure and do not require any ISP collaboration. To communicate with a remote network server, a user has to first contact the overlay network and establish communication channel with the remote server through the overlay. Measurements collected from both experimental and commercial deployments of overlay networks support the claim that multi-path routing can indeed offer performance and resiliency benefits over the ISP-provided packet routing.

This thesis introduces and analyzes mechanisms that boost the security, resilience and performance of network systems. These systems are composed of large numbers of untrusted and unreliable components that communicate using the existing network infrastructure. Furthermore, the majority of this network infrastructure is controlled by private companies running legacy equipment and software making it difficult to design and advocate solutions that ignore or fail to make use of current infrastructure. These limitations become even more restrictive when combined with the fact that security mechanisms typically have a negative impact on the performance of a system. However elegant in their design, most service protection architectures are impractical or infeasible because they represent a prohibitive deployment or performance cost. Exploring the tradeoffs between performance, usability, and security is therefore of paramount importance. In this thesis we propose and evaluate practical mechanisms that can protect a wide range of services while maintaining or even improving their performance characteristics. Our study indicates that an overlay architecture is both easily deployable and can effectively filter traffic close to the edge of the network. Furthermore, we show that overlay nodes can be used as vantage points to harness the hidden capacity in the network to essentially transcend the enforced ISP routing. Although the solution proposed in this thesis is generally applicable to a variety of network services, it is geared towards real-time and low latency services such as VoIP, thin clients, and health related applications including tele-medicine and remote surgery.

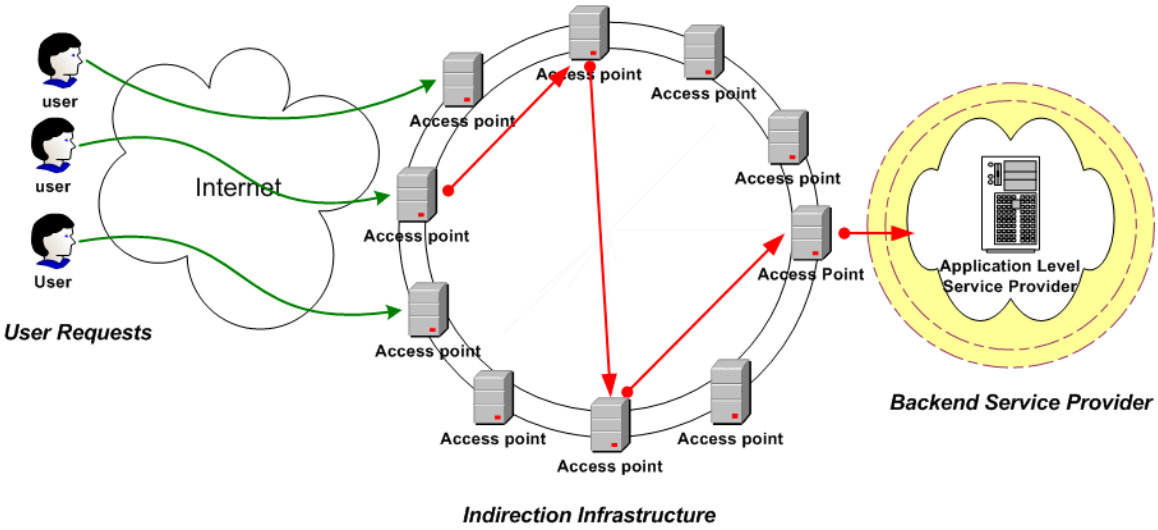


Figure 1.1: Communication architecture: Access Points represent an entry point to the overlay filtering and routing user requests to the back-end service provider

Our ultimate goal is to provide an end-to-end overlay architecture that significantly improves service availability and connectivity. Our approach should be able to scale to millions of users and accommodate any applications' requirements including network latency and throughput. To this end, we model network services as having three distinct components that inter-operate and integrate with each other for seamless service operation: the *front-end* consists of the client, the *communication fabric* consisting of multiple active and passive equipment, and a *back-end* containing the server. Figure 1.1 presents this architecture. In the left of the figure, clients reside in different networks and route their requests through the Internet to the location of the server. Clients establish connections directly to the server using the single path routing provided by their corresponding ISPs and they receive replies using the reverse path (which can be assymetric). We use this abstract model as a guide to develop techniques that significantly elevate the fault and attack tolerance of each individual component and their corresponding communications.

Our goal is to harness the latent capacity of the Internet or an organization (most nodes and links are heavily overprovisioned) for security and quality of service. Our work can be

summarized into the following two parts:

- An overlay network which provides distributed presence allowing us to take advantage of the different paths existing in the network and at the same time act as a distributed firewall filtering incoming connections. The use of a distributed overlay presence allows our solution to be both scalable and easily deployable by virtue of Bypassing all ISP enforced static routing solutions.

- We introduce scalable mechanisms that allow us to utilize and manage these new resources in a way that cannot be abused to deny service to both clients and servers. Our approach is resilient to attacks because we use multiple communication paths and thus we do not depend on a single point of failure. To prevent storing the protocol state and the authentication information for each connection to all of the overlay nodes, we designed an encrypted verifiable token that the client uses to communicate. This allows our system to scale well with the number of clients keeping only a small amount of state on the overlay nodes used to prevent attacks and protocol abuses. In addition, our system operates without significant loss or latency overhead even under adverse operating conditions: it is designed to adapt and compensate when the underlying multi-path network is unreliable. The attack detection algorithms are designed to operate and maintain reliable end-to-end connectivity even when a large fraction of the network is either attacked or taken over by attackers.

Despite our overall architecture's seeming heavy reliance on the use of overlay networks, in practice our system would work well with commodity routers; unfortunately, such machines do not typically implement request filtering and application-level routing on top of the existing BGP routing[1].

[1] Often, performance considerations are given as an argument against such additional processing.

1.1 Thesis Contributions

The list of contributions of our work includes:

- The first practical and effective overlay-based architecture using a scalable distributed spread-spectrum like communication protocol (Chapter 6) that enables us to admit users, route, and regulate traffic without depending on a single point of failure. We designed this protocol to be lightweight keeping only a small amount of state used to prevent attacks and protocol abuses. Using our protocol goes beyond mere performance gains: our system is impervious to some simple but devastating attacks we devised against all stateful Overlay networks including SOS, RON and Tor (Section 6.1)

- The implementation and evaluation of an overlay-based architecture for Internet services using Planet-Lab, an Internet2-based distributed platform. Our experimental results demonstrate that overlays can both protect against DoS and provide adequate performance.

- A fully end-to-end evaluation of our overall architecture using a real-time remote desktop infrastructure (Chapter 8). We showed that our system can effectively protect time-critical applications including video and data streaming even when 40% of the overlay network becomes unresponsive.

- The introduction of a lightweight algorithm to detect malicious overlay insiders and push-back the attack traffic with no need for ISP collaboration (Chapter 7). If we combine this system with our overlay-based communication protocol, we have a DoS architecture that can defend against attacks both from insiders and outside clients.

- The first end-to-end DDoS network service protection system that does not require strong authentication and does not depend on any network by combining lightweight process migration and overlay routing (Chapter 5).

- The first use Graphic Turing Tests (GTTs) to protect Web services against the massive parallelism of automated Denial of Service attacks (Chapter 4). Our approach shows that it is possible to associate network packet flows with some notion of a principal (for GTTs, with anonymous humans). This protection mechanism can be applied to any of the previous mentioned DoS systems.

1.2 What is not addressed in this thesis

In this thesis, we are not concerned with application-level service vulnerabilities or faults. There are other efforts towards that direction including [152, 39, 38]. For example, we do not address algorithmic-complexity DoS attacks [40], whereby attackers exploit deficiencies in applications' data structures that allow them to invoke worst-case running times. In addition, the DoS protection mechanisms presented in this dissertation are not directly applicable to link-level Denial of Service (DoS) attacks such as the ones against the MAC layer of 802.11-based wireless access networks [16]. We do, however, offer protection against malicious or unexpected increase in network-bound service requests.

1.3 Thesis Organization

Chapter 2 provides an overview of the related work on service protection mechanisms. In Chapter 3 we present our initial work on adaptive availability for static objects. We continue with a system name for Web Services in Chapter 4. Chapter 5 shows our end-to-end server protection system. We proceed with two novel attacks against overlays and our proposed solution in Chapter 6. In Chapter 7, we present a novel distributed system that offers protection against malicious insider attacks for overlay networks. With Access-Assured Mobile Desktop Computing in Chapter 8, we evaluate of our overlay protection architecture using a latency demanding remote-display thin client application.

Chapter 2

Background & Related Work

As a result of its increased popularity and usefulness, the Internet contains both interesting targets and enough malicious and ignorant users that DoS attacks are simply not going to disappear on their own; indeed, although the press has stopped reporting such incidents, recent studies have shown a surprisingly high number of DoS attacks occurring constantly throughout the Internet [116, 14], sometimes due to bad design (rather than malicious intent) [33]. A further compounding factor is the susceptibility of the basic protocols (*i.e.,* IP and TCP) to denial of service attacks [149, 68, 147, 99, 61, 150].

Some of the early work toward countering DoS attacks [44, 148, 59, 153, 163, 104, 145, 180] focused on detecting the source of DoS attacks in progress, aiming to identify the true source of the attack. A variant of the packet marking approaches creates probabilistically unique path-marks on packets without requiring router coordination; end-hosts or firewalls can then easily filter out packets belonging to a path that exhibits anomalous behavior [177]. Although this approach avoids many of the limitations of the pure marking schemes, it requires that core routers "touch" packets (rather than simply switch them), and assumes that the limited resource is the target's CPU cycles, rather than the available bandwidth (*i.e.,* preventing the DoS attack is "simply" a matter of quickly determining which packets the server should ignore). In our work, we assume that the scarce resource is bandwidth.

Diffield and Krishnamurthy [52] propose artificially varying the network conditions a network flow is exposed to as a way of detecting conforming *vs.* non-conforming flows. For example, non-malicious traffic flows will back off or otherwise take corrective action when exposed to (simulated) congestion, while malicious flows will continue transmitting at a maximum (or fixed) rate.

A description of filtering out source-spoofed packets inside the Internet core, and discussion on the effectiveness of this approach is provided by [132]. The authors suggest piggy-backing on BGP to propagate the necessary information. DDoS attacks using real IP addresses are not affected by this scheme. An interesting approach is that of [81], which proposes an IP hop-count-based filter to weed out spoofed packets. The rationale is that most such packets will not have a hop-count (TTL) field consistent with the IP addresses being spoofed. A recent study [20] showed that a significant number of ASes do not employ any anti-spoofing mechanism

PRIMED [167] is a proactive approach to DDoS mitigation, in which users specify *a priori* to their ISP their (dis)interest in receiving traffic from specific network entities (blacklist/whitelist). ISPs can also dynamically construct groups (Communities of Interest) of badly behaving entities, and per-customer good CoIs containing remote entities (outside the ISP) that communicated with the customer in the past without raising any alarms. When such CoIs can be constructed, they can help considerably in mitigating the effects of DoS attacks. Mao *et al.* [110] analyzed several DDoS attack traces, and discovered 70% of attacks involved sources from fewer than 50 ASes, and that a small number of ASes produces 70% of total attack volume. They also found little use of address spoofing by attackers, suggesting that most DDoS attacks are invisible to backscatter measurement techniques [116].

SOS [91] first suggested the concept of using an overlay network to preferentially route traffic from legitimate users to a secret node (that can change over time), which is allowed to reach the protected server. All other traffic is restricted at the ISP's POP, which in most cases

has enough capacity to handle all attack and legitimate traffic (the bottleneck is typically in the protected server's access link). Since the routers perform white-list filtering, the overhead of the system is negligible. In the original SOS approach, admission to the overlay was done based on public-key (or, more generally, cryptographic) authentication, requiring prior knowledge of the set of legitimate users. WebSOS [117] relaxes this restriction by adding a Graphic Turing Test (GTT) to the overlay, allowing the system to differentiate between human users and attack zombies (a technique since used also in [84]). SOSMP [157] shows how to integrate a payment mechanism with WebSOS, to provide deployment incentives to ISPs or other commercial entities. MOVE [155] eliminates the dependency on network filtering at the ISP POP routers by keeping the current location of the server secret and using process migration to move away from targeted locations. This is similar to the concept of "hidden servers" that was introduced by anonymity systems such as Tor [135, 49], and the Roaming Honeypots system [94]. Mayday [9] explores separately the two main facets of the SOS architecture, filtering and overlay routing, with several alternative mechanisms considered. It is observed that in some cases, the various security properties offered by SOS can still be maintained using mechanisms that are simpler and more predictable. However, some second-order properties, such as the ability to rapidly reconfigure the architecture in anticipation of or in reaction to a breach of the filtering identity (*e.g.,* identifying the secret servlet) are compromised. In most other respects, the two approaches are very similar. An analysis of some security/performance design tradeoffs in IONs appears in [176]. Wang *et al.* [169] used an online network simulator to investigate the resistance of proxy networks against simple DoS attacks. They conclude that the resistance of a proxy network to flooding attacks increases linearly with its size. However, they assume that users can instantaneously detect attacked ION nodes and switch to new ones with zero overhead, an assumption that did not hold for any ION architecture prior to ours. Gore [32] is an instantiation of SOS where all the access point (indeed, the whole "overlay") are located in a large, over-provisioned facility. Injected routes are used to redirect traffic from a DDoS target to this

facility, which performs scrubbing using authentication, GTTs, payments, or detection and shaping heuristics such as the ones proposed by Xu *et al.* [174]. Tunneling is then used to divert the "clean" traffic to the target of the DDoS attack. A similar approach is proposed by Greenhalgh *et al.* [60].

Another approach to mitigating DoS attacks against information carriers is to massively replicate static content around the network [158]. To prevent access to the replicated information, an attacker must attack all replication points throughout the entire network — a task that is considerably more difficult than attacking a small number of, often co-located, servers. Replication is a promising means to preserve information that is relatively static, such as news articles. However, there are several reasons why replication is not always an ideal solution. For instance, the information may require frequent updates complicating large-scale coherency (especially during DoS attacks), or may be dynamic by its very nature (*e.g.,* a live web-cast). Another concern is the security of the stored information: engineering a highly-replicated solution without leaks of information is a challenging endeavor.

Siff[178] is the first system to create stateless flow filtering by having each router add "capabilities" to packets that traverse them; the receiver of these packets is then responsible for sending these capabilities to its peers, which will allow them to send traffic at higher rates (privileged traffic). Unprivileged traffic is limited to a fraction of the available bandwidth; thus, although a DoS attack can prevent new connections from being established (by overloading the control channel used to communicate these capabilities), existing connections will be unharmed. Casado *et al.* [27] propose extending the notion of the SYN cookie [19] to encode capability-like information in TCP packets at a trusted network element (*i.e.,* not at the end nodes). Their system uses BGP to redirect traffic to such trusted elements. Estrin *et al.* first proposed a capability-like mechanism for network packets in [53]. A similar idea, packet passports, was proposed by Liu *et al.* [105] to deal with the problem of forged source addresses. Farhat also proposed a TCP-specific capability-like mechanism [54].

Gligor [57] proposed the use of a server that can produce tickets at line speeds. Clients

must obtain a ticket from this server before they are allowed to access a protected service. The approach is primarily geared towards application-level DoS protection. Anderson *et. al* [10] subsequently proposed a similar system for use at the network layer of an Internet-like architecture designed with a clean slate, assuming a distributed token server architecture and rate-limiting/filtering traffic on routers based on these tokens. A similar idea appears in [101] and [179]. Wang and Reiter [171] and Waters *et al.* [172] independently proposed the use of computational *"congestion" puzzles* instead of tickets. These schemes require client, server, and router support.

The authors of [175] conduct an adversarial evaluation of router-based DDoS defense mechanisms that operate at the granularity of individual flows [108] and flow aggregates [107]. Such defenses aim to detect misbehaving flows by comparing against models of reasonable congestion control behavior, and a rich literature of such techniques exists [136, 111, 73, 86]. The amount of throttling on suspicious traffic is typically proportional to its deviation from the expected behavior, as specified by the model. The evaluation shows that, while these approaches do well against "dumb" CBR attackers, they fail to detect and counter stealthier attacks that slowly rump-up or pulse the attack traffic. [99] discusses a similar attack, aimed at TCP's retransmission time-out mechanism, that can throttle TCP flows to a small fraction of their ideal rate while eluding detection. The authors study the use of randomized time-out mechanisms, which are shown to be inherently limited in an environment exhibiting protocol homogeneity. Similar attacks against TCP are described in other work [106, 61]. Chan *et al.* [30] describe a "herding" attack against auto-refreshing clients, such as web browsers that periodically download the same page: by launching an intense but short-lived periodic attack, it is possible to force all such clients to synchronize their access, thereby causing temporary congestion. Wang and Reiter [35] present an empirical analysis of several anti-DoS techniques that use filters near the target of an attack, using traces of real DDoS attacks to determine the impact of the filters on the attack traffic.

Pushback [76] extends previous work on aggregate congestion control [107] to allow pushing such filters closer to the sources of the attack, assuming collaborating (and mutually trusted) ISPs. A similar scheme is proposed in [11]. In COSSACK [130], participating agents at edge networks exchange information about observed traffic and form multicast cliques to coordinate attack suppression.

Several different DDoS mitigation technologies and their interactions are studied in [23]. Among their conclusions, they mention that requiring the clients to do some work, *e.g.,* [45], can be an effective countermeasure, provided the attacker does not have too many resources compared to the defender. Wang and Reiter [170] introduced the idea of a puzzle auction as a way to ease some of the practical deployment difficulties, *e.g.,* selecting the appropriate hardness for the puzzles. Their intuition is to let clients bid for the resources by tuning the difficulty of the puzzles they solve. When the server is attacked, legitimate clients gradually increase their bids (puzzle difficulty), raising the cost outside the adversary's capabilities. The authors envision combining their scheme with some anti-DoS mechanism that counteracts volume-based attacks [76, 177, 91]. Zou *et al.* [181] explore the use of an adaptive-defense framework that can be superimposed on existing defensive strategies, as a way of minimizing collateral damage and the cost of the defense (in terms of resources used).

Application-specific networks are not immune to denial of service attacks. For example, Daswani and Garcia-Molina [43] discuss the effect of query-flooding attacks in the Gnutella peer-to-peer network. Athanasopoulos *et al.* [12] describe the use of Gnutella (and possibly other similar systems) to launch DoS attacks against third-site web servers, by redirecting all queries to these with carefully crafted responses. One interesting observation from their real-world experiments was that such attacks persisted long after the malicious client left the P2P network.

Algorithmic-complexity denial of service attacks is the focus of [40], whereby attackers exploit deficiencies in applications' data structures that allow them to invoke worst-case

running times. They show how hashing and randomizing techniques can help protect against such attacks with little performance impact. [89] discusses various approaches to mitigating IP re-assembly DoS attacks against UDP-based protocols, which are another form of low-rate, complexity-based DoS attack. The proposed solution is to either expose application-level information to the network layer, or to establish the validity of a peer through an extra protocol round-trip, and treat packets from these addresses preferentially. TCP is also susceptible to various types of DoS attacks [149, 68, 147].

Early work defending against resource-depletion attacks focused around the concept of the "cookie," an opaque bit-string that the initiator of a connection request needs to return verbatim to the server before the request is allowed to proceed. Thus, cookies were used only to establish the validity of the peer in terms of network address reachability; in other words, cookies protect against attackers spoofing their IP address. Cookie-based solutions [103] were used against TCP connection-depletion (also known as TCP SYN) attacks [149, 68], and in security protocols such as Photuris [88], IKE [66], JFK [4], and others [126, 69]. More generally, The advantages of being stateless, at least in the beginning of a protocol run, were recognized in the context of security protocols in [80] and [13].

Computational client puzzles as a means to defend against denial of service attacks were first introduced in [82]. In that work, client puzzles were used to counter TCP SYN attacks from attackers that were willing to expose their IP address. Although TCP cookies are ineffective in that scenario, client puzzles can mitigate the effects of such an attack by an adversary that is CPU-limited. However, in recent years attackers have demonstrated their ability to effectively utilize large numbers of subverted hosts in their attacks [70]. Client puzzles have also been used in the context of security protocols [79, 102], most notably for protecting SSL against computational denial of service attacks [45]. Jakobsson and Juels [79] first proposed the concept of a useful puzzle, which they call a "bread pudding protocol." The particular scheme they use, applied to minting e-coins for the MicroMint micropayment system [139], is specific to MicroMint and does not appear to be easily generalizable to other

types of useful work. Diament *et al.* [46] showed the feasibility of a puzzle useful for other types of computation. Abadi *et al.* [3] introduced the concept of a memory-bound puzzle, which aims to impose the same solving delay as traditional client puzzles by increasing the number of memory accesses a client needs to perform to solve the challenge. Doshi *et al.* [51] provide an algorithmic foundation for memory puzzles, allowing for easier tuning of the basic parameters of such schemes.

In [63], the authors discuss the use selective authentication of broadcast streams to address the problem of denial of service targeted against the Forward Error Correction (FEC) component of receiving systems. A variant of erasure codes, called distillation codes, which allow the receiver of a multicast stream to correctly and efficiently discard invalid symbols injected by an attacker intent on causing a computational DoS is introduced by [87].

DoS attacks on 802.11-based wireless access networks are analysed in [16]. This study shows that the lack of basic security measures in the MAC layer allows attackers to completely disrupt service to the network using relatively few packets and low power consumption. The authors discuss several non-cryptographic countermeasures to these attacks, which address most of the problems. The authors of [67] discuss several attacks, including denial of service, against the 802.11i wireless standard. The use of directional antennas in mobile *ad hoc networks* to avoid wormhole attacks, whereby malicious nodes advertise false routes to various destinations and drop any traffic forwarded to them is proposed in [71].

Chapter 3

Adaptive Availability for non-Dynamic Content Distribution Services

3.1 Introduction

In our early efforts to provide service resilience for online objects, including static web pages and images, we introduced a client-based randomized P2P network with the ability to locate and retrieve such objects. We call this system **PROOFS**: **P**2P **R**andomized **O**verlays to **O**bviate **F**lash-crowd **S**ymptoms. The proposed P2P mechanism modifies the way the regular Internet routing *i.e. the communication fabric* would operate for the participating clients and for the protected server. To retrieve an object, Clients do not communicate directly with the server but instead attempt to retrieve the object through the randomized overlay. Only if the object cannot be located by the overlay, the clients attempt to establish a connection to the server to retrieve the requested object making the object available to the rest of the overlay nodes. In Figure 3.1, we show the differences between direct object retrieval and through the PROOFS randomized overlay.

PROOFS falls into a class of systems termed *First Generation P2P systems* that also contains P2P systems such as Gnutella [58] in which an object (or information about the

precise location of an object) is equally likely to be available at any node within the P2P system. In contrast, *Second Generation P2P systems* (e.g., see [162, 41, 134, 142]) form overlays that, using a variety of clever distributed and dynamic hashing strategies, assign each object to a particular set of clients in the overlay. For an "unpopular" object that resides at a small, fixed number of locations, second generation systems can locate an object using $O(\log n)$ queries, whereas first generation systems require $O(n)$ queries. Thus, second generation systems can provide considerable savings in levels of traffic used for searching as n grows large. However, mathematical and simulation analysis in [143] shows that such searches have low expected traffic requirements and low latency when searching for objects that are the interest of a flash crowd. This is understood intuitively in that whenever a client locates and subsequently retrieves a copy of the desired object, that client can then service any subsequent queries, cutting down the costs in terms of both time taken and transmissions made of subsequent searches, in effect making the amortized cost of each client's search $O(\log n)$ as well. While in some respects, PROOFS is a step backwards from second generation systems, it has the following advantages:

- *Clients are not required to cache any objects or pointers to objects other than that which the client has explicitly expressed interest in receiving*. To date, second generation systems that address the flash crowd issue do so by requiring participating clients to explicitly cache copies of objects that are not necessarily of direct interest on the behalf of the system (i.e., for other clients). While technical complications are arguably solvable, it remains unclear whether users would feel comfortable using their own disk space to host unknown content.

- PROOFS handles dynamic changes in overlay membership (i.e., participants joining and leaving the system with time) without any additional mechanism or modification to its fundamental design. In addition, PROOFS is naturally robust even when there exist a substantial number of clients who "take advantage" of the system by using it to obtain popular objects, but who do not fully participate in assisting other

clients by either refusing to forward content or even secretly dropping all queries it receives. While some second generation systems have demonstrated certain degrees of robustness against changes in overlay membership, it is unclear how they perform in environments where some clients vary their levels of participation in the forwarding queries and/or delivering stored objects.

- The system is amenable to the formation of complex queries that contain keywords or temporal restrictions (e.g., a copy of an object generated within the last 5 minutes). This is much more difficult to do within second generation systems in which the object description must hash to a unique identifier. In first generation protocols, each client visited parses the query for itself.

Our design was motivated by the observations in [125] that more attention should be paid to the manageability, reliability and robustness of communication systems. Rather than target our main efforts toward minimizing traffic levels and delivery latencies, the system is designed to achieve "good" traffic levels and latencies while remaining robust, reliable, and manageable under a variety of network settings.

In particular, our goal is not to minimize network traffic levels or delivery latencies of objects to receivers. Instead, we seek to build a light-weight, easy-to-implement system that can perform its required function using "good" levels of traffic and with "good" delivery latencies. While there are numerous possible optimizations that can be added to improve performance along these lines, we do not focus here optimizing PROOFS along those lines. Instead, we focus on evaluating the robustness, reliability, and manageability of our prototype.

- Through analytical modeling and simulation, we show that the likelihood that the constructed overlay separates a client from reaching a large fraction of other clients is extremely rare, even in the presence of clients dynamically joining and leaving the overlay.

- Through simulation, we show that traffic levels, latency, and connectivity grow in a tolerable manner as a function of the fraction of overlay nodes cease to perform query and object forwarding (i.e., non-cooperative nodes).
- We evaluate a prototype implementation on a testbed comprised of end-systems scattered around the world. Although small in scale compared to how we hope the system will eventually be used, the testbed demonstrates that latencies and traffic utilization by the system are low enough to make the approach feasible in today's networks.

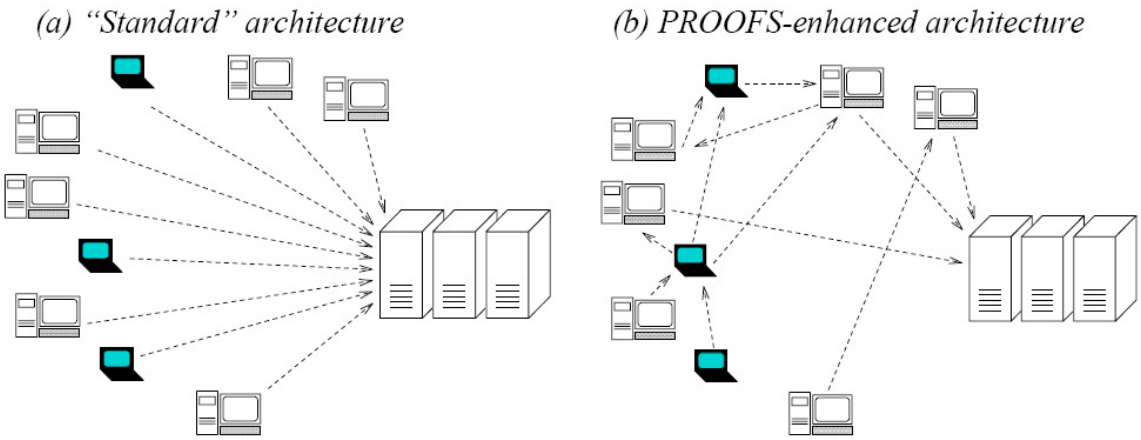


Figure 3.1: PROOFS operation for timely object location and retrieval

3.2 PROOFS Architecture

PROOFS is a randomized P2P system that consists of two protocols. The first forms and maintains a network overlay that connects the clients that participate in the P2P protocol. The overlay construction is an ongoing, randomized process in which nodes continually exchange neighbors with one another at a slow rate. The second protocol performs a series of randomized, scoped searches for objects atop the overlay formed by the first protocol. Objects are located by randomly visiting sets of neighbors until a node is reached that contains the object.

3.2.1 Design Description

In this section, we describe the application for which the PROOFS system was designed and describe the details of that design. PROOFS purpose is to provide timely delivery of web objects that are stored at locations whose availability is compromised as a result of a heavy request load for the objects. Proofs was designed with the following design objectives:

- Minimize operational complexity: Each client should be responsible for performing a small number of simple tasks (perhaps repeated several times). A flow-chart diagram of a node's operation should be short and simple, minimizing the likelihood of implementation error.

- Minimize state: To form an overlay, clients must maintain a list of neighbors that can be contacted for the purposes of a search. Clients also cache those pages that are of interest to them. It is preferred not to require clients to maintain additional state for purposes such as monitoring or sharing of network statistics, or for the caching of objects not explicitly requested by that client. Furthermore, the state should be "soft" in that any incorrect perceptions about the operating environment do not prevent the system from performing its task (but may decrease system efficiency) such that this state can simply expire with time.

- Limit recovery code: often protocols require additional complexity to "heal", e.g., recover from network partitions or adapt the overlay to dynamic changes in membership (clients joining/leaving the overlay). We wish to remove any such additional mechanism except for what is required to bootstrap the system into operation.

- Naturally cope with limited participation: some clients may refuse to deliver cached copies of objects. Others may refuse to forward queries, and worse yet, some clients may not wish to reveal their refusal to assist. The system should continue to function properly and efficiently even as these non-participants grow to significant, but not overwhelming proportions.

- Robustness of functionality that can be offered (e.g., the ability to perform searches for documents whose creation time is temporally restricted).

Figure 3.1 pictorially demonstrates how PROOFS alleviates symptoms associated with flash crowds. In Figure 3.1(a), a set of end-systems is attempting to receive the same object at roughly the same time directly servers containing the object. DNS requests issued by these end-systems would point these systems to a small set of servers from which they can receive the object. For instance, a query to DNS for cnn.com returned 6 IP addresses to which http queries can be transmitted. While deployed content delivery solutions are able to offset the load imposed on these servers for sub-objects (JPGs, ads, etc.), they are unable to offset the load for the original request for what is often referred to as the *container page*. Figure 3.1(a) is an example where several end-systems place temporally-adjacent requests for the same object, overloading the capacity of the servers. As a result, the majority of requests remain unanswered and fail, frustrating many of the users that placed the request. In Figure 3.1(b), with the PROOFS system activated, end-systems can query other end-systems for a copy of the object. Presumably, the system works well if end-system queries for the object reach end-systems that have already obtained a copy of the requested object. Our design leverages off the theoretical and simulation results in [143] that analytically demonstrate that in theory, randomized scoped searches between P2P participants all looking for the same popular object scale well in terms of number of packets transmitted and time taken to retrieve the object.

3.2.2 PROOFS Design

Here we consider the architectural design of the PROOFS system without attempting to optimize its performance in any way whatsoever, i.e., no functionality is added beyond what is necessary to make it functional and robust. There are two components to the system, the *client* and the *bootstrap server*. From the perspective of PROOFS (without optimizations added), clients are a set of homogeneous end-systems that form the P2P overlay and are

used to send searches. Bootstrap servers provide a means by which clients can learn about and gain access to the overlay. In our current implementation, we utilize a single bootstrap server. For the system to be robust, it is likely necessary to have multiple bootstrap servers. Below, we limit discussion to a system that contains a single bootstrap server. We briefly discuss some straightforward ways to provide multiple servers in Section 3.5. A detailed exploration is beyond the scope of this paper.

Each PROOFS client runs two protocols, `ConstructOverlay` and `LocateObject`. `ConstructOverlay` is responsible for determining which sets of clients a client is permitted to query when searching for objects. `LocateObject` is the protocol that participates in searches upon the overlay network formed by `ConstructOverlay`. `ConstructOverlay` is in essence the passive component, running continually, whereas `LocateObject` runs only when flash crowd phenomena exist within the network. Below, we give brief descriptions of these two protocols. These protocols rely heavily on randomness to be both simple and robust. All communications between clients occur at the IP level, i.e., each client has an IP address and port that it uses to send and receive communications. The underlying routing system is not of concern in this paper.

`ConstructOverlay`

When a client wishes to participate in the PROOFS system, the `ConstructOverlay` protocol first contacts a bootstrap server to obtain a preliminary list of *neighbors* (an IP address:port combination). A client A's neighbors are the set of nodes with which it is permitted to initiate contact. Hence, if the P2P overlay is viewed as a graph G in which the set of clients are the nodes, then the neighbor relation is indicated via a directed edge. Because we use directed edges, it is possible for node B to be node A's neighbor (such that A can initiate contact with B) while A is not B's neighbor (such that B can only communicate with A directly by responding to A). This set of neighbors is the only state maintained by the `ConstructOverlay` protocol that varies with time. There is a fixed bound, C, on the

maximum number of neighbors that a client will maintain.

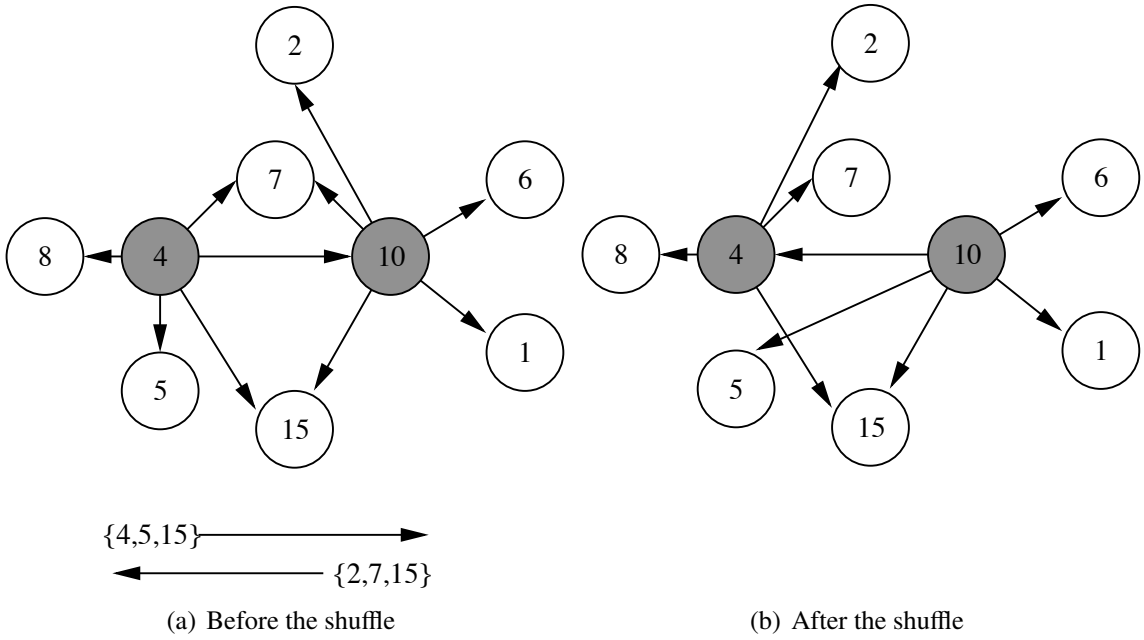
Clients continually perform what is called a *shuffle* operation. The shuffle is an exchange of a subset of neighbors between a pair of clients and can be initiated by any client. The client C_1 that initiates a shuffle chooses a subset of neighbors of size c that is no greater than its current number of neighbors. It selects one neighbor, C_2 from this subset and contacts that neighbor to participate in the shuffle. C_1 sends the subset of neighbors it selected with C_2 removed from the subset and C_1 added. If C_2 accepts C_1's shuffle, it selects a subset of neighbors from its list of neighbors and forwards this subset to C_1. Upon receiving each other's subsets of neighbors, C_1 and C_2 update their respective neighbor sets by including the set of neighbors sent to them. The replacement is done according to three rules:

1. No neighbor appears twice within the set.
2. A client is never its own neighbor.
3. Empty space in the neighbor cache before overwriting previous entries
4. Neighbors in the neighbor cache can only be overwritten (i.e., removed) if they were sent to the other neighbor during the shuffle.

A sample shuffle operation is shown in Figure 3.2. There, clients are represented by numbered circles. Directed edges indicate the neighbor relation, where an arrow pointing from A to B means that B is a neighbor of A. Neighbors are depicted only for the darkened clients numbered 4 and 10. These nodes start with the set of neighbors depicted in Figure 3.2(a) and end with the set of neighbors depicted in Figure 3.2(b).

Note two important points: Any node that was pointed to by somebody is still pointed to by somebody except the receiving end of the shuffle, and if A was B's neighbor and B initiated a shuffle with A, then after the shuffle, B is A's neighbor (i.e., the edge reverses direction).

In our current implementation, a client waits a random amount of time sampled from an exponential distribution. A shuffle request is only rejected by neighbors that have placed



(a) Before the shuffle (b) After the shuffle

Figure 3.2: An example of a shuffle operation

a request to shuffle but have not yet received a response. Upon receiving a rejection (or a timeout), a client continues the process of choosing the next time to initiate the shuffle from a uniform distribution. The rejection must be explicitly acknowledged. Clients that do not respond to shuffle requests are assumed to be inactive (i.e., are no longer part of the overlay) and are removed from the requesting client's cache.

Shuffling is used to produce an overlay that is "well-mixed" in the a client's neighbors are essentially drawn at random from the set of all clients that participate in the overlay. There is no attempt to optimize the overlay such that neighbors are topologically adjacent. There is no reason to ever terminate the shuffling operation. Once a "random" state is reached, additional shuffles will keep the overlay in a "random" state.

`LocateObject`

The `LocateObject` protocol is the protocol that attempts retrieval of the desired object by searching among the participating clients that are connected together by the overlay that was

constructed using the `ConstructOverlay` protocol. Once a client decides to use PROOFS to retrieve an object (how such a decision can be made is discussed in Section 3.5), a *query* is initiated at the client. A query contains the following information:

- **Object**: a description of the object being searched for. In our current implementation, the description is restricted to the URL that describes the original location of the page. However, the description can easily be extended to handle more sophisticated queries (keywords, temporal specifications, etc.) since the set of locations searched are independent of the object specification.
- **TTL**: a counter that counts the maximum number of additional hops in the overlay that the query should propagate if a copy of the requested object has not been located.
- **fanout**: a value f that indicates to how many neighbors a client should forward a query that it has received when it does not have a copy of the requested object (assuming the TTL has not expired).
- **Return Address**: the address of the client that initiated the query such that once a suitable object is located, it can be returned directly.

When a client receives a query or initiates a query from another client, it first checks to see if it contains a copy of the requested object in its object cache. If so, it forwards the object to the return address specified in the query. Otherwise, it decrements the TTL of the query, and if the TTL is non-negative, randomly selects f neighbors from its neighbor cache and forwards the query with the decremented TTL to those neighbors. Neighbors that receive the query are expected to acknowledge receipt by sending an ACK packet back to the client that forwarded the query. If no ACK is returned from a client then another client is selected at random and the query is instead forwarded to that client.

If a client that initiates a query does not receive a copy of requested object after a certain period of time, the client assumes that no clients reached by the query had a copy of the

object and repeats the query. Currently, we increment the TTL value by one each time a query fails until reaching a given value. Because each search is randomized, even the first few hops of the new query can visit clients that were not visited on previous queries. The time time a client waits between subsequent queries is $t\mathcal{T}$, where t is the TTL of the query and $\mathcal{T}$ is some rough estimate of propagation delay. The size of the TTL must be chosen carefully. The number of visits to clients grows exponentially in the size of the TTL, so rapid increases in its value can cause unnecessary flooding within the network. $\mathcal{T}$ must also be set carefully: large values increase potential waiting times, but small values can lead to the initiation of new queries prior to the completion of previous queries that may yet return a copy of the object. We investigate how the setting of $\mathcal{T}$ affects retrieval times and traffic levels in Section 3.4, and discuss ways to avoid traffic flooding in Section 3.5.

3.3 Robustness

In this section, we evaluate the natural robustness of PROOFS, meaning that no additional functionality is introduced beyond what is discussed in Section 3.2.1 that implements the most basic functions needed by the protocol. Even in this elementary form, we demonstrate that PROOFS operates in a robust manner even in environments that contain the two following anomalies:

- **Joins/Leaves**: One expects that over time, clients will join and leave the PROOFS system, and leaves may occur without warning.
- **Pseudo-participants**: There may exist clients that wish to retrieve objects using the PROOFS system but not participate in assisting other clients within `LocateObject`. We envision four types of possible participants:
  - Type I: these participants are "well-behaved" in that they accept queries, respond with the object when they have it, and forward queries further when they do not.

- Type II: these participants act as if they do not have a copy of the object even if they have previously received the copy. They forward queries as specified by the `LocateObject` Protocol.
- Type III: these participants are passive. Upon receiving a query, the participant neither responds with the object nor does it contact an additional f of its neighbors with a query whose TTL has been decremented. Instead, it randomly chooses a single neighbor and forwards the query without decrementing the TTL to that neighbor. In effect, the client acts as if the query was not sent to it but to another node in its place.
- Type IV: these participants receive queries (acting as good neighbors) but do not forward queries further, nor do they respond with copies of the object if available.

Clients that do not participate in the `ConstructOverlay` Protocol maintain a fixed set of neighbors. This will only hurt their own performance as some of these neighbors may leave the system.Note that under the bootstrapping process, these nodes are assigned as neighbors to other nodes and remain as parents until explicitly removed.

3.3.1 Graph Partitioning

We say that an overlay is **partitioned** if there exists a pair of nodes, n_1 and n_2 in the overlay where no path exists from n_1 to n_2. Such an occurrence would prevent any queries forwarded by n_1 from reaching n_2. In our discussions below, we will consider both partitions in the directed graph (that takes into account the directions of the edges) as well as partitions of the underlying undirected graph (where directions of edges do not matter). Clearly, if the undirected graph is partitioned, then the directed graph must be partitioned as well, but the reverse need not be true.

Partitioning of the undirected, connected graph is of particular concern. It is easy to show

that a partitioned undirected graph cannot be repaired via shuffling. In contrast, it is easy to show that a directed graph that is partitioned but whose underlying undirected graph is not partitioned can be repaired by shuffling. Our simulation results presented below demonstrate that in realistic network conditions, these types of partitions are usually short-lived and that the manner in which the graph is partitioned will rarely impact performance of PROOFS.

3.3.2 Theoretical Results

There are practical complications in maintaining an undirected overlay graph (where an edge permits bi-directional communication between the nodes it connects). In particular, when a node leaves the overlay, the nodes with which it connected must find new neighbors that are willing to take on new neighbors as well. This complication compels us to use an overlay whose edges are unidirectional. Unfortunately, it is conceivable that shuffling operations will partition the underlying overlay. However, we now present a theoretical result that demonstrates that any partitioning of the graph due to shuffling is only temporary. With probability one (given enough time), the shuffling process will eventually reconnect the separated participants. We emphasize that this theoretical result holds conclusively only when nodes do not leave the overlay.It is trivial to construct cases where leaving nodes can create a partition that cannot be healed without outside intervention. Subsequent simulation results will demonstrate that such permanent partitions are extremely rare.

The result is proved by considering the underlying undirected graph (i.e., removing the directions on edges in the overlay graph). We first prove that shuffling will not partition such a graph, and then show that if the underlying undirected graph is not partitioned, then eventually a path will exist from any node n_1 to another node n_2 within the directed graph.

An undirected graph G is said to be *connected* if a path exists between every pair of nodes, n_1 and n_2.

Lemma 1 *Let G be an undirected connected graph, and let G' be the graph that is derived from G by applying an arbitrary shuffle operation. Then G' is an undirected connected*

graph.

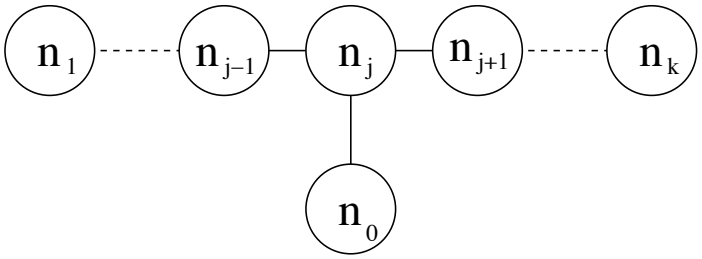
Proof: A shuffle consists of an exchange of a pair of m nodes. This exchange can be done by first removing those nodes that appear in both shuffle sets and then performing the exchange one node at a time (where an exchange might be in a single direction for the case where one node has fewer than m nodes in its cache to swap).Once duplicates are removed, the swapping operation is associative, i.e., the order in which nodes are actually exchanged does not alter the final outcome. Hence, we can restrict our attention to the case where two nodes exchange at most one entry. It follows from induction that if the graph remains connected after a single swap, it remains connected after all swaps performed within the shuffle.

Let $P = \langle n_0, \cdots, n_k \rangle$ be an arbitrary sequence of nodes that forms a path in G as depicted in Figure 3.3(a). Since we are considering a single swap, there are three cases to consider:

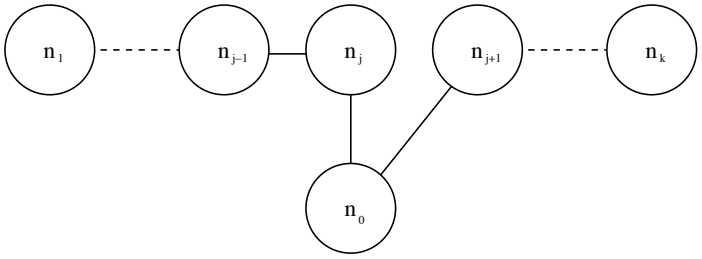
- Case 1: neither node implementing the swap lies on the path. This means that while there may be nodes on the path whose edges change (as a result of the swap), the changed edges connect to the nodes implementing the swap. Hence, no edges that form the path are changed, so the path remains after the swap is complete.

- Case 2: one node, n_j, that implements the swap lies on the path and the other lies off the path (call this other node n_0). Since the nodes that implement the swap are connected both before and after they perform the swap (but the direction of the edge changes within the directed graph), two possible scenarios occur: the node on the path swaps away no edges or swaps away one edge. As can be seen in Figure 3.3(b), a path between n_1 and n_k remains after the swap.

- Case 3: both nodes lie on the path. It follows that if n_j and n_{j+m} are the nodes implementing the swap, then either n_j is a neighbor of n_{j+m} of n_{j+m} is a neighbor of n_j. In either case, there is an edge connecting n_j and n_{j+m} in the undirected graph. We can restrict our attention to the alternative path $\left\langle n_1, \cdots, n_j, n_{j+m}, \cdots, n_k \right\rangle$ connecting n_1 to

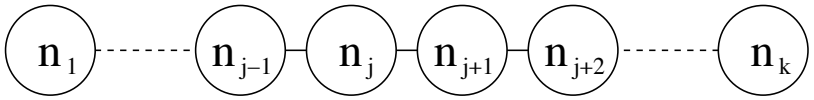


(a) Initial topology and no swaps

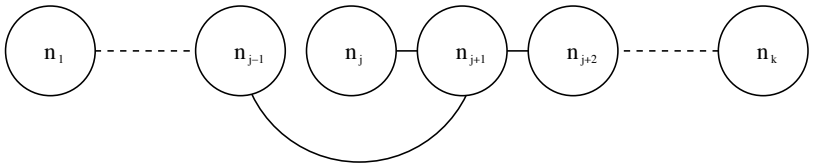


(b) One swap on the path

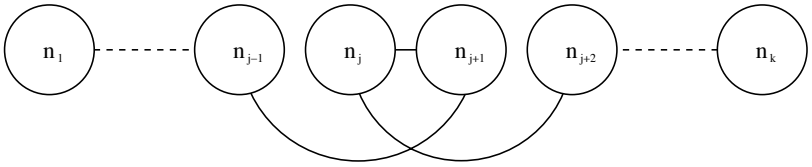
Figure 3.3: A generic path where the node being swapped with is off the path.



(a) Initial topology and no swaps



(b) One swap on the path



(c) Two swaps on the path

Figure 3.4: A generic path where the node being swapped with is on the path.

n_k in G. We use this path instead and relabel all $n_\ell, \ell > j$ as $n_{\ell-m+1}$ such that n_j and n_{j+1} are the nodes that implement the swap. Here, there are three cases to consider: a) no edges on the path are swapped between the nodes, b) n_j forwards to n_{j+1} its connection to node n_{j-1} and n_{j+1} forwards n_j a node that lies off the path or no node (this case also covers the case where the roles of n_j and n_{j+1} are reversed), and c) n_j forwards node n_{j-1} to n_{j+1} and n_{j+1} forwards edge n_{j+2} to n_j. As shown in Figure 3.4, for all three cases, the resulting graph remains connected. For case b), the new path skips over node n_j and for case c), the new path goes from n_{j-1} to n_{j+1} to n_j to n_{j+2}.

■

3.3.3 Non-cooperating nodes

We also evaluate the robustness of PROOFS in the presence of fraction of nodes that are non-cooperative in the search process. It is important to consider this scenario as in a large P2P overlay network, a node could either deliberately be non-cooperative or it may be faulty and hence unable to participate in the search process. In either case, it is important that the designed system still remains operative, even though it may be more inefficient in terms of bandwidth utilization and response time required in the search process.

Using experimentation, we evaluate the impact of fractions of nodes being non-cooperative on the bandwidth utilization and response latency. We examine whether the search process succeeds within reasonable performance degradation of PROOFS. We consider three types of non-cooperating nodes. Query-only non-cooperation refers to the case in which a node behaves as if it does not have the document and forwards the queries to its neighbors. In tunnel-only non-cooperation, a node is passive in that it does not respond to the query, and simply forwards to exactly one of its neighbors. Mute non-cooperation refers to the case where the nodes do not respond to the query, nor do they forward the query to their neighboring nodes. Figures 3.6, 3.7 and 3.5 depict the performance of our system when

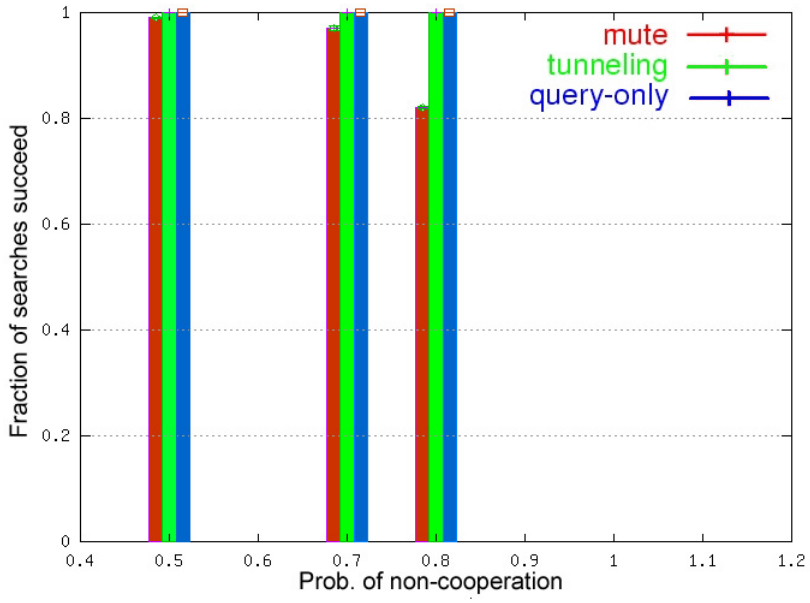


Figure 3.5: Success probability for search requests and for different types and percentage of non-cooperative nodes. Notice that for most cases 98% of the requests are completed successfully.

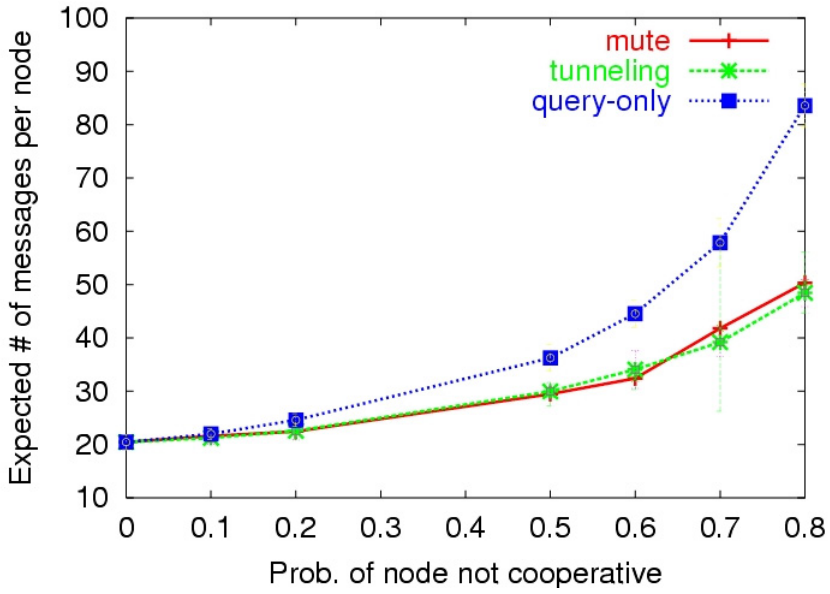


Figure 3.6: Number of messages for 180 clients, non-collaborative nodes and simultaneous searches.

there are nodes that have non-collaborative behavior. In [158], we present an even more elaborate theoretical analysis for the non-collaborating nodes.

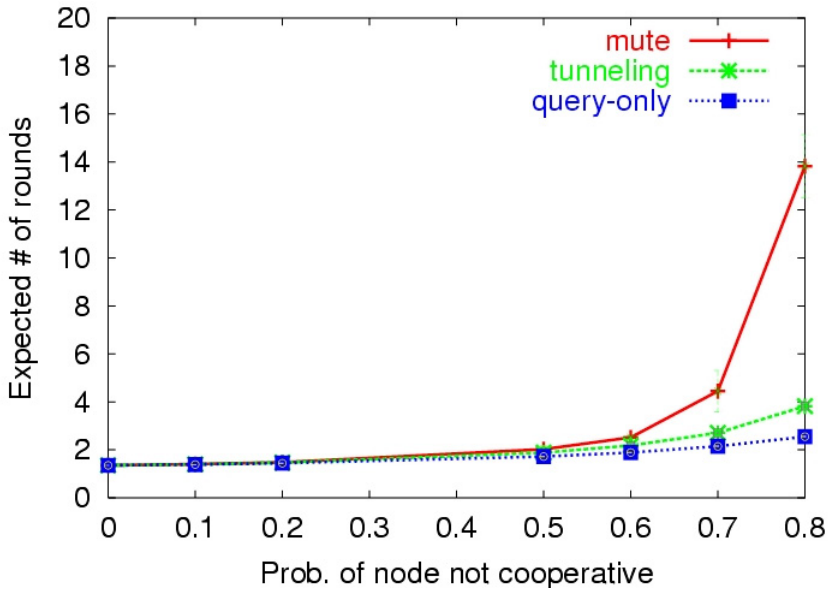


Figure 3.7: Number of rounds for 180 clients, non-collaborative nodes and simultaneous searches.

3.4 Experimental Results

In this section, we present results of our use of an experimental prototype within a wide-area network setting. Our experimental testbed consists of a variety of machines gathered at academic institutions around the globe. In all experiments, the times at which each client initiates shuffle operations are exponentially distributed with two minutes expected between shuffle intervals. We let the shuffling proceed for a half hour before initiating our experiments.

Figures 3.8 and 3.9 plot results of 8 experiments using an overlay consisting of 180 clients. In each experiment, a single client starts with a copy of the object. All other clients simultaneously search for that object. Figure 3.8 plots, for each experiment, the average number of query requests received by all clients, as well as the maximum number

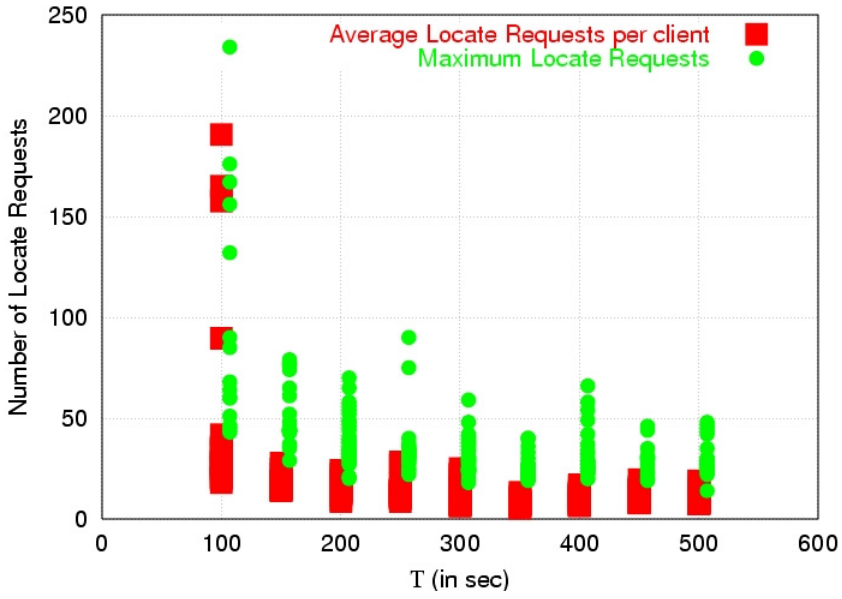


Figure 3.8: Traffic Levels, for 180 clients, simultaneous searches

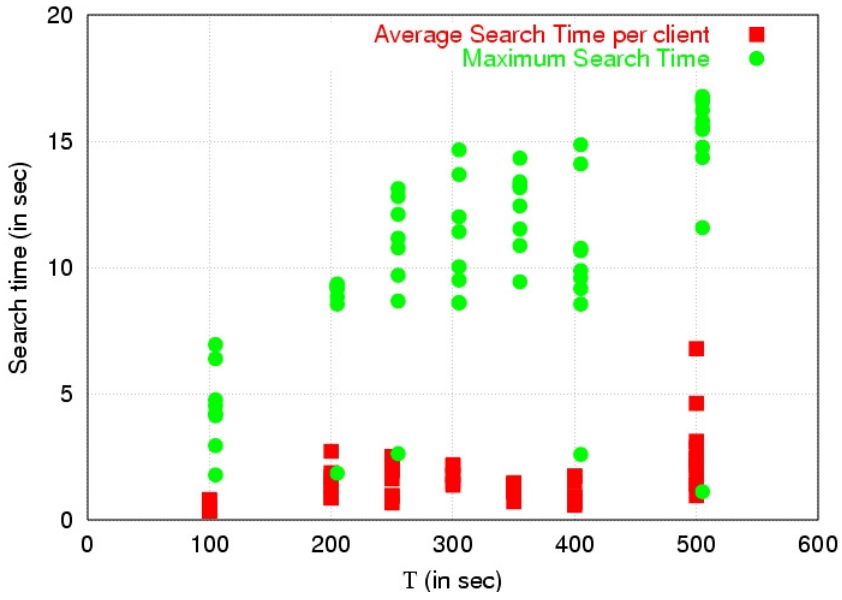


Figure 3.9: Delivery time for 180 clients, simultaneous searches

of requests received over all clients. On the x-axis, we vary $\mathcal{T}$, where a client waits $\mathcal{T} \cdot t$ milliseconds after initiating a query with TL t before initiating its next query. Figure 3.9 plots the corresponding average and maximum times taken from the time a client's search is initiated to the time that the client retrieves the object.

3.5 Discussion

The appeal of PROOFS is the simplicity, scalability, and robustness of its basic architecture. The fact that often nodes receive redundant copies of queries does increase the levels of traffic it adds to the network.. However, this redundancy proves to be helpful in naturally prevent partitions and allows the system to operate effectively even when a large fraction of clients limit their participation.

While we have demonstrated PROOFS' ability to and robustly deliver objects under heavy demand, we have not evaluated the potential damages to the network via misuse or intentional abuse. PROOFS' scalability relies on the fact that the object a client searches for is also being searched for by many other clients in the network.

There are several ways to go about optimizing the manner in which the overlay is constructed that could potentially improve the protocol's performance. Our simulations assumed that all clients had identical capabilities and our experiments were conducted upon well-connected, well-provisioned end-systems at academic institutions. One immediate direction of future work is to determine how to construct overlays upon which randomized searches proceed efficiently through the overlay graph with an increased use of high bandwidth clients and a reduced use of low bandwidth clients.

One example is using the method described in [129] to generate bounded-diameter overlays. Another would be to give preference to neighbors who are nearby (either topologically, via hop-count, or end-to-end delay). A third is to focus design toward graphs that exhibit small world phenomena. It would also be interesting to theoretically prove results involving

the "shape" of the graphs generated by shuffling (are they truly random, and if so, how quickly do they converge to a random shape?) It would also be of interest to analytically model the behavior of this type and other protocols as membership, levels of participation, and processing capabilities of clients are varied.

In practice, it is necessary to limit the amount of flooding caused by searches that are not looking for popular content. We envision two simple ways to control such flooding:

- Place limits on the rate at which clients are willing to service queries. If all clients bound the rate at which they process queries by some fixed r, then each client can only inject queries into the network at a maximum rate of fr (the rate can be lower due to queries for which a copy of the object can be returned). This can lead to a high query drop rate (i.e., processing only one out of every f queries), worsening performance. However, it should effectively bound the amount of traffic that PROOFS adds to the network, irrespective of the number of clients searching or participating in the overlay. Second, we have run other sets of simulations (not presented here) in which each client participating in the overlay drops requests with a probability of p. The results are more favorable than what we have observed when a fraction, p, of clients drop all requests. Hence, PROOFS should continue to locate objects efficiently even when the query rate modestly exceeds r.

- Place limits on the maximum TTLs for queries. Large TTLs are required when few clients are searching for an object so that their queries cover the majority of nodes in the overlay. In contrast, when numerous clients search for a common object, repeated searches with small TTLs will spread the objects around the overlay quickly as a result of the overlay's randomly connected structure. Hence, the number of small scoped searches that find the object is expected to grow exponentially with time.

The following is a list of open issues that still require further investigation:

- **Search Initiation**: We are interested in automating PROOFS inside of a web browser

to automatically retrieve objects during flash crowd conditions. A reasonable approach is to first attempt to contact the server and, after a short timeout, initiate `LocateObject`.

- **Insider DoS attacks**: Protocols that fan out requests can be used to generate large amounts traffic in the network by placing bogus queries. While the rate-limiting and TTL-bounding techniques can protect the rest of the network from being overwhelmed with queries, the generation of a large number of bogus queries can suppress the ability to service valid queries. One fix would be to prioritize the servicing of queries to sites that are more likely to legitimately contain flash crowds such as news sites. Another possibility is to prioritize service of the more frequently occurring queries, which are more likely to be legitimate.

- **Unavailable Objects**: When an object does not reside anywhere within the overlay, it cannot be retrieved, nor replicated at intermediate points in the search space. This means that searches for that object will flood the system. Handling this case remains an open problem. We point out that this problem also exists within generation approaches that rely on caching to prevent flooding the focal point of a directed query for a popular object [41, 134, 142].

- **Neighbor Proximity**: We have not made any attempt whatsoever to shape the overlay to the underlying network topology. We find that clients can recover popular objects in a small number of seconds upon an overlay produced by simple shuffle operations. It remains to be seen whether or not optimizing overlay topology can significantly reduce search times. However, the fact that the presented search time is the minimum time over a fairly large set of searches leads us to believe that large improvements are unlikely.

- **Failed Bootstrap Server**: Having a single bootstrap server is a limitation that is easily addressed by replicating the bootstrap server at several fixed locations.

- **Object verification**: A client participating in the overlay could easily transmit a fake copy of the requested object upon receiving a query. For sites that are visited frequently, a browser could obtain a copy of a public key used by the site before a flash crowd arrives at the site. By including a unique certificate into an object (such as an MD5 digest of the object [138]), encrypted by the originating site's private key, this certificate could be used to verify that an object did indeed originate from where it was claimed to have originated from.

- **Support for Dynamic Content**: The design of PROOFS was geared towards mitigating flash-crowds or Denial of Service against static (non-dynamic) objects. Therefore, it fails to handle dynamic objects that depend on information on a back-end server including PHP or MYSQL database. For dynamic objects, we need to define something that can mitigate the DDoS or flash-crowd effect on the database back-end in addition to the front-end database. In the next chapter we consider such a system for Web services.

Finally, we note that it is convenient to have connections between neighbors in the overlay maintained via open TCP connections so that we need not worry about lost messages. Since these connections are bidirectional, it would be worth considering allowing them to be used by both endpoints, effectively making the overlay graph an undirected graph. In theory, since directed overlays can partition much more frequently than their underlying undirected structures, making each edge in the overlay graph bidirectional would improve clients' expected reachabilities. Exploring the trade-offs between that complexity of having bi-directional connections and graph partitioning is an interesting open problem which is not part of our focus in this thesis.

Chapter 4

Enhancing the Availability of Dynamic Web Services using WebSOS

4.1 Introduction

The Web is increasingly being used for different kinds of services and interactions with, and between humans. Beyond displaying static content such as home pages or academic papers, the web is actively used for such diverse tasks as e-mail, banking, consumer purchasing, marketing, stock-quote dissemination and trading, and real-time communication. The wide availability of high-quality browsers and servers, as well as programmers' and users' familiarity with the tools and concepts behind web browsing ensure that ongoing creation of additional services.

We introduce *WebSOS,* an adaptation of the *Secure Overlay Services (SOS)* architecture [91]. Our intent is to prevent congestion-based DDoS attacks from denying *any* user's access to web servers targeted by those attacks. The novel aspects of WebSOS are *(a)* its use of *graphic Turing tests* in *lieu* of (or in addition to) strong client authentication (as was proposed in SOS) to distinguish between human users and automated attack *zombies*, and *(b)* its transparency to browsers and servers, by taking advantage of browser extensibility.

Although WebSOS itself protects only web traffic, it can be used to enable routing of other types of traffic by establishing IPsec tunnels through the overlay; the web-based authentication is leveraged to create a channel for other traffic, as we showed in [18]. We should also add that we envision WebSOS used when an attack is detected, *i.e.,* when a client cannot directly reach the destination web server. Thus, when no attacks are underway, the impact of WebSOS in network performance and other overheads is zero. An additional design goal of WebSOS, which was achieved, was avoiding changes to protocols and network elements (such as routers).

WebSOS protects the portion of the network immediately surrounding attack targets (*i.e.,* the web servers) by high-performance routers that aggressively filter and block all incoming connections from hosts that are not approved, as shown in Figure 4.1. These routers are "deep" enough in the network that the attack traffic does not adversely impact innocuous traffic, typically in an ISP's Point Of Presence (POP). The identities of the approved nodes is kept secret so that attackers cannot impersonate them to pass through the filter. These nodes (which can be as few as 2 or 3) are picked from a set of nodes that are distributed throughout the wide area network. This super-set forms a *secure overlay*: any transmissions that wish to traverse the overlay must first be validated at any of the entry points of the overlay using either a cryptographic authentication mechanism, or a Graphic Turing test to distinguish humans from attack scripts [48]. Once inside the overlay, the traffic is tunneled securely to one of the approved (and secret from attackers) nodes, which can then forward the validated traffic through the filtering routers to the target. Thus, there are two main principles behind our design. The first is the elimination of communication pinch-points, which constitute attractive DoS targets, via a combination of filtering and overlay routing to obscure the identities of the sites whose traffic is permitted to pass through the filter. The second is the ability to recover from random or induced failures within the forwarding infrastructure or the secure overlay nodes. In [91], we gave an analytical model for the resilience of SOS against attackers that concentrate their attacks on the overlay nodes. We summarize these

results in Section 4.2.4.

To demonstrate the potential of WebSOS as a value-added serve, we also extended the basic WebSOS architecture to use a credential-based micropayment scheme that combines access control and payment authorization. Our architecture allows ISPs to accurately charge web clients and servers. Clients can dynamically decide whether to use WebSOS, based on the prevailing network conditions (*e.g.,* the client can try to contact the web-server through WebSOS if a direct connection fails, either through manual user intervention or by detecting the connection failure in our client-side proxy). Although practically any micropayment system can be used in our model, we chose a payment system that can inter-operate with WebSOS' distributed architecture and provide the necessary user credentials. OTPchecks [77] encompasses all these properties: it is a simple distributed scheme, intended for general Internet-based micropayments that produces bank-issued users' credentials which can in turn used to acquire small-valued payment tokens. It has very low transaction overhead and can be tuned to use different risk strategies for different environments making it a suitable payment solution for a wide range of on-line services. Although the pay-per-use component of the architecture is technically independent of the DoS protection guarantees, we feel it is necessary to at least investigate the "market friendliness"[1] of any new proposed anti-DoS system.

WebSOS is the first instantiation of the SOS architecture. We use this instantiation to evaluate the performance of the underlying overlay routing mechanism both in a local area scenario and over the Internet using the PlanetLab testbed [133]. The results show that the average increase in end-to-end latency is a factor of 2 to 3 beyond what is achieved using the standard web infrastructure. We believe this modest increase is an acceptable alternative to providing no service. Such a service can be used on an as-needed basis, and hence need not impact performance when no attack is in progress. These results validate our simulation analyzes, where we used real ISP topologies to determine the added average

[1] Of course, market friendliness involves more that the ability to charge a user, which is a necessary but not sufficient condition for deployment.

latency imposed by the WebSOS mechanism.

4.1.1 WebSOS Architectural Scope

DoS attacks can take many forms, depending on the resource the attacker is trying to exhaust. For example, an attacker can try to cause the web server to perform excessive computation, or exhaust all available bandwidth to and from the server. In all forms, the attacker's goal is to deny use of the service to other users. Apart from the annoyance factor, such an attack can prove particularly damaging for time- or life-critical services (*e.g.,* tracking the spread of an real-world epidemic), or when the attack persists over several days[2]. Of particular interest are *link congestion* attacks, whereby attackers identify "pinch" points in the communications substrate and render them inoperable by flooding them with large volumes of traffic. An example of an obvious attack point is the location (IP address) of the destination that is to be secured, or the routers in its immediate network vicinity; sending enough attack traffic will cause the links close to the destination to be congested and drop all other traffic. *It is such attacks that WebSOS was designed to address. Solving the much harder general denial-of-service problem where attackers could potentially have enough resources to physically partition a network is not addressed in this paper.* Furthermore, we do not consider algorithmic denial of service attacks [40].

We assume that attackers are smart enough to exploit features of the architecture that are made publicly available, such as the set of nodes that form the overlay. However, we do not specifically consider how to protect the architecture against attackers who can infiltrate the security mechanism that distinguishes legitimate traffic from (illegitimate) attack traffic: we assume that communications between overlay nodes remain secure so that an attacker cannot send illegitimate communications, masking them as legitimate. In addition, it is conceivable that more intelligent attackers could monitor communications between nodes in

[2]In one instance of a persistent DoS attack, a British ISP was forced out of business because it could not provide service to its customers.

the overlay and, based on observed traffic statistics, determine additional information about the current configuration.

4.2 The WebSOS Architecture

Because our approach is based on the Secure Overlay Services (SOS) [91] architecture, we first highlight its important aspects. We also briefly describe Graphic Turing tests, which implement human-to-overlay authentication, and the microbilling architecture we integrated in WebSOS. We close this section with a description of WebSOS itself, and the sequence of operations in using it.

4.2.1 Overview of SOS

Attackers in the network are interested in preventing traffic from reaching the target. These attackers have the ability to launch DoS attacks from a variety of points around the wide area network. The number and bandwidth capabilities of these compromised locations determine the intensity with which the attacker can bombard a node with packets, to effectively shut down that node's ability to receive legitimate traffic. Knowledge of the target's IP address is all that is needed in order for a moderately-provisioned attacker to saturate the target site.

Fundamentally, the goal of the SOS infrastructure is to distinguish between authorized and unauthorized traffic. The former is allowed to reach the destination, while the latter is dropped or is rate-limited. Thus, at a very basic level, SOS requires the functionality of a firewall "deep" enough in the network that the access link to the target is not congested. This imaginary firewall performs access control by using protocols such as IPsec [90]. This generally pre-supposes the presence of authentication credentials (*e.g.,* X.509 [29] certificates) that a user can use to gain access to the overlay. *We consider this one of the the largest drawbacks to SOS, as it precludes casual access to a web server by anonymous, yet benign users.* In WebSOS, we extend the authentication model to include Graphic Turing

Tests, thereby allowing previously-unknown humans to access a protected site without allowing automated attacks to do the same. Both types of authentication (cryptographic and GTT based) can be used simultaneously in WebSOS, allowing pre-authorized automated processes to access the overlay in times of attack. Note that when the system is not under attack, automated processes such as Google indexing (as well as human users) can access the site directly.

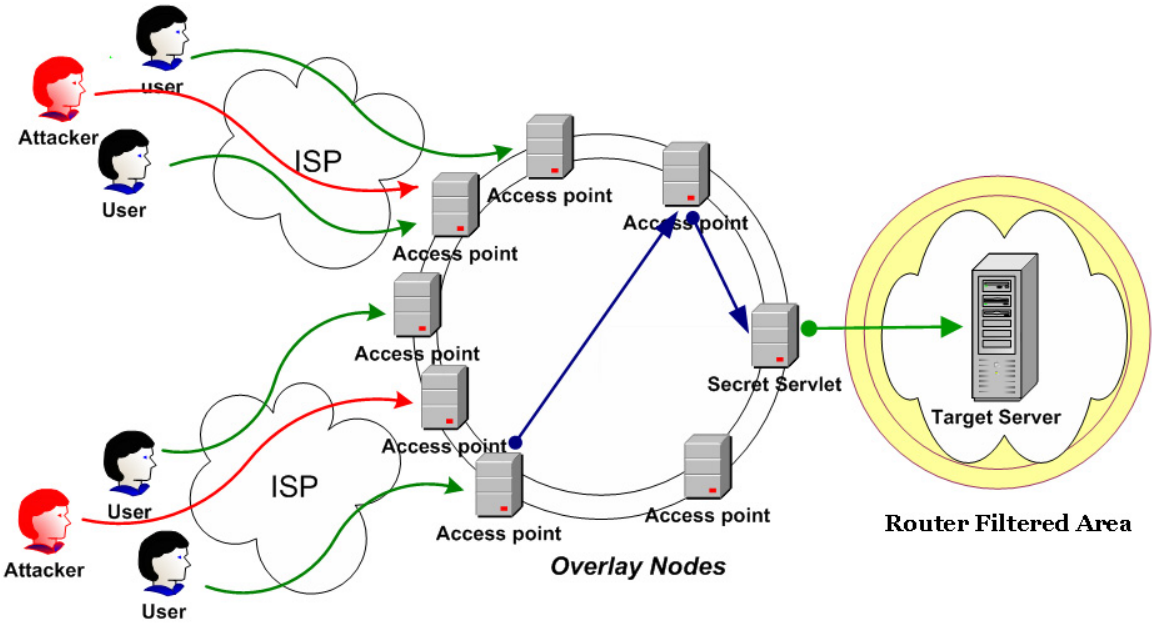


Figure 4.1: Basic SOS architecture. Access Points represent an entry point to the SOS overlay. SOS nodes can serve any of the roles of secure access point, beacon or Secret Servlet.

Since traditional firewalls themselves are susceptible to DoS attacks, what is really needed is a distributed firewall [17, 78]. To avoid the effects of a DoS attack against the firewall connectivity, instances of the firewall are distributed across the network. Expensive processing, such as cryptographic protocol handling, is farmed out to a large number of nodes. However, firewalls depend on topological restrictions in the network to enforce access-control policies. In what we have described so far, an attacker can launch a DoS attack with spoofed traffic purporting to originate from one of these firewalls, whose identity

cannot be assumed to remain forever secret. The insight of SOS is that, given a sufficiently large group of such firewalls, one can select a very small number of these as the designated authorized forwarding stations: only traffic forwarded from these will be allowed through the filtering router. In SOS, these nodes are called *secret servlets.* All other firewalls must forward traffic for the protected site to these servlets. Figure 4.1 gives a high-level overview of a SOS infrastructure that protects a target node or site so that it only receives legitimate transmissions. Note that the secret servlets can change over time, and that multiple sites can use the same SOS infrastructure.

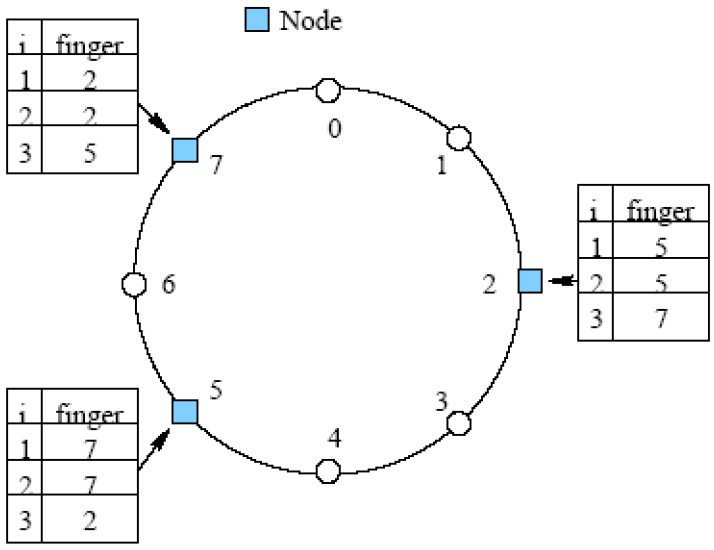


Figure 4.2: Chord-based overlay routing.

To route traffic inside the overlay, SOS uses Chord [162], which can be viewed as a routing service that can be implemented atop the existing IP network fabric, *i.e.,* as a network overlay. Consistent hashing [85] is used to map an arbitrary identifier to a unique destination node that is an active member of the overlay.

Each overlay node maintains a table that stores the identities of m other overlay nodes. The i^{th} entry in the table is the node whose identifier x equals or, in relation to all other nodes in the overlay, most immediately follows $x + 2^{i-1}(\pmod{2^m})$, as shown in Figure 4.2. When overlay node x receives a packet destined for ID y, it forwards the packet to the overlay node

in its table whose ID precedes y by the smallest amount. In the example, if node 7 receives a packet whose destination is the identifier 20, the packet will route from 7 to 16 to 17. When the packet reaches node 17, the next node in the overlay is 22, and hence node 17 knows that 22 is responsible for identifier 20. The Chord algorithm routes packets around the overlay "circle", progressively getting closer to the desired overlay node. $O(m)$ overlay nodes are visited. Typically, the hash functions used to map nodes to identifiers do not attempt to map two geographically close nodes to nearby identifiers. Hence, it is often the case that two nodes with consecutive identifiers are geographically distant from one another within the network.

The Chord service is robust to changes in overlay membership, and each node's list is adjusted to account for nodes leaving and joining the overlay such that the above properties continue to hold.

SOS uses the IP address of the target (*i.e.,* web server) as the identifier to which the hash function is applied. Thus, Chord can direct traffic from any node in the overlay to the node that the identifier is mapped to, by applying the hash function to the target's IP address. This node, where Chord delivers the packet, is not the target, nor is it necessarily the secret servlet. It is simply a unique node that will be eventually be reached, after up to $m = \log N$ overlay hops, regardless of the entry point. This node is called the *beacon*, since it is to this node that packets destined for the target are first guided. Chord therefore provides a robust and reliable, while relatively unpredictable for an adversary, means of routing packets from an overlay access point to one of several beacons.

Finally, the secret servlet uses Chord to periodically inform the beacon of the secret servlet's identity. Should the servlet for a target change, the beacon will find out as soon as the new servlet sends an advertisement. If the old beacon for a target drops out of the overlay, Chord will route the advertisements to a node closest to the hash of the target's identifier. Such a node will know that it is the new beacon because Chord will not be able to further forward the advertisement. By providing only the beacon with the identity of

the secret servlet, traffic can be delivered from any firewall to the target by traveling across the overlay to the beacon, then from the beacon to the secret servlet, and finally from the secret servlet, through the filtering router, to the target. This allows the overlay to scale for arbitrarily large numbers of overlay nodes and target sites. *Unfortunately, this also increases the communication latency, since traffic to the target must be redirected several times across the Internet.* If the overlay only serves a small number of target sites, regular routing protocols may be sufficient.

Figure 4.3: Web Challenge using CAPTCHA PIX. The response to the challenge in this case is “baby or babies”.

4.2.2 Graphic Turing Tests

Graphic Turing Tests(GTTs) are tests designed to provide a way of differentiating a human from a machine by presenting the user with a set of images and asking a questions about

the content of the images. CAPTCHA (Completely Automated Public Turing test to Tell Computers and Humans Apart) is a program that generates and grade GTTs [168]. Our design exploits GTTs as a form of Reverse Turing Tests that enables us to detect automated scripts, preventing them from abusing our system. In practice, the type of RTT is not crucial. Any RTT that can be presented to the end-user will suffice to protect our system. For example, an RTT using speech is presented in [96] and in [144] the authors present an RTT using facial features.

The particular CAPTCHA realization we use is PIX. It consists of a large database of labeled images. All of these images are pictures of concrete objects (a horse, a table, a house, a flower, etc). The program picks an object at random, finds 6 random images of that object from its database, distorts them at random, presents them to the user and then asks the question ”what are these pictures of?” as shown in Figure 4.3. PIX relies on the fact that humans can relate the objects within the distorted image and current automated tools cannot. The human authenticates himself/herself by entering as the description of the object in ASCII text. Graphic Turing Tests are an independent component of our architecture and thus we can update it without changing any other component.

When a user passes the GTT, the access point issues a short-lived X.509 [29] certificate. This certificate is signed by the entity operating the overlay, authorizing the holders to access the web service. The certificate is set to expire after 30 minutes (configurable), and contains the IP address of the client (to avoid reuse by multiple zombies). The overlay securely proxies all traffic from the source to the target via one of the beacons, as before.

Although recent advances in visual pattern recognition [118] can defeat some of the CAPTCHAs, there is no solution to date that can recognize complicated images or relation between images like PIX or Animal-PIX. Although for demonstration purposes in our prototype we use PIX, we can easily substitute it with any other instance of graphic turing test in case a solution to the problem presented by this specific CAPTCHA is discovered.

4.2.3 OTPchecks Micropayment System

In addition to the Graphic Turing test, we also integrated an (optional) micropayment mechanism to WebSOS. Our goal was to demonstrate that it is possible to design DoS-protection mechanisms that offer incentives to ISPs (or other entities) to deploy and manage, as they offer a clear way of recouping the associated costs. Here, we provide some background on the microbilling system we used; we explain its use in WebSOS in Section 4.3.

The general architecture of our microbilling system [25] is shown in figure 4.4. The Check Guarantor plays the role of *Provisioning*, the Network User plays the role of *Payer*, and the Network Storage Provider (or another NU acting as an NSP) plays the role of the *Merchant*. *Clearing* is done either by a financial institution (if real money is used) or by a selected user of the system (when loyalty points or "play money" are used).

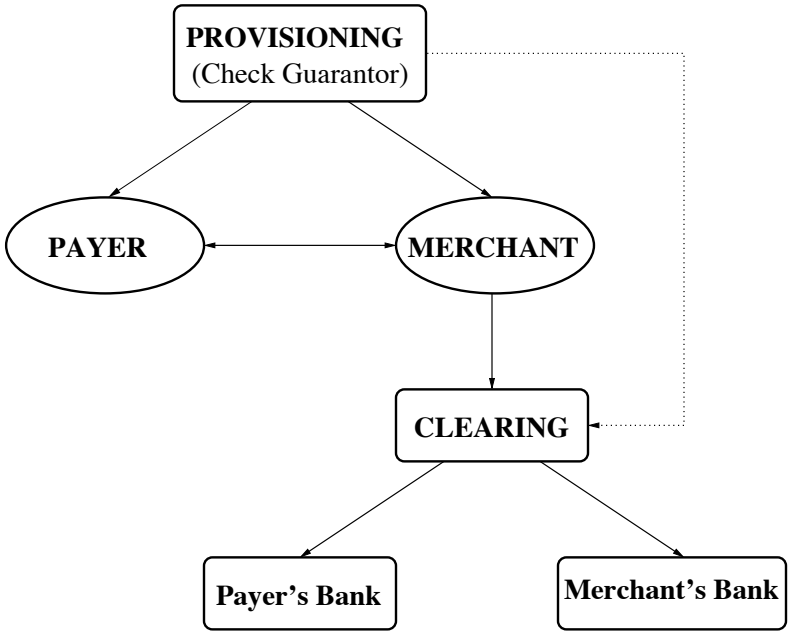


Figure 4.4: Microbilling architecture diagram. We have the generic terms for each component. The arrows represent communication between the two parties.

In this system, the *Provisioning* agent issues KeyNote [24] credentials to *Payers* and *Merchants*. These credentials describe the conditions under which a Payer is allowed to perform a transaction, and the fact that a Merchant is authorized to participate in a particular

transaction. When a Payer wants to buy something from a Merchant, the Merchant first encodes the details of the proposed transaction into an *offer* which is transmitted to the Payer.

If the Payer wishes to proceed, she must issue to the Merchant a microcheck for this offer. The microchecks are also encoded as KeyNote credentials that authorize payment for a specific transaction. The Payer creates a KeyNote credential signed with her public key and sends it, along with her Payer credential, to the Merchant. This credential is effectively a check signed by the Payer (the Authorizer) and payable to the Merchant (the Licensee). The conditions under which this check is valid match the offer sent to the Payer by the Merchant. Part of the offer is a nonce, which maps payments to specific transactions, and prevents double-depositing of microchecks by the Merchant.

To determine whether he can expect to be paid (and therefore whether to accept the payment), the Merchant passes the action description (the attributes and values in the offer) and the Payer's key along with the Merchant's policy (that identifies the Provisioning key), the Payer credential (signed by Provisioning) and the microchecks credential (signed by the Payer) to his local KeyNote compliance checker. If the compliance checker authorizes the transaction, the Merchant is guaranteed that Provisioning will allow payment. The correct linkage among the Merchant's policy, the Provisioning key, the Payer key, and the transaction details follow from KeyNote's semantics [24].

If the transaction is approved, the Merchant should give the item to the Payer and store a copy of the microcheck along with the payer credential and associated offer details for later settlement and payment. If the transaction is not approved because the limits in the payer credentials have been exceeded, then, depending on their network connectivity, either the Payer or the Merchant can request a transaction-specific credential that can be used to authorize the transaction. Observe that this approach, if implemented transparently and automatically, provides a continuum between online and offline transactions tuned to the risk and operational conditions.

Periodically, the Merchant will 'deposit' the microchecks (and associated transaction details) it has collected to the *Clearing and Settlement Center (CSC).* The CSC may or may not be run by the same company as the Provisioning, but it must have the proper authorization to transmit billing and payment records to the Provisioning for the customers. The CSC receives payment records from the various Merchants; these records consist of the Offer, and the KeyNote microcheck and credential from the payer sent in response to the offer. In order to verify that a microcheck is good, the CSC goes through the same procedure as the Merchant did when accepting the microcheck. If the KeyNote compliance checker approves, the check is accepted. Using her public key as an index, the payer's account is debited for the amount of the transaction. Similarly, the Merchant's account is credited for the same amount.

The central advantage of this architecture is the ability to encode risk management rules for micropayments in user credentials. Other electronic systems have focused on preventing fraud and failure, rather than on managing it. In many cases with such systems, the prevention mechanisms can be too expensive for micropayments, making the risk management approach particularly attractive.

4.2.4 Sequence of Operations in WebSOS

To illustrate the use of the WebSOS architecture by servers and clients, we describe the steps both sides must undertake to protect their communication channel (see Figure 4.5.For simplicity, we omit the micropayment component in this description; we describe the steps involved in a client using the micropayment scheme following this discussion.

- A site (target) installs a filter on a router in its immediate vicinity and then selects a number of WebSOS nodes to act as "secret servlets" that are allowed to forward traffic through the filter to the target site. Routers at the perimeter of the site are instructed to only allow traffic from these servlets to reach the internal of the site's network. These routers are powerful enough to do filtering using only a small number

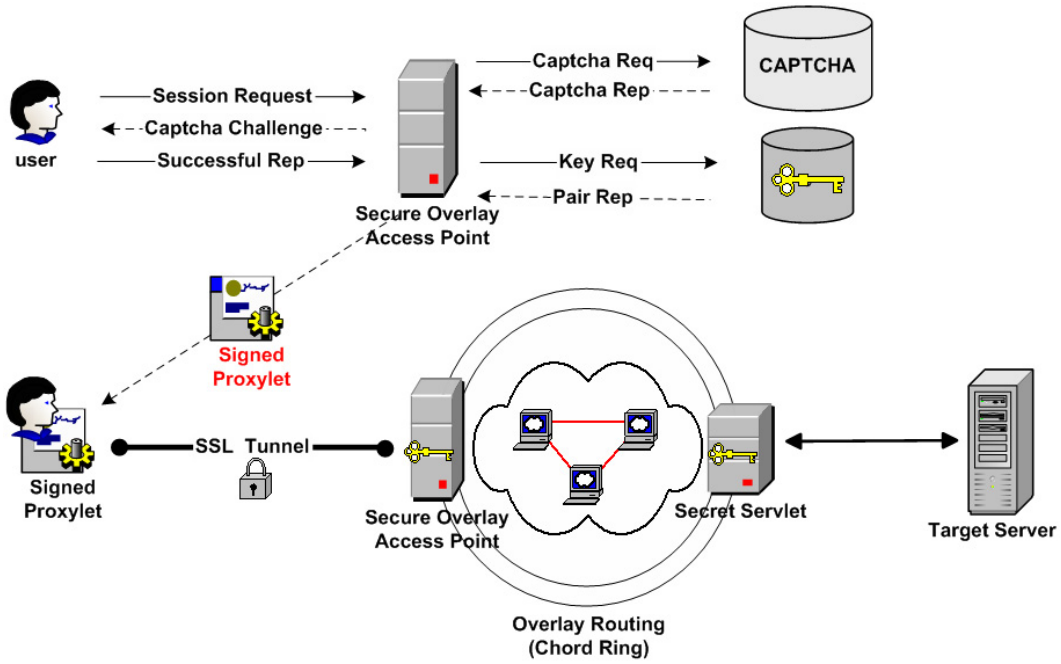


Figure 4.5: WebSOS connection establishment sequence of operations. The user is connected to an access point that in turn authenticates the user credentials and issues an X.509 certificate.

of rules on incoming traffic without adversely impacting their performance. In order to make guessing the identity of a secret servlet for a particular target harder for the attacker, the filtering mechanism uses packet fields with potentially high entropy. For example, only GRE [55] packets from a particular source (the secret servlet) containing a specific 32-bit value in the GRE Key field [50]. An attacker trying to slip attack traffic through the filter must guess not only the current servlet's IP address, but the correct 32-bit key as well. Although we expect 32 bits to be sufficient for this application, we can easily use larger keys to avoid brute-force attacks.

- When a WebSOS node is informed that it will act as a secret servlet for a site (and after verifying the authenticity of the request, by verifying the certificate received during the SSL exchange), it computes the key k for a number of well-known consistent hash functions, based on the target site's network address. Each of these keys will identify a number of overlay nodes that will act as beacons for that web server.

- Having identified the beacons, the servlets or the target will contact them, notifying

them of the servlets' association with a particular target. Beacons will store this information and use it for traffic-forwarding purposes.

- A source that wants to communicate with the target contacts a random overlay node, the Secure Overlay Access Point (SOAP). This is accomplished by controlling the forward DNS record of the source to respond with the set of SOAP nodes when a query for this service is requested by a client interested in the service. After authenticating and authorizing the request via the CAPTCHA test, the overlay node securely proxies all traffic from the source to the target via one of the beacons. The SOAP (and all subsequent hops on the overlay) can proxy the HTTP request to an appropriate beacon in a distributed fashion using Chord, by applying the appropriate hash function(s) to the target's IP address to identify the next hop on the overlay[3]. To minimize delays in future requests, the client is issued a short-lived X.509 certificate, bound to the SOAP and the client's IP address, that can be used to directly contact the proxy-server component of the SOAP without requiring another CAPTCHA test. The overall interaction of clients with the overlay is shown graphically in Figure 4.5. We shall explain the role of the signed proxylet in Section 4.5.

This scheme is robust against DoS attacks because if an access point is attacked, the confirmed source point can simply choose an alternate access point to enter the overlay. Any overlay node can provide all different required functionalities (Access Point, Chord routing, beacon, secret servlet). If a node within the overlay is attacked, the node simply exits the overlay and the Chord service self-heals, providing new paths over the re-formed overlay to (potentially new sets of) beacons. Furthermore, no node is more important or sensitive than others — even beacons can be attacked and are allowed to fail. Finally, if a secret servlet's identity is discovered and the servlet is targeted as an attack point, or attacks arrive at the

[3] As we shall see in Section 4.6, the SOAP actually uses the Chord ring to determine the identity of the secret servlet, and then proxies the traffic directly to that node. The SOAP queries in parallel all beacons for a particular destination for the identity of the secret servlet.

target with the source IP address of some secret servlet, the target can choose an alternate set of secret servlets.

Use of GRE for encapsulating the traffic between the secret servlet and the filtering router can offer an additional benefit, if we also use transparent proxies and IPsec for packet encapsulation between the proxies (replacing SSL). In that implementation scenario, as far as the target web server is concerned the HTTP/HTTPS connection from the browser was received directly. Thus, any return TCP traffic will be sent directly to the browser's IP address. This asymmetric connection routing will considerably improve the end-to-end latency and reduce the load on the overlay network (less traffic to proxy). While asymmetric routing was once considered potentially harmful, empirical studies show that most of the long-haul traffic (*e.g.,* non-local traffic) over the Internet exhibits high asymmetry [5]. Most of the arguments against this asymmetry arise from the difficulty of configuring packet classification mechanisms, which preclude stateful filtering and required synchronized configuration of multiple nodes (those the traffic may traverse). This would not be a problem in our case, as the asymmetry is exhibited far enough in the network (beyond the filtering router) that the local administrative tasks, such as configuring a firewall, remain unaffected. IPsec and transparent proxying techniques are well-known and (in the case of transparent proxies) widely used, thus we believe such an implementation is not infeasible.

In [91], the authors performed a preliminary analysis using simple networking models to evaluate the likelihood that an attacker is able to prevent communications to a particular target. This likelihood was determined as a function of the aggregate bandwidth obtained by an attacker through the exploitation of compromised systems. The analysis included an examination of the capabilities of static attackers who focus all their attack resources on a fixed set of nodes, as well as attackers who adjust their attacks to "chase after" the repairs that the SOS system implements when it detects an attack. The authors demonstrated that even attackers that are able to launch massive attacks are very unlikely to prevent successful communication. For instance, attackers capable of launching debilitating attacks against

50% of the nodes in the overlay have roughly one chance in one thousand of stopping a given communication from a client who can access the overlay through a small subset of overlay nodes. For more details on the analysis, see [91].

Finally, it is worth estimating the volume of attacks that a WebSOS-protected system can withstand. Since the Internet (and ISPs') backbones are well provisioned, the limiting factors are going to be the links close to the target of the attack. Consider the common POP structure used by ISPs shown in Figure 4.8. The aggregate bandwidth for the POP is on the order of 10 to 20 Gbps. If the aggregate bandwidth of the attack plus the legitimate traffic is less than or equal to the POP capacity, then legitimate traffic will not be affected, and the POP routers can drop the attack traffic (by virtue of dropping any traffic that did not arrive from a SOS secret servlet). Unfortunately, there do not exist good data on DDoS attack volumes; network telescopes [116] tend to underestimate their volume, since they only detect response packets to spoofed attack packets. However, we can attempt a simple calculation of the effective attack bandwidth available to an attacker that controls X hosts that are (on average) connected to an aDSL network, each with 256Kbps uplink capacity. Assuming an effective yield (after packet drops, self-interference, and lower capacity than the nominal link speed) of 50%, the attacker controls $128 \times X$ Kbps of attack traffic. If the POP has an OC-192 (10 Gbps) connection to the rest of the ISP, then an attacker needs 78,125 hosts to saturate the POP's links. If the POP has a capacity of 20 Gbps, then the attacker needs 156,256 hosts. Although we have seen attack clouds of that magnitude, they are not very common. Thus, a SOS-protected system should be able to withstand the majority of DDoS attacks even when deployed within a single ISP. If attacks of that magnitude are a concern, we can expand the scope of the filtering region to neighboring POPs of the same ISP (and their routers); this would increase the link capacity of the filtered region significantly, since each of the neighboring POPs see only a fraction of the attack traffic. Our discussion is not meant as a proof of security against DDoS attacks, but as an exploration of the limits of our mechanism. Additional work is needed to determine the

practical limits of the system, although we are encouraged by our findings to date.

4.2.5 Forwarding Specifics

WebSOS uses SSL to provide two layers of security. First, messages are encrypted end-to-end, so that only the end-points of the exchange (user and web-server) can view the data actually being transmitted. Additionally, WebSOS uses SSL over each hop of the overlay as a means of verifying the authenticity of the previous hop, both when a forwarding session is established and for the forwarded traffic, to avoid attacks from adversaries impersonating as legitimate WebSOS nodes. Note that we do not actually need message confidentiality for the internal communications of WebSOS; however, the additional cost of encryption with the RC4 algorithm is small, since we need to provide peer and data authentication (we shall see the total overhead of WebSOS in Section 4.6). No special functionality is required by the overlay nodes to perform these tasks; the user browser simply has to be supplied with the appropriate certificate(s) from the WebSOS administrator.

In the original SOS architecture, the path established from the user to the target through the overlay was unidirectional. Traffic in the reverse direction could also traverse the overlay, by reversing the roles of user and target. In that case, the path taken by requests and responses would be different. Alternatively, traffic from the target to the user could be sent directly (without using the overlay); this is usually not a problem, since most communication channels are full-duplex and, in the event of a DDoS attack, only the downstream portion (to the target) is congested. An additional benefit of this asymmetric approach is reduced latency, since most client/server traffic (especially in web environments) is highly asymmetric (*i.e.,* clients receive a lot more information than they transmit). This was possible because routing decisions in SOS are made on a per-packet basis.

In WebSOS, routing decisions are made on a per-connection basis. Any subsequent requests over the same connection (when using HTTP 1.1) and any responses from the web server can take the reverse path through the overlay. While this makes the implementation

simpler, it also introduces increased latency, as the bulk of the traffic will also traverse the overlay. We give some thoughts on how to address this issue in Section 4.6.

4.3 Pay-Per-Use Mechanism

We now examine the use of a payment mechanism in our architecture. We begin by describing the necessary hardware and software a service provider needs in order to deploy our mechanism, and then examine its implications for clients.

4.3.1 ISP Provisioning

The ISP first creates an overlay network of WebSOS access points ('servlets'). Per our previous discussion, the routers at the perimeter of the site are instructed to allow traffic only from these servlets to reach the interior of the site's network. These routers are powerful enough to do filtering using only a small number of rules on incoming traffic without adversely impacting their performance.

For a payment scheme, we chose the OTPchecks system because of its inherent flexibility to accommodate different services and its ability to inter-operate with a distributed system like WebSOS. Refer to the roles presented in the OTPchecks functional description, in Figure 4.4; the Payer is the client connecting to the access points, the Merchant is the ISP providing the DoS protection service, and the web service provider (Target) is the clearing entity. The web service provider controls the usage of the service provided via the ISP's network by having the access points delegate payment credentials to each of the clients. In this manner, the service payment can be charged either to the client or to the web service provider. The ISP, using the same transaction information, charges the site providing the web service. The web service itself may charge the user at the same or even a higher rate for the DoS protection and possibly for other Internet commodities (bandwidth, time *etc.*) using the data presented by the access points. The overall system is presented in Figure 4.6.

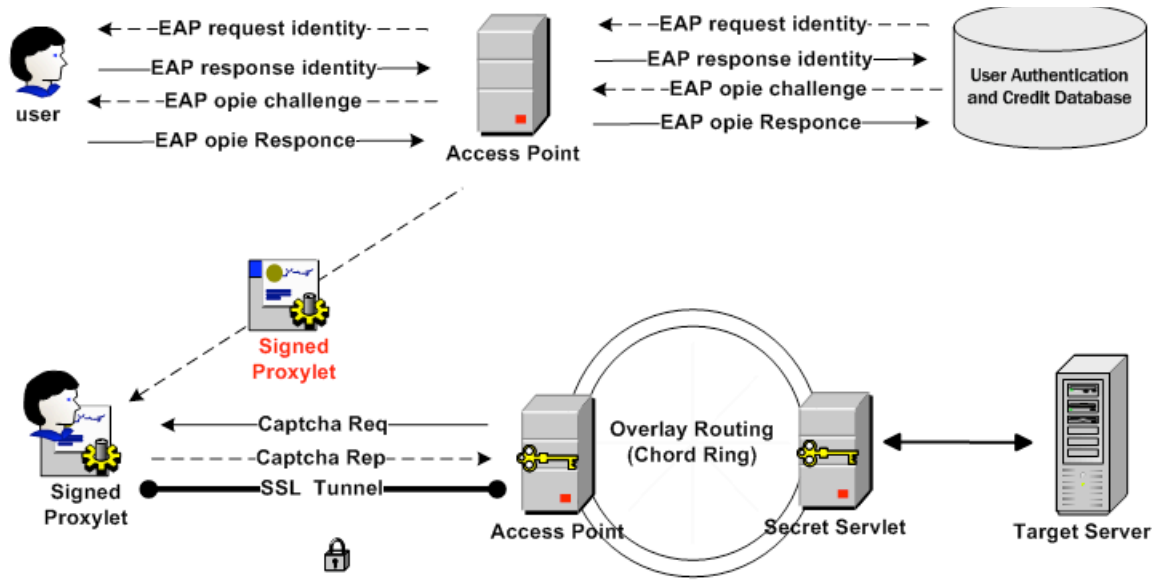


Figure 4.6: Pay-per-use DoS protection system operation overview. The user is connected to an access point which in turn authenticates the user credentials and issues an X.509 certificate and a signed proxylet that allows the user to connect securely to the web service for a limited amount time. The Access Point can act as a proxy for the EAP authentication between the authentication database and the user.

4.3.2 Buying One-Time Coins

Whenever a new client host wants to access a service that the ISP protects from DoS attacks, the access point attempts to run the Extensible Authentication Protocol (EAP) [100] Over LAN (EAPoL) protocol [2, 124] with the client. The status of the client is kept unauthenticated as long as the client fails to authenticate through EAPoL. In our case, we provide unauthenticated clients limited access so that they can buy one-time "coins" (OTC), modeled after One-Time Passwords (OTP) [64], used for the actual EAPoL level authentication (see below).

4.3.3 Using One-Time Coins

Once the Client has acquired a set of one-time coins, it runs the standard EAPoL protocol towards the local access point. The protocol run is illustrated in Figure 4.6.

Upon connection, the access point requests a user identifier from the client. The client

answers with a string that identifies the microcheck used for buying the one-time coins, and the web service the coins were bought for. This allows the access point to contact the correct back-end authenticator, the web service provider (Target). The microcheck fingerprint identifies the relevant unused OTC pile.

Once the back-end authenticator receives the identity response, it checks the OTC pile and sends a request for the next unused one-time password, *i.e.,* an one-time coin. The Client responds with the next unused coin, H_{i+1}. The back-end authenticator checks the coin, records it as used, and replies with an EAP SUCCESS message. As the access point receives the EAP SUCCESS message from the back-end authenticator, it changes the status of the client into authenticated, and passes the message to the client. Shortly before the OTC is used up, the back-end authenticator sends a new request and a GTT to the client.

For the client to continue, it has to reply with the next OTC, and the user must answer correctly the CAPTCHA challenge. This gives us the ability to have a strong protection against malicious code, such as a worm or a zombie process, using a user's micropayment wallet. The lifetime of a coin can be easily configured by the service provider. We expect to prompt the user with a CAPTCHA challenge every 30 to 45 minutes, depending on the service. The client needs to both pay and pass the CAPTCHA challenge before being issued an X.509 certificate (per our discussion in Section 4.2.2), which authorizes it to access the WebSOS overlay.

On the other hand, if the client does not want to continue access for any reason, it simply does not respond to the request. Thus, if the client goes off-line, the access point automatically changes the status of the client's address into unauthenticated once the coin has been used up.

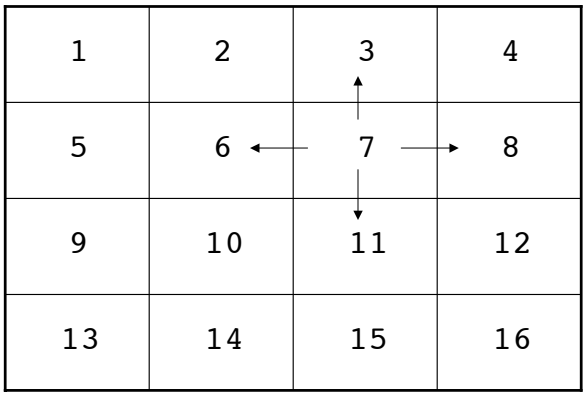


Figure 4.7: Overlay nodes serving regions of a coordinate-space.

4.4 Simulation

To understand the impact of the overlay network on the routing of packets between the source and target nodes, we have applied the WebSOS algorithm to two models of ISP networks [36]. One model, indicative of a U.S. topology, is derived from AT&T's U.S. backbone network. The other, indicative of a European topology, is derived from Worldcom's (now MCI's) European backbone network. Remote access points were excluded from the AT&T model, as were connections from Worldcom's European POPs to points outside the geographical area. For each model, two algorithms for routing traffic through the overlay were tested, one based on Chord, which uses a random ordering of the overlay nodes, and a heuristic variation of CAN that uses geographical ordering of the overlay nodes. In both cases, we tested variations on how the beacons and servlets were chosen in relation to each other, the target, and the source, *e.g.,* requiring some minimum distance between the servlet and target.

We first give a brief description of CAN [134], and then discuss the specifics of the simulation environment, such as ISP structure, the distribution of overlay nodes across ISP Points of Presence (POPs), and the selection strategies for beacons and secret servlets.

4.4.1 CAN

Like Chord, CAN uses a hash function to map overlay nodes to identifiers. However, a CAN identifier maps a node to a region within a *d*-dimensional space. Each overlay node contains a table of overlay nodes responsible for neighboring areas in the coordinate space. As shown in Figure 4.7, overlay node 7 would contain pointers to nodes 3, 6, 8, and 11. In its basic form, CAN does not assume any relationship between node positions of the coordinate space and their geographical positions in the real world. A variation suggested in [134] that assigns positions within the coordinate space being representative of the geography provided the basis for the heuristic used in the model.

4.4.2 Network Layout

A POP-level representation of the ISP was used, where each POP is assumed to consist of a hierarchy of routers as shown in Figure 4.8. At the top level are routers with links to other POPs. At the lowest level are links to client networks.

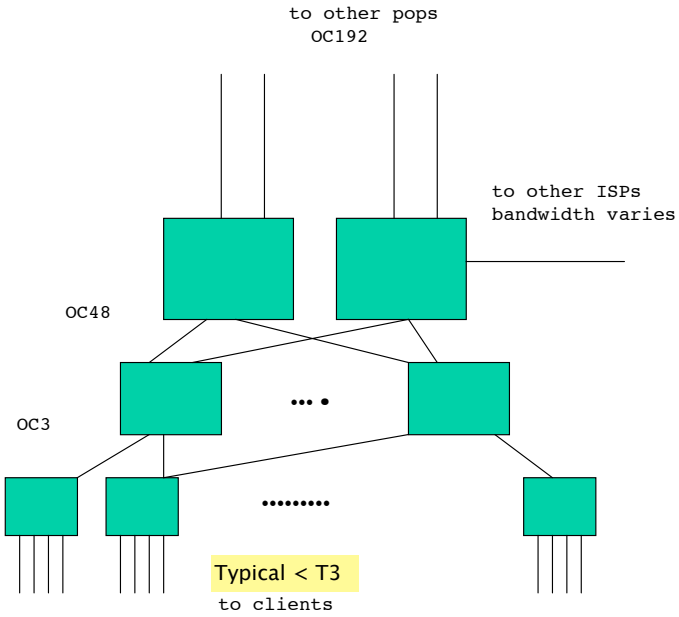


Figure 4.8: ISP POP structure used in the simulation.

Latencies between POPs were estimated from a subset of known latencies. Distances between POPs were estimated using airline miles. Three routers were included at the second level and twelve at the lowest level of each POP; however, for the statistics computed, the exact number of routers within a POP was not relevant, only the latency from the time a packet entered a router within a POP to the time it left the POP was needed.

The model assumes that there is ample bandwidth between POPs and that the choke points are the links to clients. All latencies and distances to clients to their local POP are assigned the same value.

There were 19 POPs in the US model and 18 in the Europe model. Overlay nodes participating in the overlay were evenly distributed across POPs, meaning each POP served the same number of client nodes eligible to be overlay nodes. In the cases where servlets and beacons were randomly chosen, this allowed each POP to be equally likely to have a client site that was a servlet or beacon. In the cases where the servlet and beacon nodes were not randomly chosen, there were more eligible nodes per POP than utilized and the even distribution did not impact selection. A node was not allowed to serve more than one purpose for a specific source-target pair, for example, a node could not be both a beacon and a servlet for the same target. Removing the restriction would result in shorter routes on average because some scenarios tested would pick the same node for both the servlet and beacon.

In each case, two client nodes served by each POP were included in the overlay. Since each source / target pair was tested individually, at most two nodes per POP would be selected to serve the functions of beacon and servlet. When ordering the overlay nodes according to the geographic heuristic described below, designating more than two nodes per POP could only change a route between a source and target by possibly passing through a different client on a given POP. When ordering the overlay nodes randomly and using Chord as the routing algorithm for the overlay, the probability that a client on a specific POP was picked as a beacon or servlet, or was at a certain position in the overlay impacted the route.

Since it was assumed overlay eligible nodes were evenly distributed across all POPs, having 2 versus 100 overlay nodes per POP would not impact the probabilities and thus would not affect the results. The node for the source was chosen to be a client on the same POP as the source. The impact due to it being served by a different POP than the source would be to add the cost of the normal route between the source and SOAP to the cost of the route between the SOAP and target.

4.4.3 Routing Algorithms

In WebSOS, traffic from a source to a target utilizes a route which contains the following sequence of nodes in order: source, access point, beacon, servlet and target. Normal routing is used to reach the SOAP. Also, since the beacon knows the specific servlet for the target, and the servlet knows the location of the target, normal routing is used between the beacon and servlet, and between the servlet and target. An overlay route is used between the SOAP and beacon. The increase in the route length over that of the normal route between the source and target is due not only to the requirement that the route pass through specific nodes, but also due to the need to route through an overlay network between the SOAP and beacon as opposed to using the normal route between the two nodes. For normal routing, each node in the model contained a routing table populated via Dijkstra's algorithm, using minimum hops as the criteria for shortest path. Each node in the overlay network also contained a table with the destination address and overlay node id of a subset of overlay nodes. The table was populated based on the routing algorithms described below.

A routing algorithm for use in overlays is required to send traffic between the SOAP and beacon. The Chord algorithm was utilized in the first set of experiments. The overlay nodes were randomly ordered. The tables within each overlay node were populated using the method described previously involving powers of 2. The size of a node's table is $O(\log n)$, where n is the size of the overlay.

The second set of experiments used a heuristic which divided the POPs into geographical

areas. This method is based on modifications suggested to the basic algorithm for CAN. For a specific area, A, a node n_A was chosen as the area's representative. Each n_A was an entry in each overlay node's table. In addition, if n_i is an overlay node in area A, n_i's table would include entries for each n_j in A, $i \neq j$. Thus an overlay node maintained pointers to every other overlay node in the same geographical area and to one overlay node in each other geographical area. For an overlay of size n, the size of a node's table is $O(n/\#areas) + \#(\text{areas})$, which is $O(n/\#areas)$ when n is large compared to the number of areas, assuming each area contains roughly the same portion of overlay nodes. The US model involved 6 areas, one contained 2 POPs and the other contained 3 or 4 POPs each. The Europe model contained 4 areas with 4 to 5 POPs each.

4.4.4 Beacon/Servlet Selection Scenarios

Seven source-target pairs were chosen in each of the two models. They were selected to represent a variation in source-target relations. Factors considered when selecting the pairs included the distance between cities, whether they were served by neighboring POPs and the level of connectivity for the POP. In all cases a servlet and beacon for a specific target were not permitted to be the same node and neither could serve as a SOAP .

For each model and each routing algorithm, the normal route between each source-target pair was computed then the following four scenarios were tested on each pair. In the scenarios, minimizing the number of hops refers to the number of hops as calculated by normal routing.

1. Randomly select the servlet and beacon (100 trials per source-target pair were run).

2. Select the servlet to minimize the number of hops between the servlet and target, then select the beacon to minimize the number of hops between the beacon and servlet, with the restriction that the servlet and beacon not be served by the same POP.

3. Select the servlet to minimize the number of hops between the servlet and target, then select the beacon to minimize the number of hops between the beacon and source.

4. Select a servlet randomly from those approximately X miles from the target then select a beacon randomly from those approximately X miles from the servlet, where X was 1000 in the US model and 500 in the Europe model. In the case of the Europe model, a few POPs did not have neighbors within this distance, in which case the next closest available overlay node was used.

The first scenario was used to obtain an understanding of the impact when no selection criteria was utilized for the servlet and beacon. This would be the simplest version to implement. The second and third scenarios were aimed at keeping the intermediate nodes in the route near the end points to determine if the route between the source and target would then be similar to the normal route. These two scenarios using minimum distance instead of hops were tested on the US version, but the results were not noticeably different from the scenarios using hops. The fourth scenario was used to understand the impact of selecting the servlet and beacon so they would be served by different POPs than the target, which may be desired for diversity, but at the same time guaranteeing they would be relatively close in an attempt to avoid an unnecessarily long route.

Table 4.1: Average ratio: latency with WebSOS *vs.* normal routing.

	US Chord	US CAN	Europe Chord	Europe CAN
scenario				
1 random selection	4.51	4.16	5.69	4.11
2 min hops	3.45	2.4	3.25	2.54
3 min hops	7.19	1.75	6.77	1.74
4 diversity	5.18	4.08	5.6	2.88

4.4.5 Results

Results are presented in terms of the ratio of the measurement for the WebSOS route to that of the normal route between the source and target. The measurements are for one direction only, source to target, and are averaged over 100 simulation runs. Table 4.1 shows the ratio of the latency using WebSOS to the latency expected when using normal routing. The scenario number corresponds to the previous list. These were averaged over all source-target pairs. The worst case from all source-target pairs is shown in Table 4.2. Table 4.3 indicates the increase in the number of ISP POPs involved in a route compared to that of the normal route.

Table 4.2: Worst-case ratio: latency with WebSOS *vs.* normal routing.

	US/Chord	US/CAN	Europe/Chord	Europe/CAN
scenario				
1 random selection — worst individual source-target average over 100 trials	8.76	6.05	8.05	5.81
2 min hops	7.57	3.76	4.74	3.26
3 min hops	10.9	2.14	11.29	2.14
4 diversity	10.57	6.24	8.1	3.57

Table 4.3: Numbers of POPs in WebSOS routing *vs.* normal routing.

	US/Chord	US/CAN	Europe/Chord	Europe/CAN
scenario				
1 random selection — worst individual source-target average over 100 trials	4	3	4	2.5
2 min hops	2	1.5	2	1.5
3 min hops	5	1	4.2	1
4 diversity	3.5	2.5	4.2	2

When using scenario 3 with the geographic heuristic, the servlet was always selected

from a node on the same POP as the target and the beacon was selected from a node on the same POP as the source and SOAP because there were eligible nodes at every POP. This resulted in the WebSOS route being identical to the normal route with the addition of a few detours to clients within the first and last POPs in the route, thus it was expected to produce the best results in terms of latency.

The results reported for random selection are averaged over 100 trials run per source-target pair. The actual increase in latency may be much higher depending on the specific servlet and beacon chosen. The greatest increase occurs when the source and target are close together. The overlay route may involve points geographically far from the source and target, turning a normally short route into one that may traverse every POP in the ISP at least once. Among all trials involving random selection, the worst case in the Europe model was an increase in latency 15 times that of the normal route between London and Paris when using Chord and 9.5 times when using the geographical heuristic. In the US model, the worst case also involved a latency 15 times normal between NY and Philadelphia when using Chord and 8.86 times when using the geographical heuristic. For NY to Philadelphia, the worst case increase using the geographical heuristic is approximately the same as the average (8.76) when using Chord. The worst cases from all trials involved latencies of 378*ms* using Chord and 230*ms* using the geographical heuristic.

The number of POPs serves as a measure of the complexity of the route but does not necessarily imply a physically long route because several POPs may be geographically close. In scenario 3, the beacon would be selected on the same POP as the SOAP. The ratio for scenario 3 using Chord is high due to a couple of source-target pairs in which the beacon's overlay id was just prior to that of the SOAP's id, resulting in routing through several overlay nodes in the path between the SOAP and beacon.

When using Chord, other variations for populating the overlay node's tables using powers of 3 and $i + x_j$, where x_j is the j^{th} number in the Fibonacci series, for j = 3,4,5.., were tested on a subset of source-target pairs but had no noticeable impact on the length of

the route between the SOAP and beacon. A geographic ordering of the overlay nodes was also tested while maintaining the Chord routing. Nodes that were geographically close were assigned IDs placing them close together on the overlay network. While this shortened the route in cases where nodes X and Y were physically close, a packet was being routed from X to Y using the overlay and X was assigned a lower overlay id than Y; it resulted in a worst case scenario when Y was assigned the overlay id just prior to X's because the packet would route to $O(\log n)$ overlay nodes before reaching the one that knew about X.

4.4.6 Other Considerations

If the overlay nodes are placed within POPs, as opposed to clients' networks, we eliminate the latency due to the connection between the POP and client, and it could be more difficult to attack. In contrast to a client's LAN which may receive traffic for multiple reasons and has a relatively low bandwidth connection to the POP, a server dedicated to WebSOS and attached to a router within a POP allows most invalid traffic to be filtered out in a high-capacity area. However, the use of special purpose servers would result in fewer potential overlay nodes. Furthermore, such servers would not remove the delay due to cross-country routes through the overlay.

Having the overlay network span multiple ISPs will increase the latency of the WebSOS route. There will be a larger number of POPs serving potential overlay nodes. Even if the overlay nodes are geographically distributed in the same manner as with one ISP, the route between any pair of overlay nodes will increase on average due to having to route between ISPs. When the overlay nodes are in the same city but are served by different ISPs, having to route from one ISP POP to another ISP's POP, as opposed to routing between nodes within the same POP, will increase latency. Furthermore, if there is no peering point between the ISPs for that city, the route will require a path to a different city.

4.5 WebSOS Implementation

While the simulation results are encouraging, we felt that experimentation in real networks was necessary to validate our approach. To that end, we developed a prototype of WebSOS, consisting of three main modules. The components are a communications module, a WebSOS routing module, and an overlay routing module running on each node in the WebSOS overlay.

The communications module is responsible for forwarding HTTP requests and responses among the nodes in the WebSOS overlay. When a new proxy request (in the form of a new TCP connection) is received, the communications module calls the WebSOS routing module with the target's destination address to obtain the address of the next hop in the overlay. It then opens a new TCP connection to that node and relays the received HTTP request. Any traffic received in the opposite direction (*i.e.,* the HTTP responses and web content) are relayed back to the source. Authentication of the requesting node by the access point (SOAP) and by internal nodes is accomplished through SSL. Authorized users and WebSOS overlay nodes are issued X.509 [29] certificates signed by the SOAP, once the user has succeeded in the CAPTCHA authentication.

The main *WebSOS routing module* receives requests from the communications module and responds with the IP address of the next node in the WebSOS overlay to which the request should be forwarded. The module first checks whether the current node serves a specific purpose (*i.e.,* whether is it a beacon or secret servlet for that target). If the node serves no such purpose, the module calls the overlay routing module to determine the next hop in the WebSOS overlay and passes the reply onto the communications module. Presently, the WebSOS routing module is initialized with configuration data at startup indicating which nodes serve specific purposes. We are working on an administrative module with increased flexibility to avoid this static provisioning.

The overlay routing module is a general routing algorithm for overlay networks. An implementation of Chord was written for the initial tests. However, this module can be

replaced with any other routing algorithm, *e.g.,* CAN [134]. It receives queries containing a destination IP address (the web server's) and responds with the IP address of the next node in the overlay to which the request should be forwarded. For maintenance of its own routing algorithm, the Chord implementation also communicates with other overlay nodes to determine their status, as described in [162].

When a request is issued by the browser, it is tunneled through a series of SSL-encrypted links to the target, allowing the entire transmission between the requester and target to be encrypted. The SSL connections between WebSOS nodes are dynamically established, as new requests are routed. One problem we ran into while developing the WebSOS prototype is that web browsers do not provide support for the actual proxy request to be encrypted. To solve this problem, we wrote a port forwarder that runs on the user's system, accepts plaintext proxy requests locally, and forwards them using SSL to the access point node. This is implemented as a Java applet that runs inside the browser itself. The Java applet is not considered part of the WebSOS overlay and is not trusted to perform any access control decisions; it is simply a "helper" application.

Thus, to use WebSOS, an authorized user simply has to access any access point , successfully respond to the CAPTCHA challenge, download the applet, and set the browser's proxy settings to the localhost. Java applets typically cannot communicate with any host other than the one they were downloaded from, but this is not a problem in our case. If the user is successful in his/her reply, then the web server connects to a DBMS system (local or remote) and associates a pair of RSA keys (a private key and a certificate) with the host. This set of keys are unique per IP and have an expiration time that can be configured by the system administrator. The user is prompted to download a signed applet that runs locally using one browser window and contacts the Web Server via a temporary HTTPS connection to fetch the X.509 certificate.

The applet then starts listening for HTTP/HTTPS connections on a local port (*e.g.,* 8080) and establishes an SSL-tunnel connection with the proxy server running on the access point

(or elsewhere, since the signed applet has the ability to connect to any server by changing the Java Policy files on the users' machine). The proxy server matches the X.509 certificate and the IP from client to the private key obtained from the DBMS system and allows the connection to be proxied. The only imposition on the user is that he/she must change the Proxy settings of the local browser to point to the socket that listens for the applets.

Initial prototyping of the communications module used Apache, whose proxy module was modified to query the routing module for the next hop. This worked well when unencrypted HTTP requests were issued by the browser. However, when we encountered the requirement for end-to-end authentication and encryption, we changed the implementation to use a stand-alone proxy server instead of Apache.

An adaptation of the initial implementation was created, to improve performance: rather than transporting the request and response through the full overlay network, only routing information travels through the overlay. As before, the requester makes a proxy request to the access point. At that point, the access point sends a UDP message into the overlay, specifying the target. The message is routed to the beacon, which responds directly to the overlay node with information on the secret servlet for that target. The overlay node then connects to the servlet, which proxies the request as before, in effect creating a *shortcut* through the overlay.

4.6 Experimental Evaluation

A secure system is one that meets or exceeds an application-specified set of security-policy *requirements.* So, for example, in message delivery, the high-level requirements may be that the correct information gets to the right person, in the right place, *at the right time*. The details of "right" are determined by the application's needs. Traditional security mechanisms have addressed the first two parts of this informal definition of security, but largely ignored the timeliness issue.

In order to quantify the overhead associated with use of WebSOS, we created a simple topology running on the local network (100 Mbit fully-switched Ethernet). For our local-area network overlay, we used 10 commodity PCs running Linux Redhat 7.3. We measured the time-to-completion of https requests. That is, we measured the elapsed time starting when the browser initiates the TCP connection to the destination or the first proxy, to the time all data from the remote web server have been received. We ran this test by contacting 3 different SSL-enabled sites: *login.yahoo.com, www.verisign.com,* and the Columbia course bulletin board web service at `https://www1.columbia.edu/sec/bboard`). For each of these sites, we measured the time-to-completion for a different number of overlay nodes between the browser and the target (remote web server). Our measurements sum up all WebSOS overheads, including network latency, software processing at each overlay node, and encryption (specifically, double SSL encryption, as discussed in Section 4.2.5). Note that a measurement involving n nodes implies a Chord overlay of 2^n nodes, since for an overlay of size x, it takes approximately $log(x)$ hops to reach a destination. Thus, our experiments with 10 nodes represent an overlay with 1024 nodes.

The reason for this configuration was to introduce some latency in the first-hop connection (from the browser to the SOAP), thus simulating (albeit using a real network) an environment where the browsers have slower access links to the SOAPs, relative to the links connecting the overlay nodes themselves (which may be co-located with core routers). By locating all the overlay nodes in the same location, we effectively measure the aggregate overhead of the WebSOS nodes in the optimal (from a performance point of view) case.

Table 4.4 shows the results for the case of 0 (browser contacts remote server directly), 1, 4, 7, and 10 overlay nodes. The times reported are in seconds, and are averaged over several HTTPS GET requests of the same page, which was not locally cached. For each GET request, a new TCP connection was initiated by the browser. The row labeled "Columbia BB (2nd)" shows the time-to-completion of an HTTPS GET request that uses an already-established connection through the overlay to the web server, using the HTTP 1.1 protocol.

As the figure shows, WebSOS increases the end-to-end latency between the browser and the server by a factor of 2 to 3. These results are consistent with our simulations of using SOS in an ISP topology, where the latency between the different overlay nodes would be small, as discussed in Section 4.4. The increase in latency can be primarily attributed to the network-stack processing overhead and proxy processing at each hop. It may be possible to use TCP splicing [34] or similar techniques to reduce connection handling overhead, since WebSOS performs routing on a per-request basis. Also, in the experiments we ran, we did not make use of the asymmetric routing option possible with the use of GRE as both a filtering and an encapsulation mechanism, as discussed in Section 4.2.4.

Table 4.4: Latency (in seconds) when contacting various SSL-enabled web servers directly and with different numbers of (intermediate) overlay nodes over the local-Ethernet network.

Server	Direct	1 node	4 nodes	7 nodes	10 nodes
Yahoo!	1.39	2.06	2.37	2.79	3.33
Verisign	3.43	4.22	5.95	6.41	9.01
Columbia BB	0.64	0.86	1.06	1.16	1.21
Columbia BB (2nd)	0.14	0.17	0.19	0.20	0.25

Table 4.5: Latency (in seconds) when contacting various SSL-enabled web servers directly and with different numbers of (intermediate) overlay nodes using the PlanetLab network.

Server	Direct	1 node	4 nodes	7 nodes	10 nodes
Yahoo!	1.39	3.15	5.53	10.65	14.36
Verisign	3.43	5.12	7.95	14.95	22.82
Columbia BB	0.64	1.01	1.45	3.14	5.07
Columbia BB (2nd)	0.14	0.23	0.28	0.57	0.72

Furthermore, there is an SSL-processing overhead for the inter-overlay communications. A minor additional cryptographic overhead, relative to the direct access case, is the certificate validation that the SOAPs have to perform, to determine the client's authority to use the overlay, and the SSL connection between the proxy running on the user's machine and the SOAP. As shown in [115], such overheads are typically dominated by the end-to-end com-

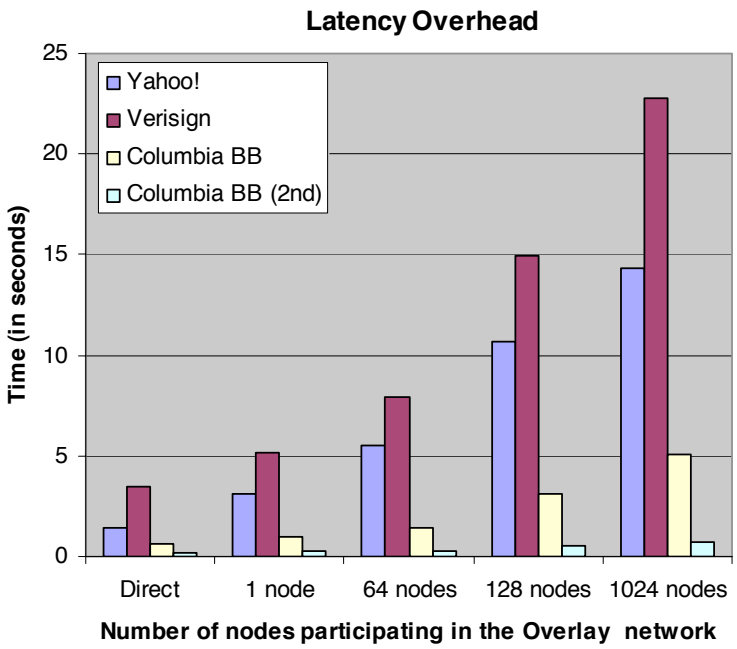


Figure 4.9: Latency (in seconds) when contacting various SSL-enabled web servers directly and with different numbers of (intermediate) overlay nodes using the PlanetLab network.

munication overheads. Use of cryptographic accelerators can further improve performance in that area. One further optimization is to maintain persistent SSL connections between the overlay nodes. However, this will make the task of the communication module harder, as it will have to parse HTTP requests and responses arriving over the same connection in order to make routing decisions.

Table 4.5 and Figure 4.9 show the same experiment using PlanetLab [133], a wide-area overlay network testbed. The PlanetLab nodes are distributed in academic institutions across the country, and are connected over the Internet[4]. We deployed our WebSOS proxies PlanetLab and ran the exact same tests. Naturally, the direct-contact case remains the same. We see that the time-to-completion in this scenario increases by a factor of 2 to 10, depending on the number of nodes in the overlay. In each case, the increase in latency over the local-Ethernet configuration can be directly attributed to the delay in the links between

[4]In fact, the majority of PlanetLab nodes are inside Internet2. This can give somewhat better results than the current Internet, since connectivity is better over Internet2.

Table 4.6: Latency (in seconds) when contacting various SSL-enabled web servers directly and while using the shortcut implementation of the WebSOS system. The testing was performed on a 76 node subset of the PlanetLab testbed using the Chord overlay. The hops to the beacon ranged from 4 to 8 and did not have a significant effect on latency. The *cached* column refers to subsequent requests using the same SOAP, whereupon the Secret Servlet information has been cached.

Server	Direct	Original Request	Cached Requests
Yahoo!	1.39	4.15	3.67
Verisign	3.43	7.33	6.77
Columbia BB	0.64	3.97	3.43
Columbia BB (2nd)	0.14	0.55	0.56

the WebSOS nodes. While the PlanetLab configuration allowed us to conduct a much more realistic performance evaluation, it also represents a worst-case deployment scenario for WebSOS: typically, we would expect WebSOS to be offered as a service by an ISP, with the (majority of) WebSOS nodes located near the core of the network. Using PlanetLab, the nodes are distributed in (admittedly well-connected) end-sites. We would expect that a more commercial-oriented deployment of WebSOS would result in a corresponding decrease in the inter-overlay delay. On the other hand, it is easier to envision end-site deployment of WebSOS, since it does not require any participation from the ISPs.

Finally, while the additional overhead imposed by WebSOS can be significant, we have to consider the alternative: no web service while a DoS attack against the server is occurring. While an increase in end-to-end latency by a factor of 5 (or even 10, in the worst case) is considerable, we believe it is more than acceptable in certain environments and in the presence of a determined attack.

Table 4.6 shows the results when the shortcut implementation was tested on the PlanetLab testbed, with a graphical depiction in Figure 4.10. This variant provides significant performance improvements, particularly on subsequent requests for the same site, because of the caching. To simulate the effects of an attack on individual nodes in the overlay, we

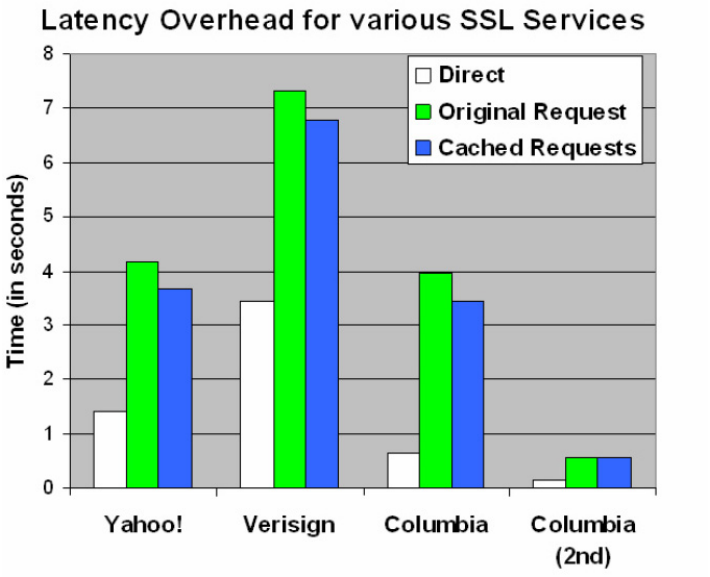


Figure 4.10: Latency (in seconds) when contacting various SSL-enabled web servers directly and while using the shortcut implementation of the WebSOS system. The testing was performed on a 76 node subset of the PlanetLab testbed using the Chord overlay. The hops to the beacon ranged from 4 to 8 and did not have a significant effect on latency. The *cached* column refers to subsequent requests using the same access point, whereupon the Secret Servlet information has been cached.

simply brought down specific nodes. The system healed itself within 10 seconds.

Table 4.7: Signing and verification times for 1024-bit RSA keys.

Sign	Verify	Signatures/sec	Verifications/sec
0.0037 secs	0.0002 secs	270.0	5055.9

To complete the overhead analysis, we measured the number of public key verifications an access point can perform, which indicates how many microchecks it can validate in unit time. We used a 3GHz Pentium4 processor machine running Linux with the OpenSSL V 0.9.7c library for the measurements. The contribution of the micropayment system to the overall system latency overhead is minimal even, when we issue 1024-bit RSA certificates for the client credentials, as shown in Table 4.7. These measurements show that the impact of the user verification process on the access points is minimal: since users only need to use a microcheck every half hour or so, a single node can handle 18 million users per hour, even

without hardware accelerators [93].

Chapter 5

Enabling End-to-End Service Protection Using Process Migration

5.1 Introduction

We present **M**igrating **OVE**rlay (MOVE), a system that aims to provide an *e2e*-compatible anti-DoS mechanism. MOVE is a solution to the denial of service (DoS) problem that does not rely on network infrastructure support, conforming to the end-to-end (*e2e*) design principle. Our approach is to separate "good" traffic from unknown traffic, and treat the former preferentially, using an indirection-based network (IBN)in a manner similar to SOS [91, 92, 37, 117] but without using packet filtering. MOVE nodes are located at edge networks, requiring no infrastructure support. Traffic is differentiated on a per-session basis, using cryptographic authentication and/or Graphic Turing Tests (GTTs) to determine valid users (which may simply mean "humans"). The overlay routes traffic from legitimate users to the current network location of the protected service. When an attack against the hosting site is detected, we use a lightweight process-migration mechanism to re-locate the service to an unaffected site. Legitimate traffic is routed transparently to the new location, while malicious (or simply unknown) traffic will continue flowing to the old location.

Unlike SOS [91, 92], WebSOS [37, 117, 157], and Mayday [9], MOVE does not require any filtering perimeter to be constructed around the target site; instead, we use process migration to move (and obscure) the current location of the attacked service, and "stepping stone" hosts to maintain connectivity between the original site and the new location of the service. Architecturally, SOS introduced the general idea of using an overlay and filtering to protect against some classes of DDoS attacks. WebSOS enhanced the front-end of the overlay (its interface with the remote clients) to enable more *ad hoc* interactions than SOS allowed. MOVE concerns itself with the back-end of the overlay (its interface with the protected sites), removing the dependency on network filtering. Our approach is similar to the concept of "hidden servers" in anonymity systems such as Tor [135, 49], although our use of server migration in MOVE allows us to protect against a larger class of attackers.

No aspect of MOVE depends on the network infrastructure itself, although it makes certain assumptions about the threat model. In particular, (*a*) there is a notion of *legitimate users*, (*b*) the attackers cannot take over arbitrary routers or eavesdrop at will on arbitrary network links, and (*c*) there exists a relatively large number of potential hosting sites. We discuss these assumptions further in Section 5.2.

Where these assumptions hold, we believe MOVE to be the first anti-DoS mechanism that does not require any additional functionality from the network. We hope to demonstrate that by making careful assumptions and relaxing the threat model in realistic ways, it is possible to design *efficient* and *effective* protection mechanisms that do not violate prevalent system and network design principles. To that effect, we test our experimental prototype on PlanetLab [133], a testbed for experimentation with network overlays. As we show, the overlay mechanism increases end-to-end latency by a factor of 2 to 3. Migrating a web server and its associated state causes less than 10 seconds of service disruption for the end user, and connectivity resumes transparently to the end applications; in the case of a VNC server with substantially more state, the service disruption time ranges between 17 and 22 seconds. The attacker is left with no indication as to the new location of the

service, thus having to either distribute the attack traffic among various potential targets or try to guess the correct hosting site. An attacker that cannot guess the new location faster than 10 seconds (for the web server case) cannot permanently disrupt access to the service. User-perceived performance of the web server degrades somewhat after a migration, since data such as the back-end database and the filesystem remain in the old location and must be accessed remotely. Similar results are obtained when migrating a remote display application, VNC. Furthermore, clients need to use MOVE only when their connectivity to a service is disrupted; under regular network conditions, direct access to the servers would typically be used, minimizing the performance impact of MOVE.

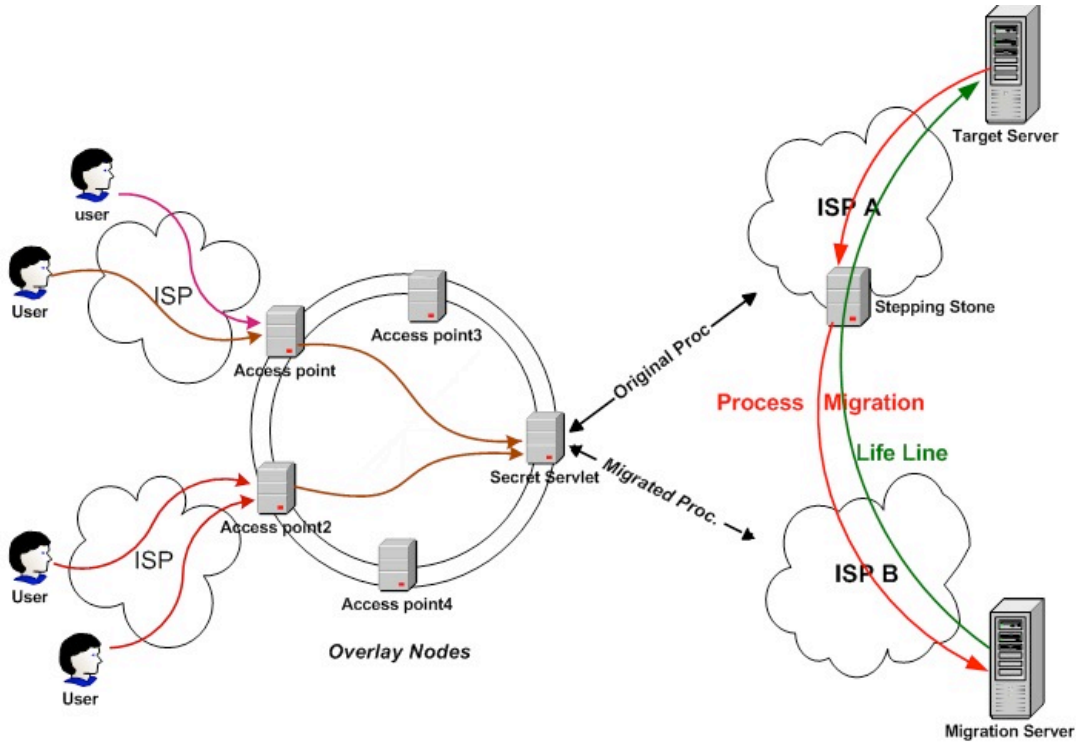


Figure 5.1: Migrating Overlay system architecture.

5.2 Threat and Application Model

DoS attacks can take many forms, depending on the resource the attacker is trying to exhaust. For example, an attacker can try to cause the web server to perform excessive computation, or exhaust all available bandwidth to and from the server. In all forms, the attacker's goal is to deny use of the service to other users. Apart from the annoyance factor, such an attack can prove particularly damaging for time- or life-critical services (*e.g.,* tracking the spread of an real-world epidemic), or when the attack persists over several days. Of particular interest are *link congestion* attacks, whereby attackers identify pinch-points in the communications substrate and render them inoperable by flooding them with large volumes of traffic. An example of an obvious attack point is the location (IP address) of the destination that is to be secured, or the routers in its immediate network vicinity; sending enough attack traffic will cause the links close to the destination to be congested and drop all other traffic.

We assume that attackers are smart enough to exploit features of the architecture that are made publicly available. We do not specifically consider how to protect the architecture against attackers who can infiltrate the security mechanism that distinguishes legitimate traffic from (illegitimate) attack traffic: we assume that communications between overlay nodes remain secure so that an attacker cannot send illegitimate communications, masking them as legitimate. In addition, it is conceivable that more intelligent attackers could monitor communications between nodes in the overlay and, based on observed traffic statistics, determine additional information about the current configuration. Such attackers would have the ability to subvert arbitrary routers and/or eavesdrop at will on network links. As such attacks are considerably more difficult than denial of service, we consider them outside our scope. In [176], the authors analyze our overlay architecture under a model allowing for attackers that can compromise arbitrary nodes in the overlay, toward determining the identity of the beacon and/or secret servlets. They conclude that by layering multiple overlays on top of each other, one can trade off increased resistance to such attackers with end-to-end performance.

Our prototype implementation is focused on two applications (although not limited to these): a web server and a remote display access application (VNC) [137]. We chose the Web as an implementation mechanism due to the facilities that common servers and browsers provide and the ease with which we could develop a prototype implementation. VNC is a good example of an application that maintains considerable state that cannot be easily replicated, without being "storage-heavy". In general, the applications we are most interested in are real-time, server-assisted applications, and other applications that require some state to be maintained by the server but are not fundamentally storage-oriented (*i.e.,* their processing component dominates the system performance costs). Our system supports applications that require access to a storage back-end, by maintaining a "lifeline" between the new and the old location of the service. The service can access the storage back-end over this lifeline, with some loss of performance, which we measure in Section 5.5. We discuss the lifeline in more detail in Section 5.3.1.

Note that applications that do not require any state to be maintained can simply be load-balanced across several sites and contacted using Anycast, RR-DNS, *etc.*. Likewise, content-delivery applications can use data replication services such as Akamai to increase availability.

Finally, we assume that there exists a number of hosting sites that can accept the migrating service. These can be statically provisioned, *e.g.,* through a co-operative agreement among various service providers, or allocated on demand from a commercial entity selling (or renting) CPU time. In either case, we assume that there exist enough such hosting sites that an attacker cannot simply overwhelm all of them with a coordinated DoS attack. Instead, the attacker can successfully attack some (small) percentage of such sites. Note that these hosting sites *are not* part of the overlay itself. They only host migrated services, presumably under some contractual agreement with the owners of such services.

5.3 MOVE Architecture

The overall architecture of our system is shown in Figure 5.1, and shares the same front-end (client operation) and communication fabric alteration (overlay network) as WebSOS. The system differs in the way we protect the back-end (the servers).

5.3.1 Server Protection In MOVE

Servers that are to be protected inform the overlay of their current location. Such servers are also authenticated using a standard security protocol such as SSL/TLS [47] with client certificates, or IPsec [90]. When a server migrates to a new location, it simply informs the overlay of its new location. Note that multiple servers, belonging to different organizations, may be using the system at the same time; the certificates they hold allow them to change the location status only for themselves, *i.e.,* they cannot cause the overlay to redirect and capture a competitor's traffic. We assume the existence of enough locations to choose as the new destination that it is impractical for an attacker to simply attack each site and determine (based on service response latency) which one is hosting a particular service; that is, it will take too long for an attacker to locate a service using that (or a similar) approach, compared to how quickly the system can migrate to a new location. We experimentally quantify this delay in Section 5.5.

However, notice that the process migration mechanism itself uses the network, which is presumed to be under a DoS attack. In keeping with the spirit of the *e2e* design, we preclude use of any form of QoS provisioning (although such arrangements can be very effective and do not require a lot of overhead, since the endpoints are fixed and known *a priori*). Instead, we assume that each hosting site has a secondary, potentially low-bandwidth connection to the Internet with a different IP address (either with the same or, better yet, with another ISP), which is not advertised through BGP. Thus, attack traffic from outside the home ISP cannot reach that interface, even if the attacker knows this alternate address.

We also require "stepping stones" in the same home ISP (but not necessarily operated by the ISP, *i.e.,* they can be located in end-networks), which allow the migrating process to reach a host that can communicate with other nodes outside the home ISP. During process migration, the server binary and state are saved (as we shall describe in Section 5.3.2), transfered to one of the stepping stones, and thence to a "random" hosting site, where the service is restarted. The service will then notify the overlay of its new location. The overlay will then redirect traffic from legitimate clients to this new location. The stepping stones can be part of the overlay (admitting new clients and routing their traffic), dedicated nodes, hosting sites that use the same home ISP, or any combination of these. The only important characteristic is that they can communicate beyond the local ISP, and with the ISP-specific address of the attacked site. For example, the ISP may be using internally a net-10 address (*i.e.,* an address in the private `10.0.0.0/8` prefix) for the secondary address of stepping stones and protected sites. The stepping stone node also uses a temporary address that belongs to the ISP, that *is* globally routable. This address is changed periodically, such that an attacker cannot target a stepping stone with attack traffic. Thus, we have created an one-way communication channel through the stepping stone (outside connections through it are possible, incoming ones are not).

Finally, a "lifeline" connection is maintained between the new location and the old location *via the stepping stone,* such that the migrated service can access any storage that is attached to the old location. For example, consider a typical web server such as the one shown in Figure 5.2. In MOVE, we will migrate the front-end web server component, and use the lifeline connection to communicate to the business logic and database components. Although the stepping stone's address may change over time, we can use tunneling to maintain the connection with the new location. We eventually intend to use SCTP for this connection, since this allows us to easily do live-connection migration to a new IP address. Since the secondary link connects to another ISP (or we ensured that it uses a different set of links than the main Internet attachment point), the DDoS attack will not affect the lifeline.

To avoid adding up to the lifeline overheads when a server is migrated several times, we collapse the lifeline by instructing the previous (original) stepping stone to connect to the new location.

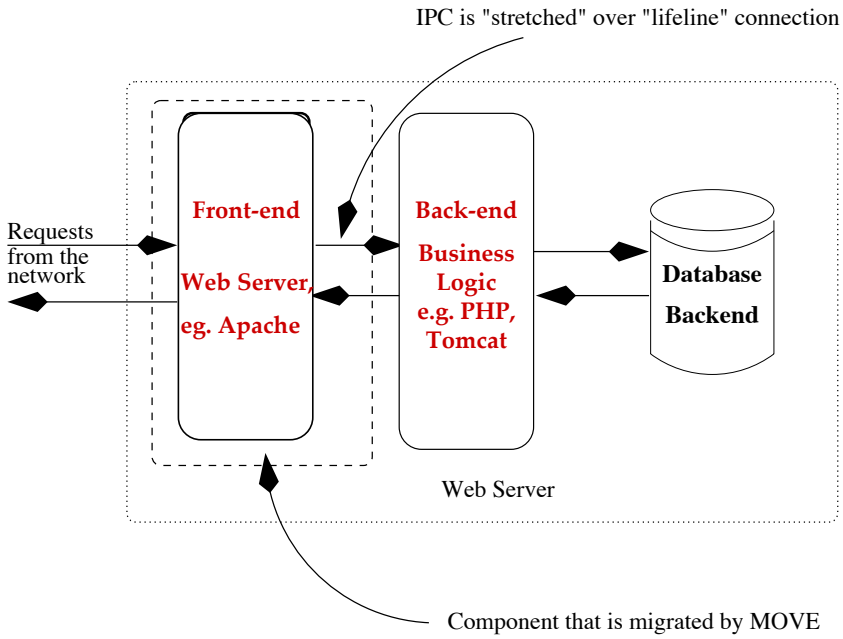


Figure 5.2: Typical model of a web server.

5.3.2 Lightweight Process Migration

Process migration is the ability to transfer a process from one machine to another. It is a useful facility in distributed computing environments, especially as computing devices become more pervasive and Internet access becomes more ubiquitous. Although many approaches have been proposed [114], achieving process migration functionality has been difficult in practice.

Toward this end, there are four important goals that need to be met. First, given the large number of widely used legacy applications, applications should be able to migrate and continue to operate correctly without modification, without requiring that they be written using uncommon languages or toolkits, and without restricting their use of common

operating system services. For example, networked applications should be able to maintain their network connections even after being migrated. Second, migration should leverage the large existing installed base of commodity operating systems. It should not necessitate use of new operating systems or substantial modifications to existing ones. Third, migration should maintain the independence of independent machines. It should avoid creating residual dependencies that limit the utility of process migration by requiring machines where a process was previously executed to continue to service a process even after it has migrated to another machine. Fourth, migration should be fast and efficient. Overhead should be small for normal execution and migration.

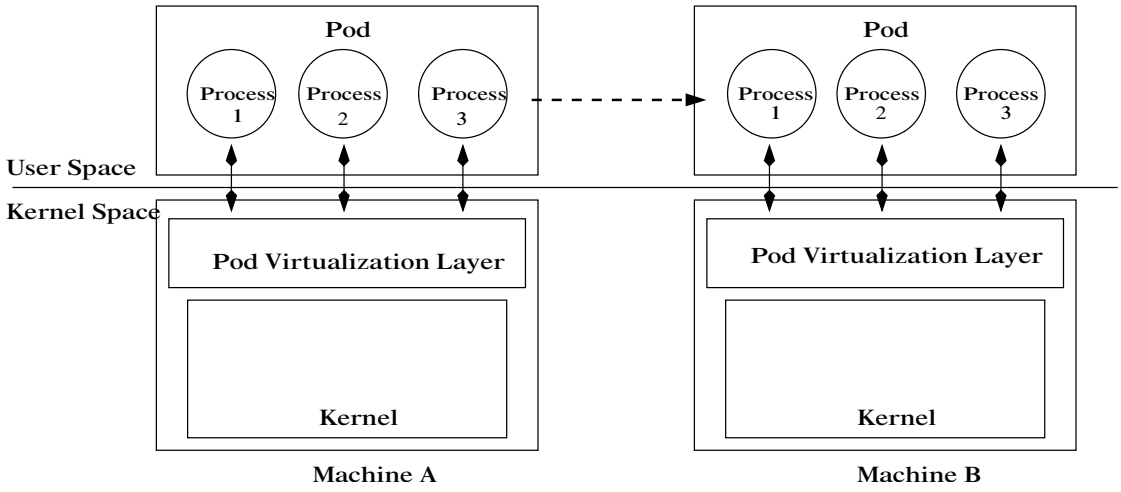


Figure 5.3: Lightweight system migration by capturing state in the interface between operating system kernel and processes.

For our system, we use an approach based on [127], by effectively providing a thin virtualization layer, called a *POD,* on top of the operating system that provides a group of processes with a private namespace, as shown in Figure 5.3. The sandboxed process group always sees the same virtualized view of the system, which associates virtual identifiers with operating system resources such as process identifiers and network addresses. This decouples sandboxed processes from dependencies on the host operating system and from other processes in the system. This virtualization is integrated with a checkpoint-restart mechanism that enables the sandboxed processes to be migrated as a unit to another machine.

These process groups are independent and self-contained, and can thus be migrated freely without leaving behind any residual state after migration, even when migrating network applications while preserving their network connections. We can therefore allow applications to continue executing after migration even if the machine on which they previously executed is no longer available. To support transparent network-connection migration, client-side support is required, which is straightforward to implement on the overlay nodes (*i.e.,* no changes to client (user) software are necessary). For our prototype implementation, this was not required, since the web model allows for temporary TCP connection failures. For application domains where this is not an option, we can augment the overlay nodes accordingly. We do not explore this option further in this paper.

The process migration system is designed to support migration of unmodified legacy applications while minimizing changes to existing operating systems. This is done by leveraging loadable kernel module functionality in commodity operating systems that allows us to intercept system calls as needed for virtualization and save and restore kernel state as needed for migration. The system's compatibility with existing applications and operating systems makes it simple to deploy and use. The system is implemented as a kernel module, allowing transparent migration among separate machines running independent versions of Linux (with unmodified kernels). Note that these systems do not need to share a single-system image.

In our web server experiments, all the server processes were contained in the same POD. When migrating the VNC environment, a number of different processes were contained inside the same POD: the VNC server itself, X11 terminals, a web browser, and a few other applications. We describe the configuration in more detail in Section 5.5.

5.3.3 Example of System Operation Under Attack

To illustrate the use of our architecture by servers and clients, we describe the steps both sides must undertake to protect their communication channel:

- A server contacts any overlay node and informs it of its location. The connection is protected by SSL/TLS, and the server presents a certificate that proves to the overlay its right to specify a location for a particular hostname/URL. The overlay node confirms the validity of the certificate and informs other nodes in the overlay of the new location of the service.

- A client that wants to communicate with the service contacts a random overlay node. After authenticating and authorizing the request via the CAPTCHA test, the overlay node securely proxies all traffic from the source to the target. Alternatively, a pre-authorized client that possesses a valid certificate can connect to the overlay without requiring any user interaction. As explained in [157], this step may also involve some type of payment (by the end user or the service owner) to the entity managing the overlay infrastructure. Following the discussion of [9], a number of overlay nodes may be involved in the routing, depending on the precise threat model and performance requirements. For example, to avoid the situation where an attacker can eavesdrop on an overlay node and determine the location of the service, we may want to use a two-hop overlay routing approach; if this is not a concern, we may use one-hop redirection instead. Alternatively, we can use full Chord-like routing [162] as in SOS [91], obscuring traffic patterns.

- When an attack is detected, the server process is suspended and migrated, using the system described in Section 5.3.2. A random hosting site is selected and, after querying its current status with respect to DoS attacks and other suspicious activity, the server is migrated there. To perform the migration, a "stepping stone" host that resides in the same ISP as the source hosting site is used, to achieve routability from an unpredictable source address (one that cannot be attacked from outside the ISP).

5.4 MOVE Implementation

Our prototy uses the regular WebSOS [156] indirection infrastructure which we have already discussed in Chapter 4.

Our prototype system is based on WebSOS [117, 37]. Each overlay node is responsible for resolving the location of the requested service and creating a security communication tunnel with it. To that end, we use Chord to distribute the location information for each site: when a service informs MOVE of its current location, its hostname is hashed and the node thus indicated is informed of the location. In that sense, this node acts in a manner analogous to SOS *beacon* nodes. Similarly, when a MOVE node needs to forward a legitimate user's request to the service, it hashes the service hostname and sends a query to the Chord node whose address is closest to the hash result. Thus, in contrast to [37] and [91], rather than transporting the request and response through the Chord overlay, only routing information travels through it; data connections are proxied directly to the protected service's location. The information is cached and periodically refreshed by consulting the authoritative MOVE node for that target.

When a new request (in the form of a new TCP connection) is received, the MOVE node to which the client is connected (called an *access point*) first checks the local cache database for the current location of the requested service. If the lookup succeeds, the access point opens a new SSL connection to a random overlay node (to borrow from SOS terminology, a "secret servlet"), which allows us to avoid some of the eavesdropping attacks identified in [9]. Thus, a two-way communication channel is established between the client and the service, through the overlay. Authentication of the user by the overlay is accomplished through SSL. Authorized users are issued X.509 [29] certificates signed by the MOVE access point that administered the GTT. These certificates are only valid for a limited time (30 minutes), after which the user must pass another GTT. Furthermore, the certificates are bound to the IP address from which the GTT authentication came, and can only be used with the specific MOVE access point. Thus, an attacker cannot simply authenticate once and redistribute the

same certificate to a large number of attack zombies. Each overlay node also communicates with other MOVE nodes over SSL connections. If the lookup fails, the access point queries the resolving node, as described previously.

When a request is issued by the client for a specific service, it is tunneled through a series of SSL-encrypted links to the target, allowing the entire transmission between the requester and target to be encrypted. The SSL connections between MOVE nodes are dynamically established, as new requests are routed. To accomplish this, we wrote a port forwarder that runs on the user's system, accepts plain-text proxy requests locally, and forwards them using SSL to the access point node. This is implemented as a Java applet that runs inside the browser that a user uses to authenticate himself. This Java applet is responsible to encrypt and forward to the access point requests from any service initiated by the client and can be configured to accept a proxy. This last requirement can be removed if we use interception of the socket communication at the operating system level.

Thus, to use MOVE, an authorized user simply has to access any access point, successfully respond to the Graphic Turing Test challenge, download the applet, and set the service proxy settings to the localhost, as shown in Figure 5.4. The client to server communication establishment has been explained previously when we discussed WebSOS in 4.5.

The access point caches the server's location for use in future requests. That information is timed out after a period of time to allow changes to propagate correctly. The same basic mechanism is used by services to announce their presence to (and periodically update the information stored by) their corresponding resolving nodes.

In a DoS attack, the target server migrates to a randomly-chosen location and, once there, notifies the overlay of its new location. The migration is done using the process migration mechanism we described in Section 5.3.2. System migration is largely implemented as a loadable Linux kernel module. In our system, this mechanism was used to load and checkpoint the Apache and VNC servers respectively, creating the necessary images of the running processes. These process images, which included the current process state, were

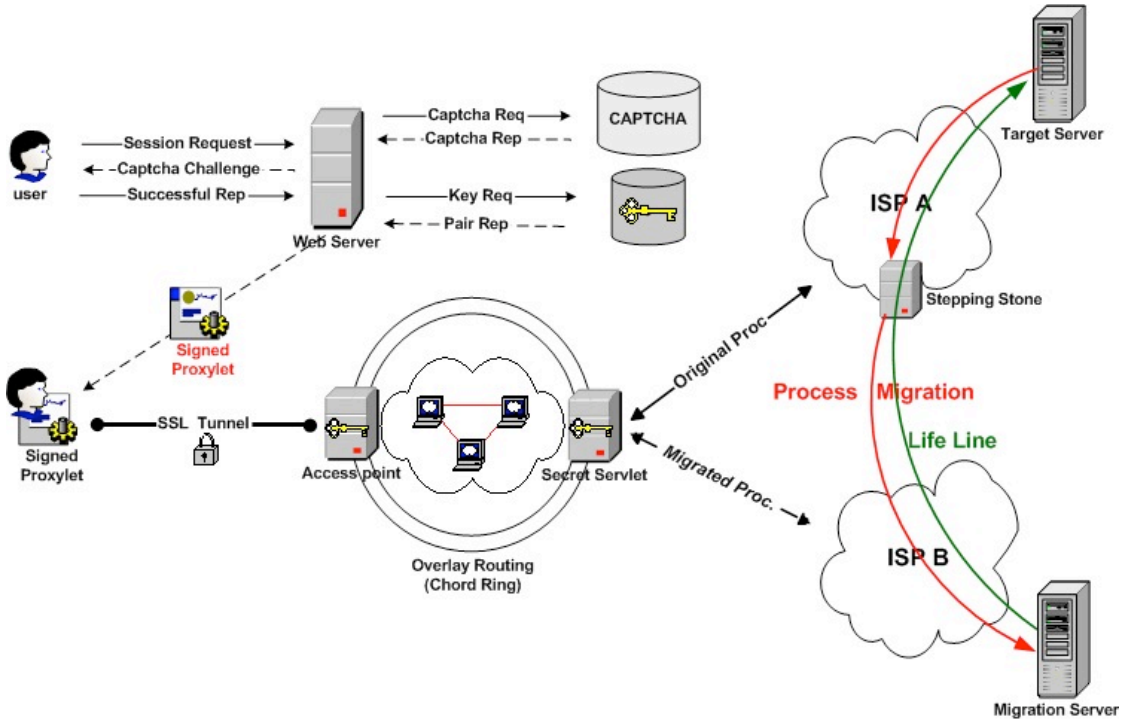


Figure 5.4: MOVE client session initiation diagram.

then transfered to the remote server and restarted.

5.5 Experimental Results

To evaluate MOVE's impact on performance and availability, we deployed the prototype implementation on a number of PlanetLab nodes [1] [133], distributed across the Internet. We measured (1) the impact of using MOVE to the client's end-to-end latency, (2) the delay in making a server available again at a new site once a DoS attack has been launched, and (3) the impact of the lifeline connection to the server's performance. In our experiments, we used the following entities (see Figure 5.4):

- A node acting as the client, using an off-the-shelf web browser.
- An http *target* server (Apache) and a VNC server, in two separate experiments.

[1] http://www.planet-lab.org/

- A set of PlanetLab nodes participating in the overlay network and providing the necessary traffic redirection facilities.

- A *migration* server, to which the server is migrated from its original location when attacked.

In our experiments, the legitimate client was located inside Columbia University's network and the traffic was redirected from various nodes inside the PlanetLab network toward the web and VNC servers, which were initially located on the local (Columbia) network. We deployed the MOVE implementation on 76 PlanetLab nodes, which formed a Chord ring.

Table 5.1: Latency (in seconds) when contacting a number of web servers directly and while using MOVE; in all cases, we download the initial web page. The last column shows the factor increase in latency The testing was performed on a 76 node subset of the PlanetLab testbed using the Chord overlay. The numbers are averaged over 25 requests.

Server	Direct	MOVE	Ratio
Yahoo!	1.32	3.67	2.78
VeriSign	3.41	6.77	1.98
BBC News	1.11	3.17	2.85
Microsoft	1.51	4.01	2.65
Slashdot	3.66	7.21	1.96
FreeBSD	1.49	3.81	2.55

To determine the end-to-end latency experienced by a client using our system, we used the MOVE overlay to contact various web servers and download the initial page. The results are shown in Table 5.1. The difference in performance over previous work [37] that used the same benchmark is due to the fact that traffic in MOVE only traverses two overlay nodes, as opposed to the full Chord overlay, meaning fewer redirections between overlay nodes. As shown in [115], the increase in latency is typically dominated by the end-to-end communication overheads. An additional delay cost is the SSL-processing overhead from the generation of the SSL tunnel and the encryption of the data from client to the overlay

and inside the overlay; use of cryptographic accelerators may further improve performance in that area [93]. One last optimization is to maintain persistent SSL connections between the overlay nodes. However, this will make the task of the communication module harder, as it will have to parse HTTP requests and responses arriving over the same connection in order to make routing decisions. For our prototype implementation, we opted for simplicity.

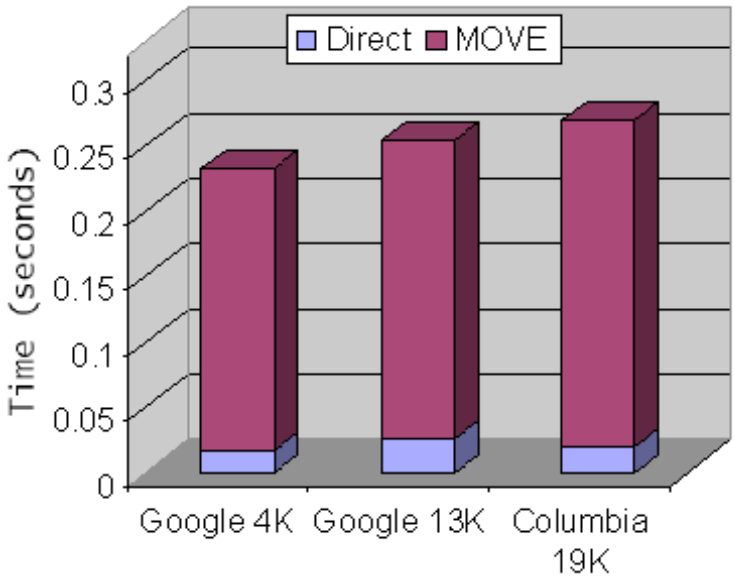


Figure 5.5: Average end-to-end transfer completion times (in seconds) for a client making a request directly, as well as through MOVE. In both cases, the web server is located on the same network as the browser. The testing was performed on a 96 node subset of the PlanetLab testbed using a Chord topology. Times are given in seconds.

We measured the end-to-end completion times of http requests as perceived by the client. That is, we measured the elapsed time starting when the client's browser initiates the http request for the web pages stored in the web server, to the time when all data from the web server have been received. We first contacted the web server directly (*i.e.,* without using MOVE), and then through MOVE. The downloaded page size was varied, with a different mix of images and text for each site. The latency overhead along with the number of files and their size for both direct and MOVE-assisted communication are given in Figure 5.5.

Table 5.2: Average end-to-end transfer completion times (in seconds) for a client making a request directly, as well as through MOVE. In both cases, the web server is located on the same network as the browser. The testing was performed on a 96 node subset of the PlanetLab testbed using a Chord topology. Times are given in seconds.

Server (Files, Size in Bytes)	Direct	MOVE
Google (index.html, 4,596)	0.016	0.216
Google (2 files, 13,154)	0.025	0.229
Columbia (index.html, 18,864)	0.020	0.25
Columbia (17 files, 61,331)	0.219	3.61

Table 5.3: Delay in re-establishing availability after disruption (due to DDoS) for an httpd server, migrated from the initial site to a co-located server (using NFS/UDP and SHFS/TCP), and to a remote site (using SHFS). The size of the server state was 9.8MB on average. We also include the round-trip latency between the target and migration servers in all cases.

Migration server	RTT Latency	Migration Time
Co-located (NFS)	1.02ms	0.761s
Co-located (SHFS)	1.02ms	1.162s
U. Penn (SHFS)	10.6ms	6.632s

To measure the delay in re-establishing the server's availability, which is the time during which a client using MOVE experiences service disruption, we used an Apache httpd server running under the default configuration. The server, initially located inside our local network, was moved to the *migration* server. The state of the server processes amounted to 9.8 MBytes on average. We measured the migration time in the following cases: when the migration server was located inside our local network, and when the migration server was located at the University of Pennsylvania, approximately 11 hops or 10ms *ping* time away (over Internet2). In both scenarios, the state files were transfered using the (Secure) SHell FileSystem (SHFS) [151], which operates over the popular SSH/SCP protocol suite. We chose SHFS because of its ease of installation and use; other filesystems, such as LBFS [119] or CODA [146] can also be used instead SHFS. For the local-migration case, we used NFS (over UDP) as a second way of transferring state. We show the results in Table 5.3.

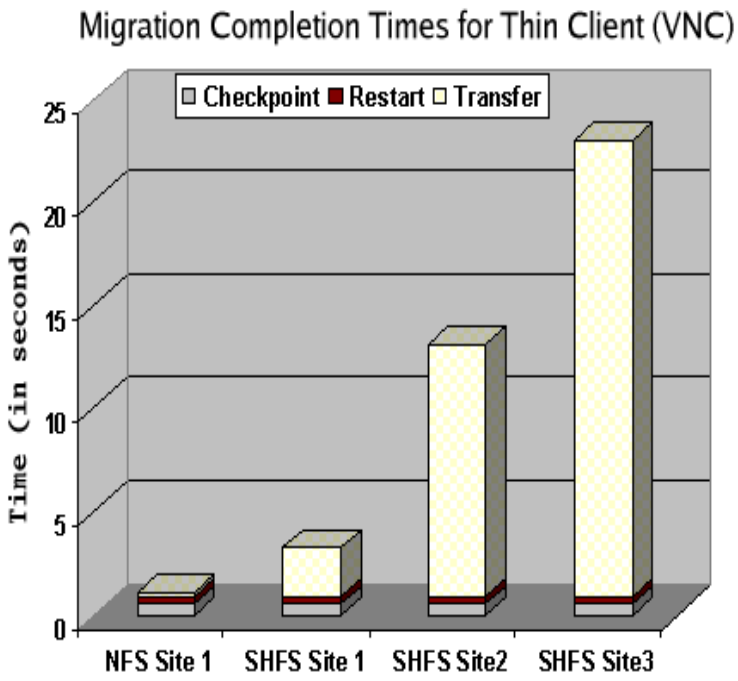


Figure 5.6: Average migration time (in seconds) for a VNC server migrates to a co-located server (*Site 1*) and to remote servers (*Site 2* and *Site 3*) using NFS/UDP and SHFS/TCP using a tunnel through the lifeline. We observe that the total time is dominated by the transfer time. Prior to the migration the VNC server was running Mozilla browser, Gaim, Kword, PS/PDF viewer and two terminal applications.

The availability delay varies between 1 second to 6.6 seconds, as round-trip times increase from 1ms (for the local-migration case) to 10*ms*. We believe that this level of disruption is acceptable in the presence of a DoS attack, when the alternative is total loss of service. One may observe that an attacker may be considered successful in some scenarios if they can cause service downtime of 7 seconds even few minutes, if only because the end users will be annoyed. One possibility we plan to examine in future work is the use of "hot spares" that are kept synchronized with the live service and to which we can divert traffic at sort notice through MOVE.

Note that, the case of accessing a local web server represents the worst possible scenario: we compare the access latency of an almost-direct connection between a browser and the web server to one that involves routing through 2 overlay nodes, for a total of over 20

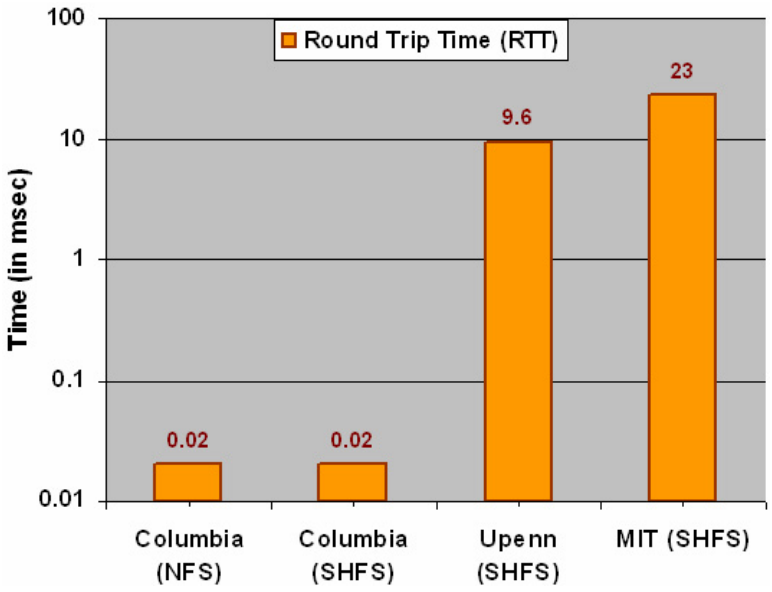


Figure 5.7: Round Trip Time (RTT) between the target server and the migration sites when using the lifeline tunnel. Note that the Y axis uses logarithmic scale.

intervening routers.

To better analyze the contribution of the lifeline connection to the overall latency for large interactive, non-caching applications like thin clients, we constructed the following experiment: initially, a VNC [137] server is located on the target server. The client connects via MOVE and creates a VNC session consisting of Mozilla browser, Gaim Instant Messenger, Kword, a PS/PDF viewer and two terminals connected to the target server. Upon detection of the attack, we checkpointed the POD containing the VNC session and server, transfered the state to the migration server and restarted it. The state transfered to the migrate server amounted to 55.9 MBytes. For the migration, we used a co-located server, *Site 1,* and two remote servers, *Site 2* and *Site 3,* with different network proximity to the target server. Figure 5.6 shows the average availability delay for this scenario, which is dominated by the transfer of the POD state; the relatively small checkpoint and restart overheads remain constant for all experiments.

In addition, the underlying file-access mechanism seems to play a significant role, especially when we establish a low-bandwidth link between the target and the migration

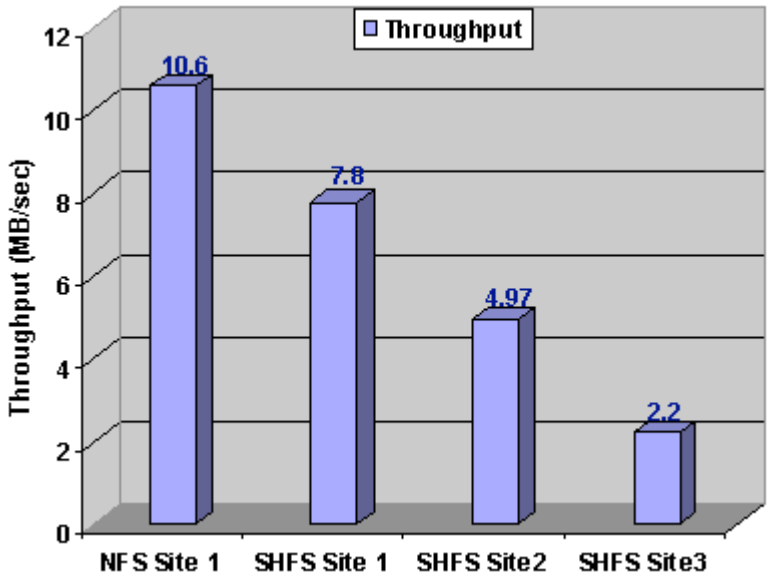


Figure 5.8: Average data throughput in MB/sec when accessing target server files from the migration sites using the lifeline tunnel. Notice that as we increase the network distance, we have a proportional decrease in the average data throughput

server. Accessing the target's database or file system via the lifeline can also increase the end-to-end latency. For servers that are either co-located or connected through a low-latency link (<5ms), we can use NFS/UDP since it is fast and reliable. However, if the established connection is of high latency (and low bandwidth), NFS/UDP becomes unresponsive. To measure the performance of our system when using a high-latency connection, we instead used SHFS. Figures 5.7 and 5.8 show the average round trip time (RTT) and effective throughput respectively for all the migration sites. We can see that as we increase the distance between the target and the migration server, we have a proportional decrease in the average data throughput. Figure 5.8 shows that when we use NFS to connect the target and migration servers for *Site 1,* we achieve better throughput compared to using SHFS.

Choosing a file-access method to connect through the lifeline depends both on the network proximity of the migrating site and the maximum network latency allowed by the application we migrate. For thin-client applications like VNC, all the applications are already loaded prior to migration and the user communicates with the target server only

to read or write data from within an application. Thus, the lifeline utilization is relatively low, allowing for relatively smooth operation under conditions of high network latency. For applications that access the filesystem frequently, such as HTTP servers, it is necessary to use a network file system with good caching characteristics. For database applications, we need to ensure that the lifeline can sustain the load from the migrated middleware server to the back-end database server.

Chapter 6

End-to-End Protection Using Stateless Multi-Path Overlays

6.1 A New Class of Attacks Against Overlay-based Services

Indirection-based overlay networks (IONs) are a promising approach for countering distributed denial of service (DDoS) attacks. Such mechanisms are based on the assumption that attackers will attack a fixed and bounded set of overlay nodes causing service disruption to a small fraction of the users. In addition, IONs depend on the inability of an adversary to discover connectivity information for a given client and the infrastructure (*e.g.,* which overlay node a client is using to route traffic). This makes them susceptible to a variety of easy-to-launch attacks that are not considered in the standard threat model of such systems. For example, adversaries may possess real-time knowledge of the specific overlay node(s) a client is routing traffic through, or may be attacking nodes using a time-based scheme that will try to maximize the impact of the attack on clients' connectivity. Such attacks can be network-oriented (*e.g.,* TCP SYN attacks [68]) or application-related "sweeping" attacks or "targeted" attacks.

In targeted attacks, an attacker that has knowledge of the client's communication parameters can "follow" the client connections and bring down the nodes that he tries to connect to. As soon as the client realizes (typically after some timeout period) that the overlay node is unresponsive and switches to a new node, the attacker also switches the attack to this new node. Thus, an attacker that can bring down a single node can create a targeted DoS for specific clients. Similar attacks, exploiting information that must only be available to trusted components of the system but which an attacker can feasibly gain access to, are possible against almost all proposed anti-DDoS mechanisms [178, 10, 76, 101].

In sweeping attacks, the attacker uses its power (which is insufficient to bring down the whole ION) to attack a small percentage of the overlay nodes at a time. This type of attack targets the application-level state maintained by the overlay node responsible for a client. Destroying this state forces the client to re-establish both network and application-level connectivity, degrading the clients' connection and leading to DoS for time-critical or latency-dependent applications. Thus, although IONs can counter blind DoS attacks, they remain vulnerable to a range of simple but debilitating attacks.

Most such systems concern themselves with *naive attackers*, *i.e.,* those without internal knowledge of the system (other than the list of participating nodes). We assume that such an attacker can mount a DoS attack against a small set of nodes in the overlay for short periods of time, which will force clients using those overlay nodes to reset their connections to new nodes. This attacker blindly "sweeps" all the nodes participating in overlay network focusing the attack from one set of overlay nodes to another, selecting nodes not previously attacked. Presently, a number of DoS attacks can be used as "sweeping" attacks: TCP SYN, ICMP flooding and TCP congestion attacks are among them. If the overhead of detecting the failure and switching to a new access point is high, compared to how long an attack must be sustained to force the connection reset, an attacker can cause significant disruption in the communications. Performance can be seriously degraded, and long-lived connections (such as a teleconference or a large file transfer) can be repeatedly disrupted,

rendering them ineffective as communication carriers. This attack is similar to a radio jammer that is randomly broadcasting noise in various channels, forcing communicating parties to continuously reset their network parameters.

Although less efficient against a single user compared to a targeted attack, this attack can be more effective, degrading the connection characteristics or preventing connectivity on most of the clients connected to the overlay. The success of the attack depends on factors such as the attack intensity[1], and the time required to detect the connection failure and then find a new overlay node that is healthy and re-establish both network connection and client authentication credentials (usually on the application level). Moreover, the client's authentication can be complex, *e.g.,* using X.509 certificates for authentication or Graphic Turing Tests [168] to allow anonymous human users. Most such authentication mechanisms require time and user interaction, which make these sweeping attacks a serious problem for real-world deployed overlays.

A more sophisticated attacker, explicitly not considered in other proposed IONs, may know which overlay node a client is using. An attacker can get this information by eavesdropping on an appropriate edge-network link: the client's wireless communications to his access point or the link to his ISP. Such an attacker can "follow" the client and direct DoS traffic against the overlay nodes that he tries to communicate with. The client, detecting a failure in communications, will select another node to access the overlay, which will become the attacker's new target. Using the radio communications analogy, this is akin to an adversary that is eavesdropping on wireless communications, jamming frequencies where traffic is detected; after a short period of time, the adversary searches for new frequencies the attacked parties may have switched to. [9] identifies possible ways an adversary can gain such information; other possibilities include snooping on the local network link, *e.g.,* in a wide-area wireless network such as the upcoming WiMAX, or in some enterprise-wide 802.11 (WiFi) environments.

[1] In this context, attack intensity is the percentage of overlay nodes that can be brought down simultaneously by the attacker.

This threat model is considerably stronger than the typical scenarios anti-DDoS mechanism designers have considered in the past. We can address all of the above attacks by employing a nearly-stateless spread-spectrum communication paradigm in conjunction with an overlay network.

6.2 Quantifying Attack Resistance

We now evaluate the security of our scheme using a simple analytical model, which we apply to first-generation IONs that are vulnerable to targeted or sweeping attacks. We then quantify the attack resistance generally offered by IONs using a simple model of an ISP and typical POP speeds. In the next section we will characterize the impact of our system on latency and throughput in a series of experiments over the Internet using PlanetLab. We don't need to quantify the impact of targeted attacks since they are successful in all cases where the attacker has eavesdropping abilities and can bring down a single overlay node.

6.2.1 Impact of Sweeping Attacks

First-generation IONs were geared towards service connection availability. No provision is made for attacks that cause the user to reset his connection, either because the overlay node is unresponsive or because the connection quality is low. After resetting the connection, the user has to re-establish connectivity and re-authenticate himself, making the system unrealistic for real-time applications. Moreover, frequently forcing the user to re-authenticate through a challenge-response or a CAPTCHA will render the system unusable for any type of application.

We assume that an attacker can mount a DoS attack against a small set of nodes in the overlay for short periods of time, which (in first-generation IONs) will force clients using those overlay nodes to reset their connections to new nodes. This attacker blindly sweeps all the nodes participating in the overlay network, focusing his attack from one set of overlay

nodes to another, keeping the sets disjoint. Not all of these attacks can be easily detected by the current infrastructure: an attacker can mount a low-rate TCP attack[99] reducing the effective bandwidth of the victim to zero. Thus, a sweeping attacker can cause significant disruption in the end-to-end communication.

To analyze a sweeping attack and quantify its impact to the clients' connection characteristics in first-generation IONs, we create a simple static model. We assume that the attacker can bring down p_d percentage of the overlay nodes simultaneously. For an attack to be successful on these nodes, it needs t_a time of sustained attack. This is the time required to either drop or severely rate-limit the connections of all the clients connected to nodes under attack. Let t_u be the average time a client is connected to the system. Also, let t_d be the time that is necessary for the client to detect the attack and connect to another overlay node. Moreover, we assume that the overlay repairs the nodes under attack immediately after the attack focus has shifted to another set of nodes (zero reboot or repair time) so the time to repair $t_r = 0$. Finally, we assume that clients are connected uniformly across all overlay nodes in a first-generation ION, *i.e.,* if there are N clients and O overlay nodes, each has $\frac{N}{O}$ clients. The percentage of clients that will have their connection reset by a sweeping attack at least once during the time that they use the system is:

$$P_1(t_u, t_a, p_d) = \frac{t_u}{t_a} \cdot p_d \;\text{ assuming }\; t_{det} = 0$$

.

We prove that this is the case using the following propositions:

Lemma 2 *The percentage of users that will have to reset their connections at least $k > 1$ times during the attack is:*

$$P_k(t_u,\ t_a,\ p_d) = \sum_{i=1}^{\lfloor \frac{t_u}{t_a} \rfloor} P_{(k-1)}(t_u - i \cdot t_a,\ t_a,\ p_d) \cdot p_d \tag{6.1}$$

with t_u: avg user time, t_a: attack time, p_d: % of nodes attacked simultaneously. We assume immediate attack detection ($t_d = 0$)

Proof: The percentage of users that will be affected by the attack at least once is:

$$P_1(t_u,\ t_a,\ p_d) = \frac{t_u}{t_a} \cdot p_d \tag{6.2}$$

Notice that the above probability can go above 100% if $t_u >> t_a$, meaning that the attack will certainly affect the clients possibly more than once. When $P_1 > 100\%$ we say that $P_1 = 100\%$, *i.e.,* $P_1 = min(100, \frac{t_u}{t_a} \cdot p_d)$. We will prove (6.1) using induction.

Base case for $k = 2$, in that case (6.1) becomes:

$$P_2(t_u,\ t_a,\ p_d) = \sum_{i=1}^{\lfloor \frac{t_u}{t_a} \rfloor} P_1(t_u - i \cdot t_a, t_a, p_d) \cdot p_d \tag{6.3}$$

In our model, the attacker can only attack $\frac{t_u}{t_a} \cdot p_d$ sets of nodes. We say that a client suffers an attack when the set of overlay nodes that he is connected to is attacked. The probability for a client to be at the first node is p_d. After realizing an attack is underway, in t_d time, the client will select a new overlay node. The probability that this new overlay node is part of the attack window, and thus the client will suffer another attack, is $P_1(t_u - t_a, t_a, p_d)$ since the attacker will have to spent t_a time attacking the first set of nodes.

Thus, the probability to be attacked at least twice when the client happens to be in the first set of attacked nodes is $P_1(t_u - t_a, t_a, p_d) \cdot p_d$. For a client connected to the second set of nodes the probability to be attacked twice is $P_1(p_d, t_u - 2 \cdot t_a, t_a) \cdot p_d$ since the attacker will have to spent $2 \cdot t_a$ time attacking the first and the second node before reaching any other node. Another way of saying the same thing is that the user will have $t_u - 2 \cdot t_a$ time left in the system reducing the probability of being attacked. A client that is connected to a node in the i^{th} set has a probability $P_1(t_u - i \cdot t_a, t_a, p_d)$ to be re-attacked. A client has probability p_d to be connected to a set and by summing up the fraction of clients connected to i^{th} set for

which $t_u - i \cdot t_a > 0$, we get (6.3).

We assume that the formula holds for $k = j$ and we will prove that it holds for $k = j + 1$. P_k is the probability that a client will be re-attacked at least k times. If the client is on the first set attacked, the probability of being attacked $j + 1$ times is the probability of initially being at the first set, which is p_d, multiplied by the probability that he will select overlay nodes which can be re-attacked j times in the $t_u - t_a$ remaining time. The probability of both being in the first attacked set and being re-attacked j more times is: $P^1_{j+1} = P_j(t_u - t_a, t_a, p_d) \cdot p_d$. For a node that connects initially to the i^{th} set we get that the probability of being attacked $j + 1$ times is $P^i_{j+1} = P_j(t_u - i \cdot t_a, t_a, p_d) \cdot p_d$. If we sum all the sets i for which $t_u - i \cdot t_a > 0$, we get (6.1). ■

The above formula is very intuitive: from the attacker's perspective, there are $\frac{1}{p_d}$ disjoint sets of nodes in the overlay network. To attack all of them the attacker needs $\frac{t_a}{p_d}$ time. Assuming a system where we have no joins, an attacker will affect the connectivity of $\frac{t_u}{t_a} \cdot p_d$ clients. Note that some of the clients may never experience the attack because they might have finished their connection by the time the attack reaches them. For this simple model, we have assumed that there is no detection time: the user selects another overlay node to connect to as soon as the attack starts to affect him. Even with this very conservative model ($t_d = 0$, $t_r = 0$, no client arrivals while the system is under attack) we can see that the attack can be significant, depending on the usage time, the size of the attack compared to the size of the overlay and the time required for an attack to be successful. For example, assuming that we have clients with average usage time of an hour, an adversary that can attack 2.5% of the overlay nodes and shifts the attack every 5 minutes will affect 30% of the clients. We can also compute the percentage of nodes that will have to reset their connections more than once. The percentage of nodes that will have to reset their connections at least $k > 1$ times during the attack is:

$$P_k = \sum_{i=1}^{\lfloor \frac{t_u}{t_a} \rfloor} P_{(k-1)}([t_u - i \cdot t_a],\ t_a,\ p_d) \cdot p_d$$

In general, for $t_d < t_a$, we have:

$$P_k = \sum_{i=1}^{\lfloor \frac{t_u}{t_a} \rfloor} P_{(k-1)}([t_u + t_d - i \cdot t_a],\ t_a,\ p_d) \cdot p_d$$

whereas if $t_d \geq t_a$, we have: $P_k = (\frac{t_u}{t_a} \cdot p_d)^k$

Lemma 3 *In the general case. where $t_d \geq 0$, the percentage of nodes that will have to reset their connections at least $k > 1$ times during the attack is:*

a) if $t_d \geq t_a$ we have that: $P_k(t_u,\ t_a,\ p_d) = (\frac{t_u}{t_a} \cdot p_d)^k$

b) if $t_d < t_a$ we have:

$$P_k(t_u,\ t_a,\ p_d) = \sum_{i=1}^{\lfloor \frac{t_u}{t_a} \rfloor} P_{(k-1)}([t_u + t_d - i \cdot t_a],\ p_d,\ t_a) \cdot p_d \tag{6.4}$$

Proof: To compute the probability when $t_d > 0$, we assume that the user is not going to be discouraged by the attack and will want to use the system for t_u time.

We derive (3) using the fact that since $t_d \geq t_a$, the client will have the same probability to select a set of overlay nodes that will be attacked as it had at the beginning of the attack: $\frac{t_u}{t_a} \cdot p_d$. The percentage of the users that will be attacked k times is $(\frac{t_u}{t_a} \cdot p_d)^k$

Equation (6.4) follows from proposition 2 if we change the usage time of a user from t_u to $t_u + t_d$, *i.e.,* the user will have to pay a penalty of t_d each time he is attacked, increasing his total time usage time by the same amount. ■

The probability that a client will be affected does not change, since the attack will continue to another set of overlay nodes and thus when the client tries to reconnect he will have the same probability of being affected, assuming he wants to keep using the system paying a penalty of t_d for each reset.

Our spread-traffic system is invulnerable to these attacks, since there is no single node

that maintains all client-specific state for a given client. Attacking a small percentage of overlay nodes will cause a corresponding packet loss in the end-to-end communication. If the attacked nodes are a small percentage of overlay nodes (corresponding to low packet loss), the end-to-end transport protocol (*e.g.,* TCP) should be able to recover.

6.3 A novel communication paradigm

We believe that the previously mentioned, *inherent* limitations of first generation overlay-based traffic redirection mechanisms can be addressed by adopting a spread-spectrum like communication paradigm[2]. In a "spread-spectrum" approach, the client spreads its packets randomly across all access points, preventing an attack from "following". The path diversity naturally exhibited by a distributed overlay network serves as the "spectrum" over which communications are "spread." In our system, a token issued by the overlay network to the client is used to verify the authenticity of each packet communicated by the client. The use of a token (akin to a Kerberos ticket) [113] alleviates the necessity to maintain application or network-level state at any of the overlay nodes (unlike previous IONs), at the expense of bandwidth (since the ticket must be included in every packet routed through the ION). In return, our system is impervious to the attacks that use this state dependence to attack the overlay.

The main challenges we must address relate to the scheme's efficiency (in terms of performance and latency of the end-to-end path), resiliency to attacks, amount of state that needs to be maintained by each overlay node (necessary to prevent packet replay or forging attacks), and the elimination of communication pinch points on which attackers can focus their attention.

[2]Note that although we use the term "spread-spectrum" to describe our approach, our work is *not* geared towards wireless networks, nor does it touch on physical-layer issues.

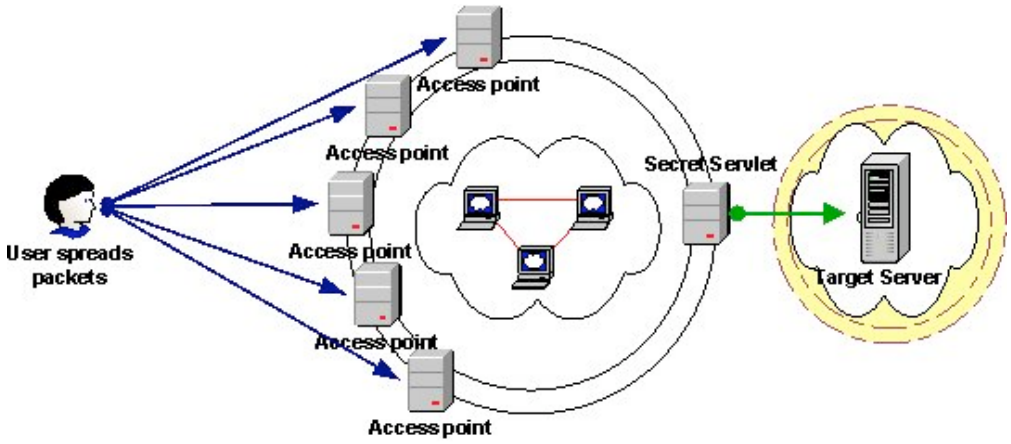


Figure 6.1: Spreading traffic across multiple overlay access points. Attacks that render a number of overlay nodes ineffective do not impact end-to-end communications.

6.3.1 Traffic Spreading

The first problem we address is how to protect the communications of a client of the overlay from attackers that either have partial knowledge of the communication parameters (*i.e.,* can determine which overlay nodes a client is communicating with), or are blindly attacking overlay nodes using "sweeping" attacks, thus forcing clients to keep re-establishing connections to new overlay nodes. For simplicity, we temporarily assume that the reverse channel (from the overlay to the client) is protected by the overlay in the same manner communications to the server are protected, or is otherwise safe from interference.

Our approach, shown in Figure 6.1, is straightforward: spread the packets from the client across all overlay nodes in a pseudo-random manner storing no network or application level state in the overlay nodes. An attacker will not know which nodes to direct an attack to; randomly attacking a subset of them will only cause a fraction of the client's traffic to be dropped. By using forward error correction (FEC) or simply duplicating packets (*i.e.,* sending the same packet through two or more different access points simultaneously), we can guarantee packet delivery with high probability, if we place an upper bound on the number of nodes an attacker can simultaneously attack. In designing our system, we must address several issues:

- First, it should not be possible for an attacker to impersonate a legitimate user and conduct a DoS attack through the overlay. This means that each packet from the user to the overlay must be properly authenticated.

- The second issue we must address is the state that each overlay node must maintain per client: since all overlay nodes can potentially receive traffic from all users, the memory requirements can quickly become prohibitive. Furthermore, a client's end-to-end connection must not depend on the network availability of a small set of overlay nodes. Keeping state that is essential for a client's network or application level connectivity makes the system vulnerable to sweeping or targeted attacks.

- Third, even legitimate clients should not be allowed to "pump" unlimited amounts of data through the overlay; verifying this is complicated due to the packet-spreading approach.

- Finally, the selection of the overlay node to forward a packet through should be as random as possible from the point of view of an external observer (*i.e.,* an attacker), yet verifiable by individual nodes, to avoid flooding attacks by compromised clients.

In the remainder of this section we describe two protocols: one used to establish a restricted ticket and secret session-key between a client and the overlay, and a second protocol used as a stateless communication protocol that allows overlay nodes to verify the validity of received packets without requiring maintenance of large amounts of state.

6.3.2 Key and Ticket Establishment Protocol

To achieve a stateless communication with the overlay network, a client has to acquire a ticket, which is then included in all subsequent packets sent through the overlay. As we will see in detail in the next section, the ticket is used by the overlay nodes to authenticate the user, validate the routing decisions, and prevent malicious (or subverted) nodes from

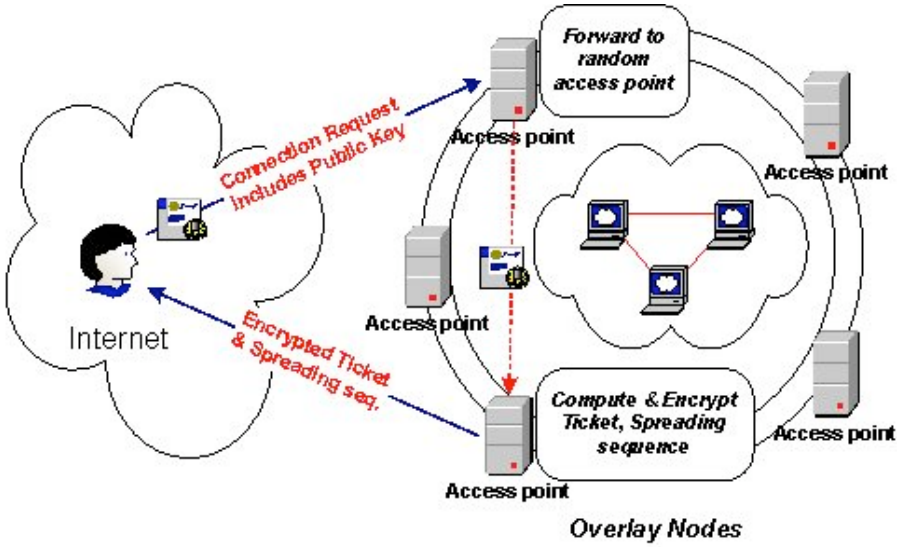


Figure 6.2: Redirection-based authentication and key establishment. An attacker observing the interactions of a user and the overlay cannot determine which overlay node(s) to target.

utilizing a disproportionate amount of bandwidth. Thus, node authentication and ticket acquisition/maintenance is a key component of our approach. Although any authentication protocol could be used, most such protocols require at least two round-trips between the two parties (as well as considerable computation).In fact, this has been shown to be the minimum number of messages in a protocol that offers certain security guarantees [4, 98]. However, an attacker that is observing communications between the client and the overlay can direct a congestion-based DoS attack[3] against any overlay node that is contacted by the client for authentication purposes. Since the client does not yet have a spreading sequence, it seems at first impossible to protect the key establishment phase.

Our proposed approach is to randomly redirect the authentication request, as shown in Figure 6.2. Briefly, the client selects an overlay node at random and sends a packet containing its public key certificate and a request to initiate authentication. The receiving node immediately forwards the request to another overlay node at random; thus, an attacker (who cannot react fast enough to prevent a packet from being forwarded on) does not have a target.

[3]Computational DoS attacks can be partially mitigated using proof-of-work techniques [79, 46].

The second overlay node selects a random session key K_u and creates a ticket for that client. The ticket contains K_u, a range of packet sequence numbers for which K_u and the ticket are valid, a randomly selected identifier for the client, the current time-stamp, and flags indicating that this is a "restricted" ticket (more on this later), all encrypted and authenticated under K_M, a secret key negotiated periodically (*e.g.,* every few hours) among all overlay nodes (see Figure 6.3). The last part of the ticket is a UMAC [22] signature of the encrypted ticket using K_M and a 64-bit nonce, which consists of the first 64 bits of the encrypted ticket. Note that only overlay nodes can validate and decrypt the ticket. The client's certificate is validated, and a second copy of K_u is independently encrypted under the client's public key. Both operations are relatively lightweight (compared to operations involving RSA private keys); as was shown in [93], a node can perform a few thousand public-key operations (*i.e.,* signature verifications or public-key encryptions) per second. The ticket and the encrypted session key are then sent to the client. An extra, optional message can be sent from the overlay to the client with the list of overlay nodes' IP addresses. This one-round-trip protocol is stateless (for the overlay) and computationally fast, resisting both memory and CPU exhaustion attacks on the overlay nodes.

To make it even more difficult for the attacker to mount a CPU exhaustion or IP spoofing attack, we can add one more round-trip on the key establishment protocol, forcing the client to send a UMAC-signed certificate before generating the ticket (which requires validation of the client certificate). Figure 6.4 displays both the one round-trip and the two round-trip key establishment protocol in detail. In the two-round-trip protocol, the client sends his certificate to overlay node A. A redirects the request to B, a randomly selected overlay node. B treats the certificate as a random number, which he UMAC-signs with the shared key K_M. The client's IP address and the system's timestamp are the nonce used in the UMAC operation. B sends the UMAC signature and the nonce to the client. To prove liveness, the client contacts another randomly selected overlay node, C, sending its certificate, the UMAC signature and the nonce. C validates the authenticity of the UMAC and redirects

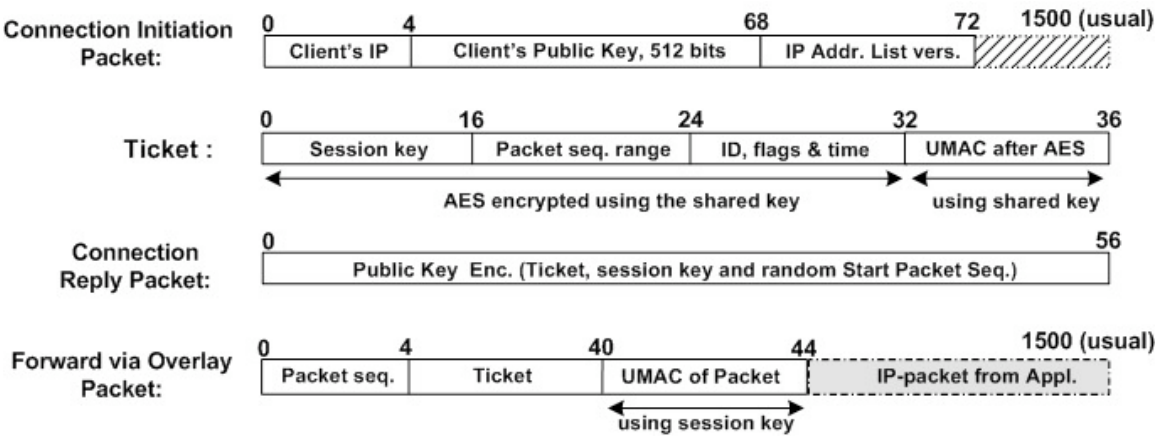


Figure 6.3: The layout of the various packets and the ticket used to establish a communication and transmit packets between the client and overlay nodes. All numbers are in bytes, unless otherwise indicated.

the request to D, another randomly selected overlay node. Finally, D generates a ticket for the client, encrypting it with the client's public key (retrieved from the certificate). In the two-round protocol, only the last step is computationally expensive (compared to simple UMAC verification). Thus, the two-round-trip protocol, guarantees client liveness. For the one-round-trip protocol we only use the first and the last communication *i.e.,* from A to D as shown in Figure 6.4. Finally, if there is a version mismatch between the list of overlay nodes' IP addresses stored locally in the client (communicated by the client in the first message) and the one stored in the overlay network, a random overlay node, E, is chosen by D to send the list differences to the client.

6.3.3 Client Authentication

The ticket obtained from the previous protocol can only be used by the client to continue the authentication protocol (*i.e.,* prove liveness for both the overlay and the client. Once two-party authentication is completed, the last overlay node provides the client with a ticket that is not "restricted," *i.e.,* the corresponding flag inside the ticket is cleared. The tickets are periodically refreshed, to avoid situations where a malicious user distributes the session

key and ticket to a large number of zombies that try to access the overlay.

This authentication step can be followed by a secondary authentication phase that uses a Graphic Turing Test (GTT) [168] to discern the presence of a human at the client node (versus a remotely controlled DDoS zombie). This step can prevent legitimate nodes that have been subverted by an attacker from being used as entry points to the overlay, but can only be used for those applications that have a GUI — such as a web browser. We can implement the secondary GTT-based authentication by issuing a second restricted ticket after the completion of the two-phase authentication step (from above), which only allows client nodes to contact the GTT server. This server is implemented locally by each overlay node, as was shown in [117]. Once the GTT step is successfully performed, the GTT server issues an unrestricted ticket to the client node. The GTT authentication can be performed periodically (to confirm the continued presence of a human). Naturally, this step is not applicable for applications where there is no human being directly controlling the client, or where displaying a graphic is infeasible or impractical (or for vision-impaired persons).

6.3.4 Client-Overlay Communication Protocol

Once the client has received a session key and an unrestricted ticket, he may start sending packets to the remote destination through the overlay. Each packet sent by a client to an overlay node contains three overlay-related fields: the ticket, an authenticator, and a monotonically increasing sequence number, as shown in Figure 6.3. The ticket contains the session key and a sequence range for which the ticket is valid, as we discussed previously, and is encrypted and authenticated under a secret key K_M, known to all overlay nodes. Note that these overlay nodes are *not* user machines, but are hosts dedicated to offering a DoS protection service.

The sequence number is a 32-bit value that is incremented by the client for each packet transmitted through the overlay with a given session key. The client identifier is a random 32-bit value that is selected by the overlay node that authenticated the client, and is used as

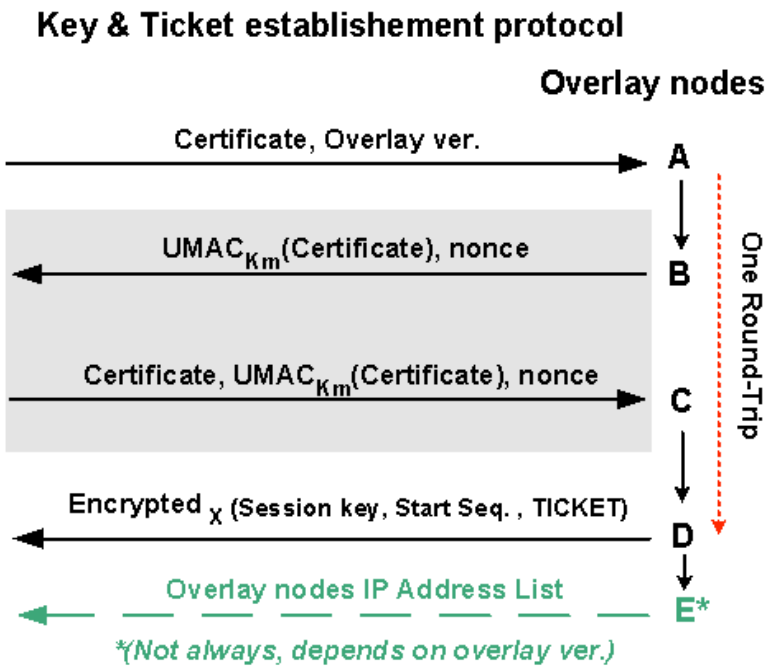


Figure 6.4: Key & Ticket Establishment protocol: The client sends node *A* his certificate. *A* immediately redirects the request to *B*, in the two-round-trip protocol, or to *D* for the one-round-trip protocol. The four-message protocol is more resilient against computational attacks since it ensures the client's liveness before generating an encrypted version of the ticket. A 5^{th} message is transmitted when the client's version for the list overlay nodes is old.

an index in the table of last-seen sequence numbers per client, maintained by each overlay node. The authenticator is a message authentication code (MAC) using a fast transform such as UMAC and the session key K_u. The UMAC is computed over the whole packet, which includes the ticket and the sequence number of the packet. For the UMAC nonce we use the sequence number concatenated with the client's IP address. Thus, the ticket is bound to a specific IP address and cannot be distributed to other clients. The only state each overlay node needs to maintain per client consists of the client identifier and the last sequence number seen by that particular client. This state is not network or application related and is used solely to prevent "replay" attacks. Assuming that both the client identifier and the sequence number are 32-bit values, each overlay node needs to maintain only 64 bits of state for each client; thus, if the overlay could support 1 million active clients (in

terms of network capacity), we will only need 8 MB of state.

Communication through the overlay.

A client transmitting a packet through the overlay uses the session key and the sequence number as inputs to a pseudo-random function (PRF). The output is treated as an index to the list of overlay nodes, through which the packet will be routed. The list of available overlay nodes does not need to change frequently, even if nodes become unavailable (*e.g.,* for maintenance purposes). There are various ways a client can obtain the list of overlay nodes. For example, it can be done the first time it connects to the overlay network by requesting it after the key establishment phase, or by downloading it independently of the protected communication. After the first time, the client can maintain the freshness of the list by comparing the version of his list with the one stored in the overlay, downloading only the differences of the two versions.

The client then encapsulates the original packet (addressed to the final destination) inside a packet for the overlay node, along with the information identified above (ticket, sequence number, authenticator). This packet is forwarded through the overlay to the appropriate secret servlet, and from there to the final destination.

Upon reception of a packet, the overlay node checks the validity of the ticket. This is a UMAC validation, a fast operation preventing computational DoS attacks against the overlay nodes. After validating the authenticity of the ticket, the ticket is decrypted and the authenticator is verified. This prevents spoofing attacks from an adversary who obtains a valid ticket and generates packets to all overlay nodes with randomly selected sequence numbers, thus preventing the client with the valid ticket to communicate. Furthermore, to detect any replay attacks, an overlay node that receives such a packet verifies that the sequence number on the packet is larger than the last sequence number seen from that client by using the client identifier to index the internal table. The overlay node also verifies that the sequence number is within the acceptable range of sequence numbers for this ticket.

Finally, it uses the key and the sequence number along with the PRF to determine whether the client correctly routed the traffic. If all steps are successful, the overlay node updates the sequence number table and forwards the packet to the secret servlet. Packets with lower or equal sequence numbers are considered duplicates (either accidental reordering or malicious replays by attackers) and are quietly dropped.

To avoid reuse of the same ticket by multiple DDoS zombies, the range of valid sequence numbers for the ticket is kept relatively small (and contained inside the ticket), *e.g.,* 500 packets. Moreover, the ticket is bound to the client's IP, since to authenticate the packet the overlay uses the UMAC including the client's IP address as part of the UMAC nonce. In addition, each packet contains a timestamp with which we can validate the freshness of the ticket. After a configurable period of time (*e.g.,* 1 or 2 hours) the overlay expires the ticket. Overlay nodes that receive valid tickets about to expire simply re-issue a new ticket with the same session key but a new range of valid sequence numbers. This approach, combined with the state kept by each node, makes it prohibitive for attackers to reuse the same ticket from a large number of distinct nodes (each of which is only transmitting to a specific overlay node), since the new valid ticket needs to be continuously propagated to all zombies.

The shared key under which the ticket is encrypted is periodically established among all overlay nodes, using a group key management protocol including [95]. The precise properties of this protocol are not relevant to this discussion, and there exist a large number of such protocols in the research literature.

We experimentally verified the validity of this analysis with the prototype on the PlanetLab network, as we discuss next.

6.4 Performance Evaluation

Just as important as security is the impact of our system on regular communications, whether under attack conditions or otherwise; a prohibitively expensive mechanism (in terms of

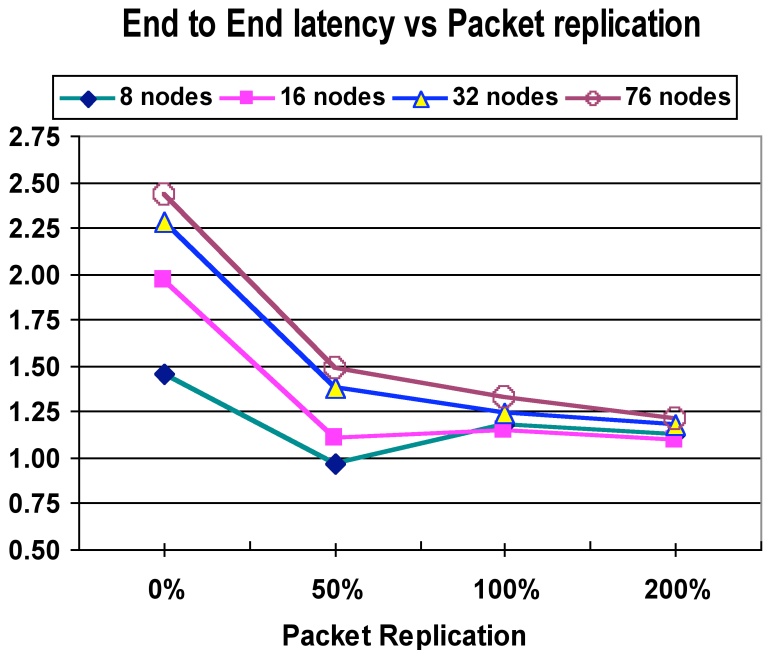


Figure 6.5: End-to-end average latency results for the index page and a collection of pages for www.cnn.com. The different points denote the change in the end-to-end latency through the overlay (T_o) when compared to the direct connection (T_d).

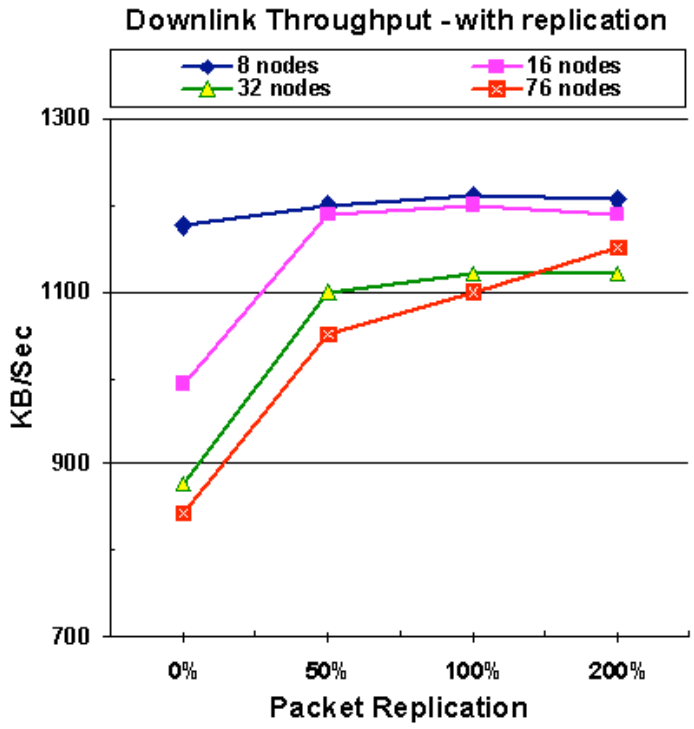


Figure 6.6: Throughput results in KB/sec. When we increase the replication, the results become closer to what we have observed for the direct connection (1250 KB/sec).

increased end-to-end latency or decreased throughput) is obviously not an attractive solution. In our experiments, we measured the communication overhead of our system in terms of end-to-end throughput and latency. To provide a realistic network environment, we deployed and used our prototype with 76 PlanetLab nodes [133].

For our evaluation, we used a testbed consisting of Planetlab machines located at various sites in the continental US. Those machines were running UML Linux on commodity *x86* hardware and were connected using Abilene's Internet-2 network. Using these fairly distributed machines, we constructed our overlay network of access points by running a small forwarding daemon on each of the participating machines. In addition, we used two more machines, acting as client and server respectively. In our experiments, we measured link characteristics such as end-to-end latency and throughput when we interposed the overlay network of access points between the client and the target server. To measure throughput, we used a target server that was located at Columbia. For our latency measurements, we used `www.cnn.com` as the target. In both cases, the goal of the client was to establish a communication with the target server. To do so, the client used UDP encapsulation on the TCP packets generated by an SCP session and then spread the UDP packets to the nodes participating on the overlay network, as we described in Section 6.3.1. Those packets were in turn forwarded to a pre-specified overlay node (the secret servlet). This node decapsulated and forwarded the TCP frames to the target server. Since our throughput connection measurements involve a client and a server that were co-located, we effectively measured the worst possible scenario (since our otherwise local traffic had to take a tour of the Internet). A non-co-located server would result in a higher latency and lower throughput for a direct client-server connection, leading to comparatively better results when we use the overlay. Surprisingly, in some cases we can achieve better latency using the overlay rather than connecting directly to the server.

Figure 6.6 shows that the impact on the downlink is only 33% in the worst case scenario, and it is easily amended by adding packet replication in the uplink direction. Again, we

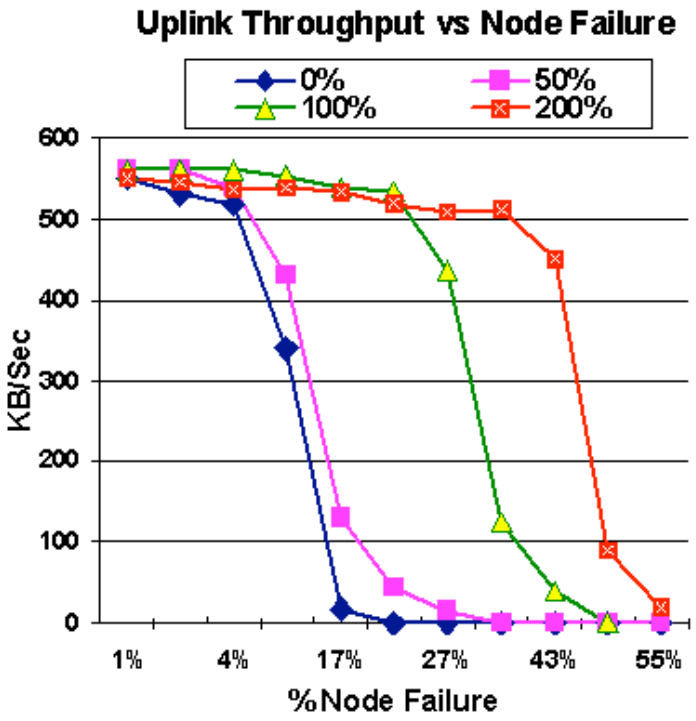


Figure 6.7: Throughput results in KB/sec when we utilize the uplink of our client under attack. The attack happens on a random fraction of the overlay nodes. Packet replication helps us achieve higher network resilience, something that we expected from our analytical results.

notice that the replication factor can cause a drop in the throughput for values > 100% in small overlay networks. Looking at the end-to-end average latency results in Figure 6.5, we notice that as we increase the replication factor, and for larger networks, we get better average latency results. In the worst-case scenario, we get a 2.5 increase in latency, which drops to 1.5 with 50% packet replication (*i.e.,* probability of replicating a packet of 50%).

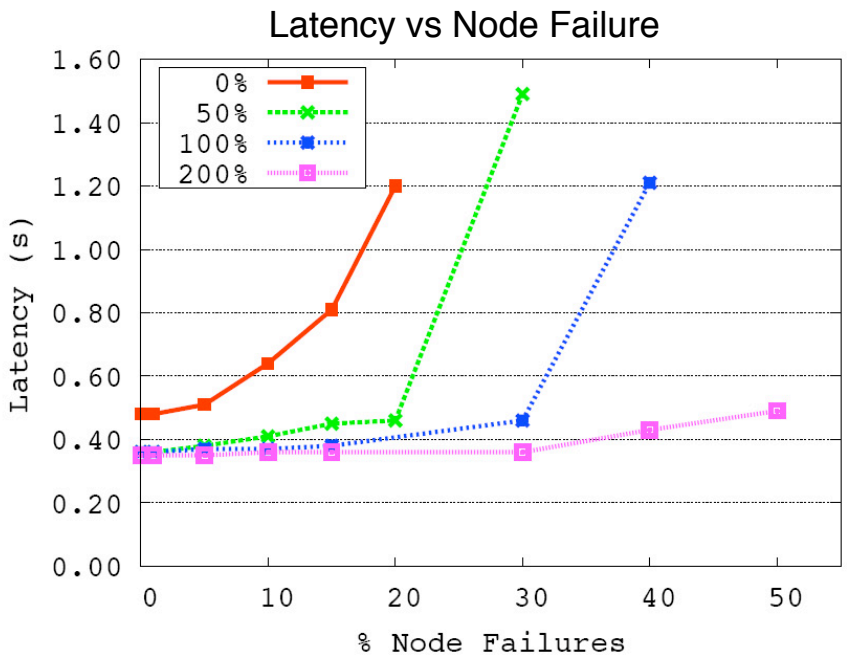


Figure 6.8: Impact of attacks against the overlay network on end-to-end latency. Different curves represent varying levels of packet replication. With 200% packet replication, latency increases by less than 25% when up to 50% of nodes are rendered unusable by an attacker.

To measure the effectiveness of our system in the presence of attacks, we performed an attack by bringing down overlay nodes at random. In our experiment, the client kept spreading data across all overlay nodes, since he was unaware which of the overlay nodes were being attacked. We then varied the portion of the overlay nodes we attacked and we measured the throughput of the resulting link. Figure 6.7 shows the decrease in the uplink throughput of the system when under attack. The attack happens on a random fraction of the overlay nodes. When we do not use any replication and depend on TCP to "recover" the lost packets, the connection performs relatively well when the losses are up to 9%-10%

of the total packets transmitted. Notice that as we increase the packet replication factor, we achieve higher network resilience, something that we also expected from our analysis. Corresponding results for latency are given in Figure 6.8.

Finally, we measured the number of tickets a single overlay node can generate. The ticket can be broken into four parts: the session key generation, the AES encryption of the ticket, the computation of the UMAC tag and the encryption of the packet using the client's public key. It appears that even for small size public keys (*e.g.,* 256 bits) the public key encryption takes up 95% of the ticket generation time. The number of tickets that can be produced by an overlay node decreases as we increase the size of the client's public key, as shown in Figure 6.9. Using a 3GHz Intel Pentium 4 machine, we were able to generate approximately 11,862 tickets/sec. In an ION with 128 nodes, the ticket-generation subsystem could sustain 1.5 million new users per second, assuming a random distribution of users across ION nodes.

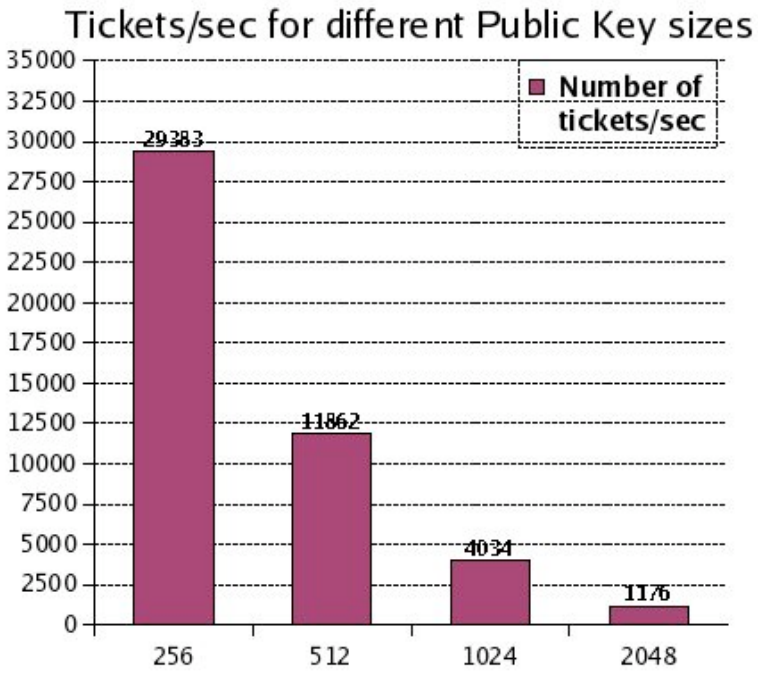


Figure 6.9: Tickets/sec produced from a single overlay node as we vary the size of the client's public key. The machine used was a 3GHz Intel Pentium 4 with 1GB of RAM.

6.5 TCP-Friendly Packet Spreading

The use of packet replication enabled us to become resilient to attacks or failures against a large fraction of our communication infrastructure. However, this resilience comes at the expense of having to utilize the network by transmitting extra information in the form of additional packets. In our threat model, we have assumed that we do not have any channel feedback or the channel feedback we have changes so fast that keeping track which of the underlying channels are operational becomes too expensive. This assumption is crucial for our because it is based on the fact that the attacker has knowledge of the underlying network connectivity and she can focus her attack to any of those links or subset of links. Therefore, when we use TCP connections between the client and the IBN or the IBN and the server (see Figure 6.10), an adaptive attacker can shift her attack from connection to connection resetting the state kept on each of the nodes. Moreover, there are attacks who attempt to exploit the TCP protocol packet retransmission algorithm with low-rate TCP-targeted DoS attacks[99]. To make things worse, when we employ TCP, we have to maintain state for **all** the TCP connections between the overlay and **every** client. For example, for a 1000-node overlay and 1 million users, we will have to maintain state for 1 billion TCP connections. TCP is clearly not scalable both with respect to the number of overlay nodes and the number of users in the system.

Furthermore, using TCP for real-time applications is not practical, since the protocol combines error control and congestion control, an appropriate combination for non-real time reliable data transfer, but inappropriate for loss-tolerant real time applications [131]. However, it is important for any communication protocol to ensure that it can coexist with current TCP-based applications. A key requirement of such a co-existence is the ability to reduce the transmission rate in the face of network congestion or packet loss to avoid starving the rest of the TCP streams. To that end, we need to equip our communication protocol with a "TCP-friendly" congestion control algorithm. We use the definition of TCP-friendliness presented in prior work by S. Floyd et.al. [109]: if a non-TCP connection

shares a bottleneck link with TCP connections, traveling over the same network path, then the non-TCP connection should receive the same share of bandwidth (i.e., achieve the same throughput) as a TCP connection. In addition, we would like the congestion algorithm to achieve the same fairness property for flows that we generate.

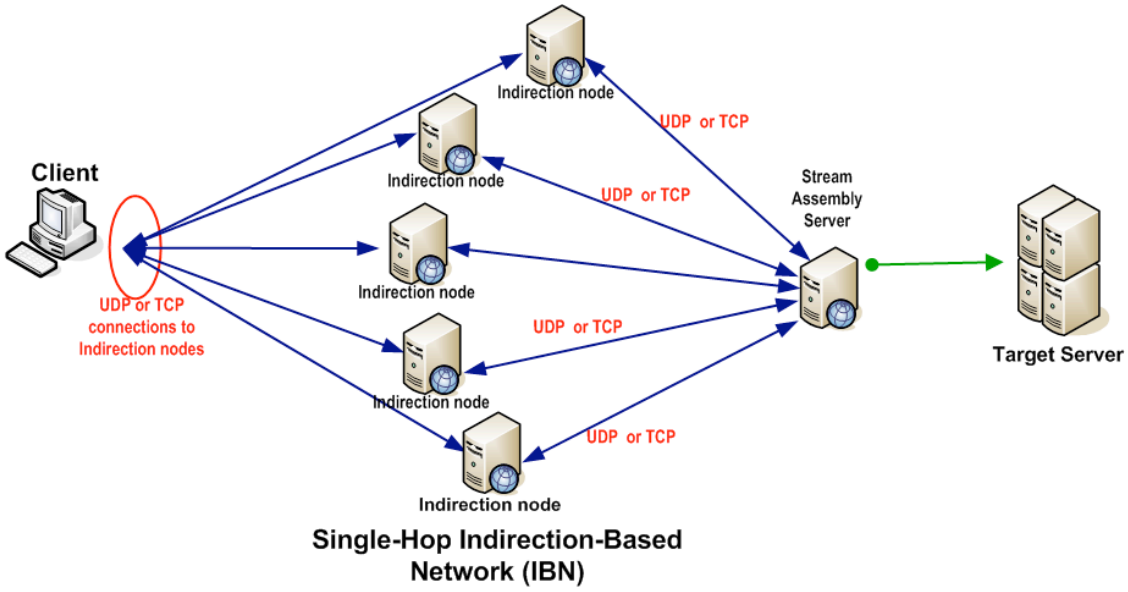


Figure 6.10: Single-hop indirection-based network using UDP or TCP connections.

Being TCP-friendly is important because we do not want to compete and obtain bandwidth by starving the rest of the TCP connections that happen to utilize the network lines that we utilize. To that end, we employ TCP-Friendly Rate Control (TFRC) protocol [128] (standardized in RFC 3448 [65]) , a two-end congestion control protocol for streaming media. Our system works by employing multiple connections and uses TFRC in *each* one of these connections, keeping independent state for each of the lines: if a line is congested then the packets send to the connection through that line get discarded if we have new data to send over that line. Initially all lines get the same amount of packets but as time goes by, we build weights for each of the lines based on the amount of packets that they were able to send on the previous round. The moment that there is a congestion, the TCP-window for the connection is re-calculated using TFRC formulas.

Both attacks and congestion of an intermediate link can lead to decreasing the weight of a line and thus essentially refraing from transmitting more traffic (unlike simple UDP). Moreover, we drop packets that are in the queue if there are more packets we have to send since we consider them stale. These packets have to be recovered using packet replication that use other, less congested lines. If all the paths we use are congested or become unavailable then our system, we will not be able to send any data and maintain any quality of service since there is no available capacity in the network that we can use. To quantify our ability to share fairly with TCP flows we measured the ratio of bandwidth achieved by regular TCP connections utilizing the same link as our UDP-TFRC implementation:

$$R = \frac{\text{Average UDP-TFRC Throughput}}{\text{Average TCP throughput}}$$

For $R > 1$ TCP flows achieves more throughput where as for $R < 1$ UDP-TFRC manages to push more traffic over the same link than TCP.

Figure figureTFRC-friendliness shows that when we share congested links with regular TCP connections, we are getting equal or less bandwidth (i.e. we are less aggressive that regular TCP-flows).

TFRC has two parts: a sender-side part and a receiver-side part. Each packet carries a sequence number and a timestamp indicating the time the packet was sent. The receiver acknowledges each packet, by sending an ACK that carries the sequence number and timestamp of the packet it is acknowledging. In addition to the sequence number and the timestamp, the ACK also carries a bit vector of 8 bits indicating whether or not each of the previous 8 packets was received. This allows some protection against ACK losses. The sender uses these ACKs to compute round trip time and base timeout value in a manner similar to [160]. The ACKs, along with the bit vectors they carry, can also be used to count the number of lost packets in a round. The sender can thus estimate the loss rate on the path between the sender and the receiver. At the end of each round, the sender uses the then value of round trip time, base timeout value and the loss rate measured over the previous round to

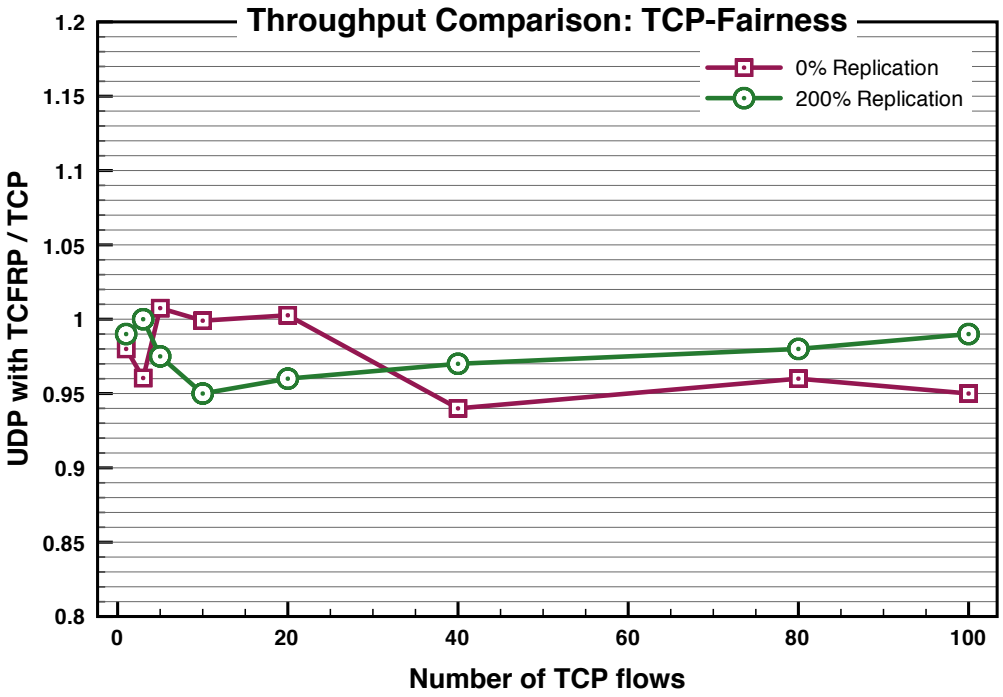


Figure 6.11: Throughput of UDP communication using TFRC when we use packet replication

calculate the sending rate to be used in the next round.

For its congestion control mechanism, TFRC directly uses a throughput equation for the allowed sending rate as a function of the loss event rate and round-trip time. In order to compete fairly with TCP, TFRC uses the TCP throughput equation, which roughly describes TCP's sending rate as a function of the loss event rate, round-trip time, and packet size. We define a loss event as one or more lost or marked packets from a window of data, where a marked packet refers to a congestion indication from Explicit Congestion Notification (ECN). The sender keeps a log of all packets it has sent in this round. The log contains two entries for each packet: the first entry indicates whether the packet has been (i) received and has been acknowledged by the receiver; (ii) presumed lost; (iii) of unknown status (neither ACKd nor yet presumed lost). This the "received status" of the packet. The second entry consists of a value that is equal to the time the packet was sent plus the current base timeout value. This the "timeout limit" for the packet. If there is no packet loss then packets are sent

twice as fast in the next round. Otherwise, we scale down using a function that depends on the round trip time estimate, the base timeout, the packet loss and the maximum window packet. A more detailed description of TFRC can be found at [65, 128].

Generally speaking, TFRC's congestion control mechanism works as follows:

- The receiver measures the loss event rate and feeds this information back to the sender.
- The sender also uses these feedback messages to measure the round-trip time (RTT).
- The loss event rate and RTT are then fed into TFRC's throughput equation, giving the acceptable transmit rate.
- The sender then adjusts its transmit rate to match the calculated rate.

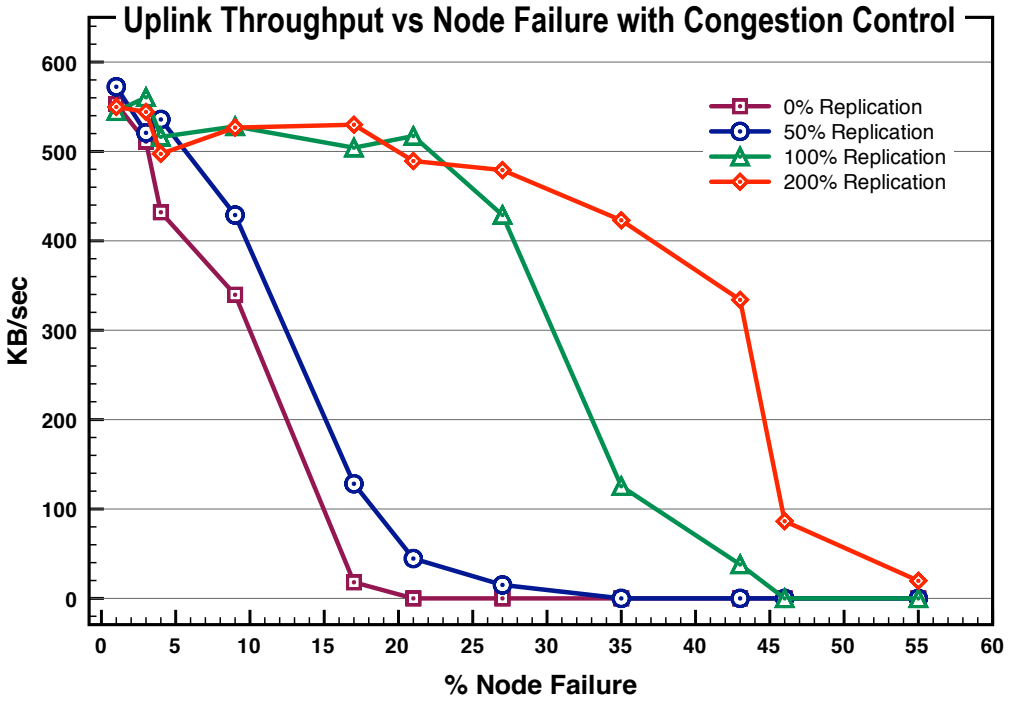


Figure 6.12: Throughput of UDP communication using TFRC when we use packet replication

To quantify the difference in resilience when under attack between UDP and UDP with TFRC, we performed two sets of experiments using exactly the same topology and nodes

in planet-lab. In the first set we used vanilla UDP sockets and in the second we used UDP equipped with TFRC congestion control. For TFRC we modified appropriately the code-base that was provided at the official TFRC site [173]. Each experiment was repeated at least 25 times and the results we are presenting in this section are based on the averages of our measurements. We used an overlay network deployed on planet-lab using continental US nodes to avoid having to traverse transatlantic lines. The network topology uses 1-hop routing *i.e.* the overlay nodes where used as reflection points. Figure 6.10 depicts the indirection topology used. Our experiments measure the worst case scenario: an attacker who can incapacitate a non-static fraction of the overlay nodes shifting his attack from one set of overlay nodes to the other. Therefore, the sender cannot utilize the link feedback mechanism efficiently but has to resort to packet replication to compensate for the attack.

The throughput we achieved when using UDP with the TFRC congestion control with packet replication is shown in Figure 6.12. These throughput results appear to be close to the results we obtained when we used regular UDP 6.7. In Figure 6.13 we compare the throughput we observed for UDP with TFRC and UDP. Notice that in most of the cases, when we use packet replication, regular UDP is more aggressive in transmitting data (sometimes even 8% on average) achieving better throughput rates. These results are not surprising because we used a single client that was transmitting data over multiple links avoiding congesting each individual link. However, when we use packet replication, this might not be the case.

Congestion control mechanisms can potentially be exploited to create denial of service through spoofed feedback. In our system, we don't have need to address that since all the communication state is stored in the ticket which is transmitted with each packet and is validated by the overlay nodes. Moreover, all acknoledgements from the overlay nodes to the client are signed with the client's public key.

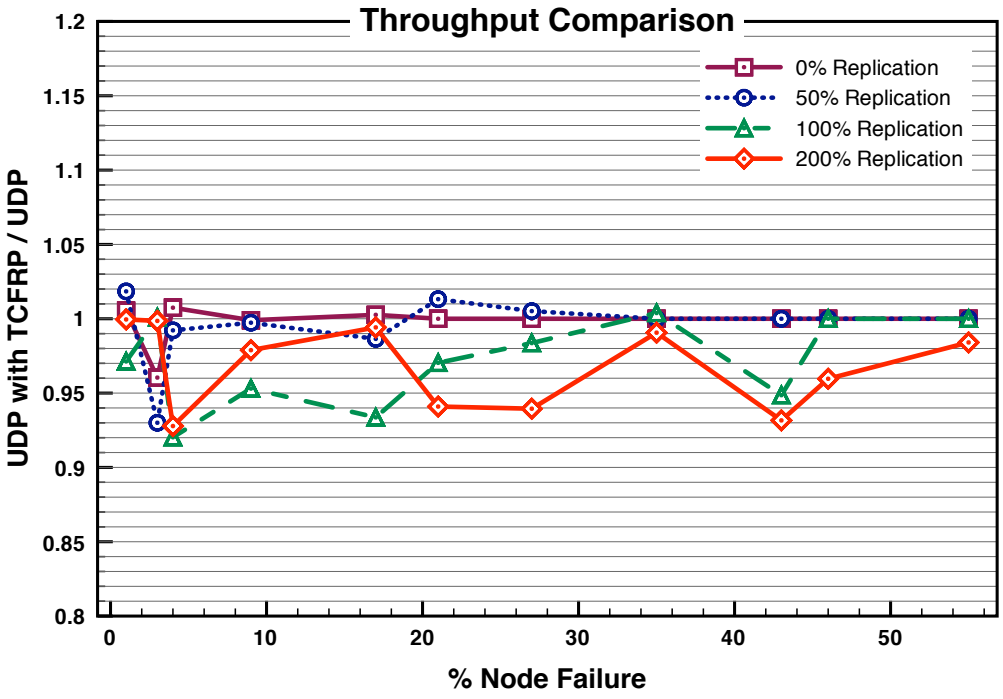


Figure 6.13: Throughput ratio between communications using UDP equipped with TFRC and vanilla UDP for various packet replication rates. Notice that when the ratio is below one plain UDP provides larger throughput. In general the two protocols perform similar but, as we expected, when we employ packet replication, UDP appears to be more aggressive than UDP TFRC achieving higher average throughput.

6.5.1 Packet Replication and Forward Error Correction

Merely replicating packets is not the optimal approach in terms of communication overhead. Source coding in the form of Forward Error Correction can provably provide the optimal information theoretic solution for an erasure channel [112, 140, 141]. Of course, there is a tradeoff between our switching capacity and our computational complexity of generating the FEC packets at high speeds since both compete for CPU resources. Fortunately, modern computers can perform simple XOR operations extremely fast and the use of Direct Memory Access (DMA) allows us to generate XOR-based FEC packets without reducing our forwarding speed [26].

However, computing the correct FEC *i.e the additional information we have to transmit* depends on the loss probability P_l. This information can be approximated by measuring the packet aggregate packet losses. Because the transmitted packets follow different, independent paths we can model the overall communication channel as an erasure channel. In such a model, the loss probability P_l (*the probability that a packet will reach the receiver*) is equal to the number of paths that are unavailable over all the available paths from the sender to the receiver. The success probability is one minus the loss probability which is what we need to compute the correct FEC source code rate.

Chapter 7

Protecting Indirection Overlay Networks against Malicious Insiders

7.1 Introduction

In this chapter, we investigate methods to identify the traffic anomalies that can arise from deliberate attacks or by a rare statistical fluke in the construction of the overlay topology. Our detection algorithms *do not depend on object popularity* but rather apply on aggregate packet flows, avoiding exorbitant storage and processing requirements, maintaining scalability on the number of overlay participants. Statistical detection on aggregate flows is not something entirely new [31, 111], but we are the first to consider these methods in an overlay network setting that takes into consideration the neighbor-structure of P2P systems.

We apply our detection algorithms on aggregate packet flows, avoiding exorbitant storage and processing requirements and maintaining scalability as the number of overlay participants grows. Statistical detection on aggregate flows is not something entirely new [31, 111], but we are the first to consider these methods in an overlay network setting that takes into consideration the neighbor-structure of P2P systems.

In general, we can protect an overlay network that has the following properties:

1. The neighbors of each overlay participant are known for a window of time
2. We can determine (within bounds) the fraction of requests we expect to receive from each neighbor

The above properties hold for almost all DHT-based Overlays (CAN is an exception) and even some randomized ones where the set of neighbors is of fixed size and the search requests have a predefined maximum length. Although we do not address misrouting directly as in [28], we do not allow mis-forwarding or neighbor spoofing: every node has a fixed list of known authenticated neighbors using pair-wise symmetric keys. Only these neighbors are allowed to route packets through the node and only to valid destinations according to the structure of the overlay. All other traffic is dropped, making objects reachable only to nodes that follow proper routing. Our work focuses on exposing traffic anomalies, which prevents nodes from dropping or injecting requests thus conforming to a "normal" forwarding behavior.

The overall protection mechanism we propose consists of two independent but complementary techniques. First, we identify a set of *invariant* traffic characteristics which depend on the structure of the P2P system and determine the relative rates at which traffic should flow between adjacent nodes. Attackers are identified either because of the change (increase or decrease) in the rate with which they inject traffic, or because their distribution of requests is unnaturally biased (with respect to un-compromised nodes) toward the target of the attack. A node can identify a traffic anomaly by comparing the traffic received from a neighbor that contains attack flows to the average he receives. To block the attack traffic before it becomes a real threat, we introduce a variant of the original Pushback[76] architecture that was designed for Internet routers. In an overlay network, we have two types of misbehaving flows: flooding flows and packet drops. For excessive flows we first rate-limit the offending flow and notify the upstream overlay neighbor of the problem, expecting it to rate-limit the offending flow. For packet drops, we contact our neighbor's neighbors requesting them to give us the traffic statistics counts for the offending flows in question until we find a

conflicting count indicating the node that drops the packets or lies about its traffic statistics. By recursive and distributed application, this pushback-like protocol allows us to isolate the attacking nodes quenching at the same time the effects of the attack.

We test our implementation using simulation and traffic from actual web server caches [1]. From the web traces, we generate the hash values of the IP address and URL of each request and we use them as the initiator and object IDs in a Chord request. This way we test the performance our system under normal traffic tuning the necessary parameters to limit false positives. For our attack scenarios, some randomly selected nodes within Chord are used by attackers to flood the system with excessive requests or drop requests toward a set of destinations. We vary both the fraction of compromised nodes and the intensity of the attack against an object (and corresponding node) of the P2P system. Our results show that we can efficiently detect and mark excessive flows even when up to 25% of the system's nodes have been compromised. Our methods require $O(log^2(N))$ of memory per node, where N is the set of nodes participating in the overlay network, and hence can fit easily within the memory constraints of participating Chord nodes. The proposed pushback-like protocol remains effective even when up to 15% of the overlay nodes have been compromised, and works even in the case of collaborating attackers. Our work is the first that attempts to detect, identify and isolate DOS flooding attacks initiated from inside an Overlay. The novelty of our approach lies in the exploitation of the properties inherent in these P2P systems with inference-based techniques.

7.2 Flow Model Description

In this section, we formalize the notation used in this paper to describe what we consider a "structured" P2P system, we formally define our notion of an attack, and we conclude with a description of invariants that structured P2P systems should exhibit unless under attack or having an "unbalanced" topology.

7.2.1 Structured P2P Systems

A structured P2P system maps a set of keys K_{ids} to a set of nodes N_s of size N ($N = |N_s|$) and provides a distributed routing algorithm among these nodes. This algorithm takes as input a key $k \in K_{ids}$ and provides a route through nodes participating in the P2P system to the node to which k is mapped. We assume that packets are forwarded in a *recursive* manner within the P2P system. When a node n wishes to forward a message to the node holding key k, each P2P node on the route forwards the packet to the next P2P node along the path. In contrast, it is possible to utilize *iterative* forwarding, where n iteratively contacts each node on the P2P path to learn the identity of the next P2P hop. Iterative forwarding is possible since every pair of nodes in the P2P system can communicate directly on top of the underlying network fabric, *e.g.,* the Internet. If iterative forwarding is permitted, stopping DoS attacks from inside the network is easier since one can simply filter messages originating from the attacking node, once these nodes are identified. However, iterative forwarding is in general unattractive because of the increased delay it adds along the forwarding path.

In most systems, there exist algorithms that maintain the routing integrity of the P2P system when nodes join or leave. For the purposes of the analysis in this paper, we will assume that the P2P system's routes are maintained and there are no reachability .

A *flow*, (s, k), consists of the set of search requests sent from node $s \in N_s$ to the key $k \in K_{ids}$. The notation can be extended to a set of flows (S, K) where $S \subseteq N_s$ and $K \subseteq K_{ids}$. We let $\lambda_{S,K}$ be the rate of packet transmissions from nodes in S to keys in K, and let $\lambda^K_{avg} = \frac{\lambda_{S,K}}{|S|}$, *i.e.,* it is the average rate at which a node transmits requests toward keys in K. Objects stored in the P2P system may have different popularity, *i.e.,* that in general $\lambda^{k_1}_{avg} \neq \lambda^{k_2}_{avg}$ for $k_1 \neq k_2$.

Initially, we also assume that for a fixed object k, the popularity of this object is the same among the participants of the P2P network: $\lambda_{S,k} = \lambda^k_{avg}$ for all S, k. We expect that as $|S|$ grows, if the nodes that comprise S are chosen at random, then $\lambda_{S,k}$ will quickly approach λ^k_{avg}. Our methods will take advantage of this assumption, and our simulation results will

evaluate the validity of our assumption.

Furthermore, we define the set of *incoming* and *outgoing* neighbors to be In_c and Out_c respectively. Any request forwarded or replied by c arrives through one of c's incoming neighbors and exits through one of its outgoing neighbors (unless c contains the key k to which the request is intended for). All structured P2P systems provide the mechanisms to uniquely calculate both In_c and Out_c. These sets are typically relatively small to the total number of nodes in the network, *e.g.,* $O(log(N))$.

7.2.2 Insider DoS Attackers & Attack Intensity

We consider DoS attacks targeted toward a specific key or set of keys in the P2P network. Such an attack could be mounted to block access to data associated with a particular key. These attacks are mounted by nodes *inside* the P2P network that inject excessive packets to be routed toward the attacked key(s). A node that is the origin point of excessive packets toward key k is said to be an *attacker* of key k.

In a healthy P2P network, the rate of a flow $\lambda_{S,k}$ from a fairly large, randomly selected group of nodes S toward a set of keys K should closely approximate the popularity of that object λ^K_{avg}. We say that a flow aggregate (S, K) is *misbehaving* if $\lambda_{s,K} > (\delta + 1) \cdot \lambda^K_{avg} = \lambda^K_{max}$ for some $\delta > 0$, where δ represents a lower bound on the proportional increase that a flow can transmit relative to the average rate before the flow is labeled as misbehaving. The previous notion can be extended to a set of nodes S as $\lambda_{S,K} > |S| \cdot (\delta + 1) \cdot \lambda^K_{avg} = |S| \cdot \lambda^K_{max}$. Note that for a set of nodes, the maximum rate allowed before we declare the aggregate flow as misbehaving depends on the size of the set. The selection of δ is a measure of the tolerance that we allow between different nodes (or groups of nodes) in the P2P system before declaring that a flow is misbehaving. If we assume a totally homogeneous system, then $\delta = 0$ — *i.e.,* no deviations from the average rate (popularity) are allowed. Larger values allow more tolerance, but also give an attacker more freedom to deviate from the average rate and avoid detection. Typically, we select $\delta = 0.1$ which means that we detect

flows that send ten percent more than the average rate of a set of keys K.

We define β to be the proportion by which one attacker increases its traffic toward k above λ^k_{avg} such that the attacker transmits packets toward key k at a rate of $(\beta + 1) \cdot \lambda^k_{avg}$. If N_a is the number of attacking nodes, the total amount of excessive search requests injected into the system by the attackers is $\beta \cdot N_a \lambda^k_{avg}$. The rate of search requests λ_{attc}, the target node experiences under attack is:

$$\lambda_{attc} = \frac{N_a}{|N|} \cdot (\beta + 1) \cdot \lambda^k_{avg} + \frac{(N - N_a)}{|N|} \cdot \lambda^k_{avg}$$

Let $f = |N_a|/|N|$ be the fraction of nodes compromised; we define the attack intensity, $\beta \cdot f$, to be the increase in the popularity of the target key k caused by the attackers' excessive search requests:

$$\beta \cdot f = \frac{\lambda^k_{attc} - \lambda^k_{avg}}{\lambda^k_{avg}}$$

For example, if $\beta \cdot f = 1$, the node that stores the object under attack has to serve twice as many search requests for that object. For the attackers' queries to be harmful to the target node that stores the keys being attacked, $\beta \cdot f$ must be large. If an attacker only controls a limited number of nodes, their only choice is to increase β. A large β and small f means that there will be a relatively small number of attack flows, and that these attack flows will inject significantly more traffic toward k.

7.2.3 Invariants of Structured DHT systems

Our methods to detect misbehaving nodes will utilize two measures that can be computed (or estimated) at a node, c. These are:

- Fixed-Key Variable-Source (FKVS):

$$\alpha(c, K, S, S') = \frac{\lambda^c_{S,K}}{\lambda^c_{S',K}}$$

- Fixed-Source Variable-Key (FSVK):

$$\gamma(c, K_1, K_2, S) = \frac{\lambda^c_{S,K_1}}{\lambda^c_{S,K_2}}$$

The FKVS measure compares the rates of two sets of sources for a particular set of keys. In a healthy P2P system, for appropriately sized (large) sets S and S', if c lies on the paths from all S and S' to the nodes storing keys K, we should have that $\alpha(c, K, S, S') = |S|/|S'|$. The FSVK measure compares the rates from a particular set of sources to two sets of keys, K_1 and K_2. In a healthy P2P system, if c is on the paths from S to nodes containing keys K_1 and K_2, then $\gamma(c, K_1, K_2, S) = \frac{\lambda^{K_1}_{avg}}{\lambda^{K_2}_{avg}}$.

7.3 Statistical Bounds of Flows

In the previous section, we presented two metrics whose values in a structured P2P system are easily estimated by a node. We now present methods that use these metrics to identify misbehaving flow aggregates and mark packets that belong to these aggregates. The total number of possible flows in a P2P system is $|N||K_{ids}|$. Thus, tracking each individual flow would require $O(|N||K_{ids}|)$ memory. Even if we fully distributed the tracking load amongst all participating nodes, $O(K)$ memory would be needed to collectively track all flows.

To avoid utilizing such a potentially huge amount of memory per node, we track a set of aggregate flows whose size is $O(\log|N|)$ by taking advantage of the fact that the number of flows in each of $|In_c|$ and $|Out_c|$ is $O(\log|N|)$. We require that each node c consider as a separate aggregate all flows that arrive via the same incoming neighbor and exit via the same outgoing neighbor. Hence, there are $O(\log^2|N|)$ aggregates to consider. By numbering the neighbor nodes in In_c and in Out_c, we can identify the flow aggregate that enters through the i-th neighbor of In_c and departs through the j-th neighbor of Out_c as $f_{i,j}$, where $i = 0$ implies the flow originates at c and $j = 0$ implies the flow terminates at c.

Aggregate $f_{i,j}$ is assigned a counter, $C_{i,j}$, where $i \in In_c$, $j \in Out_c$. When a packet arrives, we increment the counter that corresponds to the aggregate flow to which the packet belongs.

7.3.1 Comparison of Aggregate Flows

Our detection algorithm compares each flow's counter $C_{i,j}$ to the counter for all the aggregate of flows $C_j = \sum_{\ell \in In_c} C_{\ell,j}$ that exit to the same outgoing neighbor. We wish to determine the likelihood that, in a healthy P2P system, $C_{i,j}$ can have the value observed, under the assumption that C_j is made up mainly of healthy flows. More formally, let $X_{i,j}$ and X_j respectively be random variables that equal the value of these counters in healthy P2P system, and determine $\mathbb{P}(X_{i,j} \geq C_{i,j} | X_j = C_j)$. (Note that $C_{i,j}$ and C_j represent actual observed values in the real system, whereas $X_{i,j}$ and X_j are values that occur in a trial on top of a healthy P2P system.) The larger this probability, the more confidence we have that flows entering through neighbor i and exiting out of neighbor j are healthy. We define ϵ to be our *level of confidence* of the test. If $\mathbb{P}(X_{i,j} \geq C_{i,j} | X_j = C_j) < \epsilon$, then (according to this test) there is a probability $< \epsilon$ that $f_{i,j}$ is healthy.

A second test is similar to the first, except that instead of comparing a flow (i, j)'s counter $C_{i,j}$ to the counter C_j, the counter is compared to the counter of a flow (i', j) that originates from a different incoming neighbor, but heads to the same destination, i.e., we compute $\mathbb{P}(X_{i,j} \geq C_{i,j} | X_{i',j} \geq C_{i',j})$.

We don't need to assume anything about the flows distribution.Let K be the set of keys reachable by $\cup_i f_{i,j}$ through c, let S be the set of sources that reach keys in K by passing through node c, and let S_i be the set of sources that reach keys in K through c and enter c through the ith incoming neighbor. If $\alpha(c, K, S_i, S)$ is known, then we know the relative rates of the two Poisson processes and can therefore compute the first test probability value. If $\alpha(c, K, S_i, S'_i)$ is known, then we can compute the second probability value.

By comparing this value to the observed values of the counters, we can determine, within a certain confidence level, whether one flow's rate is higher with respect to the all the other

flows' rate.

While any slight deviation from the normal transmission rate could be flagged as a violation, we allow a small degree of variability. Hence, we apply our "slack" factor, δ, and say that flow $f_{i,j}$ is misbehaving when the actual ratio of observed rates is larger than $(1+\delta)\alpha(c, K, S_i, S_i')$ for the second test. Setting δ to 0 leaves no slack. If $f_{i,j}$ sends at a rate even slightly above its supposed rate in a healthy P2P system, the central limit theorem tells us that eventually, $f_{i,j}$ will be flagged as unhealthy. Setting δ to larger values allows for additional slack.

Let $\alpha(c, K, S_i, S) = x/y$. Then, when adding a slack factor δ, we say that flow $f_{i,j}$ is misbehaving when the ratio of observed rates is larger than $(1+\delta)x/(y-\delta x)$ (because S also contains $f_{i,j}$).

7.4 Overlay System Description

We now apply the tests presented in the previous sections to detect and prevent distributed denial of service (DDoS) attacks within a Chord ring. The model used in this section is the one presented in the original Chord paper [162] that assumes the ring is fully saturated, i.e., that each node carries exactly one key. This is easily implemented in practice even when $|N| < K$ using the virtual node concept described in this paper. Our discussion in the next subsection is a brief summary of Chord needed for our purposes, but assumes that the reader has basic familiarity with the Chord system.

7.4.1 Model Definitions

A Chord peer to peer system consists of a set nodes N that try to serve objects that are hashed and stored in nodes using an m bit hash function. The key identifiers K_{ids} are placed in circular order, creating a ring of length $K = ||K_{ids}|| = 2^m$. To simplify the notation, all math in the remainder of this section is performed modulo K. Each node is assigned an

identifier (id) from the key space ($N \subseteq K_{ids}$), thus creating a node ring. Since we have a circular placement, we have for each node c a successor node and a predecessor node. Each node is assigned a set of keys, meaning that the node either stores or knows the location of all the objects in its local database that hash to its value. Thus, when a search request reaches the node responsible for the key requested, the search stops and a reply to the originator of the request is issued either with the actual object or with the object's location.

7.4.2 Invariants and Tests

Proposition 1 *Let $i_1 > i_2$. If the set of flows (S_{i_2}, K) that pass through c is non-empty, then:*

$$\alpha(c, K, S_{i_1}, S_{i_2}) = 2^{i_1 - i_2}.$$

Proof: Assume the set of flows (S_{i_2}, K) that pass through c is non-empty. For each key $k \in K$, let us count the number of sources whose transmissions would pass through node c, entering node c via the i_2-th finger of node $c - 2^{i_2}$ en-route to key k. Recall that the Chord forwarding protocol requires that a transmission be forwarded on the finger that takes the transmission as far as possible in the clockwise direction without passing the destination. Since the i_2-th finger travels a distance of 2^{i_2} around the ring, all transmissions prior to reaching node c must traverse a distance greater than 2^{i_2}, and the sequence of fingers taken after reaching c to then reach k is unique.

Consider a sequence of ℓ bits, $b_1 b_2 \cdots b_\ell$, where $\ell = \log |N| - i_2$, and consider the node s that is at distance $\sum_{j=1}^{\ell} b_j 2^{j+i_2}$ from c in the counter-clockwise direction. Then, for node s to reach k, it will first traverse a series of nodes where it exits through the $j + i_2$-th outgoing finger of some node along the path when and only when $b_j = 1$. After taking this series of fingers, the transmission will end up at node c by taking the i_2-th finger of node $c - 2^{i_2}$ en-route to its final destination of k.

Each of the 2^ℓ possible bit sequences produces a unique node s. Hence, there are 2^ℓ such nodes whose transmissions to k first traverse a path that proceeds through $c - 2^{i_2}$ and takes

its i_2-th finger to reach c en-route to k. Furthermore, since the only way a transmission can originate at a source s and reach k by taking the i_2-th finger of $c - 2^{i_2}$ is to previously take a strictly decreasing sequence of fingers, this set of 2^ℓ nodes is unique.

It follows that there are $2^{\log|N|-i_2}$ sources that can reach key k by transiting through the i_2-th finger of $c - 2^{i_2}$, and there are $2^{\log|N|-i_1}$ sources that can reach k by transiting through the i_1-th finger of $c - 2^{i_1}$, so there are $2^{i_2-i_1}$ times as many sources entering the i_1-th finger of c to reach k as there are entering c through its i_2-th incoming finger to reach k.

Since we assume that all sources have (approximately) the same interest in key k, and since this ratio remains fixed irrespective of the final distance of key k from c, the Lemma holds.

Proposition 2 *If the set of flows* (S, K) *that pass through* c *is non-empty, where* $S = \cup_j S_j$*, then:*

$$\alpha(c, K, S_{i_1}, S) = \frac{1}{2^{\log|N|-i_1} - 1}$$

Proof: The proof follows the form of Proposition 1, noting that there are $\sum_{j=0}^{\log|N|-i_1} 2^{i_1-j}$ sources from S to K that enter c (through any finger).

7.5 Pushback-like protocol

Pushback[76] was a router-based mechanism for defending against DDoS attacks. The original idea was this: a router that knew that it could not successfully send packets down a congested link would, instead of dropping them itself, notify (some of) its upstream routers not to send them these packets at all. Heuristics were employed to identify the flows, or "aggregates" (packets having some common property, such as the same destination IP address and TCP port), responsible for the downstream congestion. The incoming (with respect to those flows) links would then be examined, and the fraction of upstream routers responsible for most of incoming packets belonging to the aberrant flow would be notified to rate limit that flow. These routers, in turn, would recursively apply this mechanism.

Pushback would work best when the malicious traffic was anisotropically distributed around the Internet, so that some routers would rate limit traffic more severely than others. It also required some level of trust between routers, which turned out to be an unrealistic requirement when crossing administrative boundaries (*e.g.,* an ISP's peering routers). It was also assumed that the participating routers would not be misbehaving in their application of Pushback.

There are two fundamental differences between that original setup and an overlay network: in the latter, the nodes themselves are both the originators of traffic and overlay "routers", and we have to assume that some of them will be compromised. Furthermore, not only can a compromised overlay node flood the network with traffic but it can also drop traffic meant to be routed to another destination via one of its neighbors.

Thus, in an overlay network, we have two types of misbehaving flows: flooding flows and packet drops. For excessive flows we first rate limit the offending aggregate flow and notify the upstream overlay neighbor of the problem, expecting him to rate limit the offending flow. For packet drops, we contact our neighbors' neighbors asking them to give us the counts for the aggregate flows in question, until we find a conflicting count indicating the node that drops the packets or lies about its aggregate packet count. By recursive and distributed application, this pushback-like protocol allows us to isolate the attacking nodes, quenching at the same time the effects of the attack. Of course, if our system has a lot of attackers collaborating with each other to both not comply but also to falsify their aggregate rates, pushback itself can be exploited by the attackers to generate additional traffic to the overlay network. On the other hand, even if the attackers are a significant portion of the network, if they are not coordinated they can be identified and isolated by the rest of the overlay nodes.

A silent assumption we have made is that neighboring nodes cannot fake their identities, that is, they cannot pretend to be another node in the overlay. The details of how this is achieved will depend on the underlying network and operating system facilities available

to the nodes. For example, if the underlying network is the public Internet, we can set up pairwise-authenticated tunnels using IPsec, GRE, or some other encapsulation protocol.

7.6 Simulation Results

7.6.1 Experimental Setup

Our first goal was to verify the effectiveness of our detection algorithm and to evaluate its performance. To that end, we use an implementation of the Chord peer to peer network in a series of simulations. In all of our experiments, some of the participating peer to peer nodes assume the role of the "attacker". The "attackers" select a key at random from the set of allowed keys and generate a disproportionate number of search requests towards that key. The node that stores the key under attack is the "target". The goal of the attackers is to flood the "target" with search requests, crippling its ability to respond. In general, the attackers are allowed to select multiple "targets" simultaneously. However, if we assume that attackers have abundant but nonetheless limited resources, aiming at multiple "targets" will only lower their aggregate attack ability. Indeed, the attackers will have to split their attack requests between the different targets, reducing their attack intensity. We formally defined attack intensity as the increase in the search request traffic towards the target key caused by the attackers' excessive search requests. For example, an attack intensity of $\beta \cdot f = 1$ means that the node that stores the key under attack has to serve twice as many search requests for that object as it would normally have.

We use the notion of attack intensity to formally quantify the attack ability of the compromised nodes. Where our test is working perfectly, marking only excessive search requests, only a fraction $\frac{\beta}{\beta+1}$ of packets in the attacking flow should be marked. We determine our performance by measuring our ability to detect the attack as close to the attacker as possible. Also, we want to mark the malicious search requests as many times as possible along the path from the attacker to the target object. All of these metrics are dependent both

on the attack intensity and the relative attacker's distance from attacked key.

Another goal of our experiments was to identify the limits of our detection algorithm. Our attack detection method identifies aggregate groups containing attackers by marking packets from these groups as "excessive" whenever our estimates predict that the groups are transmitting at too high a rate. Since we operate at the granularity of groups, we introduce a false positive error for the non-attacking individual flows which happen to be grouped with "excessive" ones. As we show, this error is relatively small because the vast majority of the requests from a malicious aggregate flow belong to the attacker. Moreover, the attack requests get marked multiple times along the path from the attacker to the target. Another potential source of error can arise due to the use of statistical inference. Of course, this error can become arbitrarily small by selecting a higher confidence interval. In our simulations we used a 0.999% confidence interval.

In the last set of experiments, we use a pushback-like protocol where a node upon detection of a misbehaving aggregate flow, communicates with its upstream neighbor that this flow is originated. The misbehaving aggregate flow, if it is excessive, is rate limited to the average of the rest of the flows. If the upstream neighbor fails to respond with a certain amount of packets, which is a parameter for our system, this node is considered malicious and the rest of the overlay nodes are notified. The protocol is applied recursively until the source of the anomaly is identified or the flow stops being excessive. Either way the DoS attack will be prevented allowing only small, negligible spikes of packets to reach the target node.

To ensure the statistical validity of our experiments, we used approximately 4 million search requests per simulation. In addition, each simulated experiment was repeated more than 50 times. The results we present in our graphs are the average of these simulations; the variance among the different experiments was observed to be low.

7.6.2 Detection of Single-attacker

We start by examining the scenario where a single node is compromised. Although simplistic for a real world attack, this scenario provides insights on the effectiveness of our approach. Furthermore, we can evaluate our ability to detect the malicious node for varying distances relative to the target and for a range of attack intensities.

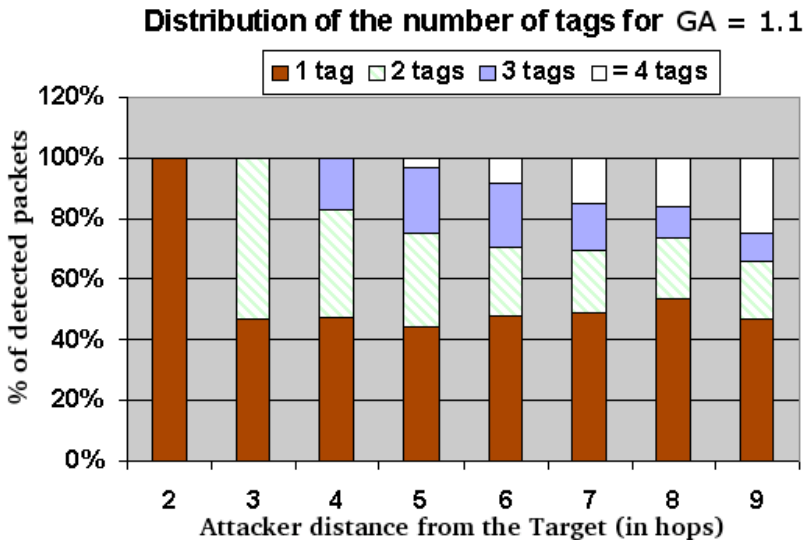


Figure 7.1: Distribution of the number of tags for the attack requests for one attacker in a 1024-node Chord ring. The different plots represent the attacker's distance in hops from the target for attack intensity of 10% ($\beta f = 0.1$).

Initially, we placed the attacker in various distances (in hops) away from the node responsible for the target key, the target node. We then measured the percentage of the excessive search requests our algorithm detected and their detection distance from the target *i.e.,* how quickly was the traffic identified as excessive. As the distance of the attacker from the target increases, the detection distance increases accordingly. In addition, we detect the malicious flow in multiple nodes along the path from the attacker to the target. It appears that the largest portion of the attack requests are detected close to the attacker. For example, if we place the attacker at a distance 9, the majority of its packets are detected by its next hop neighbor,with distance 8, then by the next node at distance 7 etc. Even when the attacker's proximity to the target is reduced to the minimum, *e.g.,* at a distance of 2, we detect 100%

of the excessive search requests at the immediate neighbor.

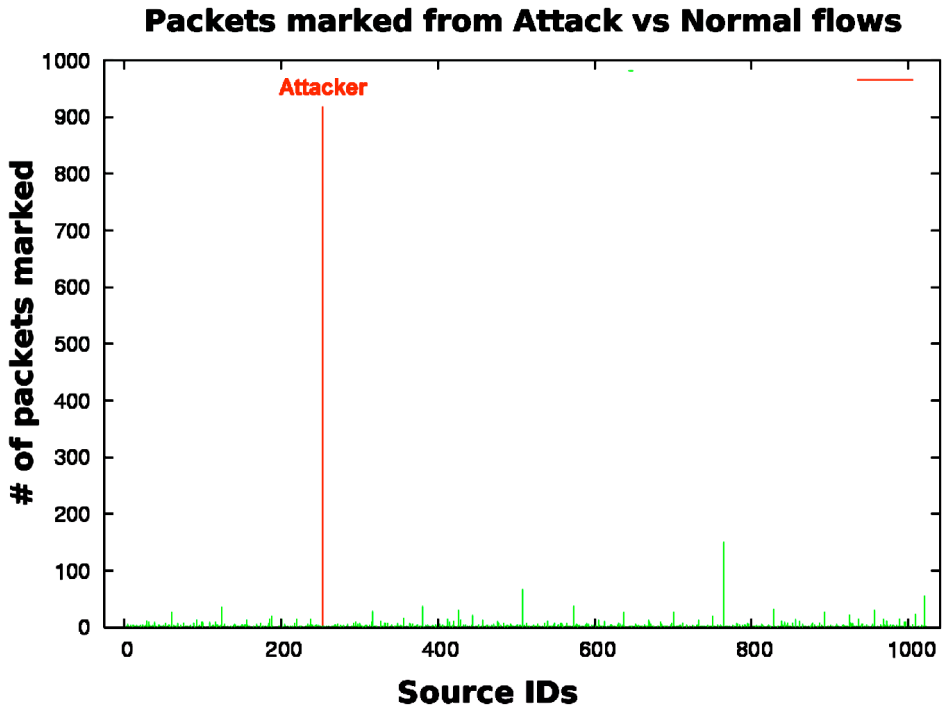


Figure 7.2: Total excessive packets detected from attacking vs normal flows: our false positive ratio is very low and distributed almost evenly among the normal flows. On the other hand, the attacking packets are marked more and the source id of the attacker stands out.

Recall that each node tags all the flows of a group that appears to be "misbehaving". Figure 7.1 shows the relative distribution of the tags when we vary the distance of the attacker to the target. We see that, depending on the distance between the attacker and the target, a significant portion of the excessive search requests are tagged at least twice by nodes along the path from the attacker to the target. The number of tags increases as we increase the number of hops between the attacker and the target since the attack packets traverse more nodes in order to reach their target.

As the distance from the target increases, we begin over-marking the attacker requests. This happens because, as we move away from the target, more flows generated by attacker participate in the groups of flows we detect and thus mark. Moreover, in this experiment we used an attack intensity of $\beta f = 0.1$ or a 10% increase in the popularity of the attacked object,

which is a relatively small value. For larger values of βf, we have minimal over-marking of the attacker search requests since the excessive requests become a bigger portion of the total search requests originating from the attacker.

To actually have impact on the search requests of the attacked object, the attackers intensity, $\beta \cdot f$ should be relatively high. For example, for $\beta \cdot f = 1$ the single attacker has to inject search requests in the system with rate $N \cdot \lambda^k_{avg}$, or with a $\beta = N$. Although this rate may appear unnecessarily high, in practice this depends on the popularity λ^k_{avg} of the object attacked. If the object is highly popular and λ^k_{avg} is large compared to the rest of the objects in the system, the attacker will need to significantly increase the number of search requests to noticeably affect its popularity.

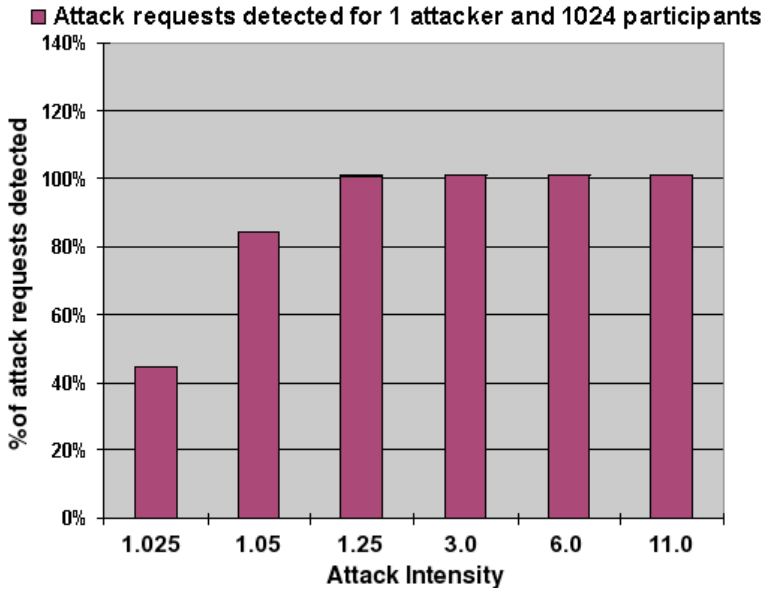


Figure 7.3: Attack packets detected for one attacker randomly placed in a 1024-node Chord ring. The different bars correspond to increasing values of attack intensity βf. In some cases, *e.g,* for $\beta f = 0.25$, we detect more attack requests than are actually injected by the attacker, because our algorithm detects groups instead of individual flows. The results presented are the average of multiple experiments (100 per bar).

We have shown that our method detects and marks search requests on groups of flows. An inherent problem is that we may end up marking legitimate flows along with the attacking ones (since they are mixed in the same aggregate). As Figure 7.2 shows, that is not the case

for our method. Although we are marking flows belonging to the same "misbehaving" group, we are punishing mostly the attacker since it is the one sending the majority of the search requests through that group. The majority of the other flows are getting marked minimally, in comparison both to the total number of requests they generate and to the attacker requests marked. Naturally, blindly marking and dropping excessive requests from a misbehaving flow is a very crude method to prevent a DoS attack, although it can be effective if resources are otherwise limited.

Through our experiments we wanted to ensure that there is no attacker placement inside the Chord ring that our algorithm fails to detect. We are now in position to present more realistic results from simulations in which we have one attacker randomly placed in a Chord ring of size 1024 (see Figure 7.3). We observe that even for very low attack intensities values, $\beta f = 0.025$, a mere 2.5% increase in the load of the end server, we can detect more than 40% of the attack packets. It is easy to see that for attack intensity values larger than 0.05 our method detects a significant portion of the excessive requests. As the intensity of the attack diminishes, our detection results become weaker. This is something we expected, since our algorithm detects excessive requests based on measurements done on groups of flows where small variations in the intensity of one flow does not have significant impact on the aggregate flow. For some values of attack intensity, especially for $\beta f = 0.25$, we have an over-marking of the attacker search requests which fades out when the attack intensity becomes more significant. This is due to the group detection nature of our algorithm and the fact that aggregate groups of flows contain both legitimate and excessive flows originating from the attacker.

Since we have a random placement of the attacker in the Chord ring we are waiting to have a average distance from the attacker $log(N)/2$. Thus, we expect that our detection should be evenly spread around this expected distance. Figure 7.4 plots the percentage of the attacker requests detected and their corresponding detection distance from the target. Indeed, for intensity values of $\beta f = 0,25$ and higher we see that most of the requests follow

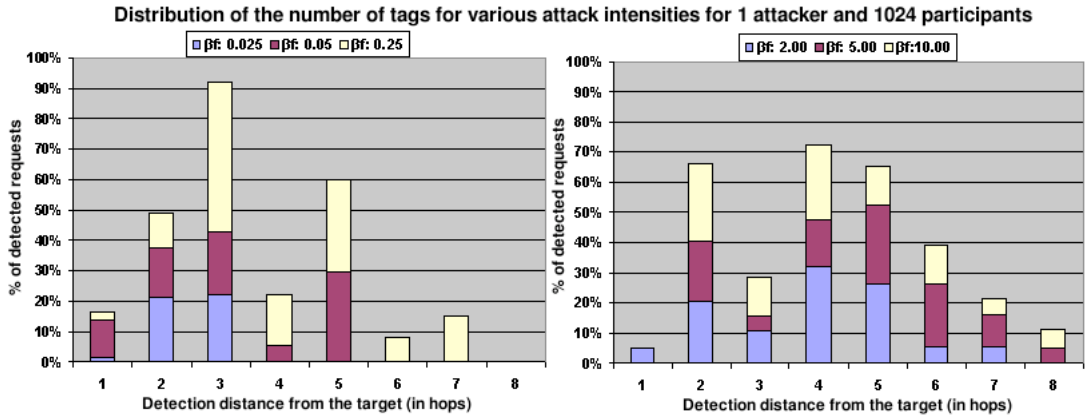


Figure 7.4: Percentage of attack requests and their detection distance from the target for one attacker in a 1024-node Chord ring. The different bars represent the detection distance in hops from the target. The areas in the bars denote the percentage of requests for various values of the attack intensity. The results presented are the average of multiple experiments (100 for each bar).

a almost normal distribution with a mean between distances of 4 and 5. Lower values of attack intensity seem to deviate from what we expect. Having a very low attack intensity, these attacks are not detected by the initial nodes they contact. Rather, they enter deeper into the DHT network, and they are detected closer to the target forming normal distributions of lower mean attacker-target distance.

The number of tags for the attack packets is an increasing function of both the average attacker distance and of the attack intensity as shown in Figure 7.5. The average attacker distance seems to play a more prevalent role. This means that as the average distance between the attacker and the target increases, so does our ability to tag the attacker on multiple locations along the attacker-target path. Thus our system works better as we increase the number of participants in the DHT system, since the average distance between two nodes in the system increases. The detection behavior of our algorithm for attack intensities that are higher than $\beta f = 10$ is similar to those measured for $\beta f = 10$ and we omit them from our figures.

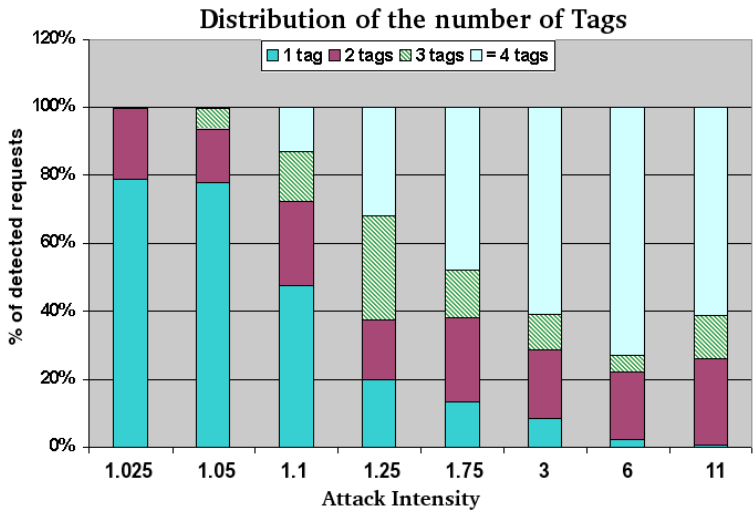


Figure 7.5: Distribution of the number of tags for the detected attack packets for a 1024-node Chord ring i^{th} randomly selected attacker-target placement. Each value on the X axis corresponds to exponentially increasing attack intensity (βf). The results averaged over 100 experiments for each attack intensity value.

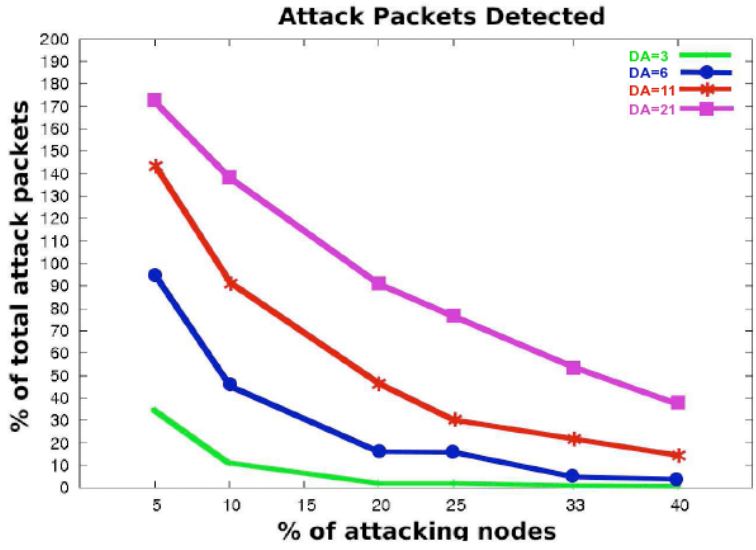


Figure 7.6: Percentage of attack packets detected when we vary both the attack intensity βf and the fraction of nodes compromised for a 4096-node Chord ring. Each line represents different attack intensity values. We can see that as we increase the attack intensity we can detect more percentage of the attackers' request even when the attack is very distributed. Conversely as the attack becomes more and more distributed (from left to right) our detection ability drops almost exponentially.

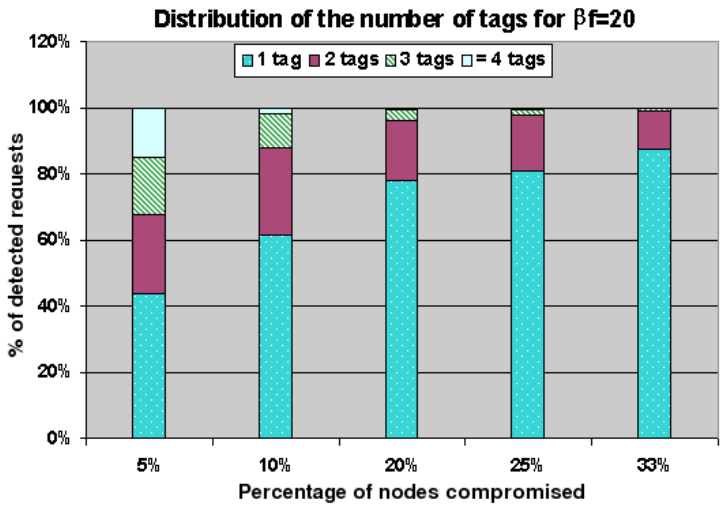


Figure 7.7: Distribution of the number of tags for the excessive search requests detected when the attack intensity is $\beta f = 20$ and we vary the fraction of nodes compromised for a 4096-node Chord ring. Notice that he number of tags on the attack requests are inversely proportional to the fraction of the attackers. The more "distributed" the attack is, the more difficult it is to detect and tag.

7.6.3 Detection of Multiple Attackers

We now study the behavior of our detection algorithm using a Chord ring where we vary both the fraction of nodes compromised and the attack intensity. Figure 7.6 presents the results for a 4096-node Chord ring with multiple attackers. It is clearly shown that there is a correlation between the excessive search requests detected and the attack intensity. Our results show that as the attack becomes more severe, our ability to detect excessive search requests increases; even when 40% percent of our nodes are compromised we are able to detect around 50% of the excess requests.

On the other hand, as the proportion of compromised nodes grows, there is a corresponding drop in our ability to detect excessive search requests, since the attack becomes more distributed on the Chord ring. Additionally the number of tags for the excessive requests are inversely proportional to the fraction of nodes compromised. For example, as Figure 7.7 shows, when the attackers constitute 5% of all the Chord participants, a large portion of their excessive requests have 2 or more tags, whereas when the attackers become 40% of the

total, only 14% of the excessive requests have 2 or more tags.

In some cases we overestimate the number of attack packets sent by the attackers. This happens because we detect groups of flows where, on average, the attacker participates with multiple flows, leading to over-marking of search requests generated from the attacker. This over-marking can be used to weed out the specific flow from the group of flows detected to be misbehaving.

Another source for this overmarking comes from the fact that we allowed the non-attacking nodes to select the popularity of each key/object from a uniform $(0, 1]$ distribution. This generates different preference distributions for each node pushing our "normality" assumptions to their limit. However, it also confirms our initial assumption that our analysis remains the same even if we allow different preference distributions. With a mean of $1/2$ and a standard deviation of $1/6$, the uniform distribution allows for wide spreading of the possible weight values in the full $(0, 1]$ spectrum. The use of a normal or exponential distribution for the same interval would have resulted in less variance, leading to a more concentrated weight distribution. The law of large numbers dictates that any type of weight distribution would eventually lead to a normal preference distribution on the limit as the number of keys grows.

In Figure 7.6.3 we can see the percentage of packets detected for a Chord ring of size 1024. Again, each line represents different attack intensities. The detection ability dissipates as the percentage of attackers in the total node population increases. Note that the detection results are better than those presented in Figure 7.6, where we used the same preference distribution function for all the nodes. Indeed, when using different preference distribution, sometimes attackers are grouped with flows that have high preference for the attacked key and thus revealed faster. However, for the same reasons, we end up having a higher false positive detection of around 0.1% compared to the 0.07% when we used the same preference function. Overall, allowing the nodes to have a different preference distribution function leads to better detection results at the cost of an slight increase in the false positive detection

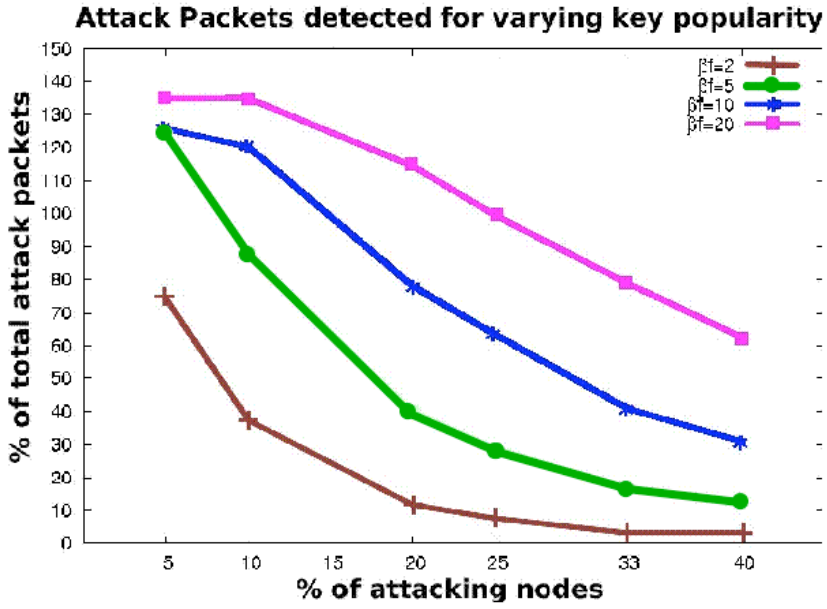


Figure 7.8: Percentage of excessive packets detected when we vary both the attack intensity βf and the fraction of nodes compromised for a 1024-node Chord ring. Each line represents different attack intensity values. For this experiment we allowed each individual non-attacking node to assign a weight to each key selected from a uniform distribution. The detection results appear to be better than using the same preference distribution function.

percentage.

In our experiments, we used a fixed value for the threshold δ, namely $\delta = 0.1$. Recall that δ is a parameter of our detection algorithm, indicating our tolerance towards deviations from the average popularity of a key. Larger values allow more tolerance and lower our false positives, but also give an attacker more room to deviate from the average rate before our algorithm starts detecting the excessive requests. Apparently, there is a trade-off between speed of detection and false positives. Figure 7.10 shows that, for $\beta f = 20$, we get an improvement in our detection as we decrease δ from 0.1 to 0.05. At the same time we see an increase in the false positive percentage, which becomes more apparent for $\delta = 0.05$. Conversely, for $\delta = 0.07$ we get an improvement of almost 12% in the detection rate, when we have a distributed attack with $> 20\%$ of attackers. Lowering δ below 0.07 does not improve the detection rate, doubling false positives.

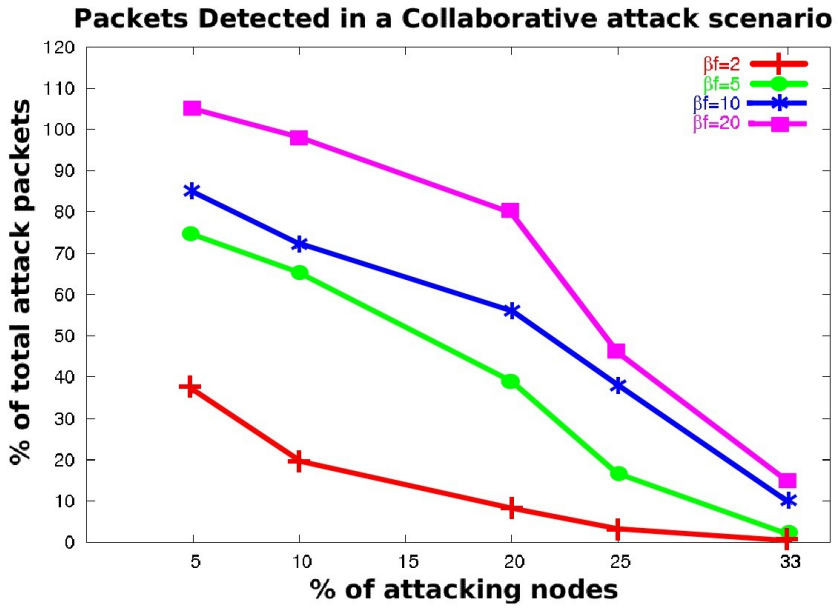


Figure 7.9: In this experiment, the attackers are collaborating by not marking the excessive packets destined for the target key only (coordinated attack). Each line represents different attack intensity values and we vary the percentage of nodes compromised. Even under a collaborative attack, and with a significant portion of all nodes being compromised, we detect a large portion of the attack requests.

Our last experiment stresses our detection algorithm under the worst-case scenario: a highly distributed attack where the attackers can collaborate both by exchanging information about the location of the target and by not marking each other's excessive flows towards the target node. Even under this adverse attack scenario, our algorithm seems to detect between 70% and 100% of all attack traffic in high-volume attacks when even up to 15% of all overlay nodes are participating in the attack, as shown in Figure 7.6.3. When 25% of the overlay nodes are participating in the attack, our mechanism correctly identifies between 40% and 50% of all attack traffic. As the percentage of attacking nodes in the population increases, the effectiveness of our mechanism decreases. This, however, should be expected: the notion of "normalcy" in such a system is shifting towards higher-volume traffic flows. Furthermore, as we have anticipated, given more knowledge, attackers can avoid detection especially when the attack is highly distributed (> 25% of the total nodes are compromised) or the attack is of low intensity ($\beta f < 5$).

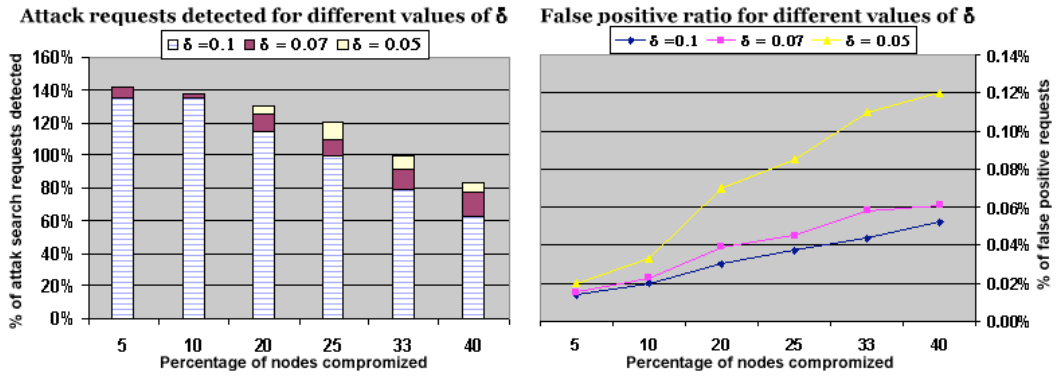


Figure 7.10: The left graph shows the increase in the detection rate as we decrease our tolerance threshold from $\delta = 0.1$ to $\delta = 0.05$. The right graph shows the impact of this increase in the false positive percentage: lowering δ below 0.07 offers little benefit to the detection rate while doubling false positives.

7.6.4 Pushback protocol performance

In this section we quantify the performance of the simple pushback protocol we devised: we run our experiments enabling nodes to rate limit the aggregate flow that contains the malicious flow using probabilistic rate limiting. The dropping probability is determined based on the deviation of the aggregate flow from the average making the flow conform to within δ from the measured average. Moreover, the node detecting the attack notified its corresponding incoming neighbor of the problem denoting also the sets of key(s) that were affected by this excessive aggregate flow. If the node was unresponsive to the rate limiting request for more than 100 packets (i.e. the rate of the detected aggregate flow remained above the average) this node was immediately marked as malicious and was removed from the overlay by being replaced with a non-attacking node to avoid reconstruction of the overlay routing table. The selection of the 100 packet threshold is a parameter of our system and allows a little more time to the neighbor to adjust to the rate limiting request. It cannot be exploited by attackers since its too low to make a significant impact to the target server.

The most important metric for the performance of our system is the additional load imposed on the attacked node. Figure 7.11 shows that the load imposed on that node during an attack for various attack intensities remains low, namely below 2 and when the percentage

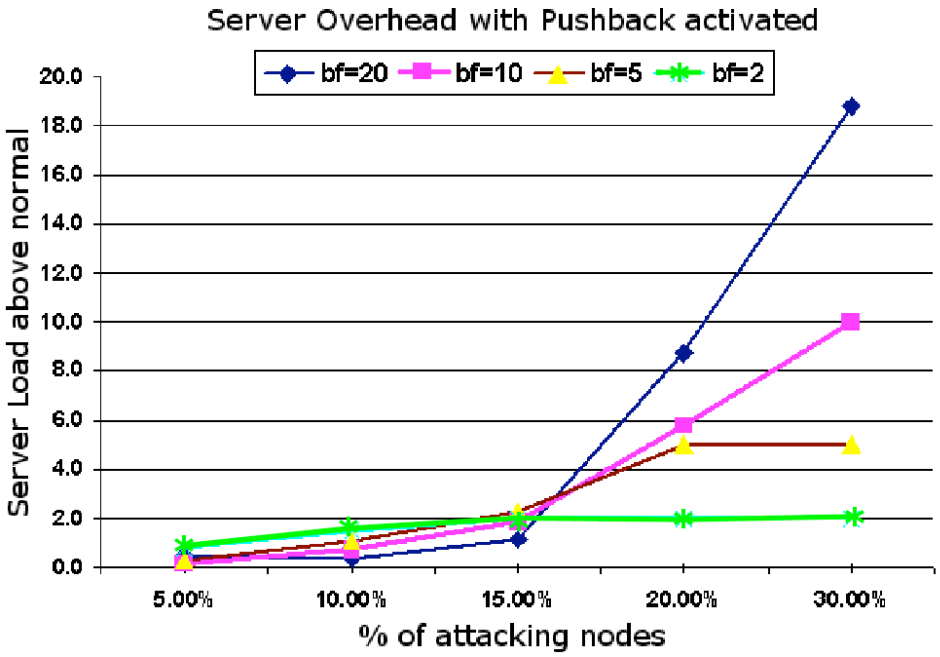


Figure 7.11: Increase in server load when under attack and with pushback protocol activated. A load of value 2 means that the server is serving twice the amount of normal requests. The performance of the pushback protocol remains acceptable even when 15% of the overlay nodes are attacking the target.

of the attackers is below 15%. If the fraction of the attacking nodes is increased beyond 15%, our system can only provide partial protection and the load of the server increases dramatically.

Figure 7.12 shows the total number of packets transmitted to the overlay network, this includes packets that traversed even one hop and then where dropped. Again, the performance of the system is not affected significantly by the attack even when 15% of the overlay nodes are subverted.

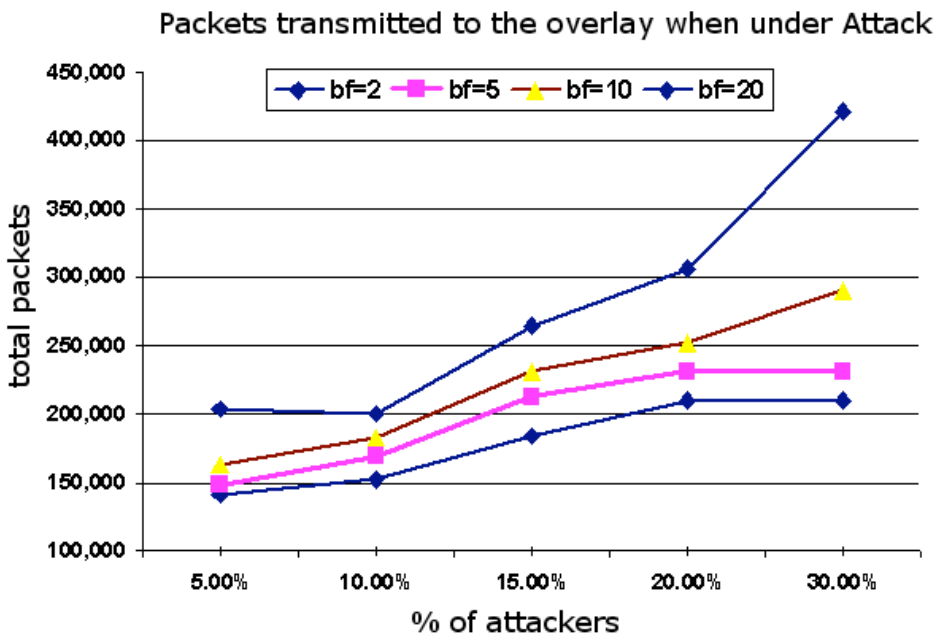


Figure 7.12: Total number of packets transmitted to the overlay network, this includes packets that traversed even one hop and then where dropped. Again, the performance of the system is not affected significantly by the attack even when 15% of the overlay nodes are subverted.

Chapter 8

Our System in Practice: Access-Assured Mobile Desktop Computing

8.1 Introduction

Using previously proposed DDoS protection mechanisms, such as SOS [91], Mayday [9] and Move [155] to protect access to a desktop service, hits a latency performance barrier: these systems cannot offer protection to time-critical or real-time applications, since they increase the end-to-end latency by more than 250% in some cases with high latency variance over time. For the large portion of the connected clients, the network latency performance degrades fast when these systems are under attack, since the clients have to detect this attack and re-establish their connections to different access points. To be of any practical use, interactive and real-time applications such as user GUI operations and multimedia streaming demand a low-latency pipe at all times.

Unfortunately, simply using existing DDoS protection mechanisms, such as SOS [91] and Mayday [9], to protect access to a desktop hosting service does not provide a viable solution. These systems cannot offer protection to interactive applications, as they increase end-to-end latency by up to 250%, with high latency variance over time. Furthermore, as

the infrastructure comes under attack, a large portion of the clients experience network latency performance degradation, as clients need to detect the attack and re-establish their connections to the hosting services using different access points. To be of any practical use, typical desktop interactive applications, such as UI interactions and multimedia streaming, demand a low-latency connection at all times. Caching of information to other nodes is not a solution since it requires vast amount of storage and network bandwidth. Additionally, it raises privacy concerns since the trust is moved to nodes, other than the originating center, that need to store and manage the cached information.

In this context, we introduce *Access-Assured Mobile* (A^2M) desktop computing, a hosted computing infrastructure that combines a remote-display architecture with a stateless indirection-based network (IBN) composed of dedicated nodes. A^2M provides both protected and efficient access to hosted desktop computing environments, even in the presence of denial of service attacks. Nodes participating in the IBN communicate only to exchange control messages, but not to route the client's data, unlike previous overlay-based approaches [91, 9, 155]. A^2M clients exploit the path diversity naturally exhibited by a distributed IBN to "spread" their traffic such that directed attacks do not cause service disruptions. To further alleviate any potential delays introduced by the IBN and reduce the latency in the end-to-end communication, A^2M uses a number of other optimizations at the remote display level to minimize the impact these delays may have on the user's experience. A^2M combines a simple low-level display protocol and a server-push model to minimize client-server synchronization and network round-trips. Atop this basic model, A^2M implements higher-lever mechanisms, such as client-managed cursor display, shortest-job-first display command scheduling, and a non-blocking drawing pipeline, further increasing the overall interactive response of the system. *We believe A^2M to be the first demonstrably practical overlay-based DDoS-protection mechanism.*

A^2M treats a connection from a client device to the server as having two parts: an upstream/uplink data channel *i.e.,* data transmitted from the client to the server, and a

downstream/downlink data channel, with data going to the client. A^{2}M's client-to-server traffic directly reflects its interactive nature, and can be characterized by small, fairly sparse upstream packets (sent by the client as a result of user-input events, TCP ACKs, *etc.*), with significantly larger downstream bursts of traffic containing the visual changes generated in response to the user's events. By routing only the upstream data through the IBN, thus protecting the upstream data channel, A^{2}M takes advantage of this traffic asymmetry to increase the resilience of the hosting infrastructure with minimum impact on the performance of the system (throughput and latency between client and server).

A^{2}M achieves this by using the following two mechanisms. First, since DoS attacks only congest the upstream link to the hosting servers, A^{2}M only needs to protect delivery of the relatively sparse client-to-server traffic. Second, to minimize the overhead that protecting traffic has on the performance of the system, A^{2}M uses several mechanisms to reduce both the quantity of upstream remote display traffic, and the effects additional delays may have on the overall performance of the system. We should note that A^{2}M can protect both directions in a communication channel, albeit at the cost of increased end-to-end latency.

Client traffic admitted in the IBN is redirected to a special node or set of nodes (that change over time) and from there forwarded to the protected server. Only that node, which we call secret forwarder, is allowed to send traffic to the server; all other traffic is filtered by the ASP's home ISP. The IBN is a distributed network of dedicated nodes that receive the traffic destined to the end server, discriminating between legitimate traffic and potentially malicious traffic, reflecting the former and dropping the latter. Thus, attacks aimed directly at the server will be stopped deep inside the ISP's network, where congestion has not yet occurred. The distributed nature of the IBN means that it will take an extremely well provisioned adversary to suppress its functionality, since attack traffic must be split among all the IBN nodes. This approach provides a very desirable solution for A^{2}M, because it has very little latency overhead since the system introduces only two extra hops for each packet. In addition, A^{2}M uses multi-path routing in the upstream, client-to-server direction, and

replicates packets in that relatively low-traffic stream to improve resilience as well as lower latency, while allowing packets in the server-to-client direction to follow regular routing paths. By exploiting multi-path routing for each replicated packet, the system can reduce the overall end-to-end latency to values that are, in some cases, even lower than the latency of the direct client-server connection, at the expense of extra bandwidth utilization.

We have implemented and deployed A^{2}M on PlanetLab [133]. We measured the impact of our prototype on the end-to-end communication latency and throughput, and the resilience of the system under attack. Our results show that in all scenarios where we enable multi-path routing using packet replication, A^{2}M imposes minimal overhead on the end-to-end latency, even when the system is under attack. Furthermore, A^{2}M significantly increases the attack resilience of the hosting infrastructure, being able to operate optimally even when up to 50% of the indirection nodes have become unresponsive, and for different types of traffic between clients and the infrastructure (web traffic, video playback and GUI actions). In all the WAN experiments, end-to-end latency increases by less than 5% even when 40% of nodes have been compromised. Our experiments in a wireless network environment further demonstrate the effectiveness of A^{2}M for truly mobile users using lightweight devices. Through a simple quantitative model based on typical ISP interconnection topologies, we quantify the resilience of our IBN (and any indirection or overlay-based anti-DDoS mechanism).

To summarize, in this chapter we present the following contributions:

- We introduce and describe A^{2}M, a single-hop indirection-based access infrastructure, based on our previous work on packet spreading [154], that protects end-user access to remotely hosted interactive applications that require a low end-to-end network latency at all times to operate and for large indirection networks.

- We implement and evaluate A^{2}M in the real Internet using PlanetLab. Our experiments show that A^{2}M introduces very little latency in most scenarios. In this context, the latency-sensitive design of A^{2}M vastly outperforms existing overlay DDoS protection mechanisms, such as SOS [91], Mayday [9] and MOVE [155].

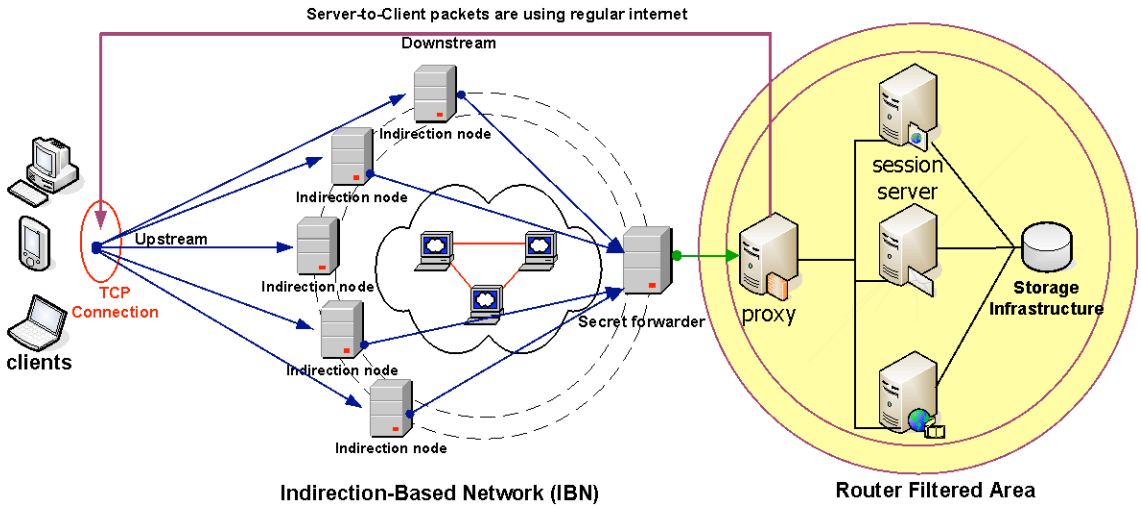


Figure 8.1: A^{2}M Architecture. The two directions of the client-server connection take different paths: the client-to-server direction goes over the indirection-based network, while the server-to-client direction goes directly to the client (not through the infrastructure). Legitimate uplink traffic is spread among the IBN nodes and competes with denial of service traffic for capacity in the links close to the server; allowing legitimate traffic through the indirection system allows us to differentiate the two. Indirection nodes can simultaneously serve as system entry points and secret forwarders, and are dedicated to this task (*i.e.,* they are not end-user-controlled nodes).

- We are the first to conduct realistic (non-simulation) experiments to evaluate the resilience of our system against DDoS attacks, and its performance under attack. Our results validate the design of A^{2}M, showing good performance for multimedia and interactive applications even with 30%–50% of the IBN nodes under attack.

8.2 A^{2}M Architecture

We have implemented the *Assured Access Network* (AAN), which extends the ideas of SOS [91], Mayday [9] and MOVE [155]. Our approach, shown in Figure 8.1, is to spread the packets from the client across all indirection nodes in a pseudo-random manner. This new communication mechanism protects the client-server connection establishment and provides uninterrupted connectivity to the target server throughout the client's session. The admitted packets are internally forwarded to a secret forwarder (selected at random, and changing

over time), which is allowed to forward traffic to the utility server. Only authorized clients are allowed to use the IBN and contact the hosting servers and these clients are provisioned in advance (*e.g.,* at registration time) with the appropriate authentication material, such as an RSA public/private key pair and a public-key certificate [24, 29]. AAN works in conjunction with filtering routers close to the hosting infrastructure, to allow only traffic from the IBN's secret forwarders to reach A^{2}M's hosting servers. All other traffic is considered unauthorized and possibly malicious, and therefore filtered out.

Contrary to previous overlay architectures, our system achieves this filtering without the use of overlay routing to transfer the client's request to the server. In our system, legitimate packets are reflected to the secret servlet(s) generating a one-hop indirection network. As shown in Figure 8.1 there is no single path between the client and the server - instead packets are spread from the client to the indirection nodes creating a single-hop multi-path effect. Both the use of the single-hop indirection and the multi-path routing permit our system to scale well in terms of latency, as we shall see in Section 8.5. For more details on the overlay architecture itself, see [154]. For completeness, we describe the most salient points of the AAN in the Appendix.

As shown in Figure 8.1, A^{2}M's architecture is divided in two major components: the hosting infrastructure and the access infrastructure. Our approach is utilizing our protection architecture presented in the previous chapters appropriately modified for the remote desktop application. The hosting infrastructure provides an environment for desktop sessions where a user's session is decoupled from any particular end-user access device, by moving all application state to hosting servers. Applications run within these servers, and their display output is redirected over the network to the access device. Redirection is performed by a per-session virtual display driver that translates from application display-draw commands to A^{2}M's display protocol commands. The protocol commands are then forwarded to the client device for display. A^{2}M extends previous work on desktop hosting infrastructures such as MobiDesk [15] by providing mechanisms that provide continuous access to hosted desktop

sessions, even in the presence of distributed denial of service attacks on the hosting servers.

A^2M's access infrastructure provides the connection between users on the network and the applications running on the hosting servers. Users make use of a simple client application that merely forwards input events to the applications running on the server, and processes display updates generated in response to these events. This application model results in a highly asymmetric network traffic pattern. On one side, input events (headed uplink, or upstream toward the server) are very small pieces of information that are generated at a relatively slow, human-dependent rate. On the other hand, display updates (headed downlink, or downstream toward the client) are orders of magnitude larger and are generated as fast bursts of activity. For example, during web browsing, a single user input event (a mouse click on a link) results in a full-screen update having to be displayed (the destination web page).

The traffic asymmetry is made more pronounced when we consider the different roles and importance of input events and display updates. In an interactive system user experience is dictated by the response time, which in turn is determined by how quickly input events are processed and display updates are made visible to the user. If response time is too high, the user will become exasperated and frustrated with the system. Since a single input event triggers the generation of display updates, guaranteed delivery of each event becomes crucial for the performance of the system. On the other hand, humans are known to be more tolerant to partial updates than to longer response times, because partial updates provide feedback to their actions. Delivery of updates should then be made such that updates can begin to be displayed as soon as possible, even if the complete update takes longer to appear.

The resource centralization around the hosting infrastructure results in a threat model where denial of service attacks on the system will only affect the uplink direction, *i.e.,* the traffic **to** the hosting servers, by saturating the network links and queuing buffers close to the servers or by directly attacking the hosting infrastructure servers. Therefore, it is crucial for A^2M to protect this communications channel from interference, blocking unwanted traffic

close to the attacker before it can reach the service providing machines. On the other hand, the downlink direction will for the most part be relatively free of noise, and without any need to be protected.

Note that denial of service attacks typically affect the uplink direction, *i.e.,* the traffic **to** the server, by saturating the network links and queuing buffer close to the server. The downlink direction is relatively free of noise. Thus, we are primarily interested in protecting the client-to-server traffic from interference; the opposite direction does need typically any such protection.

Taking advantage of both the traffic asymmetry and the threat model, A^2M partitions bi-directional connections between the client and the server into an indirected client-to-server multi-path and a direct server-to-client path. The IBN takes care of routing input events and other client-to-server traffic and protects the hosting infrastructure. Protection is performed by acting as a distributed firewall that conceptually distinguishes between authorized client-generated traffic, and unauthorized and possibly malicious traffic. Traffic permitted to traverse through the IBN is directed to a filtering router close to the hosting servers, whereas all other traffic is dropped or rate-limited providing a distributed "shield" against both network congestion and host directed attacks.

The direct server-client path in turn ensures that large and bursty display updates are delivered to the client as quickly as possible, even if parts of them are lost or delayed and need to be retransmitted. A^2M's approach represents a sharp departure from traditional interactive client-server architectures, where a vulnerable bi-directional direct connection provides the only means of communication between the client and the server. We should note that A^2M does not preclude routing both traffic directions over the IBN, albeit at a possible increase in the end-to-end latency when no replication of packets is present. Since this mode is not necessary for our usage scenario, we do not further consider it in this paper.

8.2.1 A^{2}M High-Level System Operation

To provide seamless and ubiquitous connectivity, A^{2}M encapsulates all functionality within a self-contained client application that manages communication with the indirection infrastructure, forwards user events to hosted applications, and displays application output on the local device. To access a desktop session, users must first obtain access to the IBN, which in turn allows them to authenticate with the hosting infrastructure, and then gain access to their session. Users need to be recognized as legitimate in order for the IBN to distinguish their traffic from other unauthorized, possibly malicious traffic. In contrast to traditional service providing infrastructures such as web-content distributors, A^{2}M requires users to be authenticated and does not allow anonymous users, because only authorized users should be able to connect to the hosting infrastructure. A^{2}M ties the authentication requirements of the IBN and the hosting infrastructure into a single, seamless process.

When a user attempts to connect to A^{2}M, the client application first acquires a "ticket" from one of the indirection nodes. This ticket gives it temporary access to the IBN, and allows the client to contact A^{2}M's authentication service to identify itself as a legitimate user. After being successfully authenticated and authorized, the client receives a longer-term session ticket from the IBN, and a connection to the A^{2}M server hosting the user's session. The session ticket identifies the client as a legitimate user of the system, and allows it to freely interact with the hosting infrastructure. To avoid stale sessions to be used in an attack, the session ticket needs to be renewed periodically. The specific details of ticket acquisition and renewal are further discussed in the Appendix. The connections to the hosting infrastructure are asymmetric: the client-to-server traffic will travel through the IBN, while the server-to-client traffic will use regular Internet routing. In the case where a session does not already exist, a new session is created and populated, before the client is allowed to connect to it. The authentication and connection setup process is done transparently by the client application, and it does not require special support from the underlying devices. This simplicity allows A^{2}M users to access their sessions from almost any number of

Internet-enabled devices.

Once the connection to the hosting server is established, the client will be recognized as a legitimate user, and user input events will be allowed to traverse the indirection nodes and be routed to the hosted applications. This process continues until the user disconnects from the session, at which point the client's ticket is revoked and the connections are closed. Since a disconnected client is no longer allowed to use the system, previously legitimate devices cannot be reused as attack tools on the infrastructure.

8.3 AAN Operation Details

8.3.1 Client Authentication

Before a client is allowed to send traffic through the IBN, it must obtain a *ticket,* which is then included in all subsequent packets sent to the IBN, until it expires. The ticket is used by the IBN nodes to authenticate the user, validate the routing decisions, and prevent malicious (or subverted) clients from utilizing a disproportionate amount of bandwidth. To obtain a ticket, the client contacts an indirection node at random using a ticket establishment protocol described in detail in previous work [154]. This protocol is fully distributed and resilient to CPU exhaustion attacks. Furthermore, the ticket issuing process is protected against replay and IP spoofing attacks. At the end of the protocol, the client and the IBN have authenticated each other, and the client is in possession of a ticket. The ticket contains a session key K_u, a range of sequence numbers for which it is valid (more on this later), and the IP address of the client, all encrypted under K_M, a secret key negotiated periodically (*e.g.,* every few hours) among all indirection nodes. Note that only the indirection nodes can decrypt the ticket; clients treat the ticket as an opaque value that they must provide to the AAN with each packet they need to forward. A second copy of K_u is independently encrypted under the client's public key. This ticket can only be used by the client to continue the authentication protocol (*i.e.,* prove liveness for both the IBN nodes and the client. Once the full two-party

authentication is completed, the last indirection node provides the client with a ticket that is not "restricted," *i.e.,* the corresponding flag inside the ticket is cleared. As we discussed in the previous section, the tickets are periodically refreshed, to avoid situations where a malicious user distributes a valid session key and ticket to a large number of zombies that then simultaneously send attack traffic through the IBN.

8.3.2 AAN Encapsulation

When using , every packet sent by a client to an indirection node contains four fields: a client identifier, the ticket, an authenticator, and a monotonically increasing sequence number. Recall that the ticket contains the session key and the maximum sequence number for which the ticket is valid, and is encrypted and authenticated under a secret key known only to the indirection nodes. Note that these indirection nodes are *not* user machines, but are hosts dedicated to offering a DoS protection service.

The sequence number is a 32-bit value that is incremented by the client for each packet transmitted through the IBN with a given session key. The client identifier is a random 32-bit value that is selected by the indirection node that authenticated the client, and is used as an index in the table of last-seen sequence number, maintained by each indirection node for each active client. The authenticator is a fast hash function, such as UMAC [22], computed over the session key and the whole packet (including the ticket, sequence number, and client identifier). Thus, the only amount of state each indirection node needs to maintain per active client are the client's identifier and the last sequence number seen from that particular client. Assuming that both the client identifier and the sequence number are 32-bit values, each indirection node needs to maintain only 64 bits of state for each client; thus, if the system has 1 million active clients, we will only need 8 MB of state — easily manageable even if it is stored in main memory, given current prices of RAM.

8.3.3 AAN Connection Initiation

A client transmitting a packet through the IBN uses the session key and the sequence number as inputs to a pseudo-random function (PRF). The output is treated as an index to a publicly available list of indirection nodes, through which the packet will be routed. The list of available indirection nodes does not need to change frequently, even if nodes become unavailable (*e.g.,* for maintenance purposes), and can be downloaded by clients independently of the protected communication. For the purposes of this paper, we assume that clients trust the IBN's entry points. Discussion and analysis of an environment where access points cannot be safely trusted can be found in [176].

The client encapsulates the original packet (addressed to the final destination) inside a packet for the indirection node, along with the information identified above (client identifier, ticket, sequence number, authenticator). The packet is then forwarded through the IBN to the secret forwarder for that particular destination, and from there to the final destination.

An indirection node that receives such a packet first verifies that the sequence number on the packet is larger than the last sequence number seen from that client, by using the client identifier to index the internal table. It then decrypts the ticket, obtaining the session key for that client, with which it verifies the authenticator. The indirection node also verifies that the sequence number is within the acceptable range of sequence numbers for this ticket. Finally, it uses the key and the sequence number along with the PRF to determine whether the client correctly routed the traffic. If all steps are successful, the indirection node updates the sequence number table and forwards the packet to the secret forwarder. Packets with lower or equal sequence numbers are considered duplicates (either accidental artifacts of the underlying network, or malicious replays by attackers) and are quietly dropped.

To avoid reuse of the same ticket by multiple DDoS "zombies," the range of valid sequence numbers for the ticket is kept relatively small (and contained inside the ticket), *e.g.,* for 500 packets. IBN nodes that receive valid tickets about to expire simply re-issue a new ticket with the same session key but a new range of valid sequence numbers. This approach,

combined with the state kept by each node, makes it prohibitive for attackers to reuse the same ticket from a large number of distinct nodes (each of which is only transmitting to a specific indirection node), since the new valid ticket needs to be continuously propagated to all zombies.

The key under which the ticket is encrypted is periodically established among all IBN nodes, using a simple key-distribution protocol. The precise properties of this protocol are not relevant to this discussion, and there exist a large number of such protocols in the research literature.

A client may decide to send multiple copies of the same original packet through different indirection nodes, to improve the probability that it will make it to the server even as indirection nodes are attacked. Doing so is fairly straightforward: each replica of the original packet is treated as a different packet, uses a different sequence number, and is transmitted through the corresponding indirection node. As we shall see in Section 8.5, this dramatically improves the resiliency of connections. Furthermore, packet multi-path replication often reduces average end-to-end latency by exploiting links that are not utilized efficiently by routing protocols. Our results are in line with independent work on the effect of multi-path routing on end-to-end latency [62, 8, 83] and availability [7].

8.4 General ION Attack Resistance

It is worth estimating the attack volume that any ION system can withstand. Since ISP backbones are well provisioned, the limiting factors are going to be the links close to the target of the attack. The aggregate bandwidth for most major ISP POPs is on the order of 10 to 20 Gbps[1]. If the aggregate bandwidth of the attack plus the legitimate traffic is less than or equal to the POP capacity, legitimate traffic will not be affected, and the POP routers can drop the attack traffic (by virtue of dropping any traffic that did not arrive through the overlay).

[1]For example, see http://www.verizonbusiness.com/us/about/network/

Unfortunately, there do not exist good data on DDoS attack intensities; network telescopes [116] tend to underestimate their volume, since they only detect response packets to spoofed attack packets. However, we can attempt a simple back of the envelope calculation of the effective attack bandwidth available to an attacker that controls X hosts that are (on average) connected to an aDSL network, each with 256 Kbps uplink capacity. Assuming an effective yield (after packet drops, self-interference, and lower capacity than the nominal link speed) of 50%, the attacker controls $128 \times X$ Kbps of attack traffic. If the POP has an OC-192 link (10 Gbps) to the rest of the ISP, an attacker needs $78,000$ hosts to saturate the POP's links. If the POP has a capacity of 20 Gbps,see Figure 8.2, the attacker needs $156,000$ hosts. Although we have seen attack clouds of that magnitude (or larger), the ones used in actual attacks seem to be much smaller in practice. Thus, an overlay-protected system should be able to withstand the majority of DDoS attacks. If attacks of that magnitude are a concern, we can expand the scope of the filtering region to neighboring POPs of the same ISP (and their routers); this would increase the link capacity of the filtered region significantly, since each of the neighboring POPs see only a fraction of the attack traffic. Our discussion is not meant as a proof of security against DDoS attacks, but as an exploration of the limits of such mechanisms. It is important to note that these findings agree with other similar studies [169].

These numbers give us a baseline from which to determine how much more resistant our spread-spectrum system is compared to a basic indirection approach. Assume an attacker can create an effective attack bandwidth of K Mbps, and that each overlay node can be disabled through an attack sustaining D Mbps; thus, an attacker can simultaneously disable $\frac{K}{D}$ out of the N overlay nodes. When an attacker can observe a client's actions (*i.e.,* which overlay nodes a client routes traffic through), the effectiveness of the attack (defined as the probability of disrupting communications) is 1, as long as $K > D$. What is a likely value for D? The Click software router with commodity hardware [97] claims a switching capacity of 435,000 64-byte packets, or 222 Mbps. Taking a more conservative value of

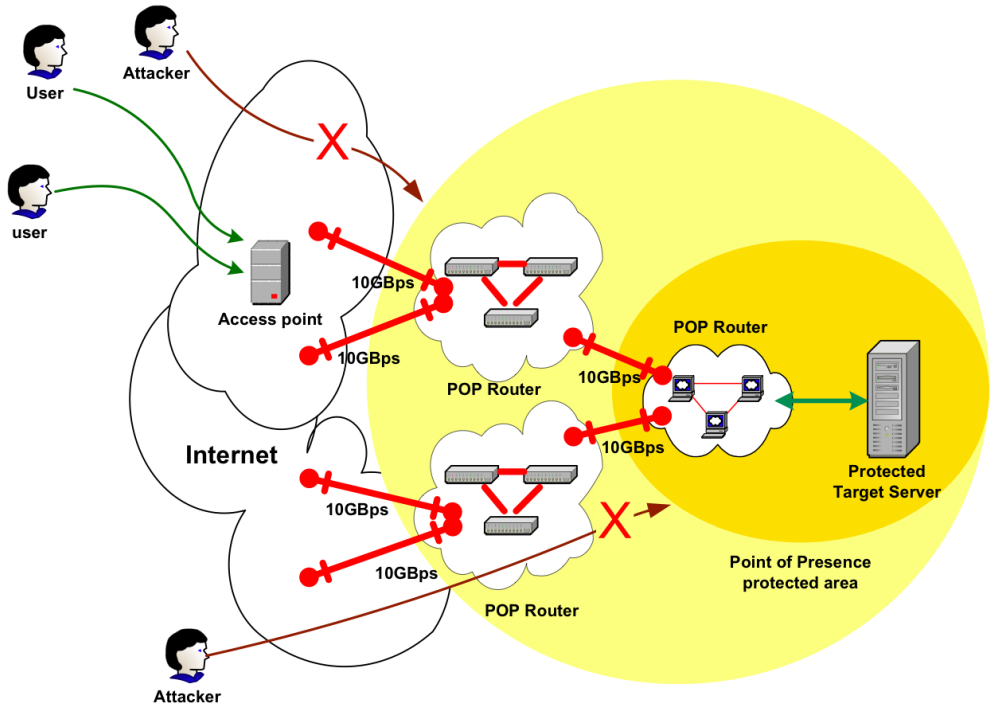


Figure 8.2: Filtering at various locations in the home ISP *viz.* attack volume that can be withstood.

50 Mbps, an attacker can saturate an overlay node by using $1,740$ hosts. Furthermore, an attacker controlling $100,000$ nodes (not enough to directly attack the target) can render approximately 60 geographically dispersed, well connected overlay nodes inaccessible at a time. Assuming an overlay network of a size comparable to Akamai's (approximately $2,500$ nodes), the attacker can render 2.5% of the overlay unusable. Moreover, if we use the results of the packet switching measured by The Click router, we have an 440% increase in our ability to defend.

To guarantee packet delivery at a given probability P_s in the presence of such attacks, we need to select the number of packet replicas R such that $P_s = 1 - (\frac{K}{D \times N})^R$ or $P_s = 1 - f^R$, where f is the percentage of the attacked nodes. If we assume that users initiate TCP connections with the protected server, then P_s should be no less than 90%, otherwise the connections stall [120]. From the formula for P_s and using the fact that $P_s = 0.9$ for TCP, we can compute the required bandwidth given the size of the network, or the fraction of

nodes that need to be successfully attacked to disrupt the user's TCP session. For example, if we send each packet twice, *i.e.,* have a packet replication $R = 2$, the attacker has to bring down 32% of the nodes participating in the overlay network. For an overlay network of $2,500$ nodes, an attacker needs to gain access and coordinate a network of $1,375,000$ or $6,125,000$ zombies assuming 50 Mbps and 222 Mbps switching capacity respectively. In addition, if we increase the packet replication value to $R = 3$, the percentage of nodes that need to get compromised jumps to 46%. This corresponds to almost half of the nodes in the overlay network which further increases by almost 50% the number of bots required to incapacitate our system.

To avoid imposing extra traffic on the network by replicating each packet, we can instead select the packets that we replicate at random with a probability P_r. Now P_s becomes $P_s = 1 - f(1 - P_r(1 - f))$ since the probability that a packet will fail the first time transmitted is f and the failure probability for the possibly replicated packet is $(1 - P_r(1 - f))$. Again, using $P_s = 0.9$ for TCP, we see that if we replicate 50% of the transmitted packets, the fraction of the nodes that need to get compromised is 17%, which is significant for medium to large overlay networks.

8.5 Implementation and Experimental Results

To demonstrate the feasibility of the proposed architecture, we have implemented an A^2M prototype which hosts and protects Linux-based desktop sessions. We deployed the indirection nodes of our prototype in 80 PlanetLab nodes, while having the client and server reside in our local network. Our architecture spreads the packets across all indirection nodes. Perhaps the most surprising aspect of our implementation is its size: excluding cryptographic libraries and the JFK protocol, the code implementing the complete functionality of the system consists of 1,600 lines of well commented *C* code. The JFK implementation itself adds another 2,500 lines of code. Although this is a prototype implementation and does not

include management code and other facilities that would be required in a production system, we feel that the system is surprisingly lightweight and easy to comprehend.

The implementation consists of the code for the indirection nodes, as well as code running on each client that does the encapsulation and initial routing. A detailed description of MobiDesk may be found in [15]. On the client, a routing-table entry redirects all IP packets destined for the protected servers to a virtual interface, implemented using the *tun* pseudo-device driver. This device consists of a linked pairs of virtual network interfaces and character devices that a user-level process can read and write. IP packets sent to the *tun0* network interface can be read by a user process reading the device */dev/tun0*. Similarly, if the process writes 1a complete IP packet to */dev/tun0* this will appear in the kernel's IP input queue as if it were coming from the network interface *tun0*. Thus, whenever an application on the client tries to access a protected server, all outgoing traffic is intercepted by the virtual interface. A user-level proxy daemon process reading from the corresponding device captures each outgoing IP packet, encapsulates it in a UDP packet along with authentication information, and sends it to one of the indirection nodes according to the protocol. The code running on indirection nodes receives these UDP packets, authenticates and forwards them to the secret forwarder, which forwards them to the final destination. There, the packets are decapsulated and delivered to the original intended recipient (*e.g.,* web server). The decapsulation can be done by a separate box or by the end-server itself. In addition to the decapsulation code on the indirection nodes, there is also a daemon listening for connection establishment packets from the clients.

The implementation of A^2M's prototype access infrastructure is divided into the client-access application, a forwarder in each indirection node, and the secret forwarder. To communicate with the IBN, the client application encapsulates outgoing messages along with the IBN authentication information in UDP packets that are sent to one of the indirection nodes, according to the protocol. The forwarder application running in the indirection node receives the UDP packets, authenticates them, and if valid and authorized, forwards them to

the secret forwarder. The secret forwarder decapsulates the packets and delivers them to the entry point of the hosting infrastructure. On every indirection node, there is also a daemon waiting for connection establishment packets from new clients.

In evaluating A^2M, we focused on two metrics: the quality of service in terms of latency, as this is perceived by the end user, and the system's resilience when under attack *i.e.,* node failures. PlanetLab provides a realistic network environment for our experiments that stresses the performance of our system because the packets follow different, highly variant paths to reach the protected server. In our experiments, we protected the uplink traffic from the client to the server routing it through the IBN, while the return path followed normal Internet routing (outside the IBN).

Our testbed consisted of a client PC simulating a typical remote-display access device, a laptop used as wireless access device, a server where the benchmark applications executed, and 80 indirection hosts deployed across various PlanetLab locations in the US and Canada. The client computer had a 450Mhz Intel Pentium-II CPU and 128MB RAM running Debian with Linux 2.4.27. Our client PC was chosen to reflect the type of low-power, thin-client devices which we expect to become A^2M's access devices. The laptop PC had a 1.5Ghz Intel Pentium M and 1GB RAM running Debian with Linux 2.6.10. The server was an Intel dual-Xeon 2.80GHz with 1GB of RAM running RedHat 9 with Linux 2.4.20.

We measured the performance of A^2M in web, video, and basic interactive tasks as representative applications of typical desktop usage. Our web measurements used the Mozilla 1.6 browser to run a benchmark based on the Web Page Load test from the Ziff-Davis i-Bench benchmark suite. The benchmark consists of a sequence of 54 web pages containing a mix of text and graphics. The browser window was set to full-screen resolution for all platforms measured. Video playback performance was measured using Mplayer 1.0pre3 to play a 34.75 second video clip of original size $352x240$ pixels displayed at full-screen resolution. For our interactive tests we recorded a number of sessions where simple interactive tasks were performed. Recording the sessions allowed us to reliably play

back the exact same tasks under different network conditions. The measure of performance for these tests was the latency experienced by a user performing the specific task. The primary measure of web browsing performance was the average page-download latency in response to a mouse-click on a web page link. To minimize any additional overhead from the retrieval of web pages, we used a conservative setup where the web server was directly connected to the hosting server through a LAN connection. The primary measure of video playback performance was video quality [123], which accounts for both playback delays and frame drops that degrade playback quality. For example, 100% video quality means that all video frames were displayed at real-time speed. On the other hand, 50% video quality means either that half the video frames were dropped when displayed at real-time speed or that the clip took twice as long to play even though all of the video frames were displayed.

Operation	**Direct**	**0%**	**50%**	**100%**	**200%**
ls -l	0.81	0.80	0.82	0.88	0.79
start gvim	1.53	1.58	1.53	1.55	1.53
hello world (1)	8.50	8.47	8.69	8.48	8.51
hello world (2)	7.90	7.87	7.92	7.94	7.88
quit gvim	2.61	2.60	2.61	2.60	2.61
move window (outside)	2.86	2.90	2.89	2.88	2.86
move window (within)	2.20	2.38	2.45	2.17	2.20
resize window	3.44	3.49	3.55	3.50	3.48
open mozilla	6.72	6.73	6.94	6.72	6.72
go to ncl/pubs	6.84	6.93	6.87	6.89	6.88
page down once	0.17	0.14	0.16	0.17	0.18
up arrow to top	3.21	3.24	3.27	3.24	3.26
quit mozilla	0.52	0.52	0.71	0.53	0.50

Table 8.1: Interactive Results

We first examined the effects that the basic indirection network and various levels of packet replication had on the overall performance of the system. The levels of replication tested were no replication, 50% (meaning one extra copy of each packet with probability 0.5), 100% replication (one extra copy of each packet) and 200% replication (two extra copies of each packet). We also measured the impact of the IBN size by running our experiments on 8

Operation	Direct	5%	10%	20%	40%	50%
ls -l	0.81	0.82	0.83	0.79	0.81	0.79
start gvim	1.53	1.53	1.53	1.53	1.53	1.55
hello world (1)	8.50	8.53	8.51	8.51	8.51	8.52
hello world (2)	7.90	7.91	7.89	7.89	7.91	7.89
quit gvim	2.61	2.61	2.60	2.60	2.61	2.61
move window (outside)	2.86	2.89	2.90	2.85	2.86	3.08
move window (within)	2.20	2.22	2.15	2.13	2.16	2.15
resize window	3.44	3.45	3.44	3.42	3.47	3.47
open mozilla	6.72	6.73	6.72	6.73	6.74	6.98
go to ncl/pubs	6.84	7.08	6.87	6.89	6.86	6.94
page down once	0.17	0.61	0.19	0.19	0.18	0.19
up arrow to top	3.21	3.32	3.28	3.28	3.25	3.22
quit mozilla	0.52	0.52	0.50	0.53	0.52	0.52

Table 8.2: Interactive Results under DDoS 200% replication

and 80 nodes participating in the IBN. We ran a baseline test where we used a direct LAN connection between the client and the server. Since the indirection nodes were deployed over a wide area with varying network latency, this test provided us with a very conservative measurement of the indirection overhead. In a realistic A^2M deployment, the client and server will typically reside at different, topologically distant locations. In that case, it is entirely possible for the indirection path to provide better connectivity characteristics than a direct connection due to the multi-path effect, which allows the packets originating from the client to follow a route with lower latency towards the end server [62, 8, 83, 164]. Although not shown in our results for ease of viewing, we also compared the performance of A^2M to that of MobiDesk and found it to be the same on the direct connection case.

Figure 8.3 illustrates the end-to-end average web latency results as perceived by the client. We can see that even for the worst-case scenario, an 80-node IBN without packet replication, the overhead from the indirection results in a latency increase of only 2 (*i.e.*, twice the latency of the baseline direct connection). When 50% packet replication is used (*i.e.,* replicating a packet with probability 0.5), the overhead drops significantly to 40% for the 80-node IBN. The drop in the overhead is due to the variant path latency of nodes

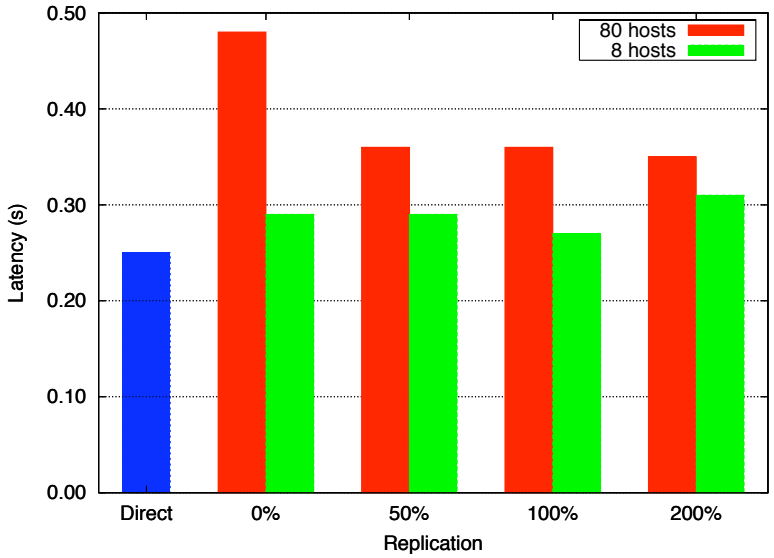


Figure 8.3: Web latency *vs.* packet replication when measured close to the client and for 8 and 80 nodes participating in the IBN. The direct bar shows the latency when we fetch the page directly from the server locally using a LAN and without protection. The latency overhead drops to 40% at 50% packet replication (*i.e.,* duplicating a packet with probability 0.5).

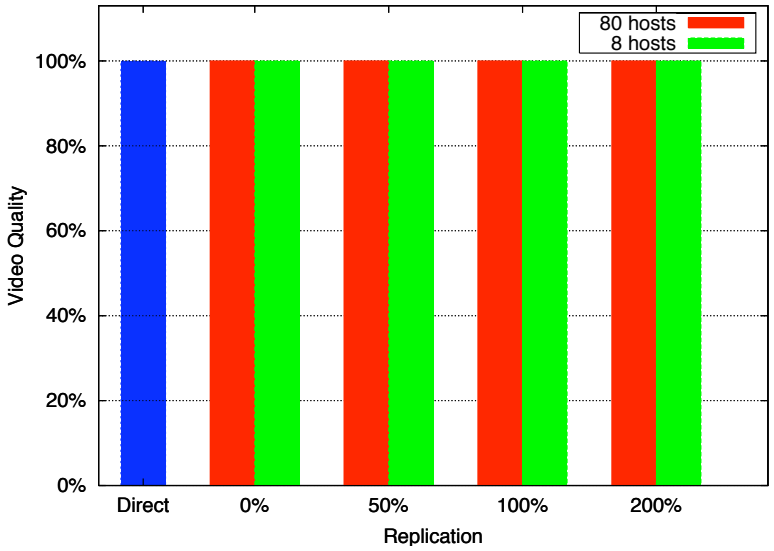


Figure 8.4: Video quality *vs.* packet replication — video quality remains 100% under all test scenarios even for a 80-node IBN with no packet replication, despite the use of indirection.

Host	Round Trip Time
planetlab(1,2).comet.columbia.edu	0.288 ms
planetlab(1,2).cs.cornell.edu	8.57 ms
planetlab(1,2).cis.upenn..edu	9.30 ms
planetlab(1,2).cs.virginia.edu	8.08 ms

Table 8.3: Average Round Trip Time (RTT) between nodes participating in the 8-node IBN and the protected server.

participating in the IBN. TCP does not behave optimally when packets appear to have high variance when arriving at the end server out of order. Adding packet replication decreases this variance, as the same packet follows more than one paths with different latency and the end server uses the one that arrives first. Boosting the replication beyond 50% follows the law of diminishing returns, as each additional increase in replication gives us less latency improvements. Care must be taken however, as too much packet replication can cause performance degradation, since bandwidth is "wasted" on duplicate packets. This is better exemplified by the results on the 8-node network using 200% replication. The 80-node network does not exhibit the same adverse affect because its average path latency is higher, allowing the secret gateway enough time to process the encapsulated packets received by the IBN. Moreover, for the 8-node network, we amplified this effect by lowering the average latency, using PlanetLab nodes that were "close" to the protected server, as shown by the average round trip times in Table 8.3.

To measure our system with an application that could generate more upstream traffic and required the system to maintain its quality of service above a threshold for latency, we used video playback. Figure 8.4 shows the results for video quality as measured at the client side. We can clearly see that A^2M performs optimally under all test scenarios, providing the same perfect video quality as the direct LAN connection scenario, even for the worst-case scenario of the 80-node IBN deployed over a WAN with no packet replication.

The average per-page data transfer during the web benchmark in both directions for various packet replication settings is shown in Figure 8.7. The results demonstrate that since

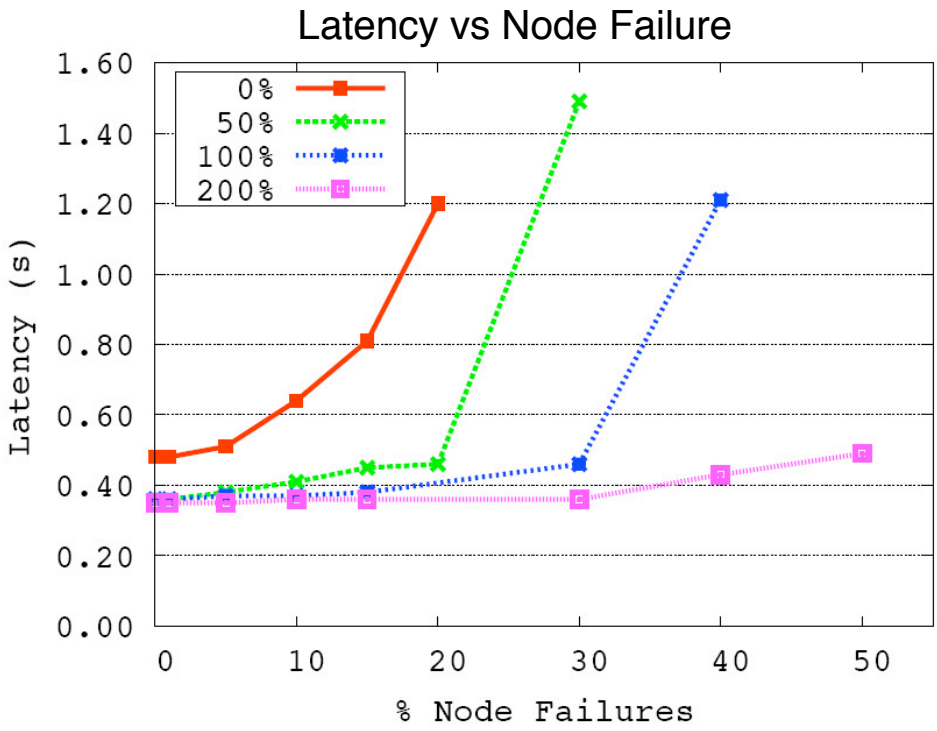


Figure 8.5: Web latency under DDoS attack. Latency increases in response to increased nodes failure. Allowing packet replication, higher resilience is achieved, while maintaining almost constant latency even in the presence of large node failures.

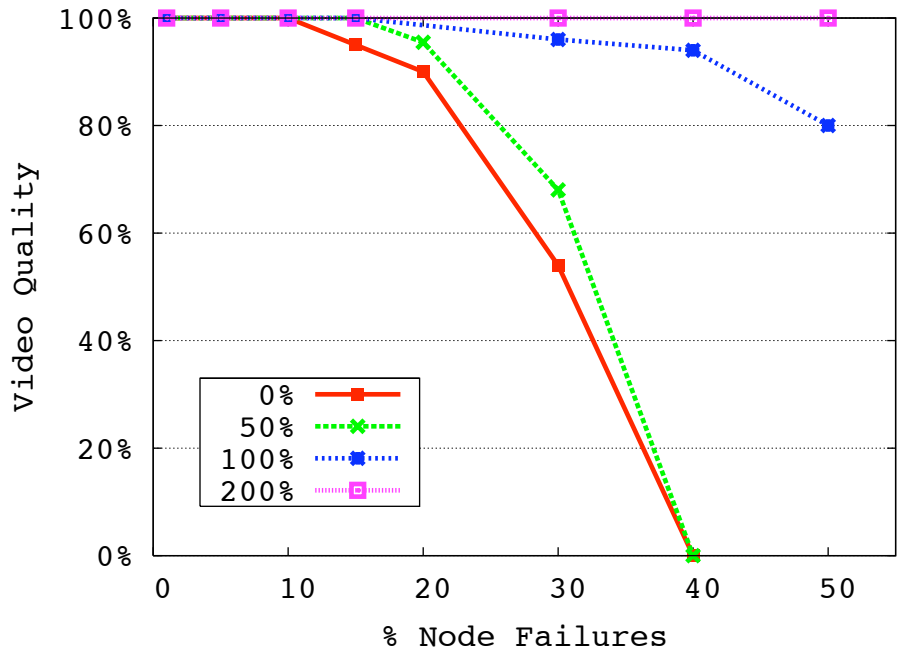


Figure 8.6: Video quality under DDoS attack. Video quality drops only after a substantial percentage of nodes become unresponsive. At 200% replication, latency does not increase even with 50% node failures.

the upstream channel carries only input events (in this particular case, mouse clicks) and data ACKs, packet replication has very low overhead (~2.3KB to ~8.5KB) even for large pages. Similarly, Figure 8.8 shows the amount of data transferred during video playback. Although the upstream data size is significantly larger when compared to the web benchmark (from around 900KB to 2.2MB for 200% replication), it is still only a fraction of the traffic generated in the downstream channel, and confirms the large asymmetry of remote display traffic, and the low overhead of A^{2}M's access infrastructure.

To examine the behavior of the overall system when under attack, we measured its resilience to a simulated denial of service attack that targeted the IBN itself. Our threat model assumes that the attacker can render a fraction of the nodes participating in the IBN unresponsive, thus inducing packet loss in the TCP connection of a user connected to the hosting server. All resilience tests were run on the 80-node IBN network. When attacked, a node stops forwarding packets from the client to the end host, acting as a mute node.

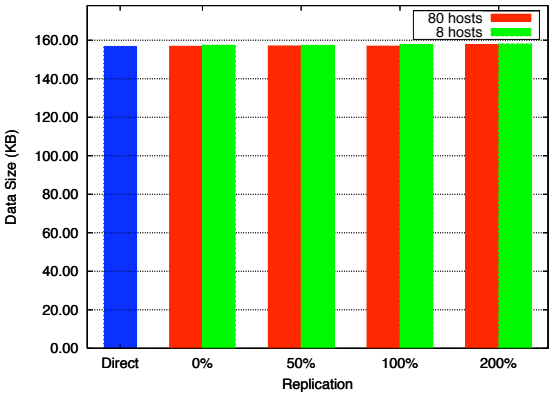


Figure 8.7: Average per-page data transfer *vs.* packet replication for an 8-node and an 80-node IBN. Notice that the data replication does not show up in the graph, since the upstream link is only used to send input events, which are a small fraction of the total data transmitted.

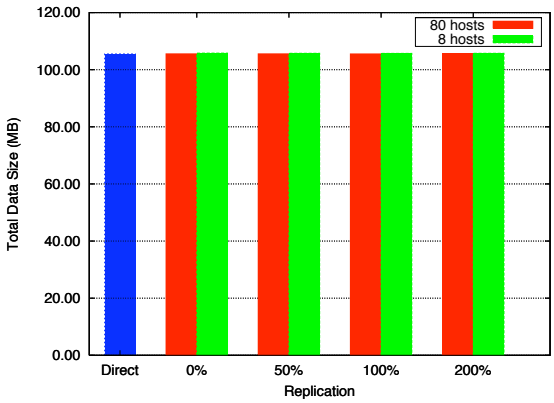


Figure 8.8: Total video data transmitted *vs.* packet replication for 8 and 80-node testbeds. The upstream is only used for TCP ACKs, which are a tiny portion of the actual downstream video data.

Since there is no immediate feedback, clients do not know which A^{2}M nodes are operating and which are suppressed by the attacker. Figure 8.5 illustrates the effects on the average web page latency as we increase the percentage of node failure, and demonstrates both the resilience of A^{2}M and the advantages of packet replication. Without packet replication, latency quickly degrades to twice that of the direct connection when we have 15% of node failures, and reaches three times for 20% node failure. On the other hand, employing packet replication allows A^{2}M to maintain an almost constant latency that is very close to the direct connection, even under 50% A^{2}M node failure, in the case of 200% replication. These results are reinforced when we consider the video playback measurements. Figure 8.6 demonstrates that excellent video quality can be maintained even after a substantial percentage of nodes become unresponsive. As we increase packet replication, the threshold can be drastically increased, to the point that A^{2}M is able to provide perfect video playback for up to 30% node failure, and very good (80%) video quality with 50% node failure.

8.5.1 Interactive Applications

Although video streaming and web browsing are both representative and demanding applications, we felt that we needed to include another set of experiments that require a high level of synchronization between the upstream and the downstream channel. We performed four different tests, each representing typical interactive operations on a desktop environment. The tests were performed by first recording a session of a user performing the appropriate operation, and then playing back the session in a number of different experimental scenarios. Our measure of performance was the user-perceived latency in response to the interactive operations. The four tests performed were: echo, minimize/maximize window, scroll, and move window. The echo test measured the time it takes for a line of text to appear on the screen after the user has pressed and depressed a key. The minimize/maximize window tests measures the time it takes to maximize a window after the user has pressed the maximize button, and then (after the window has been maximized) to minimize it after the user has pressed the minimize button. The scroll test measures the time it takes to scroll down a full-screen web page in response to a single `Page Down` key-press, and then the time it takes to scroll back to the top by leaving the `Arrow Up` key pressed. Finally, the move window test measures the time it takes to move a window across the screen. The window's size is about one fifth of the screen's size, and it is moved by dragging the window while the left-mouse button is pressed. The window operation is opaque, *i.e.,* the contents of the window are continuously redrawn as the user performs the move operation.

The end-to-end latency the end users experience for these operations is shown in Figures 8.9, 8.10, 8.11, and 8.12. These measurements show that without using packet replication, and for attacks up to 20% of the indirection nodes, the client's end-to-end latency increases only by a factor of 2.5 when compared to the direct, non-protected case. On the other hand, if we permit packet replication, we notice an increase in latency only after 50% of the indirection nodes become unresponsive. In some cases, for attack intensities that exceeded 20% of the indirection nodes and without replication the network conditions were

Operation	Direct	5%	10%	20%	40%	50%
ls -l	0.81	0.81	0.82	0.82	0.78	1.00
start gvim	1.53	1.53	1.55	1.60	1.47	1.29
hello world (1)	8.50	8.52	8.51	8.73	8.60	8.47
hello world (2)	7.90	7.90	7.94	7.90	7.87	7.91
quit gvim	2.61	2.61	2.62	2.58	2.57	2.81
move window (outside)	2.86	2.89	2.88	2.89	2.95	3.08
move window (within)	2.20	2.15	2.15	2.17	2.21	2.41
resize window	3.44	3.46	3.41	3.50	3.58	3.42
open mozilla	6.72	6.74	6.73	6.73	5.63	5.67
go to ncl/pubs	6.84	6.91	6.90	6.91	6.72	6.69
page down once	0.17	0.16	0.18	0.19	0.19	0.23
up arrow to top	3.21	3.26	3.29	3.23	3.42	3.41
quit mozilla	0.52	0.51	0.52	0.52	0.50	0.56

Table 8.4: Interactive Results under DDoS 100% replication

Operation	Direct	5%	10%	20%
ls -l	0.81	1.06	0.95	1.02
start gvim	1.53	1.62	1.85	1.70
hello world (1)	8.50	8.53	8.50	10.09
hello world (2)	7.90	8.56	7.91	8.46
quit gvim	2.61	2.60	2.61	2.85
move window (outside)	2.86	3.29	2.86	2.84
move window (within)	2.20	2.44	2.15	2.62
resize window	3.44	3.57	3.55	4.18
open mozilla	6.72	6.73	5.63	10.40
go to ncl/pubs	6.84	6.88	6.92	7.82
page down once	0.17	0.19	3.26	0.35
up arrow to top	3.21	3.28	0.69	4.65
quit mozilla	0.52	0.75	2.13	0.72

Table 8.5: Interactive Results under DDoS 50% replication

Operation	Direct	5%	10%	20%
ls -l	0.81	1.04	0.94	0.99
start gvim	1.53	1.85	1.60	1.82
hello world (1)	8.50	8.46	8.48	8.85
hello world (2)	7.90	7.98	7.98	8.55
quit gvim	2.61	2.57	2.73	2.98
move window (outside)	2.86	2.84	2.92	3.01
move window (within)	2.20	2.17	2.36	2.80
resize window	3.44	3.47	3.73	3.47
open mozilla	6.72	6.72	6.78	6.78
go to ncl/pubs	6.84	7.10	6.90	7.24
page down once	0.17	0.37	0.14	0.14
up arrow to top	3.21	3.47	3.40	5.12
quit mozilla	0.52	0.72	0.53	0.77

Table 8.6: Interactive Results under DDoS 0% replication

too adverse for the test to complete.

8.5.2 Performance using Wireless links

Our next step was to replicate some of the experiments we had for the wired network using a mobile client with a wireless connection. With the wireless tests we want to explore the performance of both traditional laptop computers, and more common mobile access devices, such as PDAs. To this end, we restricted ourselves to using an 802.11b network (as opposed to a fast 802.11a or 802.11g). Furthermore, to realistically show the performance on an expected A^2M deployment, we conducted all the tests over our university's public wireless network. Since wireless connections introduce an additional source of error to our system, we chose to show experiments using video playback, because this type of application stresses the bandwidth and latency requirements of the overall system. In order to simulate a PDA connection, we reduced the client's window size to a resolution of $320x240$ pixels. Similar to our wired tests, we obtained a baseline case where we played back the video using only the wireless connection. Once again, A^2M's baseline case performance is the same as MobiDesk's. We then added the Planetlab-based indirection network to carry the

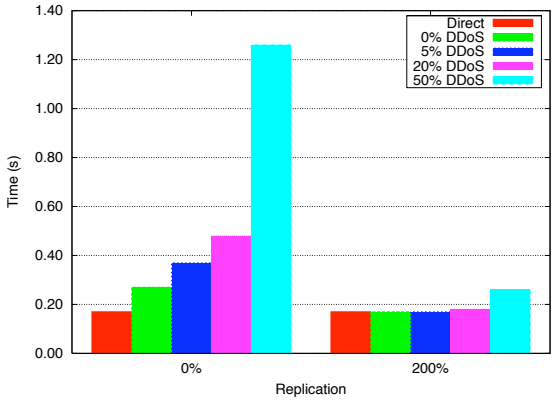


Figure 8.9: Interactive performance for the echo test. Even without replication and with attacks affecting up to 20% of the IBN nodes, the client's end-to-end latency increases only by a factor of 2.5 when compared to the direct, non-protected case. With packet replication, latency rises only after 50% of the nodes become unresponsive.

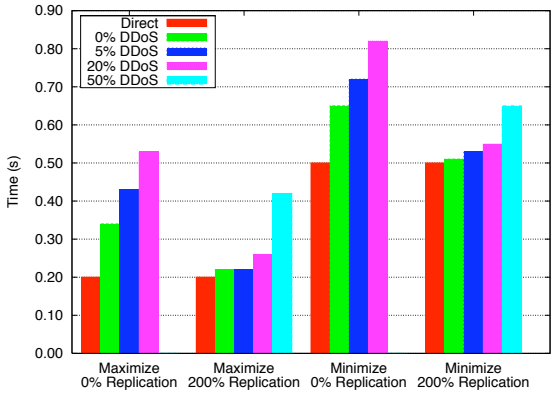


Figure 8.10: Interactive performance for minimize/maximize window test. Without replication and for attacks affecting up to 20% of the IBN nodes, the client's end-to-end latency increases only by a factor of 2. (Over 20%, the tests could not complete.) With replication, attacks on up to 50% of the IBN nodes had no impact on latency.

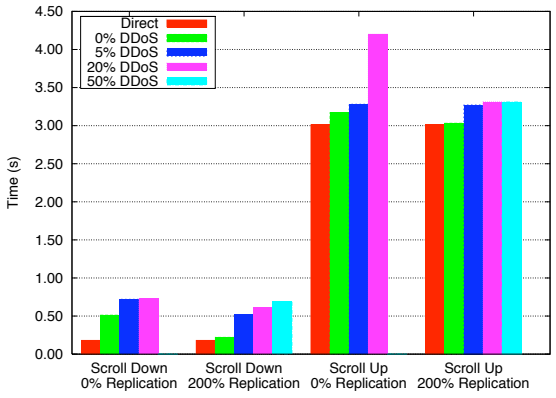


Figure 8.11: Interactive performance for the scroll test. With packet replication, latency is close to a direct connection even when under severe attack. Without packet replication, latency increases by less than a factor of 3, *vs.* the direct case.

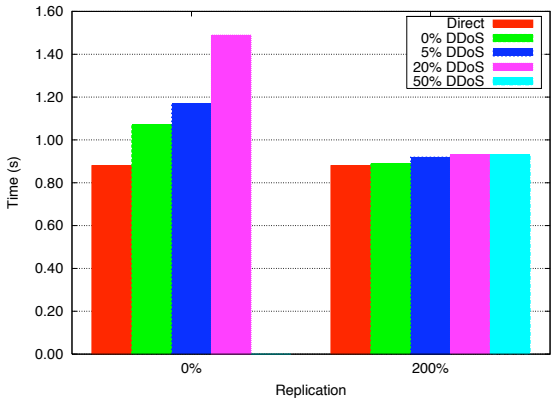


Figure 8.12: Interactive performance for the move window test. Latency increases significantly only after 20% of IBN nodes are attacked, with no replication. With 200% packet replication, latency does not increase even for attack intensities of 50%.

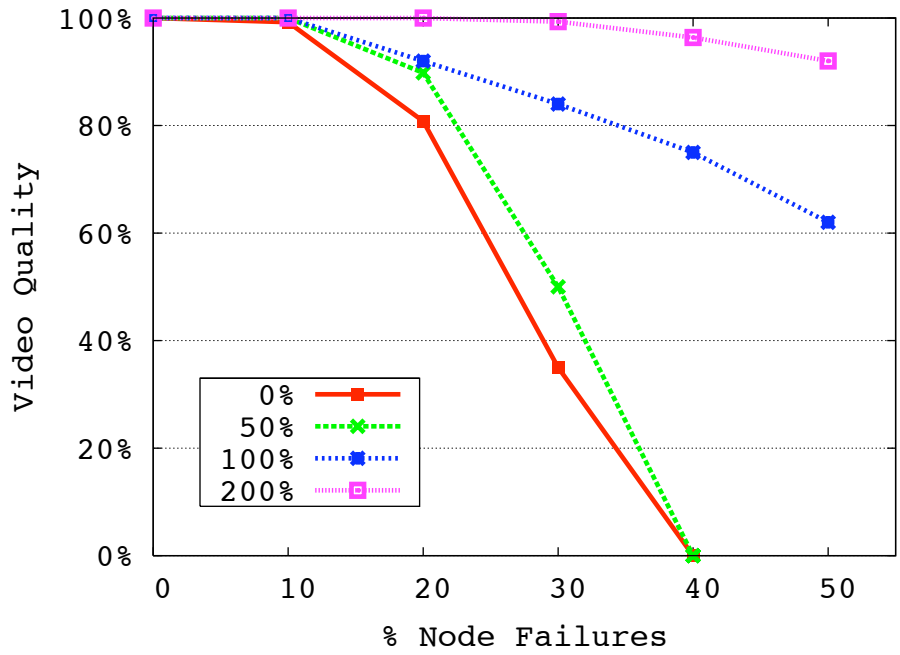


Figure 8.13: Video quality under DDoS attack in the wireless scenario. Video quality suffers only after a large portion of the indirection nodes are attacked, allowing correct system operation even when 40% of the nodes have been attacked, for 200% packet replication.

upstream packets from the wireless client back to the server, while continuing to use the direct path for the downstream traffic. Figure 8.13 shows video quality as a function of node failure in the indirection network and for different percentages of packet replication, while using a wireless network. We see the same trend as in our wired-network attack scenario: video quality suffers only after a large portion of the IBN nodes are attacked, reaching 40% node failure for 200% replication.

All the previous results demonstrate that A^2M can significantly increase the attack resilience of the desktop hosting infrastructure with minimum overhead, providing web browsing latency comparable to traditional (and vulnerable) direct LAN connections, and perfect video quality even in the presence of severe attacks on the access infrastructure.

8.6 Conclusions

We presented A^{2}M, an attack-resilient and latency efficient desktop hosting infrastructure based on a single-hop indirection network. A^{2}M exploits multi-path routing, packet replication, and the high asymmetry inherent to interactive display traffic, to assure access to remote desktop sessions, even in the presence of high-volume DoS attacks. Contrary to the current DoS protection mechanisms, our system guarantess both availability and uninterrupted connectivity to the end server providing a truly secure end-to-end connectivity model. Furthermore, in a departure from traditional client-server systems, A^{2}M provides an asymmetric client-server connection consisting of an indirected client-to-server multi-path, and a direct server-to-client connection. A^{2}M's indirection-based overlay acts both as a first-level distributed firewall and as a redirection mechanism for performance-critical user input-events going from the client device to the hosting servers. In turn, the direct server-to-client connection provides quick delivery of display updates, to provide quick response time and good user experience.

We implemented an A^{2}M prototype in Linux and evaluated its performance on PlanetLab. Our experimental results show that, as opposed to existing DDoS protection mechanisms, A^{2}M has minimum latency overhead and can provide good interactive performance for web, video, and general interactive applications. Furthermore, our experimental results show that A^{2}M significantly increases the attack resilience of the hosting infrastructure, being able to provide perfect video playback and low-latency web browsing and GUI interactions even in the presence of large attacks on the infrastructure. A^{2}M maintains 100% video quality in a number of remote video display scenarios, despite the use of overlay routing. Furthermore, end-to-end latency increases by less than 5% even when 40% of nodes have been rendered unusable by an attacker. Given its performance and resilience to DoS attacks, A^{2}M represents a step forward towards realizing the vision of computer utilities that provide ubiquitous, secure, and assured-access desktop computing.

Chapter 9

Conclusion

9.1 Summary

Our initial exploration of this subject area resulted in our development of PROOFS [158, 159], a system that helps sustain the availability of online objects in the face of Internet *flash crowds*. Flash crowds represent a sudden, unpredicted increase in demand of a web page or any other online object's popularity, resulting in denial of service (DoS) experienced by end-users. PROOFS is a simple, lightweight peer-to-peer (P2P) system that uses randomized overlay construction and randomized, scoped searches to efficiently locate and deliver objects under heavy demand to all users that request them. The distributed nature of PROOFS takes advantage of the user's bandwidth and transforms them into servers for the popular objects that they have already received. Our approach does not depend on any network-level or server-side modifications. Unlike Bittorrent [21] and other recent P2P distribution networks that focus solely on object dissemination, PROOFS offers distributed mechanisms to both locate and retrieve objects. Thus, it does not depend on the existence of a centralized, per-object tracker to coordinate the download process among nodes. Maintaining a tracker for each web object requires an amount of resources that makes Bittorrent impractical for distributing a large number of small files such as web pages. For small files, locating the file

is more crucial than its quick retrieval. In addition, unlike paid overlay distribution services such as Akamai, our system can scale without requiring extra resources or network presence and can operate in an extremely adverse environments. Moreover, we do not impose any economic burden on any of the protected sites, thereby providing protection for even small sites.

However, PROOFS cannot be used when trusted or dynamic content (*e.g.,* real-time communications) is involved. Furthermore, PROOFS cannot guarantee that a client will be able to obtain the latest version of the requested object without flooding the network. To improve network performance and to allow the protection of web services that offer dynamic or trusted content, we introduced WebSOS [156, 117]. WebSOS was the first overlay-based architecture that enabled the protection of web servers that offer dynamic and trusted content against DoS attacks without any network requirements or modifications. Its design was based on the theoretical foundations laid by the Secure Overlay Services (SOS)[91] work, but is geared toward web services. By filtering and dropping all connections toward the web server other than the ones originating from a very small set of overlay nodes, we managed to guarantee service availability even when the server was the victim of a DoS attack. One of our main contributions is complete transparency to the underlying network; creating an overlay of firewalls guarantees access to a web server for a large number of *previously unknown* users without requiring pre-existing trust relationships between users and the system. This was the first time that reverse Graphical Turing Tests were employed to thwart denial of service attacks by differentiating humans from malicious automated scripts (often referred to as "bots"). Our design exploits GTTs as an easily recognizable form of Reverse Turing Tests (RTTs) that enables us to detect automated scripts and prevent them from abusing our system without causing significant delays to the legitimate uses. In practice, the specific type of RTT is not crucial; almost any RTT will suffice to protect our system. For example, we could employ RTTs that use speech [96] or facial features [144].

With the MOVE [155] architecture, we further increased the attack and failure resilience

of Internet services. When under attack, MOVE employs a lightweight process migration mechanism to relocate the service to another site informing the overlay of the new location. The migration is transparent to legitimate users because they use the overlay to reach the service. However, attackers are not aware of the new service location, and thus they cannot disrupt the service. MOVE is the first truly End-to-End protection mechanism that can be deployed without relying on any network-level support including firewalls and capacity reservation mechanisms. We deployed our prototypes on PlanetLab, a distributed network of nodes with global presence. We quickly realized that protection comes at a price: there was a significant end-to-end latency increase (by a factor of 2 of more) that was caused by packet indirection inside the overlay.

Although appropriate for web applications, WebSOS and MOVE cannot effectively protect low-latency, time-critical, or interactive services, including Thin Clients and Voice over IP (VoIP). Furthermore, we showed that all connection-oriented overlay-based systems that use connection state (*i.e.,* all such mechanisms to date) are vulnerable to a new class of targeted DoS attacks. Our attacks exploit the fact that clients communicate with the service through a single, static connection with the overlay network. By severing that connection (*e.g.,* through a targeted DoS attack), the user is forced to reconnect and re-authenticate multiple times over a short period of time, rendering the overall architecture impractical. To address all the latency issues and the new vulnerabilities, we proposed a *stateless, spread-spectrum-like* protocol that enhances the communication of the client with the overlay [154]. This protocol allows the client to communicate with the end-server using multiple disjoint paths through the overlay, treating the overlay as a whole as a stateless pipe.

Our solution had to be two-pronged: we wanted to remove connection state from all the overlay nodes and distribute the connection responsibility evenly among the overlay nodes. Each connection had to be treated as a contract between the client and the entire overlay, not just a single node (access point). To that end, we proposed a novel authentication and key agreement protocol. This protocol requires very little persistent state in each overlay node

(4 to 8 bytes per client). In addition, we removed the single connection point dependency: we spread traffic to overlay nodes using pseudorandom packet dispersion among all possible paths. This Multiple Input Multiple Output (MIMO) communication exploits all the paths provided by the underlying network regardless of their characteristics. The term MIMO was adopted from wireless communication systems with multiple transmission and reception antennas. Of course, in the Internet there are no antennas – overlay nodes become our "antennas", reflecting packets which generate a natural path diversity. Properly placed overlay nodes can harness the benefits of the MIMO communication model, significantly increasing the aggregate capacity and network reliability. The striking difference between this approach and the one used in WebSOS, MOVE, and other overlay-based networks is the distribution of client traffic to *all* overlay nodes (and paths) instead of choosing individual paths per client. By taking advantage of multiple path and by aggressively using packet replication or packet coding including Forward Error Correcting codes (FEC), we eliminate the latency overhead imposed by the overlay indirection. At the same time, we significantly elevate the resilience of the system against failures and attacks that target the communication network, the overlay nodes, or the end server itself.

Furthermore, we demonstrate how we can use packet spreading and replication techniques while being TCP-friendly: when we employ UDP encapsulated TCP packets, we use ACKs with link identification encoded in the TCP-options field. For regular TCP connections, we make use of the inherent TCP-congestion control. Our experiments show that our system can take advantage of the underlying multi-path link capacity without starving other flows over shared links. Although being TCP-friendly can be exploited for TCP flows, we demonstrate that there is no significant throughput or latency degradation for UDP. Moreover, when there is either a static or slow-changing attack against the communication substrate, enabling weighted diffusion can boost our performance and resilience to that of the available aggregate link capacity.

To protect the proposed overlay architecture against insider attacks, we present a set of

algorithms that attempts to detect, identify, and isolate Denial of Service and flooding attacks initiated from inside the indirection infrastructure. Related research, such as SOS [91], used structured, Distributed Hash Table (DHT) systems to protect against DoS attacks from outside attackers, but was itself susceptible to insider attacks by compromised nodes. Our approach can successfully protect overlay networks that exhibit well-defined properties due to their *structure or topology* against non-conforming (abnormal) behavior of participating nodes. We show that we can thwart insider flooding attacks for almost all DHT-based Overlays and even some randomized ones where the set of neighbors is fixed and well-known. Furthermore, our method can protect simple one-hop indirection infrastructures like the ones used in our protection overlay. In addition, even though our attack detection algorithms are applicable for a general overlay formation that serves objects of variable popularity, they are much more effective when that objects exhibits the same popularity. This is generally the case for overlays that are employed for protection using traffic spreading because the clients' packets are homogeneously spread on the indirection nodes.

Our detection algorithms operate on aggregate packet flows, thus avoiding exorbitant storage and processing requirements and maintaining scalability as the number of overlay participants grows. Statistical detection on aggregate flows is not something entirely new [31, 111], but we are the first to consider these methods in an overlay network setting that takes into consideration the neighbor-structure of P2P systems. Upon detection, we invoke a Pushback-like protocol to notify and prompt into action (e.g., throttle the traffic) the predecessor node: the node from which the offending traffic arrives. Recursive applications of the protocol can identify and isolate the offending node(s).

We measure our mechanism's ability to detect attackers via simulation of web traces from IRCache (`http://www.ircache.net`) served by a DHT overlay. The results show that our system can detect a simple attacker whose attack traffic deviates by as little as 5% from average traffic. We also demonstrate the resilience of our mechanism against coordinated distributed flooding attacks that involve up to 15% of overlay nodes. In addition,

we verify that our detection algorithms work well; they produce a low false positive rate ($< 2\%$) when used in a system that serves normal web traffic. The false positive rate is a measure of the detector's accuracy and does not necessarily measure the rate of incorrect actions that may be executed by some mitigation strategy. Mitigating actions are required only when the protected server is under heavy load, invoking load balancing to evenly spread service among clients.

When we insert attack traffic into these traces, our approach identifies the offending node(s) and squelches the attacks. The detection and containment mechanisms presented show that overlays can protect themselves from insider DoS attacks and eliminates an important roadblock to their deployment.

To evaluate the performance our protection architecture in practice, we introduce A^2M, a secure and attack-resilient desktop computing hosting infrastructure. A^2M combines a stateless and secure communication protocol, a single-hop Indirection-based network (IBN) and a remote display architecture to provide mobile users with continuous access to their desktop computing sessions. Our architecture protects both the hosting infrastructure and the client's connections against a wide range of service disruption attacks. Unlike any other DoS protection system, A^2M takes advantage of its low-latency remote display mechanisms and asymmetric traffic characteristics by using multi-path routing to send a small number of copies of each packet transmitted from client to server. This packet replication through different paths diversifies the client-server communication, boosts system resilience and reduces end-to-end latency. Our analysis and experimental results on PlanetLab demonstrate that A^2M significantly increases the hosting infrastructure's attack resilience. Using current ISP bandwidth data, we show that when we filter traffic only at a single point of presence (POP), we can protect against attacks involving up to $150,000$ attackers. Our protection incurs no performance degradation for multimedia and web applications and basic GUI interactions even when up to 30% and 50%, respectively, of indirection nodes become unresponsive. Similarly, a network-video application was able to deliver perfect playback

over A^2M even when 50% of the overlay nodes were similarly disabled. When there are multiple filtering points, an Akamai-sized overlay can withstand attacks involving over 1.3M attacking hosts while providing uninterrupted end-to-end connectivity. This number is computed when assuming a commodity PC with a mere 50 Mbps packet forwarding ability. This is a very conservative estimate since The Click software router with commodity hardware [97] claims a switching capacity of 435,000 64-byte packets, or 222 Mbps. Allowing packet replication for an overlay network of $2,500$ nodes, an attacker needs to gain access to and coordinate a network of $1,375,000$ or $6,125,000$ bots, assuming 50 Mbps and 222 Mbps switching capacity, respectively.

Furthermore, when we enable packet replication, the system can resist attacks that render up to 40% of the nodes inoperable. Not surprisingly, our experiments on PlanetLab demonstrate that in many cases end-to-end latency *decreases* when packet replication is used, with a worst-case increase by a factor of 2.5. Similarly, our system imposes less than 15% performance degradation in the end-to-end throughput, even when subjected to a large DDoS attack. Thus, our approach offers an attractive solution against congestion-based denial of service attacks in most environments, as it does not require modifications to clients, servers, protocols, or routers both in terms of hardware and in terms of existing software.

9.2 Lessons Learned

Network-level DoS attacks remain a real threat to the availability of Internet services [122, 121, 74, 110]. One prevalent argument holds that DoS is really a quality of service (QoS) problem that should be addressed by placing intelligence in the network core. Such an architecture, however, is contrary to the current end-to-end Internet design. In this thesis, we implemented and evaluated a fully end-to-end protection architecture that does require any network support. We demonstrate that if we can enforce policy decisions at the edge of the network using overlays (or even edge routers) then there is no need to have any intelligence

or QoS provisions within the network core. We can continue to treat the network as an efficient but unreliable transport mechanism.

However, we demonstrate that for such a protection architecture to be practical, it is necessary to employ end-to-end communication protocols that are resilient to failures and attacks against the underlying infrastructure. To that end, we introduce novel scalable distributed communication algorithms that can defend against large-scale attacks, including insider attacks. Even if the adversary can render a large fraction of the available links inoperable, our system provides end-to-end connectivity without significant performance loss. In addition, we show that we can discover and utilize the hidden capacity of the existing Internet infrastructure to provide security without significant performance overhead. Our techniques work even for time-critical and interactive applications.

9.3 Future Directions

The work presented in this thesis gives rise to a number of potential future directions including the following:

- Explore the effect of multi-path and information replication in networks that naturally exhibit link or packet loss such as mobile ad-hoc and sensor networks.

- Create Accurate Profiles of usage for each Internet service we try to protect. Deciding the level and detail of each profile and the actions that need to be taken when a profile is violated is an interesting open problem.

- Network-based DoS defenses have become essentially a reactive mechanism that try to address a problem that stems from the clients' host. Providing resource efficient host-based defenses would help us pro-actively protect against DoS.

- Employ our DoS architecture for quality of service assurance in dense wireless networks. Indeed, instead of using a single wireless access point to route data, it

seems to be beneficial both for the communication latency and bandwidth to associate the sender to multiple wireless access points. Quantifying the advantages and caveats of such communication including the packet scheduling algorithm is a future direction we would like to investigate.

Chapter 10

Bibliography

[1] Ircache web trace repository, *www.ircache.net*.

[2] IEEE Draft P802.1X/D11: Standard for Port based Network Access Control, March 2001.

[3] M. Abadi, M. Burrow, M. Manasse, and T. Wobber. Moderately Hard, Memory-bound Functions. In *Proceedings of the ISOC Symposium on Network and Distributed Systems Security (SNDSS)*, February 2003.

[4] W. Aiello, S. M. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A. D. Keromytis, and O. Reingold. Efficient, DoS Resistant, Secure Key Exchange for Internet Protocols. In *Proceedings of the* 9^{th} *ACM Computer and Communications Security (CCS) Conference*, pages 48–58, November 2002.

[5] L. Amini, H. Schulzrinne, and A. Lazar. Observations from Router-level Internet Traces. In *DIMACS Workshop on Internet and WWW Measurement, Mapping and Modeling*, February 2002.

[6] D. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proceedings of the Eigth Workshop on Hot Topics in Operating Systems*, May 2001.

[7] D. G. Andersen, H. Balakrishnan, M. Frans Kaashoek, and R. N. Rao. Improving Web Availability for Clients with MONET. In *Proceedings of the* 2^{nd} *Symposium on Networked Systems Design and Implementation (NSDI)*, May 2005.

[8] D. G. Andersen, A. C. Snoeren, and H. Balakrishnan. Best-Path vs. Multi-Path Overlay Routing. In *Proceedings of the Internet Measurement Conferencee*, October 2003.

[9] David G. Andersen. Mayday: Distributed Filtering for Internet Services. In *Proceedings of the* 4^{th} *USENIX Symposium on Internet Technologies and Systems (USITS)*, March 2003.

[10] T. Anderson, T. Roscoe, and D. Wetherall. Preventing Internet Denial-of-Service with Capabilities. In *Proceedings of the* 2^{nd} *Workshop on Hot Topics in Networks (HotNets-II)*, November 2003.

[11] K. Arguraki and D. Cheriton. Active Internet Traffic Filtering: Real-Time Response to Denial-of-Service Attacks. In *Proceedings of the USENIX Technical Conference*, pages 135–148, April 2005.

[12] E. Athanasopoulos, K. G. Anagnostakis, and E. P. Markatos. Misuing Unstructured P2P Systems to Perform DoS Attacks: The Network That Never Forgets. In *Proceedings of te* 4^{th} *Applied Cryptography and Network Security Conference (ACNS)*, pages 130–145, June 2006.

[13] T. Aura and P. Nikander. Stateless connections. In *Proceedings of International Conferenec on Information and Commun ications Security (ICICS), Lecture Notes in Computer Science volume 1334*, pages 87–97. Springer, November 1997.

[14] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson. The Internet Motion Sensor: A Distributed Blackhole Monitoring System. In *Proceedings of the ISOC Symposium on Network and Distributed Systems Security (SNDSS)*, pages 167–179, February 2005.

[15] Ricardo Baratto, Shaya Potter, Gong Su, and Jason Nieh. MobiDesk: Mobile Virtual Desktop Computing. In *Proceedings of the* 10^{th} *Annual ACM International Conference on Mobile Computing and Networking (MobiCom)*, September 2004.

[16] J. Bellardo and S. Savage. 802.11 Denial-of-Service Attacks: Real Vulnerabilities and Practical Solutions. In *Proceedings of the 12th USENIX Security Symposium*, pages 15–28, August 2003.

[17] S. M. Bellovin. Distributed Firewalls. *;login: magazine, special issue on security*, pages 37–39, November 1999.

[18] M. C. Benvenuto and A. D. Keromytis. EasyVPN: IPsec Remote Access Made Easy. In *Proceedings of the 17th USENIX Systems Administration Conference (LISA)*, pages 87–93, October 2003.

[19] D. Bernstein. SYN Cookies. http://cr.yp.to/syncookies.html, 1996.

[20] R. Beverly and S. Bauer. The Spoofer Project: Inferring the Extent of Source Address Filtering on the Internet. In *Proceedings of the Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, pages 53–59, July 2005.

[21] Bittorrent. http://www.bittorrent.com.

[22] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway. UMAC: Fast and Secure Message Authentication. *Lecture Notes in Computer Science*, 1666:216–233, 1999.

[23] W. J. Blackert, D. M. Gregg, A. K. Castner, E. M. Kyle, R. L. Hom, and R. M. Jokerst. Analyzing Interaction Between Distributed Denial of Service Attacks and Mitigation Technologies. In *Proceedings of DISCEX III*, pages 26–36, April 2003.

[24] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The KeyNote Trust Management System Version 2. RFC 2704, September 1999.

[25] M. Blaze, J. Ioannidis, S. Ioannidis, A. D. Keromytis, P. Nikander, and V. Prevelakis. TAPI: Transactions for Accessing Public Infrastructure. In *Proceedings of the 8th IFIP Personal Wireless Communications (PWC) Conference*, pages 90–10, September 2003.

[26] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman. An XOR-Based Erasure-Resilient Coding Scheme. Technical report, International Computer Sciences Institute, ICSI TR-95-048, August 1995.

[27] M. Casado, A. Akella, P. Cao, N. Provos, and S. Shenker. Cookies Along Trust-Boundaries (CAT): Accurate and Deployable Flood Protection. In *Proceedings of the Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, pages 15–22, July 2006.

[28] M. Castro, P. Drushel, A. Ganesh, A. Rowstron, and D. Wallach. Secure routing for structured peer-to-peer overlay networks. In *OSDI*, 2002.

[29] CCITT. *X.509: The Directory Authentication Framework*. International Telecommunications Union, Geneva, 1989.

[30] M. C. Chan, Ee-Chien Chang, L. Lu, and P. S. Ngiam. Effect of Malicious Synchronization. In *Proceedings of te* 4^{th} *Applied Cryptography and Network Security Conference (ACNS)*, pages 114–129, June 2006.

[31] Randy H. Katz Chen-Nee Chuah, Lakshminarayanan Subramanian. DCAP: Detecting Misbehaving Flows via Collaborative Aggregate Policing. *SIGCOMM Computer Communication Review*, 33(5), October 2003.

[32] S. T. Chou, A. Stavrou, J. Ioannidis, and A. D. Keromytis. gore: Routing-Assisted Defense Against DDoS Attacks. In *Proceedings of the* 8^{th} *Information Security Conference (ISC)*, September 2005.

[33] R. Clayton. The Rising Tide: DDoS from Defective Designs and Defaults. In *Proceedings of the Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, pages 1–6, July 2006.

[34] Ariel Cohen, Sampath Rangarajan, and J. Hamilton Slye. On the Performance of TCP Splicing for URL-Aware Redirection. In *USENIX Symposium on Internet Technologies and Systems*, 1999.

[35] M. Collins and M. Reiter. An empirical analysis of target-resident DoS filters. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2004.

[36] D. Cook. Analysis of Routing Algorithms for Secure Overlay Service. Computer Science Department Technical Report CUCS-010-02, Columbia University, April 2002.

[37] D. L. Cook, W. G. Morein, A. D. Keromytis, V. Misra, and D. Rubenstein. WebSOS: Protecting Web Servers From DDoS Attacks. In *Proceedings of the* 11^{th} *IEEE International Conference on Networks (ICON)*, pages 455–460, September 2003.

[38] Manuel Costa, Miguel Castro, and Tim Harris. Securing software by enforcing data-flow integrity. In *OSDI '06 Proceeding of the 7th USENIX Symposium on Operating Systems Design and Implementation*, 2006.

[39] Manuel Costa, Jon Crowcroft, Miguel Castro, Antony Rowstron, Lidong Zhou, Lintao Zhang, and Paul Barham. Vigilante: end-to-end containment of internet worms. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 133–147, New York, NY, USA, 2005. ACM Press.

[40] S. A. Crosby and D. S. Wallach. Denial of Service via Algorithmic Complexity Attacks. In *Proceedings of the 12th USENIX Security Symposium*, pages 29–44, August 2003.

[41] F. Dabek, F. Kaashoek, R. Morris, D. Karger, and I. Stoica. Wide-area cooperative storage with cfs. In *Proceedings of ACM SOSP*, October 2001.

[42] D Dagon. The Botnet Trackers. *Washington Post*, February 2006. http://www.washingtonpost.com/wp-dyn/content/article/2006/02/16/AR2006021601388.html.

[43] N. Daswani and H. Garcia-Molina. Query-Flood DoS Attacks in Gnutella. In *Proceedings of the* 9^{th} *ACM Conference on Computer and Communications Security (CCS)*, pages 181–192, November 2002.

[44] D. Dean, M. Franklin, and A. Stubblefield. An Algebraic Approach to IP Traceback. In *Proceedings of the ISOC Symposium on Network and Distributed System Security (SNDSS)*, pages 3–12, February 2001.

[45] D. Dean and A. Stubblefield. Using Client Puzzles to Protect TLS. In *Proceedings of the 10th USENIX UNIX Security Symposium*, August 2001.

[46] T. Diament, H. K. Lee, A. D. Keromytis, and M. Yung. The Dual Receiver Cryptogram and Its Applications. In *Proceedings of the* 11^{th} *ACM Conference on Computer and Communications Security (CCS)*, October 2004.

[47] T. Dierks and C. Allen. The TLS protocol version 1.0. RFC 2246, January 1999.

[48] S. Dietrich, N. Long, and D. Dittrich. Analyzing Distributed Denial of Service Tools: The Shaft Case. In *Proceedings of USENIX LISA XIV*, December 2000.

[49] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the* 13^{th} *USENIX Security Symposium*, pages 303–319, August 2004.

[50] G. Dommety. Key and Sequence Number Extensions to GRE. RFC 2890, September 2000.

[51] Sujata Doshi, Fabian Monrose, and Aviel D. Rubin. Efficient Memory Bound Puzzles Using Pattern Databased. In *Proceedings of te* 4^{th} *Applied Cryptography and Network Security Conference (ACNS)*, pages 98–113, June 2006.

[52] N. Duffield and B. Krishnamurthy. Stress Testing Traffic to Infer Its Legitimacy. In *Proceedings of the Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, pages 61–67, July 2005.

[53] D. Estrin, J. Mogul, and G. Tsudik. VISA Protocols for Controlling Inter-Organizational Datagram Flow. *IEEE Journal on Selected Areas in Communications*, May 1989.

[54] H. Farhat. Protecting TCP Services From Denial of Service Attacks. In *Proceedings of the* 1^{st} *Workshop on Large-Scale Attack Defence (LSAD)*, pages 155–160, September 2006.

[55] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina. Generic Routing Encapsulation (GRE). RFC 2784, March 2000.

[56] Global environment for network innovations (geni). http://www.geni.net.

[57] V. D. Gligor. Guaranteeing Access in Spite of Distributed Service-Flooding Attacks. In *Proceedings of the Cambridge Security Protocols Workshop*, April 2003.

[58] The Gnutella Protocol Specification v0.4, revision 1.2. Available from http://gnutella.wego.com.

[59] M. T. Goodrich. Efficient Packet Marking for Large-Scale IP Traceback. In *Proceedings of the* 9^{th} *ACM Conference on Computer and Communications Security (CCS)*, pages 117–126, November 2002.

[60] A. Greenhalgh and M. Handley ad F. Huici. Using Routing and Tunneling to Combat DoS Attacks. In *Proceedings of the Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, pages 1–7, July 2005.

[61] M. Guirguis, A. Bestavros, and I. Matta. Exploiting the Transients of Adaptation for RoQ attakcs on Internet Resources. In *Proceedings of the International Conference on Network Protocols (ICNP)*, pages 184–195, October 2004.

[62] K. P. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, and D. Wetherall. Improving the Reliability of Internet Paths with One-hop Source Routing. In *Proceedings of the* 6^{th} *Symposium on Operating Systems Design & Implementation (OSDI)*, December 2004.

[63] Carl A. Gunter, Sanjeev Khanna, Kaijun Tan, and Santosh Venkatesh. DoS Protection for Reliably Authenticated Broadcast. In *Proceedings of the ISOC Symposium on Network and Distributed System Security (SNDSS)*, pages 17–36, February 2004.

[64] N. Haller, C. Metz, P. Nesser, and M. Straw. A One-Time Password System. RFC 2289, IETF, February 1998.

[65] M. Handley, S. Floyd, J. Padhye, and J. Widmer. Rfc 3448: Tcp friendly rate control (tfrc): Protocol specification, 2003.

[66] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). Request for Comments (Proposed Standard) 2409, Internet Engineering Task Force, November 1998.

[67] C. He and J. C. Mitchell. Security Analysis and Improvements for IEEE 802.11i. In *Proceedings of the ISOC Symposium on Network and Distributed Systems Security (SNDSS)*, pages 97–115, February 2005.

[68] L.T. Heberlein and M. Bishop. Attack Class: Address Spoofing. In *Proceedings of the* 19^{th} *National Information Systems Security Conference*, pages 371–377, October 1996.

[69] Shouichi Hirose and Kanta Matsuura. Enhancing the resistance of a provably secure key agreement protocol to a denial-of-service attack. In *Proceedings of the 2nd International Conference on Information an d Communication Security (ICICS)*, pages 169–182, November 1999.

[70] K. Houle, G. Weaver, N. Long, and R. Thomas. Trends in Denial of Service Attack Technology. http://www.cert.org/archive/pdf/DoS_trends.pdf, October 2001.

[71] Lingxuan Hu and DAvid Evans. Using Directional Antennas to Prevent Wormhole Attacks. In *Proceedings of the ISOC Symposium on Network and Distributed System Security (SNDSS)*, pages 131–141, February 2004.

[72] G. Hulme. Extortion online. *Information Week*, September 13, 2004.

[73] A. Hussain, J. Heidemann, and C. Papadopoulos. A Framework for Classifying Denial of Service Attacks. In *Proceedings of ACM SIGCOMM*, pages 99–110, August 2003.

[74] Nicholas Ianelli and Aaron Hackworth. Botnets as a Vehicle for Online Crime. http://www.cert.org/archive/pdf/Botnets.pdf, December 2005.

[75] The Cambridge-MIT Institute. Dos-resistant internet working group meetings, February 2005.

[76] J. Ioannidis and S. M. Bellovin. Implementing Pushback: Router-Based Defense Against DDoS Attacks. In *Proceedings of the ISOC Symposium on Network and Distributed System Security (SNDSS)*, February 2002.

[77] John Ioannidis, Sotiris Ioannidis, Angelos D. Keromytis, and Vasillis Prevelakis. Fileteller: Paying and Getting Paid for File Storage. In *Proceeding of Financial Cryptography (FC) Conference*, pages 282–299, March 2002.

[78] S. Ioannidis, A.D. Keromytis, S.M. Bellovin, and J.M. Smith. Implementing a Distributed Firewall. In *Proceedings of Computer and Communications Security (CCS)*, pages 190–199, November 2000.

[79] M. Jakobsson and A. Juels. Proofs of Work and Bread Pudding Protocols. In *Proceedings of the IFIP TC6 and TC11 Joint Conference on Communications and Multimedia Security*, September 1999.

[80] P. Janson, G. Tsudik, and M. Yung. Scalability and flexibility in authentication services: the KryptoKnight appro ach. In *Proceedings of IEEE INFOCOM*, pages 725–736, April 1997.

[81] C. Jin, H. Wang, and K. G. Shin. Hop-Count Filtering: An Effective Defense Against Spoofed DoS Traffic. In *Proceedings of the* 10^{th} *ACM International Conference on Computer and Communications Security (CCS)*, pages 30–41, October 2003.

[82] A. Juels and J. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proceedings of the ISOC Symposium on Network and Distributed Systems Security (SNDSS)*, pages 151–165, February 1999.

[83] A. Kaella, J. Pang, and A. Shaikh. A Comparison of Overlay Routing and Multihoming Route Control. In *Proceedings of ACM SIGCOMM*, pages 93–106, August/September 2004.

[84] S. Kandula, D. Katabi, M. Jacob, and A. Berger. Botz-4-Sale: Surviving Organized DDoS Attacks That Mimic Flash Crowds. In *Proceedings of the* 2^{nd} *Symposium on Networked Systems Design and Implementation (NSDI)*, May 2005.

[85] D. Karger, E. Lehman, F. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 654–663, May 1997.

[86] Frank Kargl, Jorn Maier, and Michael Weber. Protecting Web Servers From Distributed Denial of Service Attacks. In *Proceedings of the W3C World Wide Web Conference (WWW)*, pages 514–524, 2001.

[87] Chris Karlof, Naveen Sastry, Yaping Li, Adrian Perrig, and J. D. Tygar. Distillation Codes and Applications to DoS Resistant Multicast Authentication. In *Proceedings of the ISOC Symposium on Network and Distributed System Security (SNDSS)*, pages 37–56, February 2004.

[88] P. Karn and W. Simpson. Photuris: Session-key management protocol. Request for Comments (Experimental) 2522, Internet Engineering Task Force, March 1999.

[89] C. Kaufman, R. Perlman, and B. Sommerfeld. DoS Protection for UDP-Based Protocols. In *Proceedings of the* 10^{th} *ACM International Conference on Computer and Communications Security (CCS)*, pages 2–7, October 2003.

[90] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401, November 1998.

[91] A. D. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure Overlay Services. In *Proceedings of ACM SIGCOMM*, pages 61–72, August 2002.

[92] A. D. Keromytis, V. Misra, and D. Rubenstein. SOS: An Architecture For Mitigating DDoS Attacks. *IEEE Journal on Selected Areas of Communications (JSAC)*, 33(3):413–426, January 2004.

[93] A. D. Keromytis, J. L. Wright, and T. de Raadt. The Design of the OpenBSD Cryptographic Framework. In *Proceedings of the USENIX Annual Technical Conference*, pages 181–196, June 2003.

[94] S. M. Khattab, C. Sangpachatanaruk, D. Moss, R. Melhem, and T. Znati. Roaming Honeypots for Mitigating Service-Level Denial-of-Service Attacks. In *Proceedings of the* 24^{th} *International Conference on Distributed Computing Systems (ICDCS)*, pages 238–337, March 2004.

[95] Yongdae Kim, Adrian Perrig, and Gene Tsudik. Simple and fault-tolerant key agreement for dynamic collaborative groups. In *Proceedings of the* 7^{th} *ACM Conference on Computer and Communications Security (CCS)*, pages 235–244, November 2000.

[96] G. Kochanski, D. Lopresti, and C. Shih. A reverse turing test using speech. In *In Proceedings of the International Conferences on Spoken Language Processing Denver, Colorado (2002)*, pages 1357–1360, 2002.

[97] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM Transactions on Computer Systems (ToCS)*, 18(3):263–297, August 2000.

[98] H. Krawczyk. Invited Talk. SIGMA: the SIGn-and-MAc Approach to Authenticated Diffie Hellman and its Use in the IKE Protocols. In *Proceedings of the CRYPTO Conference*, August 2002.

[99] A. Kuzmanovic and E. W. Knightly. Low-Rate TCP-Targeted Denial of Service Attacks. In *Proceedings of ACM SIGCOMM*, pages 75–86, August 2003.

[100] J. Vollbrecht L. Blunk. PPP Extensible Authentication Protocol (EAP). RFC 2284, IETF, March 1998.

[101] Karthik Lakshminarayanan, Daniel Adkins, Adrian Perrig, and Ion Stoica. Taming IP Packet Flooding Attacks. In *Proceedings of the* 2^{nd} *Workshop on Hot Topics in Networks (HotNets-II)*, November 2003.

[102] J. Leiwo, P. Nikander, and T. Aura. Towards network denial of service resistant protocols. In *Proceedings of the 15th International Information Security Confer ence (IFIP/SEC)*, August 2000.

[103] J. Lemmon. Resisting SYN-flood DoS Attacks with a SYN Cache. In *Proceedings of the USENIX BSD Conference (BSDCon)*, February 2001.

[104] J. Li, M. Sung, J. Xu, and L. Li. Large-Scale IP Traceback in High-Speed Internet: Practical Techniques and Theoretical Foundation. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2004.

[105] X. Liu, X. Yang, D. Wetherall, and T. Anderson. Efficient and Secure Source Authentication with Packet Passports. In *Proceedings of the Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, pages 7–13, July 2006.

[106] X. Luo and R. K. C. Chang. On a New Class of Pulsing Denial-of-Service Attacks and the Defense. In *Proceedings of the ISOC Symposium on Network and Distributed Systems Security (SNDSS)*, pages 61–79, February 2005.

[107] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling High Bandwidth Aggregates in the Network. *ACM Computer Communication Review*, 32(3):62–73, July 2002.

[108] R. Mahajan and S. Floyd. Controlling High-bandwidth Flows at the Congested Router. In *Proceedings of the International Conference on Network Protocols (ICNP)*, pages 192–201, November 2001.

[109] Floyd S. Mahdavi J. TCP-friendly unicast rate-based flow control, 1997. Note sent to the end2end-interest mailing list.

[110] Z. M. Mao, V. Sekar, O. Spatscheck, J. van der Merwe, and R. Vasudevan. Analyzing Large DDoS Attacks Using Multiple Data Sources. In *Proceedings of the* 1^{st} *Workshop on Large-Scale Attack Defence (LSAD)*, pages 161–168, September 2006.

[111] A. Matrawy, P. C. van Oorschot, and A. Somayaji. Mitigating Network Denial-of-Service Through Diversity-Based Traffic Management. In *Proceedings of the* 3^{rd} *International Conference on Applied Cryptography and Network Security (ACNS)*, pages 104–121, June 2005.

[112] A.J. McAuley. Reliable Broadband Communication Using a Burst Erasure Correcting Code. In *Proceedings of SIGCOMM'90*, pages 297–306, Philadelphia, PA, September 1990.

[113] S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer. Kerberos Authentication and Authorization System. Technical report, MIT, December 1987.

[114] D. Milojicic, F. Douglis, and R. Wheeler. *Mobility: Processes, Computers, and Agents*. Addison Wesley Longman, February 1999.

[115] S. Miltchev, S. Ioannidis, and A. D. Keromytis. A Study of the Relative Costs of Network Security Protocols. In *Proceedings of the USENIX Annual Technical Conference, Freenix Track)*, pages 41–48, June 2002.

[116] D. Moore, G. Voelker, and S. Savage. Inferring Internet Denial-of-Service Activity. In *Proceedings of the* 10^{th} *USENIX Security Symposium*, pages 9–22, August 2001.

[117] W. G. Morein, A. Stavrou, D. L. Cook, A. D. Keromytis, V. Misra, and D. Rubenstein. Using Graphic Turing Tests to Counter Automated DDoS Attacks Against Web Servers. In *Proceedings of the* 10^{th} *ACM International Conference on Computer and Communications Security (CCS)*, pages 8–19, October 2003.

[118] Greg Mori and Jitendra Malik. Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA. In *Computer Vision and Pattern Recognition (CVPR)*, June 2003.

[119] Athicha Muthitacharoen, Benjie Chen, and David Mazières. A low-bandwidth network file system. In *Proceedings of the* 18^{th} *ACM Symposium on Operating Systems Principles (SOSP)*, pages 174–187, Chateau Lake Louise, Banff, Canada, October 2001.

[120] Erich M. Nahum, Marcel-Catalin Rosu, Srinivasan Seshan, and Jussara Almeida. The effects of wide-area conditions on WWW server performance. In *Proceedings of the ACM SIGMETRICS*, pages 257–267, June 2001.

[121] Arbor Networks. Worldwide ISP Security Report. http://www.arbor.net/downloads/Arbor_Worldwide_ISP_Security_Report.pdf, September 2005.

[122] Arbor Networks. Worldwide Infrastructure Security Report. http://www.tcb.net/wisp07.pdf, September 2007.

[123] Jason Nieh, S. Jae Yang, and Naomi Novik. Measuring Thin-Client Performance Using Slow-Motion Benchmarking. *ACM Transactions on Computer Systems (TOCS)*, 21(1):87–115, February 2003.

[124] P. Nikander. Authorization and charging in public wlans using freebsd and 802.1x. In *Proceedings of the Annual USENIX Technical Conference, Freenix Track*, June 2002.

[125] Committee on Research Horizons in Networking. *Looking Over the Fence at Networks: A Neighbor's View of Networking Research*. National Academy Press, Washington D.C., 2001.

[126] R. Oppliger. Protecting key exchange and management protocols against resource c logging attacks. In *Proceedings of the IFIP TC6 and TC11 Joint Working Conference on Communications and Multimedia Security (CMS)*, pages 163–175, September 1999.

[127] S. Osman, D. Subhraveti, G. Su, and J. Nieh. The Design and Implementation of Zap: A System for Migrating Computing Environments. In *Proceedings of the* 5^{th} *Symposium on Operating Systems Design and Implementation (OSDI)*, pages 361–376, December 2002.

[128] J. Padhye, J. Kurose, D. Towsley, and R. Koodli. A model based TCP-friendly rate control protocol. *UMass-CMPSCI Technical Report TR 98-04*, 1998.

[129] G. Pandurangan, P. Raghavan, and E. Upfal. Building Low-Diameter P2P Networks. In *42nd Symposium on Foundation on Computer Science (FOCS'01)*, Las Vegas, NV, October 2001.

[130] C. Papadopoulos, R. Lindell, J. Mehringer, A. Hussain, and R. Govindan. COSSACK: Coordinated Suppression of Simultaneous Attacks. In *Proceedings of DISCEX III*, pages 2–13, April 2003.

[131] Christos Papadopoulos and Guru M. Parulkar. Retransmission-based error control for continuous media applications. *Proc. 6th Intl. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, 1996.

[132] K. Park and H. Lee. On the Effectiveness of Route-based PAcket Filtering for Distributed DoS Attack Prevention in Power-law Internets. In *Proceedings of ACM SIGCOMM*, pages 15–26, August 2001.

[133] L. Peterson, D. Culler, T. Anderson, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In *Proceedings of the* 1^{st} *Workshop on Hot Topics in Networks (HotNets-I)*, October 2002.

[134] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proceedings of ACM SIGCOMM'01*, San Diego, CA, August 2001.

[135] M. Reed, P. Syverson, and D. Goldschlag. Anonymous Connections and Onion Routing. *IEEE Journal on Selected Areas in Communications (JSAC)*, 16(4):482–494, May 1998.

[136] P. Reiher, J. Mirkovic, and G. Prier. Attacking DDoS at the source. In *Proceedings of the* 10^{th} *IEEE International Conference on Network Protocols*, November 2002.

[137] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual Network Computing. In *Proceedings of IEEE Internet Computing*, volume 2, January/February 1998.

[138] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321, April 1992.

[139] Ronald L. Rivest and Adi Shamir. Payword and micromint: Two simple micropayment schemes. In *Security Protocols Workshop*, pages 69–87, 1996.

[140] L. Rizzo. Effective Erasure Codes for Reliable Computer Communication Protocols. *Computer Communication Review*, April 1997.

[141] L. Rizzo and L. Vicisano. RMDP: An FEC-based Reliable Multicast Protocol for Wireless Environments. *Mobile Computing and Communications Review*, 2(2), April 1998.

[142] A. Rowstron and P. Druschel. Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility. In *Proceedings of ACM SOSP'01*, Banff, Canada, October 2001.

[143] D. Rubenstein and S. Sahu. An Analysis of a Simple P2P Protocol for Flash Crowd Document Retrieval. Technical report, Columbia University, November 2001.

[144] Yong Rui and Zicheg Liu. Artifacial: automated reverse turing test using facial features. In *MULTIMEDIA '03: Proceedings of the eleventh ACM international conference on Multimedia*, pages 295–298, New York, NY, USA, 2003. ACM Press.

[145] L. Santhanam, A. Kumar, and D. P. Agrawal. Taxonomy of IP Traceback. *Journal of Information Assurance and Security (JIAS)*, 1(2):79–94, June 2006.

[146] M. Satyanarayanan, J. Kistler, P. Kumar, M. Okasaki, E. Siegel, and D. Streere. Coda: A Highly Available File System for Distributed Workstation Environments. *IEEE Transactions on Computers*, 39(4), April 1990.

[147] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson. TCP Congestion Control with a Misbehaving Receiver. *ACM Computer Communications Review*, 29(5):71–78, October 1999.

[148] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical Network Support for IP Traceback. In *Proceedings of ACM SIGCOMM*, pages 295–306, August 2000.

[149] C. Schuba, I. Krsul, M. Kuhn, E. Spafford, A. Sundaram, and D. 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.

[150] R. Sherwood, B. Bhattacharjee, and R. Braud. Misbehaving TCP Receivers Can Cause Internet-Wide Congestion Collapse. In *Proceedings of the* 12^{th} *ACM Conference on Computer and Communications Security (CCS)*, pages 383–392, November 2005.

[151] SHFS Development Team. SHell File System, SHFS. http://shfs.sourceforge.net/.

[152] Stelios Sidiroglou, Michael E. Locasto, Stephen W. Boyd, and Angel os D. Keromytis. Building a Reactive Immune System for Software Services. In *Proceedings of the USENIX Annual Technical Conference*, pages 149–161, April 2005.

[153] A. Snoeren, C. Partridge, L. Sanchez, C. Jones, F. Tchakountio, S. Kent, and W. Strayer. Hash-Based IP Traceback. In *Proceedings of ACM SIGCOMM*, August 2001.

[154] A. Stavrou and A. Keromytis. Countering DoS Attacks With Stateless Multipath Overlays. In *Proceedings of the* 12^{th} *ACM Conference on Computer and Communications Security (CCS)*, pages 249–259, November 2005.

[155] A. Stavrou, A. D. Keromytis, J. Nieh, V. Misra, and D. Rubenstein. MOVE: An End-to-End Solution To Network Denial of Service. In *Proceedings of the ISOC Symposium on Network and Distributed System Security (SNDSS)*, pages 81–96, February 2005.

[156] Angelos Stavrou, Debra L. Cook, William G. Morein, Angelos D. Keromytis, Vishal Misra, and Dan Rubenstein. Websos: An overlay-based system for protecting web servers from denial of service attacks. *Journal of Communication Networks*, 48(5), August 2005.

[157] Angelos Stavrou, John Ioannidis, Angelos D. Keromytis, Vishal Misra, and Dan Rubenstein. A Pay-per-Use DoS Protection Mechanism For The Web. In *Proceedings of the Applied Cryptography and Network Security (ACNS) Conference*, pages 120–134, June 2004.

[158] Angelos Stavrou, D. Rubenstein, and S. Sahu. A Lightweight, Robust P2P System to Handle Flash Crowds. *IEEE Journal on Selected Areas in Communications (JSAC)*, 22(1):6–17, January 2004.

[159] Angelos Stavrou, Dan Rubenstein, and Sambit Sahu. A lightweight, robust p2p system to handle flash crowds. In *In Proceedings of IEEE ICNP 2002*, Paris, France, November 2002.

[160] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery. RFC 2001, January 1997.

[161] I. Stoica, D. Adkins, S. Zhuang, and S. Surana. Internet Indirection Infrastructure. In *Proceedings of ACM SIGCOMM*, pages 73–86, August 2002.

[162] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Application. In *Proceedings of ACM SIGCOMM*, August 2001.

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

[164] A. Su, D. R. Choffnes, A. Kuzmanovic, and F. E. Bustamante. Drafting Behind Akamai (Travelocity-Based Detouring). In *Proceedings of ACM SIGCOMM*, pages 435–446, September 2006.

[165] S. Tao, K. Xu, A. Estepa, T. Fei, L. Gao, R. Guerin, J. Kurose, D. Towsley, and Z. Zhang. Improving VoIP quality through path switching. In *Proceeding of IEEE INFOCOM*, March 2005.

[166] The Honeynet Project & Research Alliance. Know your Enemy: Tracking Botnets. http://www.honeynet.org, March 2005.

[167] P. Verkaik, O. Spatscheck, J. van der Merwe, and A. C. Snoeren. PRIMED: Community-of-Interest-Based DDoS Mitigation. In *Proceedings of the* 1^{st} *Workshop on Large-Scale Attack Defence (LSAD)*, pages 147–154, September 2006.

[168] Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. CAPTCHA: Using Hard AI Problems For Security. In *Proceedings of EUROCRYPT*, May 2003.

[169] J. Wang, X. Liu, and A. A. Chien. Empirical Study of Tolerating Denial-of-Service Attacks with a Proxy Network. In *Proceedings of the* 14^{th} *USENIX Security Symposium*, pages 51–64, August 2005.

[170] X. Wang and M. K. Reiter. Defending Against Denial-of-Service Attacks with Puzzle Auctions (Extended Abstract). In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2003.

[171] X. Wang and M. K. Reiter. Mitigating Bandwidth-Exhaustion Attacks using Congestion Puzzles (Extended Abstract). In *Proceedings of the* 11^{th} *ACM Conference on Computer and Communications Security (CCS)*, pages 257–267, October 2004.

[172] B. Waters, A. Juels, J. A. Halderman, and E. W. Felten. New Client Puzzle Outsourcing Techniques for DoS Resistance. In *Proceedings of the* 11^{th} *ACM Conference on Computer and Communications Security (CCS)*, pages 246–256, October 2004.

[173] Jorg Widmer. Implementation of the TCP-Friendly Congestion Control Protocol (TFRC). http://www.icir.org/tfrc/code/.

[174] K. Xu, Z. Zhang, and S. Bhattacharyya. Reducing Unwanted Traffic in a Backbone Network. In *Proceedings of the Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, pages 9–15, July 2005.

[175] Y. Xu and R. Guerin. On the Robustness of Router-based Denial-of-Service (DoS) Defense Systems. *ACM Computer Communication Review*, 35(3):47–60, July 2005.

[176] D. Xuan, S. Chellappan, and X. Wang. Analyzing the Secure Overlay Services Architecture under Intelligent DDoS Attacks. In *Proceedings of the* 24^{th} *International Conference on Distributed Computing Systems (ICDCS)*, pages 408–417, March 2004.

[177] A. Yaar, A. Perrig, and D. Song. Pi: A Path Identification Mechanism to Defend against DDoS Attacks. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2003.

[178] Abraham 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.

[179] X. Yang, D. Wetherall, and T. Anderson. A DoS-limiting Network Architecture. In *Proceedings of ACM SIGCOMM*, pages 241–252, August 2005.

[180] H. Yin and J. Li. An Efficient Probabilistic Packet Marking Scheme (NOD-PPM). In *Proceedings of the* 9^{th} *Information Security Conference (ISC)*, pages 373–382, August/September 2006.

[181] C. C. Zou, N. Duffield, D. Towsley, and W. Gong. Adaptive Defense Against Various Network Attacks. In *Proceedings of the Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, pages 69–75, July 2005.