OverDoSe: A Generic DDoS Protection Service Using an Overlay Network

Elaine Shi **Ion Stoica** **David Andersen**
Adrian Perrig

February 2006
CMU-CS-06-114

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We present the design and implementation of OverDoSe, an overlay network offering generic DDoS protection for targeted sites. OverDoSe clients and servers are isolated at the IP level. Overlay nodes route packets between a client and a server, and regulate traffic according to the server's instructions. Through the use of light-weight security primitives, OverDoSe achieves resilience against compromised overlay nodes with a minimal performance overhead. OverDoSe can be deployed by a single ISP who wishes to offer DDoS protection as a value-adding service to its customers.

Keywords: overlay network, Distributed Denial-of-Service, computational puzzle, compromised overlay nodes, request channel

1 Introduction

Distributed Denial-of-Service (DDoS) attacks continue to be a serious problem in the Internet. In a DDoS attack, an adversary controlling many hosts (often tens of thousands) uses these hosts to simultaneously send traffic to a victim, exhausting the victims bandwidth or computational capacity. These attacks may be used to capriciously bring down a prominent website, but they have also been used in extortion schemes against e-commerce sites [22].

Recently, researchers have proposed two categories of DDoS defense mechanisms. The first approach requires router support for filtering attack traffic. For instance, in Pushback [11, 21] and traceback [8, 25, 26], routers use an explicit signaling protocol to detect the source of DDoS attacks and filter bad traffic as close to the source as possible. Network-capability architectures [6, 20, 32, 33], on the other hand, use markings in the packet header to encode permission to send traffic. The primary concern with infrastructure-based approaches is deployability, as modifications to the current network infrastructure require a lengthy standardization process and significant economic lift to get off the ground. The second approach is to deploy an overlay network to route and filter traffic to the victim server. In contrast to infrastructure-based approaches, overlays are immediately deployable in today's Internet. This approach has therefore received much recent attention in the research community [5, 17, 18, 15, 28, 27]. Previous overlay-based DDoS defense solutions, including Akamai's SiteShield service [1], SOS [15, 27] and Mayday [5], rely on a secret authenticator (e.g., the IP address of the victim server) held by all or a subset of the overlay peers. If the secret authenticator is revealed, an attacker can bypass the overlay network and directly flood the victim server.

The potential existence of compromised overlay nodes poses a serious threat to the security of overlay-based solutions. Previous work has studied ways to limit the damage of compromised overlay nodes [5, 15]. Their focus is mainly on preventing malicious overlay nodes from learning and disclosing secret authenticators. The proposed countermeasures require relatively expensive mechanisms such as anonymous routing to protect the secret authenticator and the identities of the overlay nodes holding the secret. These security mechanisms have a high run-time performance overhead, and increase design-time complexity, introducing opportunities for new DDoS attacks.

In this paper, we propose OverDoSe, a high-performance and compromise-resilient overlay network offering generic DDoS protection for a spectrum of applications. OverDoSe *isolates* clients from the victim server at the IP level. Only the overlay nodes have IP reachability to the protected server; a client cannot reach the server via IP, so it must route its traffic through the overlay network to communicate with the server.

To achieve the IP level isolation between clients and servers, we take an ISP-centric approach, where the overlay network is deployed in strategic positions by a single ISP who wishes to offer DDoS protection as a value-adding service to its customers. The ISP hosting the overlay network must configure its routers to isolate the source and the victim server, but no new functionality is required on the ISP's routers. In addition, an ISP-centric approach requires no cooperation between different ISPs, imposing fewer hurdles to deployment.

One of the main goals of OverDoSe is to provide resilience against compromised overlay nodes. We examine a variety of potential DoS attacks by a malicious overlay node, and propose novel light-weight mechanisms to defend against these attacks.

Clients are recommended to install the OverDoSe client-side software, but need not modify legacy applications. We also propose ways to support legacy clients, although legacy clients get weaker protection under DDoS attacks than updated clients.

Unlike Content Delivery Network (CDN)-like approaches that offload the entire application logic to the overlay network, OverDoSe provides a set of application-independent primitives for DDoS protection.

2 Problem Definition

2.1 Terminology and Definitions

OverDoSe involves three entities: 1) *clients*, Internet hosts that request service from the server; 2) *overlay nodes* (or as *overlay peers*), who route and regulate traffic between clients and servers; and 3) *servers,* who execute application-specific logic and serve the clients' requests. Note that the server could be a single server machine or a server farm deployed in a data center.

OverDoSe considers attacks in which attackers and legitimate clients compete for some bottleneck resource. Other types of DDoS attacks, e.g., those that exploit server software vulnerabilities are outside the scope of this paper. We assume that the bottleneck resource is close to or at the server, either processing-bound (i.e., the server's computation/storage capability), or network-bound (i.e., the access link bandwidth to the server).

We assume the bottleneck resource is divided between serving connection setup requests (henceforth referred to as the *connection setup channel* or *the request channel*) and serving *established clients*. Connection setup is the process by which a client establishes its identity and starts a conversation with the server. After connection setup, a client becomes an established client. For example, consider a processing-bound web server. The web server can commit a fixed fraction of its processing cycles to admit new clients, which requires a password verification; and commit the remaining processing cycles to serving established clients, by fetching static or dynamic content from the local storage system and returning it to the client. Likewise, a network-bound server divides its access-link bandwidth into a smaller portion to serve connection setup requests and a larger portion to serve established flows.

2.2 Assumptions and Threat Model

We consider an adversary who has potentially compromised a large number of Internet hosts, which then collaborate in launching a DDoS attack.

Once the overlay network comes into play, instead of directly DDoSing the victim server, the adversary can also attack the overlay nodes. We assume that while the attacker may have enough resources to bring down a subset of the overlay nodes, she cannot disable the entire overlay network.

A sophisticated adversary can also compromise overlay nodes. In this paper, we assume that a compromised overlay node can exhibit arbitrarily malicious behavior. For instance, a compromised overlay node can flood the victim server, it can drop packets, or hijack the sessions of legitimate clients. An important goal of OverDoSe is to provide resilience against compromised overlay peers.

OverDoSe assumes that the overlay network is hosted by a single ISP. We assume that the ISP's routing infrastructure is trusted, and that a DDoS-resilient name lookup service can be achieved through means of replication and IP anycast. We assume that the hosting ISP provides a management infrastructure by installing special boxes close to the server, through which the server can turn off IP-level connectivity between a misbehaving overlay node and the server. (Section 4.1). We assume that the management infrastructure is not DDoSed.

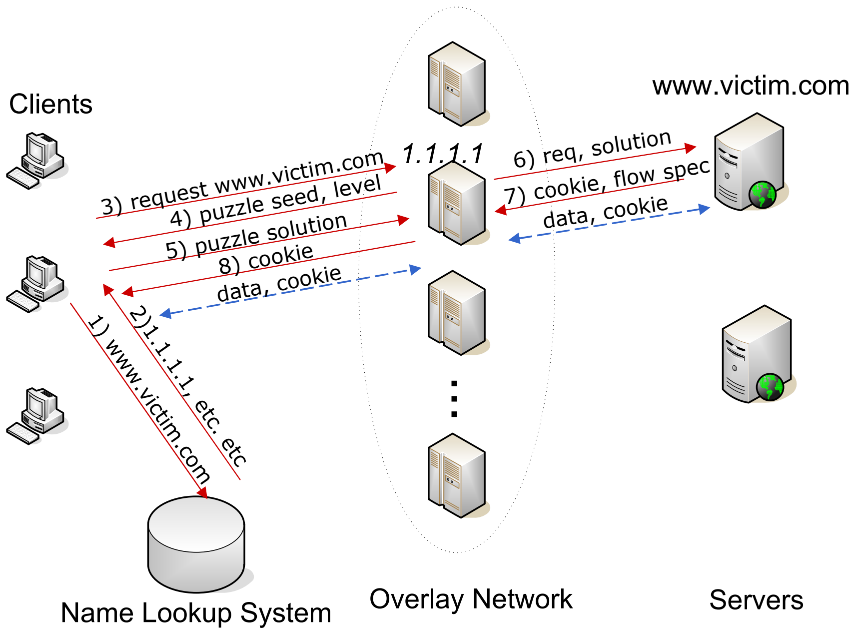


Figure 1: **OverDoSe basic protocol.** *This figure illustrates OverDoSe's basic protocol. The numbers represent the temporal ordering of messages. Solid lines represent the messages in the request channel and dashed lines represent the established channel.*

3 OverDoSe Overview

3.1 Protocol Overview

OverDoSe uses a novel computational puzzle scheme to provide fairness in the request channel. When a client wishes to connect to a server, it first sends a request to a name server to resolve the IP address of the server (step 1 in Figure 1). The name server returns a list of IP addresses of overlay nodes (step 2 in Figure 1). The client selects an overlay node to which it sends a connection request (step 3 in Figure 1). The node selection algorithm can be based on a variety of heuristics such as network proximity [24, 31] or node reputation [13]. In response to a client's connection request, the overlay node replies with the latest *puzzle seed* released by the server, as well as a *puzzle difficulty level* specified by the server (step 4 in Figure 1). The client is expected to solve a puzzle at or above the specified difficulty level in order to successfully set up a connection. The client generates a puzzle based on the puzzle seed, solves the puzzle, and sends the puzzle solution to the overlay node (step 5 in Figure 1). The purpose of the puzzle seed and the puzzle generation algorithm will be detailed in Section 4.2.2. Now the overlay node validates the puzzle solution and forwards the request and the solution to the server (step 6 in Figure 1). The server assigns a *cookie* to the requesting client, and replies to the overlay node with the cookie and a *flow specification* (step 7 in Figure 1). The flow specification is a set of rules the overlay node must enforce for regulating an established flow. The flow specification can be updated dynamically by the server, and is explained in more detail in Section 4.3. The overlay node then replies to the client with the cookie, successfully completing connection setup. The client attaches the cookie to all subsequent packets to the server. The overlay node then routes traffic between the client and the server, and polices the client's flow according to the flow specification.

3.2 Design Rationale

3.2.1 Securing Connection Setup, Established Flows

A complete DDoS solution must protect both the connection setup channel and the established flows. If a viable mechanism exists to distinguish legitimate clients from malicious clients, we should give better service to legitimate traffic, or simply blacklist malicious clients. When inferring legitimate traffic from attack traffic is hard (e.g., when zombies behave like legitimate clients), the defense mechanisms should provide some fairness among the users, both friend and foe, of the service.

Securing connection setup. Connection setup may occur before clients establish their identities with the server. As a result, differentiating between legitimate requests and malicious clients is difficult and remains an area of open research [14]. OverDoSe uses computational puzzles to share the connection setup channel resource fairly among clients. Compared with schemes such as per source-IP fairness, the puzzle-based scheme effectively bounds the rate at which a compromised overlay node can establish connections with a server. The motivation for choosing computational puzzles in OverDoSe is explained in more detail in Section 4.2.1.

Securing and regulating established flows. In contrast to connection setup, once a client establishes a session with the server, the server can often differentiate between legitimate and malicious behavior, by verifying application-level correctness, or by application-level authentication (e.g., password authentication or the CAPTCHA [29] mechanism used by many websites). In OverDoSe, once an attack has been identified, a server can instruct overlay nodes to filter the attacking traffic. OverDoSe also supports differentiated service: a server to specify to the overlay network the service level of each client. The overlay then regulates the sending rate of each client based on the client's service level. In this way, the server can give preference to high-priority customers, e.g., customers who spend more money shopping on an e-commerce website. For servers that cannot distinguish between clients, OverDoSe fair shares the bottleneck resource among all established clients.

3.2.2 Defense Against Compromised Overlay Nodes

Compromised overlay nodes can perform three general attacks: 1) flood the server themselves, or collude with malicious clients to flood the server; 2) drop packets from legitimate clients; 3) compromise data integrity, e.g., by injecting bogus data, by impersonating legitimate clients or by hijacking TCP connections.

To defeat the first attack, OverDoSe ensures that the server can always check whether an overlay node has correctly verified puzzles and performed packet filtering. A server can turn off IP connectivity between itself and a misbehaving overlay node using techniques described in Section 4.1.

Under the second attack, the victim client experiences high packet loss. To defend against this attack, clients are allowed to switch to a new overlay peer. To the client, the effect of a packet-dropping overlay peer is indistinguishable from a well-behaved overlay peer that is being DDoSed, or is simply overloaded. In all of these cases, however, the client has motivation to switch to a new overlay peer.

OverDoSe offers end-to-end authentication as an option to defeat data integrity attacks. This assumes the existence of a PKI from which clients can obtain a certificate for a server's public key, and then uses the Secure Sockets Layer (SSL) protocol to establish a symmetric key between a client and a server.

Section 4.4 discusses more sophisticated attacks and defense mechanisms.

3.2.3 Security vs. Deployability

One goal of OverDoSe is to find a good balance between security and deployability. Wide-spread overlay deployment across different ISPs increases the capacity of the overlay network, reducing the chance that the overlay network itself becomes the target of a DDoS attack; in addition, it allows the filtering of attack traffic closer to the source. On the other hand, a universal overlay deployment requires inter-ISP cooperation, and complicates the deployment process. OverDoSe investigates a single ISP deployment scenario, in which the ability to provide DDoS protection as a value-added service to its customers provides an economic incentive for an ISP to deploy an OverDoSe-like solution. We also hope that the successful deployment of OverDoSe by a single ISP can spur incentives for inter-ISP cooperation, eventually leading to the wide-spread deployment of such an overlay network.

A second goal of OverDoSe is to support legacy clients and legacy applications. Yet to achieve such backward compatibility imposes severe design constraints, and weakens the level of security we can achieve. OverDoSe encourages clients to install the OverDoSe client-side software to get better protection under DDoS attacks. The client-side software is built on top of the OCALA [12] framework, and allows legacy applications to use the overlay without being aware of its existence. OverDoSe also supports legacy clients, but they are less protected from DDoS than updated clients.

4 Detailed Design of OverDoSe

4.1 Infrastructure Support for OverDoSe

OverDoSe requires that servers only be reachable via overlay nodes, and requires that the server can revoke permission for a particular overlay node to contact it.

Isolation. OverDoSe isolates the home server by placing it on a private network that is logically separate from the ISP's IP routing infrastructure. The ISP then configures tunnels between the overlay nodes and the server, to allow only them to reach the server. In practice, such a private network can be established using an ISP's existing VPN technology, by configuring MPLS or other tunnels between the overlay nodes and the edge. The server can reside at a either public or private IP address, provided that arbitrary hosts on the Internet cannot directly reach that address. This approach provides isolation using only components widely available in today's network infrastructure.

Revocation. To deal with compromised overlay nodes, OverDoSe requires that a server be able to prevent a node from communicating with it. We assume that the hosting ISP install boxes close to the victim node running RSVP-TE [7] and similar protocols that can dynamically establish and destroy tunnels and configure QoS parameters such as capacity and priority. Using this function requires the ability for a server that is untrusted by the ISP's network to signal to the network its desire for tunnel establishment and teardown. This could be done using RSVP or via a relatively simple client interface that validates requests and translates them into internal RSVP-TE requests.

Origin authentication for overlay nodes. Compromised overlay nodes can spoof a well-behaved overlay node and flood the server. To allow servers to identify the attack source, OverDoSe requires an authenticated communication channel. One way to achieve this is to have routers one hop away from the overlay nodes filter spoofed IP packets. One could also configure each overlay node with a different MPLS label, and use the MPLS label for origin authentication. This requires that the server understand MPLS, and the last-hop router to the server retain the MPLS labels on packets before handing them to the server. Alternatively, one could configure the last-hop router to the server to rewrite the source IP address of every packet according to its MPLS label.

4.2 Puzzle-based Connection Setup

4.2.1 Why Puzzles?

As we explain in Section 3.2.1, because differentiating between legitimate and malicious clients in the request channel remains an open research topic, OverDoSe aims to share the request channel fairly among clients. The question then reduces to how to reliably identify a client.

Typically, a client can be identified by its IP address. Unfortunately IP-based fairness raises several concerns: 1) *IP spoofing by malicious clients.* Although simple IP spoofing attacks can be countered using an approach similar to SYN cookies, such approaches provide no defense against an adversary who is able to sniff packets sent to an IP address and send packets spoofing that IP address. In such cases, we say that the adversary *owns* the IP address. For instance, a zombie machine situated in a class B network can potentially spoof any IP address in the class B range, even in the presence of ingress filtering. If an IP-fairness scheme is used in the request channel, an adversary gains an advantage proportional to the number of IP addresses it owns, which could be much larger than the number of machines the adversary has compromised. 2) *IP spoofing by malicious overlay nodes.* A compromised overlay node can flood the server with fake request packets from many sources. It can then open many different connections with the server and flood the server using the spoofed IP addresses. 3) *NATs.* A practical concern with per-IP fairness is due to the presence of Network Address Translators (NATs) and firewalls which can cause many users to share a single IP address. Hence, per-IP fairness would discriminate against NATed hosts.

To address these concerns, OverDoSe instead uses computation-based fairness. A server periodically releases new puzzle seeds to the overlay nodes. A client obtains the latest puzzle seed from an overlay node, generates and solves a puzzle based on the seed, and attaches the solution to a connection setup request. The server can control the rate of connection setup requests by adjusting the puzzle level. The overlay nodes verify puzzle solutions and admit only requests with valid puzzle solutions at or above the puzzle level specified by the server.

Puzzle-based connection effectively defends against compromised overlay nodes. The server can detect a cheating overlay node that passes incorrect solutions by re-checking the puzzle solution. By using computational puzzles, malicious overlay nodes also have to solve computational puzzles to open connections with the server. The rate at which a malicious overlay node can open new connections with the server is therefore bounded by its computational resources.

If every client had equal computational power, then per-challenge fairness would be equivalent to per-source fairness. Suggestions on how to mask the asymmetry in computational ability are provided in Section 5.2.

4.2.2 Design of Puzzle Scheme

Puzzle seeds. The puzzle seeds are a sequence of pseudo-random numbers generated by a server, which periodically releases a new seed to the overlay network.

Generating and solving puzzles. After receiving the latest puzzle seed s, the client picks a random nonce r and computes a flow-specific puzzle as

$$p = H(x||r||s||\ell||\text{source IP}||\text{peer IP}), \tag{1}$$

where H denotes a cryptographic hash function such as SHA-1, and $||$ denotes concatenation.

To solve the puzzle at difficulty level ℓ, the client finds a 64-bit value x such that the last ℓ bits of p are zero. The client includes r, s, ℓ, and the puzzle solution x in a request packet.

Including the source IP in the hash computation prevents reuse of a puzzle solution by different zombie machines. Likewise, including the peer IP prevents an adversary from using the same puzzle solution with different overlay nodes.

Puzzle verification. Upon receiving a connection setup request with a puzzle solution, an overlay node first verifies that the puzzle seed s contained in the request packet is one that has been recently released by the server. Then the overlay node verifies the puzzle solution by computing the same hash shown in Equation 1, using the nonce r, the seed s, and the flow information in the request packet.

4.2.3 Resource Scheduling in the Request Channel

Request channel resource scheduling must meet four requirements: 1) An overlay node must suppress all requests containing puzzle solutions below the threshold difficulty level specified by the server. 2) To prevent an adversary from reusing the same puzzle solution at high rates, an overlay node should throttle the rate at which the same puzzle solution is used. 3) To prevent an adversary from precomputing solutions and using them all at once, puzzle solutions must expire after a finite amount of time. 4) When requests arrive at the server, the server must check the puzzle solutions, and preferentially admit clients that solved more difficult puzzles.

When request packets arrive at an overlay node, the node first verifies the puzzle solution and drops requests containing invalid puzzle solutions, expired seeds, or a puzzle difficulty below the current threshold level. Verified request packets enter a rate throttler that enforces that each puzzle solution is not excessively repeated. A puzzle solution that has been seen in the past t seconds is dropped in this stage. t is an adjustable parameter; when set to infinity, the rate throttler reduces to strict duplicate suppression.

4.3 Regulating Established Flows

Flow cookie. To receive prioritized service, a client must attach to its data packets the flow cookie it received during connection setup. The cookie identifies the flow, and can be carried across multiple TCP sessions to support HTTP-like applications that complete over multiple TCP transactions.

Compromised overlay nodes or zombie hosts can hijack a client's connection by eavesdropping on the path from the client to the server and stealing the session cookie. This attack is similar to a TCP session hijacking attack [19], where an adversary eavesdrops on a TCP session and steals TCP sequence numbers to inject bogus data. To mitigate the cookie hijacking attack, OverDoSe uses a short-lived cookie, and have the server periodically update the cookie during the life-time of the flow. When an old cookie expires, the server notifies the overlay node, who in turn notifies the client of the new cookie. The cookie expiring mechanism requires an adversary to constantly tap the session to observe with the latest cookie. A stronger but more expensive defense involves the use of end-to-end authentication, and is discussed in more detail in Section 4.4.

Flow specification. The flow specification represents the level of service the server assigns to the flow. In OverDoSe, this is expressed as a maximum inbound bandwidth. Overlay node rate limit the incoming flow beneath its maximum allowable bandwidth.

The server can dynamically update a flow's bandwidth quota. For instance, the server can instruct an overlay node to blacklist a client or reduce its bandwidth if that client is suspected of cheating. The server could also upgrade a client if it is about to spend money or after it has just spent money on a hosted website. To dynamically update a flow's quota, the server sends to the overlay node a flow update message, either piggybacked on top of an existing data packet, or in the form of an explicit control message.

4.4 Defense against Compromised Overlay Nodes

An important goal of OverDoSe is to provide resilience to compromised overlay nodes. We now discuss potential attacks from compromised overlay nodes, and techniques to defeat them. Our study is restricted to DoS attacks; other attacks such as breach of data privacy are outside the scope of this paper.

4.4.1 Data Integrity Attacks

A data integrity attack in the request channel is a packet dropping attack, for it results in the clients' inability to establish a connection with a server. We address packet dropping attacks separately in Section 4.4.2, so the discussion below focuses on data integrity attacks in the established channel.

Data integrity attacks can be detected either at the application level or at the OverDoSe level. Applications are able to detect invalid data packets, either through application-level data authentication, or through application semantic checks. OverDoSe allows applications and users to to switch to a new overlay node when such attacks are detected.

OverDoSe also provides an end-to-end data authentication option at the OverDoSe level. Built on top of the OCALA [12] framework, OverDoSe adopts OCALA's mechanism for setting up a secret authentication key between the client and the server, assuming the existence of a PKI from which clients can obtain a certificate for a server's public key and using the Secure Sockets Layer (SSL) protocol for symmetric key establishment.

4.4.2 Packet Dropping Attacks

Dropping legitimate connection setup packets prevents a client from setting up a connection through that overlay node. An OverDoSe client automatically tries a new overlay node if it fails to set up a connection through the current overlay node within a certain timeout.

In the established channel, a malicious overlay node can drop packets in transit between a legitimate client and a server. Similar to data integrity attacks, packet dropping attacks may be detected either by the application or by OverDoSe. The application perceives a packet dropping attack as a broken connection, or low application throughput. When this happens, OverDoSe allows applications and users to explicitly switch to a new overlay node.

OverDoSe itself detects packet dropping attacks. OverDoSe clients can be configured to send periodic probe messages to the server which sends back probe reply messages. A missing probe reply indicates lossy network conditions or a packet dropping attack. When the loss rate exceeds a certain threshold, the OverDoSe client automatically switches to a new overlay node.

The probe mechanism must be combined with light-weight cryptography for enhanced security:

Probe integrity. To defeat OverDoSe's probes, malicious overlay nodes can impersonate clients or a server and inject bogus probes and probe replies. Such attacks can be defeated by turning on the end-to-end authentication option as described in Section 4.4.1 to ensure data integrity.
Defense against selective forwarding attacks. A malicious node can selectively forward probes and reply messages but suppress normal data packets. To defend against the selective forwarding attack, OverDoSe disguises so that probes are indistinguishable from data packets. To do so, we assume that the client and the server shares a secret key K. The secret key can be established in a similar way as the authentication key. Meanwhile, OverDoSe dedicates a field in its packet header to contain a random nonce u, and a number x. A packet is a probe packet if and only if $x = \text{MAC}_K(r||\text{msg})$, where MAC_K denotes a message authentication code over secret key K. If probe packets were sent at regular intervals, malicious overlay nodes might also be able to distinguish probe packets from normal data through timing information. To avoid such timing attacks, OverDoSe adds randomness to the time when probe messages are sent.

4.4.3 Flooding Attacks

A compromised overlay node can attempt to flood a server to create congestion, and deny service to legitimate clients. A compromised overlay node can do so in several ways: it can collude with malicious clients and admit their flood traffic, or it can actively generate flood traffic. Floods can target the request channel or the established channel.

Request channel floods. In the request channel, a malicious overlay node can attempt to facilitate malicious clients by admitting requests without a valid puzzle solution at or above the specified difficulty level. A malicious overlay node can also generate invalid request floods.

OverDoSe ensures that servers can check whether an overlay node correctly verified puzzles in the request channel. When an overlay node forwards a request to the server, it also attaches the puzzle solution (including s, r, ℓ, x). The server can check if the seed s used in the generation of the puzzle has been recently disclosed by the server. The server then verifies the puzzle solution by recomputing Equation 1. In addition to checking the solutions, the server also checks whether each overlay node has throttled the rate at which each puzzle is reused.

In OverDoSe, a malicious overlay node cannot inject invalid request packets at arbitrary rates, because it must compute a puzzle solution for every request packet it injects to avoid detection by the server.

Established channel floods. In the established channel, a compromised overlay node can fail to shape a client's flow, or actively generate floods. To detect such attacks, the server monitors the incoming traffic rate of established flows and compares the observed rate against the flow specification. In this way, a server can detect malicious overlay nodes that fail to regulate established flows.

4.4.4 Other Attacks

Slandering. A malicious overlay node can generate floods impersonating a legitimate overlay node; hence, a server can mistakenly blame and revoke the legitimate overlay node. Packet-level authentication fails to address this attack, because a slandering attacker can flood with unauthenticated packets. The slandering attack is addressed through infrastructure supported origin authentication techniques as described in Section 4.1. OverDoSe trusts the hosting ISP's network infrastructure to prevent slandering by malicious overlay nodes.

Equivocation. A malicious overlay peer can help a zombie client by ignoring a blacklist or flow downgrade message from the server. Similarly, a malicious overlay node can admit request at lower difficulty levels by ignoring a puzzle level upgrade message. The equivocation attack can be addressed by a timeout. We require that the server periodically send flow update messages. If an overlay peer does not hear any updates for a specific flow within the timeout, it must suppress or gradually downgrade the service level of that flow. To blacklist a flow, the server first sends explicit blacklist messages to the overlay node; meanwhile, the server withholds further updates for the suspect flow. If an overlay node does not respond to the explicit blacklist message immediately, it must still suppress the flow within a short time to avoid detection. A similar approach can address equivocations on the puzzle difficulty.

Performance degradation. Instead of dropping all or the large majority of packets of a legitimate flow, a compromised overlay node can potentially avoid detection with a performance degradation attack, where it drops a small subset of legitimate packets or delays packet delivery. This attack is difficult to defeat. Although OverDoSe clients monitor real-time connection statistics, it is unclear when an alarm should be raised. The current implementation tries to switch to a new overlay node when performance drops beneath a user specified threshold (e.g., delay > 100 ms). We recognize that this is an aspect of OverDoSe left for future research. One promising defense is to spread a single flow across multiple overlay nodes, as recently proposed by Stavrou et al. [28]. More details on this approach are available in Section 8.

4.5 Supporting Legacy Clients

Thus far, we have assumed that clients install the OverDoSe software that bridges the legacy applications and the overlay. For incremental deployment purposes, we also describe techniques to support legacy unmodified clients.

Identifying the destination of legacy packets. When multiple servers are behind the same overlay network, overlay nodes must be able to route packets from clients to the correct server. Updated clients convey to overlay nodes the destination of each packet through a designated server identification field in the packet header. Since legacy clients do not provide this information, overlay nodes in OverDoSe rely on the following means to identify the destination of legacy packets: 1) For self-identifying applications such as HTTP, overlay nodes rely on the destination URL embedded in the application data to identify the destination. 2) When overlay nodes have multiple

IP addresses, they can designate a different IP for each protected server. Let n denote the number of servers, and m the number of overlay nodes. Let $(\mathrm{ip}_1^i, \mathrm{ip}_2^i, \ldots, \mathrm{ip}_n^i)$ denote the n different IP addresses associated with the $i^{th}(1 \leq i \leq m)$ overlay node. When a legacy client sends a DNS request for the $j^{th}(1 \leq j \leq n)$ server, the DNS server returns set S of IP addresses, where $S \subseteq \{\mathrm{ip}_j^1, \mathrm{ip}_j^2, \ldots, \mathrm{ip}_j^m\}$. 3) Since the OverDoSe endhost software is built on top of OCALA, it can also use OCALA's native approach for supporting legacy clients, i.e., through the use of Legacy Client (LC) gateways. The LC gateway is configured as the clients' local DNS server. It intercepts the clients' DNS packets, sends back a DNS reply with an Internet routable address to the client (a different IP address is used for each server), captures packets sent by the legacy client to that address, stamps them with the destination identifier, and sends them over the overlay [12].

Resource scheduling in the presence of legacy packets. OverDoSe reserves a small fraction of the request channel to serve legacy request packets. A server specifies to the overlay nodes the maximum rate at which to admit legacy requests. In this way, legacy requests only compete with other legacy requests during congestion.

OverDoSe uses IP fair-queuing for legacy connection setup requests. To counter IP-spoofing attacks, the overlay nodes return TCP SYN/ACKs and SYN cookies impersonating the server such that spoofed requests are filtered at the overlay network before reaching the server. Malicious overlay nodes can spoof IP addresses and inject invalid request packets, but cannot exceed the legacy request bandwidth without being detected.

After a legacy client establishes a connection with a server, the client's IP address and port number are used to identify the established flow. Without additional application-level information, an updated client's flow should be given preference over a legacy flow, because the updated client paid a certain amount of computation to set up the flow. A server can establish confidence in a legacy flow through application-level authentication, and elevate its service level. The following section explains the server's resource allocation policy in more detail.

4.6 Expressing Server Policies

The server is responsible for two decisions: 1) admission control, i.e., whether to admit or blacklist a client; and 2) resource allocation, i.e., how much of the bottleneck resource to allocate to each client. The server can often make admission control and resource allocation decisions based on application-level information or through means of application-level authentication. For instance, an e-commerce website can give higher priority to clients who have spent more money in the past. A server can also blacklist a client who sends corrupt application data, or data that contain a known worm signature.

OverDoSe provides a generic framework for servers to express their resource allocation policies. Let r_x denote the rate at which the server can admit new clients; let r_y denote the rate at which the server can serve established flows. The rates are either expressed in number of packets per second, or the bits per second.

Policy for request channel. The server specifies to an overlay node a minimum puzzle level ℓ. The overlay node must suppress requests below level l. Initially, when the request channel is free of congestion, $\ell = 0$. The server increases ℓ when congestion occurs in the request channel; and decreases ℓ when request channel load falls beneath a certain threshold.

Policy for established flows. OverDoSe implements a generic weighted fair sharing scheme for established flows. The server application specifies to OverDoSe the weight for each flow. Let f denote the total number of established flows at a particular point of time; let w_i denote the weight of the i^{th} flow, $1 \leq i \leq f$. OverDoSe computes r_y^i, the bandwidth quota for the i^{th} flow as $r_y^i = \frac{r_y \cdot w_i}{\sum_{i=1}^{f} w_i}$. Setting w_i to 0 is equivalent to blacklisting the i^{th} flow.

The OverDoSe server software allows the server application to differentiate between four types of established flows and specify a default weight for each type: 1) updated and authenticated; 2) legacy and authenticated; 3) updated and non-authenticated; 4) legacy and non-authenticated. A flow is updated if it comes from an updated client. A flow is authenticated if the server application is able to authenticate the client's identity at the application level, e.g., through password-based authentication, or a CAPTCHA [29] mechanism.

5 Discussion

5.1 Accommodating NATs

As shown in Equation 1, a sender computes a puzzle based on its IP address. If the sender's ISP uses a Network Address Translator (NAT), then a sender's internal IP address differs from its external IP address, so overlay nodes in OverDoSe will reject puzzle solutions computed for a puzzle based on the internal address. To accommodate NATs, when a sender behind a NAT requests the latest puzzle seed from an overlay node, the overlay node includes not only a puzzle seed and a puzzle level, but also the sender's external IP address in the reply. The sender can extract the external IP address from the response packet and use it in the computation of a puzzle solution.

5.2 Asymmetric Computational Ability

One concern with using computational puzzles is that it gives an advantage to endhosts with faster CPUs. To address this problem, researchers have proposed memory-bound puzzles [4, 9]. Another possibility is use a database like ARIN WHOIS [2] to obtain organizational information of an IP address. This can help us infer what type of device is associated with an IP address. We plan to study this further in future research.

5.3 Other Types of DDoS Attacks

Because clients and servers are isolated at the IP layer in OverDoSe, compromised zombies cannot directly flood the server at the IP layer. However, a sophisticated adversary can instead direct the flood towards hosts close to the targeted server. This can cause network congestion near the targeted server, hence preventing legitimate clients from accessing the server. To defend against this attack, we can either hide the location of the server or locate the protected server in an isolated and well-connected part of the network with a very high capacity.

6 Analysis

We analyze OverDoSe's puzzle-based connection setup scheme (PUZZLE), and compare it with Fighting Fire with Fire [30] (FFF), in which all clients, both legitimate and malicious, send aggressively to ensure fairness. The results show that with a single bottleneck, PUZZLE and FFF achieve the same mean and variance.

For simplicity, assume that all Internet hosts are identical in terms of computational power and access link bandwidth. Assume that the Internet backbone has sufficient capacity such that in the case of FFF, each client is capable of injecting request packets to the bottleneck as fast as their access link bandwidth can sustain. Due to space limit, we only show the conclusions of our theoretical analysis. We provide the detailed proofs in the appendix.

6.1 Response Time without Queuing

Assume $n-1$ malicious and 1 legitimate clients that concurrently try to establish a connection with a server, who is capable of handling new connection requests at a deterministic rate of μ.

For FFF, we make the simplifying assumption that every $1/\mu$ seconds, the server accepts a request at random from all requests that have been received in the past $1/\mu$ seconds. In PUZZLE, the server accepts all requests carrying a valid puzzle solution of level at least ℓ. ℓ is selected such that a client needs n/μ time in expectation to solve a level ℓ puzzle. Let T_{FFF}, T_{PUZZLE} denote the time for a legitimate client to successfully get a request accepted at the server.

Theorem 6.1. $E[T_{FFF}] = E[T_{PUZZLE}] = \frac{n}{\mu}, \quad VAR[T_{FFF}] = VAR[T_{PUZZLE}] = \frac{n^2}{\mu^2}$

6.2 Effect of Queuing

The analysis in the previous section does not take into account queuing delay in FFF and PUZZLE.

Consider FFF-Q-FCFS, a variant of FFF, where the server has a queue of length Q, and requests are serviced at a deterministic rate of μ in First-Come-First-Serve (FCFS) order.

Corollary 1. $E[T_{FFF-Q-FCFS}] = n/\mu + Q/\mu, \quad VAR[T_{FFF-Q-FCFS}] = n^2/\mu^2.$

Next, we analyze the effect of queuing on PUZZLE and show that under FCFS policy, the queuing delay is sublinear with respect to the number of attackers.

Assume the server has an infinite-size queue, and adopts a First-Come-First-Serve (FCFS) service policy. The response time is the sum of the time spent computing the puzzle and the queuing delay. To minimize the response time the server has to choose between (1) a lower puzzle level which translates to a lower computation time, but a larger queuing delay (as more puzzle are computed), or (2) a higher puzzle value, which translates to lower queuing delay, but a higher computation time.

Assume at any point of time, n users concurrently request a connection setup. Define job size S to be the time spent servicing a request. We consider general distributions of job sizes. Let μ denote the average rate at which the server can service requests, hence, $\mu E[S] = 1$. Let $x = \frac{E[S^2]}{2E[S]}$. x is a quantity that roughly characterizes the variance of the job size distribution.

Theorem 6.2. *To obtain the best expected response time, one must set the puzzle level to be* $\ell^* = \log_2 \frac{n+\sqrt{\mu x n}}{\mu\delta}$ *. And under* ℓ^**, the best expected response time is* $E[T_S]^* = \frac{n}{\mu} + 2\sqrt{\frac{nx}{\mu}}$.

Corollary 2. *Particularly, with deterministic job size,* $\ell^* = \log_2 \frac{n+\sqrt{n/2}}{\mu\delta}$, $E[T_S]^* = \frac{n}{\mu} + \sqrt{2n}$. *When job size is an exponential distribution,* $\ell^* = \log_2 \frac{n+\sqrt{n}}{\mu\delta}$, $E[T_S]^* = \frac{n}{\mu} + 2\sqrt{n}$.

7 Evaluation

7.1 Implementation Details

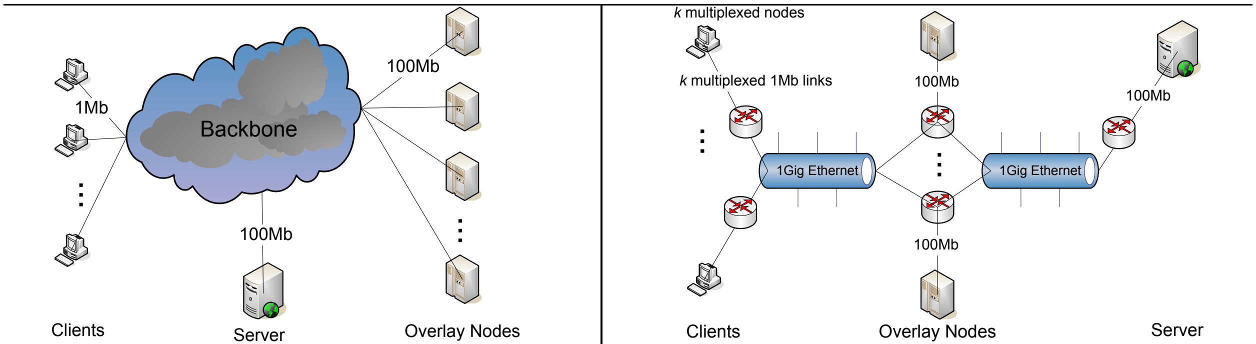
We implement the client-side and server-side OverDoSe software on top of OCALA [12], a software layer positioned below the transport layer in the IP stack, that allows legacy applications to use features provided by an overlay network without being aware of the existence of the overlay. On the client side, when OverDoSe captures a DNS request for a protected server from a legacy application, it resolves the name to a list of IP addresses of the overlay nodes. OverDoSe then establishes an end-to-end path to the protected server via the overlay. After setting up the path, OverDoSe returns to the legacy application a DNS reply containing a fake IP address. OverDoSe thereafter captures all packets sent to that IP address and routes them via the overlay network. As described in Section 4.4, OverDoSe also uses probe messages to monitor end-to-end connectivity. OverDoSe automatically tries to reroute traffic through a different overlay node when it observes low throughput or high loss.

On the server side, in addition to connecting the server application with the overlay network, OverDoSe also provides an API through which server applications can specify to OverDoSe how to regulate established flows and the connection setup packets.

Overlay nodes in OverDoSe run a single-threaded user-level daemon under Linux, capturing and forwarding packets using UDP. An OverDoSe overlay node allocates a different UDP port for each server. All established data packets from or destined for a server, as well as control messages from the server, are sent to a unique UDP port dedicated to that server. In addition, all overlay node use a unique UDP port to handle initial connection setup requests.

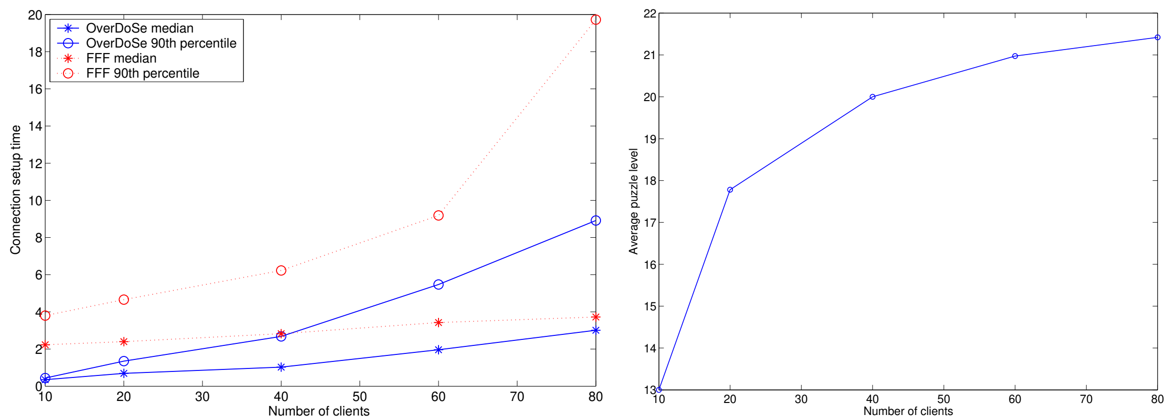
7.2 Experiment Setup

To validate the design of OverDoSe, we ran several experiments on Emulab [3]. All experiments use Emulab's pc3000 nodes, consisting of 3.0 GHz 64-bit Xeon processors, 800MHz FSB, and



(a) Virtual topology (b) Physical topology

Figure 2: **Experimental topology.** *This figure illustrates the virtual and physical topology used in the experiments. The virtual topology consists of a backbone of "infinite" capacity and endhosts that are subject to access-link bottleneck bandwidth constraints. We multiplex nodes and links to simulate many virtual nodes (links) out of one physical node (link). The two gigabit Ethernets provide abundant capacity for all experiments.*



(a) Connection setup time (b) Average puzzle level

Figure 3: **Request channel experimental results.** *The server is capable of handling 30 new connections per second. The connection setup time for OverDoSe includes the time for requesting the puzzle seed and solving the puzzle.*

2GB 400MHz DDR2 RAM. An underlying VLAN switched network can be configured to set up the topologies needed for the experiments.

We emulate an Internet-like topology, consisting of a backbone with infinite capacity, clients with access-link bandwidth of 1Mbit/s, and peers and a server each with access-link bandwidth of 100Mbit/s. This virtual topology is depicted in Figure 2(a). Figure 2(b) shows the real topology used in the established channel experiments. To emulate the effect of large DDoS attacks, we multiplex nodes and links to simulate many virtual nodes (links) out of one physical node (link). Link multiplexing is achieved using customized Click [16] routers that shape each virtual node's traffic to 1Mbit/s. The total amount of traffic sent into the backbone in every experiment is below 1Gbps, so we use a Gigabit LAN to simulate an Internet backbone with infinite capacity.

	RTT (std. dev.)	RTT reverse (std. dev.)	Throughput	Throughput reverse
OverDoSe	0.499(0.051) ms	0.503(0.082) ms	185.3 Mbps	146.7Mbps
no OverDoSe	0.193(0.051) ms	0.198(0.052) ms	694.3 Mbps	656.8Mbps

Table 1: **Micro-benchmarks.** *The measurements are taken with a single client, peer and server node connected to the same gigabit Ethernet. Round-trip-time and throughput are measured both from the client to the server and in the reverse direction.*

7.3 Micro-benchmarks

For micro-benchmarks, we connect a client, a peer, and a server to the same gigabit LAN. We measure round-trip-time and throughput from the client to the server, and vice versa; both with and without OverDoSe. Table 1 shows micro-benchmarking results. Round-trip-time is measured by sending 100 `ping` packets; and throughput is measured by running `ttcp` over 120 seconds. The round-trip-time with OverDoSe is approximately twice the round-trip-time without OverDoSe, because with OverDoSe, the packet is routed through an overlay node. The large drop in throughput is caused by OCALA, which is running in user space; similar results are reported by Joseph et al. [12].

7.4 Request Channel Evaluation

To evaluate the request channel, we configure the server to handle request packets at a deterministic rate of 30 requests per second. The server has a finite-length drop-tail queue of length 60, i.e., at most 2 seconds of buffering; and adopts a First-Come-First-Serve (FCFS) policy.

We compare three schemes, each with 10 to 80 clients: 1) OverDoSe's puzzle-based scheme; 2) a Fighting-Fire-with-Fire (FFF) scheme [30]; and 3) non-aggressive legitimate senders under no protection.

Because OverDoSe clients must solve puzzles, a computationally intensive task, we use one physical node to model each client. We use link multiplexing and have every 10 clients share a 100Mbit/s LAN. A customized Click router sits at the border of every LAN and shapes each client's traffic to 1Mbit/s. The same setup is used for the FFF and no defense experiments.

The OverDoSe experiment involves 5 overlay nodes. All clients repeatedly request for new connections as quickly as they can. Each connection setup attempt times out after 10 seconds at which time the client stops the current puzzle computation, requests a new puzzle seed, and starts a new connection setup attempt. The connection setup time for OverDoSe includes the puzzle computation time. In the FFF experiment, all clients send request packets as quickly as their access link bandwidth can sustain. Requests and server responses are tagged with a sequence number to mark which attempt a request packet belongs to. An FFF client floods with requests under sequence number i until it hears a response at which time it completes the current request, and starts to request under sequence number $i + 1$. The connection setup time for attempt i is measured as the time when the first request tagged i is sent out, till when a response tagged i is received. In the experiment with non-aggressive legitimate senders, one client is configured as the legitimate sender, sending requests at a uniform rate of 5 packets per second. The rest of clients are malicious and inject request packets as quickly as they can. A connection setup attempt times out after 10 seconds for a legitimate sender, and a new one is started. In all experiments, request and response packets are 84 bytes in size including IP headers. Ethernet headers and OverDoSe headers are not counted in the 84 bytes.

Request channel experimental results are shown in Figure 3. The experimental results confirm the theoretical analysis in Section 6, showing that FFF and OverDoSe's puzzle scheme have roughly the same mean and variance in terms of connection setup time. However, with a queue of length 60 at the server, FFF introduces a constant queuing delay of 2 seconds, since the queue is always full in FFF. The gap narrows as the number of clients grows because the queuing delay for OverDoSe increases in the presence of more clients. In Theorem A.3 of Section 6, we show that the queuing delay is sublinear with respect to the puzzle computation time, assuming the server has a queue of sufficient length.

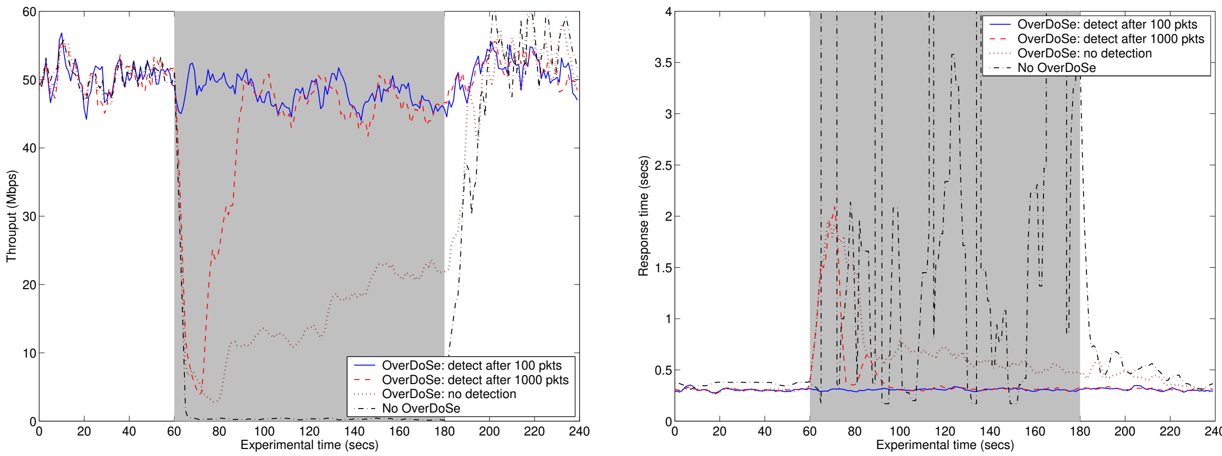
Figure 3(a) shows, unexpectedly, that with 80 clients, the 90th percentile line for FFF increases sharply. This is because the server is using a software interrupt mechanism for reading packets from the Network Interface Card (NIC). Because the clients send small request packets at an aggregate rate of 80Mbit/s, the software interrupt handler is consuming almost the entire CPU. The server request handling process therefore experiences short outages when it fails to get its scheduling quantum. We also run the experiment with 100 clients, but the FFF server is so overloaded that the request handling process gets too few CPU cycles. For this reason, the case for 100 clients is not included in the figure.

Our experiments show that a non-aggressive legitimate sender has low probability of successfully setting up a connection. In the presence of 2 attacking hosts, only 9 out of 917 connection setup attempts completed within the 10 seconds timeout. For this reason, the curve for a non-aggressive legitimate sender is not shown in Figure 3(a) along with FFF and OverDoSe.

In the OverDoSe implementation, the server dynamically adjusts the puzzle level according to real-time load, to keep the request arrival rate below 30 requests per second. The server notifies the peers of the latest puzzle level every second, and the latest puzzle level is returned to the clients along with the puzzle seed. Figure 3(b) plots the average puzzle level as seen by the clients against the number of requesting clients.

7.5 Established Channel Evaluation

To evaluate the established channel under DDoS attacks, we set up a web-server running Apache v2.2.0. The Apache server is unmodified and running with default configuration, except that we lower the client timeout value in to be 20 seconds, so the server does not wait too long for unresponsive clients under DDoS attacks. A simple detection module runs as an OverDoSe plug-in. Once a malicious client is found, the plugging informs the OverDoSe server, which then asks the overlay nodes to blacklist the malicious client. The detection delay is modeled in terms of the number of packets the server receives from each client before an alarm is triggered. Meanwhile, the server runs the default OverDoSe policy as described in Section 4.6, aiming to fair-share the server's access link bandwidth between all legitimate clients.



(a) Throughput over time (b) Response time over time

Figure 4: **Established channel under bandwidth exhaustion attack.** *This figure shows the performance of the established channel under a bandwidth exhaustion attack. Legitimate clients arrive at an Apache web-server according to a Poisson process with an average inter-arrival time of 0.05s. 300 zombie machines coordinate to inject attack packets from 60s to 180s.*

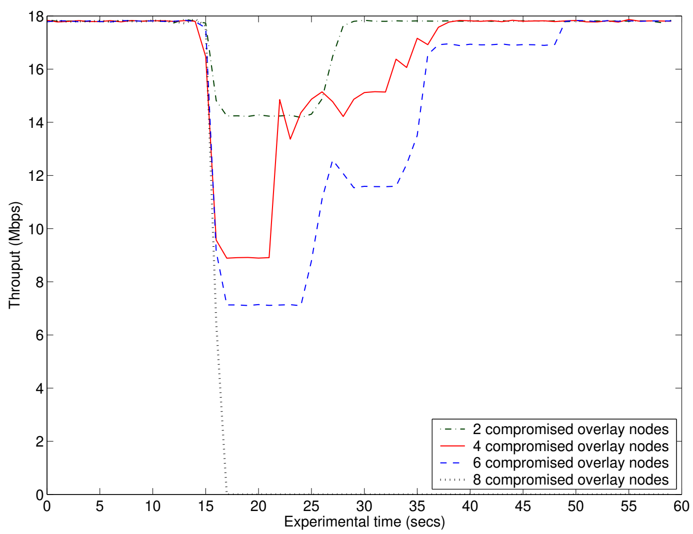


Figure 5: **Packet dropping attack by compromised overlay nodes.** *The attack happens at 15s. Detection and repair occur without interrupting TCP.*

We model an adversary who floods with established traffic. There is no resource contention in the request channel in this experiment, so we turn off the puzzle mechanism, enabling us to multiplex 5 physical nodes to emulate around 300 concurrent legitimate clients. We use httperf [23], a HTTP performance measurement tool, to emulate legitimate clients' behavior. In this experiment, clients arrive according to a Poisson process with an average inter-arrival time of 0.05s. This is achieved by having each of the 5 physical nodes simulate a Poisson process with an average inter-arrival time of 0.25s. During a client's session, it issues 15 requests for a 20KB file. The

15 requests come in a sequence of bursts of length 5, where consecutive bursts are spaced by 5 seconds of user think-time. Clients arrive in a non-blocking fashion; inside each client's session, a client waits for a response to its previous HTTP request before issuing a new one.

We model 300 zombie machines on 15 physical nodes, with each physical node simulating 20 zombie machines. Each client, attacker and legitimate client alike, has 1Mbit/s of access link bandwidth. All zombie machines attack at the same time, injecting packets as quickly as possible to congest the uplink bandwidth to the server.

Figure 4 shows the behavior of the established channel under this attack. With no protection, throughput drops and response time increases sharply in the presence of 300 attackers. The performance of the established channel improves greatly with OverDoSe for two main reasons: 1) *Fair-sharing*. The default policy fair-shares the uplink bandwidth among all legitimate clients. Hence, even without any specialized detection mechanism (the dotted line in Figure 4), the attack loses its strength after having been shaped by the overlay peers according to each client's fair-share bandwidth. 2) *Attack detection and blacklisting*. Once the server detects that a client is misbehaving, it asks the overlay network to blacklist the client; so the attack packets get dropped before the bottleneck link.

7.6 Packet Dropping by Compromised Overlay Nodes

This experiment uses Apache with 8 overlay nodes and 20 clients, who establish a persistent TCP connection with the web server and repeatedly download a 50KB file. Compromised overlay nodes start to drop all packets at the same point of time, and the client-side OverDoSe detects and repairs the connection without disrupting TCP, as shown in Figure 5. The figure plots the throughput over time under a varying number of compromised overlay nodes. In this experiment, once an OverDoSe client detects a packet dropping attack, it tries to repair the connection with two random overlay nodes at a time, and if the attempt fails after a small timeout, it tries with two other overlay nodes. A future version of OverDoSe might use a more sophisticated algorithm that incorporates proximity information [24, 31] or overlay node reputation [13].

8 Related Work

Proposed solutions to DDoS fall roughly into infrastructure-based approaches and overlay-based approaches. The two approaches are complementary; we believe that a promising direction in DDoS defense is to introduce minimal changes to the current network infrastructure, and delegate complex functionalities such as per-flow traffic shaping to a large-scale overlay network.

8.1 Overlay-based DDoS Defense

In SOS [15, 27], Keromytis et al. first propose the idea of using an overlay network to provide DDoS defense. SOS assumes a network filter in the vicinity of the protected server that blocks all incoming packets except those coming from a small subset of egress nodes in the overlay

	isolation mechanism	resilience. agst. compromised peers	overhead	secure conn. setup	modification to infrastructure.
SOS	secret dst. IP	low	high	no	edge
Mayday	secret IPs, ports, etc.	low	low $\sim$ high*	no	edge
Si3	secret dst. IP (physical)**	low	low	rate limit	no (edge)
FoNet	IP level	low	low	partial***	global
Spread-spectrum	secret dst. IP	low	high	no	edge
OverDoSe	IP level	high	low	puzzle	edge

Table 2: **Comparison with related work.** * *Mayday presents several designs allowing us to trade-off performance for security.* ** *The Si3 paper describes two designs, one leveraging the i3 overlay network, the other requiring modifications the the edge ISP's infrastructure.* *** *For services that have restricted access, the server can provide the overlay network with an access control list; however, FoNet does not protect connection setup for open services.*

network. Hence, SOS relies on the identities of the egress nodes being secret; otherwise, zombies could either spoof the source addresses of the egress nodes or selectively attack the egress nodes. Mayday [5] generalizes SOS, and proposes an expanded set of overlay routing and filtering mechanisms, allowing a tradeoff between security and performance. Like SOS, Mayday's filtering mechanism also relies on a secret authenticator that may be globally transferable across different clients. SOS and Mayday protect the secret authenticator and increase resilience against compromised overlay nodes through indirection, random routing or agile secret updates. This increases system design complexity, and creates the possibility of attacks against the overlay routing mechanism itself. Multi-hop overlay routing also increases the end-to-end latency and run-time load at the overlay nodes. A final concern with SOS and Mayday-like approaches is where to deploy the network filter. The network filter itself can be susceptible to DDoS attack if it is situated too close to the victim, while moving it towards the core raises scalability issues.

Stavrou et al. extend SOS and Mayday, spreading traffic across multiple access points in the overlay network for resilience to the "sweeping attack" [28], where malicious zombies flood a subset of the overlay nodes at a time. This approach requires all overlay nodes to share a secret key, and so is susceptible to a single compromised overlay node. However, spread-spectrum communication is potentially valuable for resilience against not only the sweeping attack, but also a performance degradation attack in which malicious nodes drop a subset of legitimate packets or delay packet delivery in a way that is hard to detect.

Si3 [18] uses the i3 overlay network to route packets to the protected server, proposing to implement traffic shaping functionalities on i3 nodes. Si3 assumes that a trusted overlay network hides the protected server's IP; once the server's IP address is revealed, attackers can bypass the overlay and directly attack the server.

FONet [17] proposes a federated overlay network across multiple ASes, offering different types of DDoS resistance to suit different applications. FONet proposes to configure the network infrastructure to isolate senders and protected receivers at the IP level, and to give preference to traffic between the overlay nodes. FONet assumes a trusted overlay network and does not provide techniques to defend against compromised overlay nodes.

8.2 Infrastructure-based DDoS Defense

Firebreak decouples senders and protected receivers at the IP level [10] through Firebreak boxes deployed near the edge of the network, that tunnel packets from sources to the server. Firebreak is complementary to our work, for it can be used to provide the infrastructure support required by OverDoSe.

IP Pushback [21, 11] and traceback [8, 25, 26] use explicit signaling protocols to discover the sources of attack traffic and install filters to remove the attacking traffic as early as possible. Network-capability systems [20, 6, 32, 33], on the other hand, use markings in the packet header to encode permission to send traffic.

9 Conclusion and Future Work

This paper presents OverDoSe, a high-performance and compromise-resilient overlay architecture to protect targeted sites against DDoS attacks. OverDoSe is intended for deployment by a single ISP who wishes to provide DDoS protection as a value-added service to its customers. We implement OverDoSe endhosts on top of the OCALA architecture to support legacy applications. We run a series of DoS experiments on Emulab and validate the design of OverDoSe.

References

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

[2] American registry for internet numbers. `http://www.arin.net/`.

[3] Network emulation testbed. `http://www.emulab.net`.

[4] Martín Abadi, Mike Burrows, Mark Manasse, and Ted Wobber. Moderately hard, memory-bound functions. In *Proceedings of the 10th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, USA, February 2003.

[5] David G. Andersen. Mayday: Distributed Filtering for Internet Services. In *Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS)*, Seattle, Washington, March 2003.

[6] Tom Anderson, Timothy Roscoe, and David Wetherall. Preventing Internet denial-of-service with capabilities. In *Proceedings of Hotnets-II*, November 2003.

[7] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow. *RSVP-TE: Extensions to RSVP for LSP Tunnels*. Internet Engineering Task Force, December 2001. RFC 3209.

[8] Steve Bellovin. ICMP traceback messages. Internet draft, 2000. `http://www.cs.columbia.edu/~smb/papers/draft-bellovin-itrace-00.txt`.

[9] Cynthia Dwork, Andrew Goldberg, and Moni Naor. On memory-bound functions for fighting spam. In *Proceedings of CRYPTO*, August 2003.

[10] Paul Francis. Firebreak: An IP perimeter defense architecture. `http://www.cs.cornell.edu/People/francis/firebreak/hotnetsfirebreak-v7.pdf`.

[11] John Ioannidis and Steven M. Bellovin. Implementing Pushback: Router-Based Defense Against DDoS Attacks. In *Proc. Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, February 2002.

[12] Dilip Joseph, Jayanthkumar Kannan, Ayumu Kubota, Karthik Lakshminarayanan, Ion Stoica, and Klaus Wehrle. OCALA: An architecture for supporting legacy applications over overlays. In *Proc. 3rd Symposium on Networked Systems Design and Implementation (NSDI) (to appear)*, San Jose, CA, May 2006.

[13] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 640–651, New York, NY, USA, 2003. ACM Press.

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

[15] Angelos D. Keromytis, Vishal Misra, and Dan Rubenstein. SOS: Secure overlay services. In *Proc. ACM SIGCOMM*, pages 61–72, Pittsburgh, PA, August 2002.

[16] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[17] Jinu Kurian and Kamil Sarac. FONet: A federated overlay network for DoS defense in the internet. Technical report, University of Texas at Dallas, 2005.

[18] Karthik Lakshminarayanan, Daniel Adkins, Adrian Perrig, and Ion Stoica. Taming IP packet flooding attacks. In *Proceedings of the Second Workshop on Hot Topics in Networks (HotNets-II)*, Cambridge, Massachusetts, November 2003.

[19] Kevin Lam, David LeBlanc, and Ben Smith. Theft on the web: Prevent session hijacking. `http://www.microsoft.com/technet/technetmag/issues/2005/01/SessionHijacking/default.aspx`.

[20] S. Machiraju, M. Seshadri, and I. Stoica. A scalable and robust solution for bandwidth allocation. In *International Workshop on QoS*, May 2002.

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

[22] Joseph Menn. Deleting online extortion. Los Angeles Times, October 25, 2004.

[23] David Mosberger and Tai Jin. httperf – a tool for measuring web server performance. *SIGMETRICS Perform. Eval. Rev.*, 26(3):31–37, 1998.

[24] T. S. Eugene Ng and Hui Zhang. Towards global network positioning. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 25–29, 2001.

[25] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Practical network support for IP traceback. In *ACM SIGCOMM 2000*, pages 295–306, 2000.

[26] Alex C. Snoeren, Craig Partridge, Luis A. Sanchez, Christine E. Jones, Fabrice Tchakountio, Beverly Schwartz, Stephen T. Kent, and W. Timothy Strayer. Single-packet IP traceback. *IEEE/ACM Transactions on Networking*, 10(6), December 2002.

[27] 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. *Computer Networks*, 48(5):781–807, 2005.

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

[29] Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. CAPTCHA: Using hard AI problems for security. In *Proceedings of Eurocrypt*, pages 294–311, 2003.

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

[31] Bernard Wong and Emin Gun Sirer. ClosestNode.com: An open-access, scalable, shared geocast service for distributed systems. *SIGOPS Operating Systems Review*, 40(1), 2006.

[32] Avi Yaar, Adrian Perrig, and Dawn Song. An endhost capability mechanism to mitigate DDoS flooding attacks. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2004.

[33] Xiaowei Yang, David Wetherall, and Thomas Anderson. A DoS-limiting network architecture. In *Proc. ACM SIGCOMM*, Philadelphia, PA, August 2005.

A Appendix

We compare three DDoS defense mechanisms, IP-based fair queuing (IP-FAIR), Fighting Fire with Fire (FFF) and a puzzle scheme like the one adopted by OverDoSe (PUZZLE). We study the time it takes for a legitimate sender to successfully transmit a packet in each scheme; and each scheme is evaluated under two different scenarios, in the presence of a single bottleneck, or multiple bottlenecks.

For simplicity, assume that all Internet hosts are identical in terms of computational power and access link bandwidth. We assume that in the case IP-FAIR, IP addresses must be authentic and cannot be spoofed.

A.1 Single Bottleneck

Assume that the Internet backbone has sufficient capacity such that in the case of FFF, each client is capable of injecting request packets to the bottleneck as fast as their access link bandwidth can sustain.

A.1.1 Mean and Variance

Assume $n-1$ malicious and 1 legitimate hosts try to establish a connection with a server, who is capable of handling new connection requests at a deterministic rate of μ.

Because different variants of the IP-FAIR, FFF and PUZZLE scheme exist, we consider the following specific variantions in the analysis. For FFF, assume that every $1/\mu$ seconds, the server accepts a request at random from all requests that have been received in the past $1/\mu$ seconds. In PUZZLE, the server accepts all requests carrying a valid puzzle solution of level at least ℓ. ℓ is selected in a way such that a client needs n/μ time in expectation to solve a level ℓ puzzle. In IP-FAIR, assume that the server has n queues, and each arriving request enters the queue corresponding to its IP address. The server polls the n queues in a round-robin fashion at a deterministic rate of μ.

Let T_{FFF}, T_{PUZZLE} and $T_{IP-FAIR}$ denote the time for a client to successfully get a request accepted at the server, for FFF, PUZZLE and IP-FAIR respectively.

In all of the analysis below, we do not take into account the take service time for the request.

Theorem A.1.

$$E[T_{FFF}] = E[T_{PUZZLE}] = \frac{n}{\mu}, \qquad E[T_{IP-FAIR}] = \frac{n}{2\mu}$$

Theorem A.2.

$$VAR[T_{FFF}] = VAR[T_{PUZZLE}] = \frac{n^2}{\mu^2} \qquad VAR[T_{IP-FAIR}] = \frac{n^2}{12\mu^2}$$

Proof. (Theorem A.1 and Theorem A.2). In the case of FFF, a client has $1/n$ chance of getting a request accepted every $1/\mu$ seconds. Let T_{FFF} denote the time for a legitimate client to get a request accepted at the server; then $\mu \cdot T_{FFF}$ follows a geometric distribution with $1/n$ being the probability of success in each trial. Hence,

$$\begin{aligned} E[T_{FFF}] &= n/\mu \\ VAR[T_{FFF}] &= VAR[\mu \cdot T_{FFF}]/\mu^2 \\ &= \frac{(1-1/n)}{(1/n)^2}/\mu^2 \approx n^2/\mu^2 \end{aligned}$$

In the case of PUZZLE, let ℓ denote the puzzle level under n concurrent users requesting connection setup. Let δ denote the time to compute one hash function. The probability of hitting a level ℓ solution with one hash computation is $1/2^\ell$, hence, a client need to perform 2^ℓ hash computations in expectation to find a level ℓ solution. The PUZZLE server selects ℓ in a way such that the expected time to solve a level ℓ puzzle equals n/μ; hence,

$$E[T_{PUZZLE}] = \delta \cdot 2^\ell = n/\mu$$

In addition, T_{PUZZLE}/δ follows a geometric distribution with probability $1/2^\ell$ of success in each trial. Hence,

$$\begin{aligned} VAR[T_{PUZZLE}] &= VAR[T_{PUZZLE}/\delta] \cdot \delta^2 \\ &= \frac{1 - 1/2^\ell}{(1/2^\ell)^2} \cdot \delta^2 \approx n^2/\mu^2 \end{aligned}$$

In the case of IP-FAIR, assume the legitimate client arrives at a random point of time, then $T_{IP-FAIR}$ is a uniform distrition on the interval $[0, n/\mu]$. Hence, $E[T_{IP-FAIR}] = n/\mu$, $VAR[T_{IP-FAIR}] = \frac{n^2}{12\mu^2}$. ■

A.1.2 Effect of Queuing for FFF and PUZZLE

The above analysis does not take into account queuing delay in FFF and PUZZLE.

Let FFF-Queue-FCFS be a variant of FFF, where the server has a queue of length Q, and requests are serviced at a deterministic rate of μ in First-Come-First-Serve (FCFS) order.

Corollary 3.

$$\begin{aligned} E[T_{FFF-Queue-FCFS}] &= n/\mu + Q/\mu \\ VAR[T_{FFF-Queue-FCFS}] &= n^2/\mu^2 \end{aligned}$$

Proof. After the attack has run for a sufficient amount of time, the queue is always full. Hence, each request gets served after a constant amount of queuing delay Q/μ. ■

We now analyze the effect of queuing for PUZZLE and show that under FCFS policy and sufficient queue length, the queuing delay is sublinear with respect to the number of attackers.

Assume the server has an infinite-size queue, and adopts a First-Come-First-Serve (FCFS) service policy. Now response time is the sum of the time spent computing the puzzle and the queuing delay. A tradeoff exists between queuing delay and the time of puzzle computation. The server can raise the puzzle level so every one spends more time computing a puzzle solution; and requests arrive at a slower rate; hence, the queue is less filled. Alternatively, the server can lower the puzzle level so every one spends less time computing a puzzle solution, but now they must be subject to a longer queuing delay.

Assume at any point of time, n users concurrently request a connection setup. Define job size S to be the time spent servicing a request. We consider general distributions of job sizes. Let μ denote the average rate at which the server can service requests, hence, $\mu E[S] = 1$. Let $x = \frac{E[S^2]}{2E[S]}$. x is a quantity that roughly characterizes the variance of the job size distribution. Let δ denote the time to perform one hash computation.

Theorem A.3. *To obtain the best expected response time, one must set the puzzle level to be*

$$\ell^* = \log_2 \frac{n + \sqrt{\mu x n}}{\mu\delta}$$

and under ℓ^*, the best expected response time

$$E[T_S]^* = \frac{n}{\mu} + 2\sqrt{\frac{nx}{\mu}}$$

Proof. With one hash computation, the probability of successfully finding a level ℓ solution is $1/2^\ell$. With n requesting clients, the number of clients who can successfully solve a level ℓ puzzle in time δ follows a binomial distribution $B(n, 1/2^\ell)$. Assume the time to compute one hash is infinitesimally small, then the clients' arrival process is approximately a Poisson process with rate $\lambda = \frac{n}{2^\ell \delta}$.

The expected queuing delay for a M/G/1/FCFS system

$$E[T_Q] = \frac{\lambda}{\mu - \lambda} \cdot \frac{E[S^2]}{2E[S]}$$

where S is a random variable representing the job size, i.e., time to service each request.

Response time T_S is the sum of the time spent solving a puzzle T_C, and the queuing delay T_Q. Hence,

$$E[T_S] = E[T_C] + E[T_Q] = \frac{n}{\lambda} + \frac{\lambda}{\mu - \lambda} \cdot x$$

$E[T_S]$ is a function of λ, and to minimize $E[T_S]$, let $\frac{\mathrm{d}E[T_S]}{\mathrm{d}\lambda} = 0$, hence, we get

$$\lambda^* = \frac{\mu}{1 + \sqrt{\frac{\mu x}{n}}}, \qquad E[T_S]^* = \frac{n}{\mu} + 2\sqrt{\frac{nx}{\mu}}$$

Hence,

$$\ell^* = \log_2 \left(\frac{n}{\delta\lambda^*}\right) = \log_2 \frac{n + \sqrt{\mu x n}}{\mu\delta}$$

■

Corollary 4. *Particularly, with deterministic job size,*

$$\ell^* = \log_2 \frac{n + \sqrt{n/2}}{\mu\delta}, \qquad E[T_S]^* = \frac{n}{\mu} + \sqrt{2n}$$

When job size is an exponential distribution,

$$\ell^* = \log_2 \frac{n + \sqrt{n}}{\mu\delta}, \qquad E[T_S]^* = \frac{n}{\mu} + 2\sqrt{n}$$

A.2 Multiple Bottlenecks

In general, consider an open communication network. A legitimate sender wants to send a packet over path $P^* = R_1, R_2, \ldots, R_m$, where $R_1 \ldots R_m$ are m routers on the path. Among the m routers, k routers, $R_{c1}, R_{c2}, \ldots, R_{ck}$, are overloaded. To simplify, we assume that it takes negligible time for a packet to traverse a non-congested router; hence, we only study overloaded routers in the picture.

We formally analyze the following variants of the PUZZLE, IP-FAIR and FFF scheme.

In PUZZLE, assume that a trusted authority distributes the puzzle seeds. Once a sender solves a computational puzzle, all routers on the path are able to verify the solution. All routers have a queue of sufficient capacity and adopts a FCFS policy. Each router independently adjusts the current puzzle level to ℓ^* as in Theorem A.3. We consider an FFF scheme with no queuing. Every $\frac{1}{\mu}$ seconds, each router randomly picks a packet received in the past $\frac{1}{\mu}$ second, and forwards it along. The IP-FAIR scheme in this section behaves exactly the same way as described in the single bottleneck scenario.

Assume $n-1$ malicious senders and 1 legitimate sender. Let T^*_{FFF}, T^*_{PUZZLE} and $T^*_{IP-FAIR}$ denote the time for the legitimate sender to successfully transmit a packet across multiple bottlenecks.

Lemma A.4. *Let λ_{cj} denote the total rate of traffic arriving at R_{cj}, let λ_g be the rate at which the legitimate sender injects packets.*

$$E[T_{FFF}] = \frac{1}{\lambda_g} \cdot \Pi_{j=1}^{k} \frac{\lambda_{cj}}{\mu_{cj}}$$

Proof. The probability for each packet to successfully transmit is $\Pi_{j=1}^{k} \frac{\lambda_{cj}}{\mu_{cj}}$. ■

Theorem A.5. *Regardless of the network topology and the adversary's strategy,*

$$E[T_{PUZZLE}] = O(\frac{n}{\mu^*} + k\sqrt{n})$$

where $\mu^* = \min_{j=1}^{k} (\mu_c j)$.

Proof. Each router picks the optimal puzzle level ℓ^* as described in Theorem A.3, then the sender spends expected $\frac{n}{\mu^*}$ to computate a puzzle solution to get accepted at all routers, and has $O(\sqrt{n})$ queuing delay at each router. ■

Theorem A.6. *Regardless of the network topology and the adversary's strategy,*

$$E[T_{IP-FAIR}] = O(\frac{k \cdot n}{\mu^*})$$

Proof. Each host has a dedicated queue at each router. Hence, the legitimate client's packet never gets dropped, but has an expected queuing delay of at most $\frac{n}{\mu_{cj}}$ at R_{cj}. ■

Theorem A.7. *Under FFF, there exists a network configuration and adversary strategy, such that the legimate sender can only transmit a packet in $\Omega(n)$ time.*

Proof. Consider a scenario where all hosts have equal access-link bandwidth B; and other than access-link bandwidth restrictions, and the bottlenecks present on P^*, the rest of the network has sufficient capacity. An adversary with n compromised hosts can use $n/2$ hosts to sustain an arrival rate of $Bn/2$ at R_{c1}, and the other $n/2$ hosts to sustain an arrival rate of $Bn/2$ at R_{c2}. According to Lemma A.4, the legitimate sender needs at least $\frac{1}{B} \cdot \frac{(Bn/2)^2}{\mu_{c1}\mu_{c2}}$ expected time to successfully transmit a packet. ■