The Design and Implementation of the A^3 Application-Aware Anonymity Platform

Micah Sherr[a,*], Harjot Gill[b], Taher Aquil Saeed[c,1], Andrew Mao[d], William R. Marczak[e], Saravana Soundararajan[f,1], Wenchao Zhou[a,1], Boon Thau Loo[b], Matt Blaze[b]

[a]*Georgetown University, Washington, DC*
[b]*University of Pennsylvania, Philadelphia, PA*
[c]*DramaFever Corporation, New York, NY*
[d]*Harvard University, Cambridge, MA*
[e]*University of California, Berkeley, CA*
[f]*Oracle Corporation, Redwood Shores, CA*

Abstract

This paper presents the design and implementation of *Application-Aware Anonymity* (A^3), an extensible platform for rapidly prototyping and evaluating anonymity protocols on the Internet. A^3 supports the development of highly tunable anonymous protocols that enable applications to tailor their anonymity properties and performance characteristics according to specific communication requirements.

To support flexible path construction, A^3 uses a declarative language to compactly specify path selection and instantiation policies. We demonstrate that our declarative language is sufficiently expressive to encode novel multi-metric performance constraints as well as existing relay selection algorithms employed by Tor and other anonymity systems, using only a few lines of concise code. We experimentally evaluate A^3 using a combination of trace-driven simulations and a deployment on PlanetLab, as well as a case-study of A^3-enabled voice-over-IP communication. Our experimental results demonstrate that A^3 can flexibly and efficiently support a wide range of path selection and instantiation strategies at low performance overhead.

Keywords: Anonymity, Extensibility, Testbeds, Declarative Networking

1. Introduction

In the past two decades, there has been intense research [5, 12, 13, 16, 38, 52, 53, 62, 67] in designing systems that enable parties to communicate anonymously in the presence of eavesdroppers. Typically, these systems achieve anonymity by

[*]Corresponding author. Author address: 3700 Reservoir Rd NW, Room 337, Washington DC 20057 USA. Telephone: 202-687-4381. Fax: 202-687-1835. Email: `msherr@cs.georgetown.edu`.
[1]Worked performed while at the University of Pennsylvania.

sending a message through a path of relays before delivering it to its final destination. Broadly speaking, recent innovations have improved *relay selection* – choosing a path of relays to provide high anonymity and good performance – and *path instantiation* – establishing necessary state at each relay to enable anonymous communication [1, 11, 39, 58, 63].

Despite the proliferation of proposed techniques, we argue that no *one-size-fits-all* anonymity system exists. The appropriate relay selection and path instantiation strategies can vary according to application requirements, performance characteristics, and additional constraints imposed by the underlying network. For example, in the context of relay selection, an anonymous video conferencing system may be willing to achieve weaker anonymity in exchange for a path that meets its high-bandwidth, low-latency performance demands. In contrast, an anonymous email system may require very strong anonymity guarantees while imposing no constraints on bandwidth or latency.

Similarly, several path instantiation approaches exist. Onion Routing [67] and Tor's *telescoping* scheme [12] build paths by recursively encrypting and shipping key material to their constituent nodes. The former constructs anonymous paths that have constant length over their lifetime, while the latter adds the ability to extend existing anonymous paths. On the other hand, the *Crowds* [52] approach relies on the network to make routing decisions on behalf of the source. Crowds is best suited for an environment where source routing is not available and intermediate relay nodes can be trusted with the identity of the receiver.

This paper presents the design and implementation of *Application-Aware Anonymity* (A^3): an anonymity platform that enables developers to rapidly prototype and evaluate anonymity protocols. The high level goal of A^3 is to enable researchers to develop and study strategies that intelligently trade off between performance and anonymity; once understood, these strategies can then be applied to deployed anonymity services. A^3 aims to support a wide range of anonymity-based networked services with different application-specific constraints. Our target users are primarily researchers of anonymity systems, who can leverage A^3 in a *policy-driven* fashion by specifying path instantiation and relay selection techniques that meet their applications' performance and anonymity requirements.

Importantly, A^3 is not intended to replace existing anonymity systems such as Tor. Rather, A^3 provides a language and platform for quickly formulating and testing anonymity protocols in a controllable and methodical manner.

Motivation. Anonymity protocols are often complex, and even small changes in relay selection and path instantiation may significantly impact performance and anonymity [1, 58, 63]. There is a critical need to understand the effects of anonymity protocols *before they are deployed on a live network*. Although specialized simulators [21, 39, 44] and emulators [3, 20] exist, they are often closely and inseparably tied to a particular approach (e.g., onion routing) and rely on hard-coded relay selection and path instantiation policies that are difficult to customize. Anonymity researchers and protocol designers currently lack a flexible platform for quickly prototyping, deploying, and evaluating novel and diverse algorithms. A^3 provides (1) a simulation environment, (2) an implementation framework, and (3) a language for specifying two core aspects of anonymity system design: how relays are chosen and how paths are constructed. A^3 allows developers to quickly test their protocols under either simulation or deployment, before undertaking the time-consuming task of modifying existing systems or constructing new software.

Contributions. We make the following contributions in this paper:

- **Declarative relay selection and path instantiation:** We propose the use of *declarative networking* [27, 29] techniques in A^3 as a policy engine for specifying and executing relay selection and path instantiation policies. Declarative networking provides a high-level logic-based framework that can efficiently execute a high-level protocol specification using orders of magnitude less code than an imperative implementation. Our proposed A^3LOG declarative language extends previous declarative networking languages with constructs that are added specifically to enable the specification of anonymity systems. For example, we have integrated the ability to specify user-defined cryptographic primitives for secure communication. We have also adapted recently proposed extensions for declarative network composition [35] to enable us to develop reusable components that are ideal for specifying and customizing anonymous routing. We demonstrate how these extensions enable the concise expression of relay selection and path instantiation algorithms. By providing a flexible framework for realizing both relay selection and path instantiation policies, A^3 enables the rapid development, deployment, and testing of both existing and novel anonymity protocols.

- **Extensible anonymity via flexible relay selection.** A^3 is sufficiently extensible to support both traditional *node-based* as well as recently proposed *link-based* [57, 58] relay selection strategies. Node-based strategies select relays with desirable node properties (usually bandwidth), whereas link-based strategies bias relay selection in favor of link characteristics such as latency, AS hop count, or jitter. We demonstrate that both link- and node-based relay selection strategies, including those used by Tor and other systems, can be concisely represented in a few lines of A^3LOG code. Moreover, one can combine node and link-based metrics to implement hybrid relay selection policies. In many instances, we can quantify the *selectivity* of the tested relay selection strategy and provide the developer with feedback as to the strategy's anonymity properties.

- **Implementation and experimentation.** We describe the design and implementation of the A^3 platform, developed using the RapidNet declarative networking engine [30, 40]. We conduct extensive trace-driven network simulations in the ns-3 network simulator [43] to demonstrate that A^3 flexibly supports a wide range of path selection and instantiation strategies at low performance overhead. We additionally describe a PlanetLab deployment of A^3 and show how our platform may be easily and mostly transparently employed to study the effects of relay selection and path instantiation policies *when anonymizing legacy applications*. In particular, we present a case study that examines how A^3 may be used to evaluate the benefits of real-time constraint-based relay selection strategies for anonymizing voice-over-IP (VoIP) communication. Finally, we introduce the

RapTor system that integrates A^3's relay selection engine with Tor, permitting customizable anonymous paths on the live Tor network.

Code release. A^3 is well-suited for security and anonymity researchers, allowing one to quickly develop and deploy new relay selection and path instantiation policies and techniques. The goal of our work is to provide an extensible platform for anonymity researchers, and to this end, we have released the source-code of our A^3 system; the software is available for download at `http://a3anonymity.com`.

2. Related Work

We begin by reviewing existing approaches for enabling anonymous communication (Section 2.1), comparing A^3's features with existing anonymity simulators, emulators, and testbeds (Section 2.2), and describing how this paper improves upon our prior work with A^3 (Section 2.3).

2.1. Anonymous Routing Overlays

To support diverse applications, the Internet uses a simple routing scheme in which packets are forwarded on a best-effort basis towards their intended destinations. With the exception of fragmented portions of the Internet that support IP quality-of-service features, applications usually have little control over the performance aspects of their network connections. An *overlay network* built on top of the Internet routing infrastructure can allow users to exercise greater control over the manner in which their messages are relayed, as forwarding can be based on application layer information. When combined with source-routing, these networks allow applications the ability to select paths that meet their specific requirements.

Overlay networks may also enable anonymous routing on the Internet. For example, Tor [12], Onion Routing[2] [67], Crowds [52], Tarzan [16], Hordes [62], JAP [13], and MorphMix [53] (among many others) utilize application-layer overlay routing. These anonymity systems exploit two features of overlay networks: (i) the ability to obfuscate the addresses of the *initiator* (sender) and *responder* (receiver) while still providing reliable message delivery; and (ii) in some instances, the ability to produce anonymous paths that achieve some desirable property (usually high bandwidth) [12, 16, 63].

A large volume of existing literature examines methods for generating high performance anonymous paths. Tor [11, 12] attempts to achieve high bandwidth paths by imposing a probability distribution over the set of potential anonymous relays. The probability of a relay being selected is proportional to its advertised bandwidth. Murdoch and Watson have demonstrated that such a strategy delivers both performance and strong anonymity [39]. Snader and Borisov offer refinements to Tor's strategy,

[2]Throughout this paper, we differentiate between traditional onion routing [67] in which the initiator establishes a path using a single multi-layered onion and Tor's telescoping strategy [12] in which anonymous circuits are incrementally extended. Tor's technique borrows heavily from and has similar anonymity properties to the original onion routing approach, but offers greater flexibility.

allowing an initiator to *tune* the performance (quantified in their work as bandwidth) of its anonymous paths [63] by defining the degree to which relay selection is biased in favor of bandwidth. At one extreme, initiators consistently choose relays with the highest bandwidth, achieving very high bandwidth paths at the expense of allowing a small subset of relays to view a significantly disproportionate amount of anonymous traffic [2, 45, 58]. At the other extreme, initiators may opt to favor anonymity while disregarding performance by selecting relays uniformly at random. Given the bandwidth requirements of the particular application, Snader's and Borisov's technique enables the sender to select a point in this anonymity-vs-performance spectrum. Similarly, we previously introduced tunable *link-based* routing [58], where initiators can weigh relay selection based on the expected e2e cost computed using link performance indicators such as latency, AS hop count, and jitter. We showed that biasing selection on link characteristics offers some anonymity benefits over node-based (i.e., bandwidth-weighted) techniques, since link-based routing reduces "hotspot nodes" in the network that appear attractive to all initiators.

TorCtl [70] is a Python library for interfacing with an instance of Tor. Compared with TorCtl, A^3's use of declarative networking provides a higher level abstraction for selecting and setting up paths as logical specifications. TorCtl provides a limited API for selecting the next relay node. As such, users of TorCtl still need to write Python code to implement policies for relay selection. One can in principle compile our declarative specifications into Python code written by TorCtl. In addition to a code size reduction, our approach also derives other benefits of declarative programming — namely, the ability to verify and debug rules using a high level language. Finally, since TorCtl is specific to Tor, it does not support flexible path instantiation.

To our best knowledge, A^3 is the first system that provides *extensible anonymous routing*. Unlike the above anonymity services that provide hardcoded relay selection and path instantiated functionality, our anonymity platform allows researchers to efficiently formulate and evaluate different strategies. In particular, A^3 enables developers to construct and test "application-aware" policies that are compatible with the underlying application's communication requirements.

2.2. Anonymity Simulators, Emulators, and Testbeds

Perhaps the most significant challenge of developing techniques for private communication is conducting an analysis or experiment that accurately reflects how the system will perform *in reality*. At the extreme, experimentation can be performed on a deployed network such as Tor; however, such an approach raises ethical concerns [64] since experimenting on a live network risks compromising the anonymity of its users. Moreover, modifications to algorithms and protocols cannot be easily propagated to all of the network's participants, enabling only small segments of the network to run any modified code.

At the other extreme, mathematical modeling ensures that live users' anonymity is not breached. However, the complexities involved in anonymous networking require such analytic work [6, 39] to rely on simplifications which may not hold in an actual deployment. Similarly, there are also network simulators [21, 39, 44] that attempt to model the behavior of anonymity systems. These approaches often trade off realism for efficiency, simplifying the behavior of both the network and end hosts in order to efficiently produce simulations. Network emulators [3, 20] adopt a hybrid approach, running actual software over an emulated network.

A^3 functions in either simulation or deployment mode. The former allows "network"-wide testing of novel relay selection and path instantiation protocols using user-provided network topology specifications. (As described in Section 6, A^3 is constructed using the ns3 [43] network simulator, permitting both high- and low-level network simulations.) Additionally, A^3's deployment mode permits testing on actual networks. In contrast to other approaches that model, simulate, or emulate a particular system (with its particular relay selection and path instantiation policies), A^3 allows for rapid prototyping and evaluation of novel policies.

2.3. *Improvements over Prior A^3 Papers*

This paper is an extension of our prior conference publication [61] on applying declarative techniques to enable anonymity systems. Extensions that we have made in this paper include:

- **Adapter interface specification:** In Section 3.1, we provide a specification of the adapter interface used by A^3 to communicate with *Information Providers* for various scalable network measurements.
- **Improved implementation:** In Section 6, we describe our implementation, which is based on a vastly enhanced architecture developed using the RapidNet declarative networking system. The new system not only outperforms its predecessor P2 system [46] used in our prior system [61], but also fixes execution ambiguities [34] through the use of atomic rule execution and local fixpoint computations in between external network events. Language constructs on secure communication and composition are fully integrated into the A^{3}Log language and runtime system. Moreover, we have added support for integration with the emerging ns-3 network simulator [43], enabling more realistic network simulations in our evaluation. Finally, we have introduced transparent tunneling functionality that provides support for legacy applications, without the need to recompile source code or reconfigure applications to use proxy servers.
- **Application-driven evaluation:** Section 7 presents more comprehensive experiments that extend earlier results based on micro-benchmarks [61]. In particular, we evaluate the performance and anonymity offered by A^3 for an existing application – a SIP [54] VoIP softphone – with real-time latency, jitter, and bandwidth requirements on the PlanetLab testbed using actual audio streams.
- **Comparison with other anonymity testbeds:** We have expanded our related work (Section 2.2) to contrast the A^3 platform to existing efforts to model, simulate, and emulate anonymity networks.
- **Integration with Tor:** We introduce the *RapTor* system that integrates A^3's flexible relay selection engine with Tor. We execute RapTor on the live Tor network and validate its capabilities to select tunable routes at low overhead.

- **Code release:** An open-source code release is now available under the GNU GPLv2 license at `http://a3anonymity.com`. As of June 2012, there have been more than 1000 downloads of the software.

3. A^3 Design Goals and Architecture

A^3 allows an investigator to provide a *relay selection policy* that precisely specifies the manner in which relays are chosen for anonymous paths. This is in contrast to existing anonymity systems in which an immutable relay selection algorithm is hard-coded into the anonymity service. The A^{3}Log policy language (Section 4) enables the researcher to not only intelligently tune relay selection in favor of performance or anonymity [58, 63], it also allows her to easily define her individual characterization of performance in terms of bandwidth, latency, loss, jitter, etc., or some combination of the above. In addition to supporting flexible relay selection, A^3 also permits the customization of *path instantiation policies* (Section 5).

A^3's use of declarative networking provides the capability for investigators to rapidly customize and refine the policies that best meet their underlying applications' constraints. Declarative networking technologies have been widely applied to a wide range of domains in distributed systems programming, including fault tolerance protocols, cloud computing, sensor networks, overlay network compositions, anonymity systems, mobile ad-hoc networks, secure networks, network configuration management, network forensics, optimizations, and as a basis for course projects in a distributed systems class (cf. Loo *et al.* [28] for a survey of recent use cases).

In addition to ease of implementation, another advantage of the declarative networking approach is its amenability to formal and structured forms of correctness checks. These include the use of theorem proving [72], algebraic techniques for constructing safe routing protocols [73], and runtime verification [78]. These formal analysis techniques are strengthened by recent work on formally proving correct operational semantics of NDlog [42]. Finally, the dataflow framework used in declarative networking naturally captures information flow as distributed queries, hence providing a natural way to use the concept of *network provenance* [77] to analyze and explain the existence of any network state, a useful feature for debugging anonymity policies.

The anonymity offered by an anonymous path depends in no small part on the mechanisms for relay selection [39, 45, 58] and path instantiation. A thorough review of the performance and anonymity properties of various relay selection and path instantiation algorithms is outside the scope of this paper. Our goal in this paper is to provide a flexible architecture for developing, testing, and studying path strategies and implementations.

System Overview. Figure 1 shows the architecture of an A^3 client running on the initiator's host. An investigator provides relay and path instantiation policies; the *Relay Selection Engine* interprets the relay selection policy and applies that policy to produce (but not instantiate) an anonymous path consisting of relays from the *Local Directory Cache*. To populate the cache, the A^3 instance periodically contacts a *Directory Server* to ascertain membership information – that is, a listing of available relays – and, optionally, one or more *Information Providers*. Information Providers are data aggregating

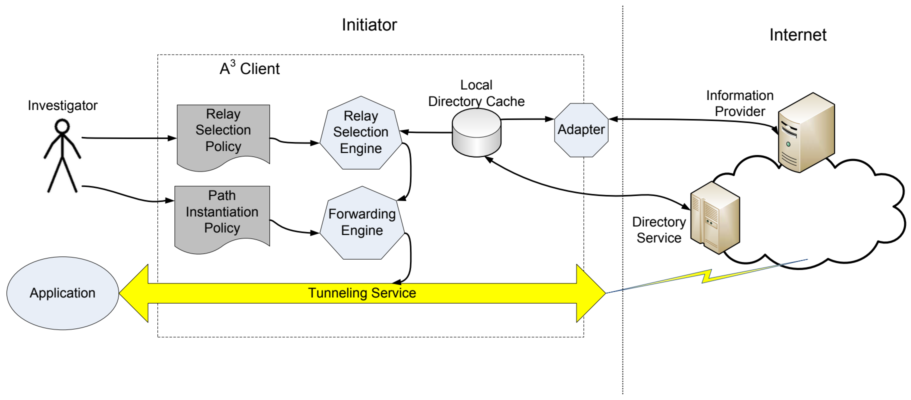


Figure 1: The A^3 architecture.

services that report performance characteristics of relays (e.g., bandwidth) and links (e.g., the latency between two relays). The Relay Selection Engine uses cached data to generate paths that conform to the provided relay selection policy.

Once the Relay Selection Engine produces a path, the *Forwarding Engine* instantiates that path according to the provided path instantiation policy. After path establishment, a *Tunneling Service* on the local machine transparently intercepts the application's traffic and relays it through the anonymous path. Likewise, incoming data from the anonymous channel is forwarded through the Tunneling Service to the application. The tunneling service emulates a network interface, permitting legacy applications to communicate anonymously via A^3.

Below, we describe each component of A^3 in more detail.

3.1. Information Providers

To support non-trivial relay selection policies, A^3 makes use of *Information Providers* (also referred to as *Providers*) that aggregate node and/or link performance data. Policies may utilize such information to more precisely define their requirements (e.g., "include only relays that have been online for at least an hour").

A^3 imposes few restrictions on the types of Information Providers. Each Information Provider is interfaced through an *adapter* that resides on the A^3 relay. Adapters are small programs or scripts that periodically query a Provider for new information, storing the results in the Local Directory Cache. Our current implementation includes adapters for the Vivaldi [9] embedded coordinate system (described below) and CoMon [47], although others can be easily constructed.

3.1.1. Provider Interface

A^3 defines an interface (Table 1) through which a client may communicate with an Information Provider. Clients use the interface both to send updates to the Providers when their positions change as well as to query information about potential relays.

Function	Description
`post(location,[credentials])`	updates the Provider with the client's specified `location`
`get(node_id,[credentials])`	returns the location of the specified node, or $\emptyset$ if no such node or location is known
`find(filter,[credentials])`	returns the node identifiers and locations of nodes that match the provided query `filter`

Table 1: Adapter interface. The optional `credentials` argument contains sufficient information (e.g., digital signatures) to authenticate the client to the Information Provider.

The interface serves as an abstraction layer, enabling A^3 to retrieve data from (and later base routing decisions off of) *any* searchable information service. By implementing the three functions described in Table 1, lightweight adapters can easily be constructed, effectively serving as translation services between A^3 and an Information Provider. Note that the interface does not restrict the Information Provider to any particular architecture – for example, our current implementation of A^3 (Section 6) uses adapters to query information from both centralized (CoMon [47]) and fully distributed (Chord DHT [66]) sources. The optional `credentials` argument may be used to authenticate the client (for example, using a digital signature). Authentication and authorization processes are Provider-specific, and the Provider should define its expected format for the `credentials` argument.

3.1.2. Example Information Providers

Below, we highlight the flexibility of the A^3 architecture by describing several potential Information Providers:

Network Coordinate Information Providers. Traditional anonymous relay selection algorithms (most notably Tor [12] and the refinement proposed by Snader and Borisov [63]) bias selection in favor of relays that advertise high bandwidths. However, in addition to bandwidth, an application may also prefer paths that exhibit low latency. Unlike bandwidth, latency is not a *node characteristic* that can be associated with an individual relay. Rather, latency is a *link characteristic* that has meaning only when defined in terms of a connection between a pair of relays.

Given that there are $\binom{N}{2}$ links in a network composed of N relays, maintaining up-to-date link characteristics for all links in the anonymity network is infeasible. One practical solution to succinctly capture pairwise link latencies is via the use of *virtual coordinate embedding systems* (also called network coordinate systems). These distributed algorithms enable the pairwise latencies between all participating relays to be estimated to high accuracy with low overhead. Network coordinate systems, such as Vivaldi [9], PIC [8], NPS [41], and Big Bang Simulation [56] map each relay to multidimensional coordinates such that the Euclidean distance between any two relays' coordinates corresponds to the latency between the pair. By representing pairwise distances using N virtual coordinates, these systems effectively linearize the information that must be stored and maintained by the Information Provider.

Coordinate systems use distributed algorithms in which each participant periodically measures the distance between itself and a randomly selected peer. By comparing the empirical measurement with the Euclidean distance between the two nodes' coordinates, the relay can adjust its coordinate either towards (in the case of over-estimation) or away from (for under-estimation) the neighbor's coordinate. Although network distances cannot be perfectly represented in Euclidean space due to the existence of triangle inequality violations on the Internet, virtual coordinate systems efficiently estimate pairwise distances with very low error [9]. Since network coordinate systems require only periodic measurements (on the order of a single ping every 15 seconds), participation in the system does not incur a significant bandwidth cost.

A *Network Coordinate Information Provider* maintains the current coordinates of the relays in the A^3 network. Relays periodically send updates to the Provider whenever its coordinate changes from its last reported value (e.g., by more than 10ms).

Unfortunately, the distributed nature of coordinate systems make them particularly vulnerable to insider manipulation [7]. Recent studies [23] on Vivaldi have shown that when 30% of nodes lie about their coordinates, Vivaldi's accuracy decreases by a factor of five. Attacking the coordinate system provides a vector for an adversary to either prevent high performance routing or bias routing decisions in favor of relays under their control.

Several schemes have been proposed [4, 8, 22, 55, 59, 75] for securing coordinate systems. These schemes are typically applied on top of the embedding system in order to ensure the veracity of advertised coordinates. They rely on spatial and temporal heuristics to spot false coordinate advertisements [75], use machine learning techniques [4] to thwart so-called *frog-boiling attacks* [7], utilize a small set of trusted surveyor nodes [22, 55], or assess coordinate accuracy using a distributed voting protocol [59]. These techniques can be used together with A^3; A^3 is agnostic to the specific security technique being employed, so long as it can query coordinates using the Adapter Interface API.

Relay-Assisted Information Providers. Relays have access to a significant number of local performance indicators. For instance, a relay can measure its current upstream and downstream throughput, processor usage and available memory, and estimate its bandwidth capacity. Such information can be collected and stored in a *Relay-Assisted Information Provider*. The CoMon Monitoring Infrastructure [47] that operates on the PlanetLab testbed [48] is one such example.

As has been pointed out by Øverlier [45] and others [2, 58, 63], malicious relays may purposefully attract a large fraction of anonymous traffic by falsely advertising favorable performance, consequently increasing their view of traffic in the anonymous network. To mitigate such attacks, Snader and Borisov propose the use of *opportunistic measurements* in which relays report the observed throughput of their network peers. The Provider (or in their case, the directory service) reports the median of the reported measurements [63]. Similar protection schemes—where relays report the bandwidth and responsiveness of peer relays with whom they interact—are applicable to A^3 Information Providers. Certain metrics (e.g., memory usage) cannot easily be probed by remote parties, and if reported by Information Providers, should be treated with some degree of skepticism by the relays that make use of them.

Other Potential Information Providers. There have been a number of proposed systems (e.g. iPlane [32, 33], IDMaps [14], OASIS [15], Meridian [74], and Sequoia [50]) that attempt to provide estimates of latency and (in some cases) bandwidth between arbitrary hosts, by succinctly capturing the structure of the Internet. Such systems have typically been deployed to provide proximity-based routing [31, 60], neighbor selection in overlays [10], network-aware overlays, and replica placement in content-distribution networks. However, these systems are also applicable to anonymity services in which the initiator is interested in discovering the cost of routing through a particular relay. All that is required for an investigator to use information from such a system in A^3 is to write an adapter to its interface.

3.2. Other Components of A^3.

We briefly describe the other components of the A^3 system:

Directory Service. Node discovery is facilitated by a *Directory Service* (or simply Directory) that maintains membership information on all the relay nodes currently participating in the A^3 network. Relays that join the network publish their network addresses and public keys to the Directory Service. Initiators periodically poll the Directory to discover peer nodes that may potentially be used as routers in anonymous paths.

Local Directory Cache. The *Local Directory Cache* periodically queries and stores performance data from Information Providers. The rate at which the cache polls Providers affects both the freshness of cached data as well as the relay's communication overhead. The tradeoff between update intervals and bandwidth costs depends on the rate at which performance characteristics change in the network, and is explored in more detail in Section 7.

Relay Selection Engine. At runtime, an investigator provides relay selection policies. Using the information stored in the Local Directory Cache, the *Relay Selection Engine* forms routes according to the specified policy. The participants of generated paths are relayed to the Forwarding Engine that instantiates the path. Relay selection is explored in more detail in Section 4.

Forwarding Engine. The *Forwarding Engine* consists of a declarative networking engine enhanced with low-level cryptographic primitives. The Forwarding Engine provides methods for composing these primitives to form high-level operations. For example, the one-way authentication and symmetric key-exchange primitives used in Tor path instantiation are constructed by composing RSA digital signatures with Diffie-Hellman key exchange.

The Forwarding Engine instantiates the anonymous path provided by the Relay Selection Engine according to rules specified in the *path instantiation policy*. Additionally, the Forwarding Engine supports message relay over instantiated anonymous paths. That is, the Forwarding Engine is used both to construct paths as well as to relay application messages over the anonymous route. Often, the rules for path instantiation and message relay are distributed recursive queries. We revisit the routing engine in Section 5.

Tunneling Service. The *Tunneling Service* provides legacy application support, enabling an investigator to experiment with different relay and path instantiation policies using legacy applications, without requiring recompilation or application reconfiguration. The Tunneling Service operates by emulating a network interface on the client's host. Local routing rules redirect application traffic through the virtual interface, where communication is tunneled through the anonymous circuits that are constructed and maintained by the Forwarding Engine.

4. Relay Selection Policies

In this section, we demonstrate how a variety of strategies used by the relay selection engine can be expressed using the A^{3}Log declarative networking language. Our goal is to highlight the flexibility, ease of programming, and ease of reuse afforded by a declarative query language. We show that routing protocols can be expressed in a few A^{3}Log rules, and additional protocols can be created by simple modifications to previous examples.

4.1. A^{3}Log

A^{3}Log extends existing work on *declarative networking* [29]. The high level goal of declarative networking is to enable the construction of extensible architectures that achieve a good balance of flexibility, performance, and safety. One specifies a declarative networking protocol as a set of queries in a high-level language. Because such a specification expresses *what* a program achieves as opposed to *how* it operates, declarative queries are a natural and compact way to implement routing protocols and overlay networks [25–27, 29].

Our A^{3}Log declarative language is primarily based on Datalog [49]. A Datalog program consists of a set of possibly recursive declarative queries, also referred to as *rules*. Each rule has the form `q :- p1, p2, ..., pn.`, which can be read informally as "`p1` and `p2` and ... and `pn` imply `q`". Here, `q` is the *head* of the rule and `p1, p2, ..., pn` is a list of *literals* that constitutes the *body* of the rule. Literals are either *predicates* (also called *relations*) with *attributes* (variables or constants) or boolean expressions that involve function symbols (including arithmetic) applied to attributes. A^{3}Log extends Datalog by allowing the specification of rules with multiple head literals, i.e. rules of the form `q1, q2, ..., qm :- p1, p2, ..., pn.`. A rule of this form is short-hand for the set of *m* rules where the i^{th} rule is of the form `qi :- p1, p2, ..., pn`. A Datalog program is said to be recursive if a *cycle* exists through any predicate – such as when a predicate that appears once in a rule's body appears in the head of the same rule. A recursive Datalog program is continuously executed until a *fixpoint* is reached, i.e. no new facts are derived.

The order in which the rules are presented in a program, as well as the ordering of predicates in a rule body, is semantically immaterial. By convention, the names of predicates, function symbols, and constants begin with a lowercase letter, while variable names begin with an uppercase letter. Function calls are prepended with `f_`. An aggregate construct, which defines an operation on multiple results from the rule body, is represented as a special function in the rule head with its attribute variables enclosed in angle brackets (`<>`). To support anonymous relay selection, A^{3}Log enhances Datalog with *cryptographic functions*, *random* and *ranking aggregates*, and *composability*.

Algorithm	**Description**	**Benefits**	**Example Usage**	**A^{3}Log Rules**
RANDOM	Relays selected uniformly at random	Offers strong anonymity	Email mixing	5
TOR	Relays biased proportionally to bandwidth	High bandwidth and network utilization [39]	Web browsing	5
SNADER-BORISOV	Tunable bias in favor of bandwidth	Tunable anonymity and performance	File transfer	6
CONSTRAINT	Specification of end-to-end performance requirements	Expresses communication requirements	VoIP	6
WEIGHTED	Bias relay selection in favor of link-properties	Extends support to multiple metrics (latency, jitter, etc.)	Streaming multicast	6
HYBRID	Combines above techniques	Supports highly flexible routing policies	Video conferencing	varies

Table 2: Example relay selection policies.

Example: All Pairs Reachability. We illustrate A^{3}LOG using a simple example of two rules that compute all pairs of reachable nodes in a network.

```
r1 reachable(S,N) :- neighbor(S,N).
r2 reachable(@N,D) :- neighbor(S,N), reachable(S,D).
```

Rules `r1` and `r2` specify a distributed transitive closure computation that derives all pairs of nodes that can reach each other through paths of neighbors. The rules take as input a local `neighbor` table stored at each node `S`. Each fact in the `neighbor(S,N)` relation denotes that `N` is a neighbor of `S`. Rule `r1` computes all pairs of nodes reachable within a single hop from all input neighbor links. Rule `r2` expresses that "if `N` is the neighbor of `S`, and `S` can reach `D`, then `N` can reach `D`." The output of interest is the set of all `reachable(S,D)` facts, representing reachable pairs of nodes from `S` to `D`. By modifying this simple example, we can construct more complex routing protocols, such as distance vector and path vector routing protocols.

Rule `r2` introduces the *location specifier*, which is the argument prefixed with the `@` symbol. This argument denotes the location of each fact derived by the rule head. For example, in rule `r2`, all derived `reachable(N,D)` facts are exported based on the address encoded in their first attribute (`@N`). This means that the execution of rule `r2` results in each node propagating its reachability information to its neighbors until a distributed fixpoint is reached—i.e., until no node can derive any new facts.

4.2. Example Relay Selection Policies

We next present sample relay selection policies written in A^{3}LOG. In all our examples below, we assume that node and link information obtained from Information Providers is stored on each node in a table called `node`. The `node` table is indexed by the IP addresses of remote peers. These node and measurement data are then used as input to A^3 for executing A^{3}LOG rules that will select candidate relays. In all our example programs, the output of interest is an `ePathResult(Src,Dst,P)` tuple, where `P` is the list of relay node tuples (a relay node tuple includes includes the node's address, bandwidth, coordinates and any other attributes relevant to the query) from `Src` to `Dst`. The relay selection strategies described in this section are summarized in Table 2.

4.2.1. Node-based Relay Selection

As its name suggests, node-based relay selection selects nodes based on node characteristics (in most cases, bandwidth).

Random Selection (Random Policy). Given a request to generate a path from `Src` to `Dst`, the following program produces a path of three randomly selected relays (excluding `Src` and `Dst`).

```
r1 ePathResult(Src, Dst, RAND(3)<IP>) :- ePathRequest(Src, Dst), node(IP),
                                         Src != IP, Dst != IP.
```

Rule `r1` takes as input a path request, in the form of an event tuple `ePathRequest(Src,Dst)`, where `Src` is the address of the node that issued the request, and `Dst` is the address of the responder. Rule `r1` is essentially a typical database query with group-by attributes (`IP` in this case) and a *random aggregate*.

Unlike a regular aggregate that computes, for instance, the minimum and maximum value, a random aggregate is a function of the form `RANDAGG`$(a_1, a_2, ..., a_m)\langle p_1, p_2, ..., p_n\rangle$ that takes in $(a_1, a_2, ..., a_m)$ as m argument parameters, and n arguments in $\langle p_1, p_2, ..., p_n\rangle$ that denote the output (projection) attributes of the resulting group-by value. Given result tuples generated in the rule body, for each group-by value, `RANDAGG` performs the appropriate random selection algorithm based on its function definition, and then returns a list of tuples with the appropriate n attributes being projected from the results.

For instance, in rule `r1`, `RAND(3)<IP>` is a random aggregate with argument 3 and projecting by IP. With these parameters, the aggregate will return 3 randomly selected nodes without replacement from the result of executing the rule body. The output of executing the rule `r1` is the `ePathResults(Src,Dst,P)` event tuple, where `P` is a list of tuples each containing the IP address field of the selected relay nodes from `Src` to `Dst`. The additional selection predicates in `r1` ensure that neither `Src` or `Dst` are selected as relay nodes.

Bandwidth-weighted Selection (Tor Policy). The Random strategy chooses three nodes as relays without taking into consideration their node characteristics. As an enhancement, the following rules implement Tor's relay selection [11] and select nodes with probabilities that are proportional to their bandwidths. A node with higher bandwidth has a greater probability of being selected, and the likelihood of selection relative to other nodes is linearly proportional to bandwidth.

```
t1 eCandidateRelay(Src, Dst, PathsSoFar, RANDWEIGHTED(1,BW)<IP>) :-
     ePathRequest(Src, Dst, PathsSoFar), node(IP,BW), Src != IP, Dst != IP.

t2 ePartialPath(Src, Dst, PathNew) :-
     eCandidateRelay(Src,Dst,PathsSoFar,Relay),
     PathsSoFar.inPath(Relay[0]) = false,
     PathNew = f_append(PathsSoFar, Relay[0]).

t3 ePartialPath(Src, Dst, PathsSoFar) :-
     eCandidateRelay(Src,Dst,PathsSoFar,Relay),
     PathsSoFar.inPath(Relay[0]) = true.

t4 ePathRequest(Src, Dst, P) :- ePartialPath(Src, Dst, P), f_size(P) < 3.
```

```
t5 ePathResult(Src, Dst, P) :- ePartialPath(Src, Dst, P), f_size(P) = 3.
```

The initial route request is triggered by the requesting event `ePathRequest(Src,Dst,())`, where the last attribute is the current path (initially initialized to the empty list `()`). Rule `t1` is similar to the earlier rule `r1`, except that it uses the aggregate function `RANDWEIGHTED(1,BW)<IP>` which selects one tuple randomly from the tuples derived from executing the rule body, with probability linearly weighted by the bandwidth attribute `BW`. (To bias the selection using another metric such as average node load, one would simply have to modify the parameter to `RANDWEIGHTED`.) The resulting output is a list containing one tuple, which can be retrieved as the first element of the list (indicated by the index `[0]`) followed by a projection on the `IP` field.

Rules `t2` and `t3` generate a new `ePartialPath` if the chosen `Relay` is not already in the current partial path; otherwise, they add the relay's IP to the path. The process repeats in `t4` if the number of relays selected is less than three. Eventually, the resulting path `ePathResult` is returned via rule `t5` when three relay nodes have been chosen.

Tunable Performance/Anonymity Selection (Snader-Borisov **Policy).** Snader's and Borisov's recent proposal introduces a tunable weighting system that allows the initiator to trade between anonymity and performance [63]. Briefly, their proposal defines the family of functions

$$f_s(x) = \begin{cases} \frac{1-2^{sx}}{1-2^{s}} & \text{if } s \neq 0 \\ x & \text{if } s = 0 \end{cases} \tag{1}$$

where s is a parameter chosen by the initiator that allows for a tradeoff between anonymity and performance. After ranking the relays by bandwidth, the initiator chooses the relay with index $\lfloor n \cdot f_s(x) \rfloor$, where x is chosen uniformly at random from $[0, 1)$, and n is the number of nodes. By applying higher values of s, the initiator is able to more heavily bias her selections towards bandwidth. On the other hand, for $s = 0$, a relay is chosen uniformly at random. Each relay is selected independently and without replacement according to the distribution imposed by Eq. 1.

Snader and Borisov's algorithm may be represented in A^{3}Log by modifying the `t1` rule from above into two rules:

```
s1 eRelayList(Src,Dst,PathsSoFar,S,SORT(BW)<IP>) :-
     ePathRequest(Src,Dst,PathsSoFar,S),
     node(IP,BW), Src != IP, Dst != IP.

s2 eCandidateRelay(Src,Dst,PathsSoFar,Relay) :-
     eRelayList(Src,Dest,PathsSoFar,S,SortedRelayList),
     sbRand = (1 - 2^(S * f_rand01())) / (1 - 2^S)),
     Relay = f_selectIndex(SortedRelayList,sbRand).
```

`SORT(BW)<IP>` is a *ranking aggregate* which follows a similar syntax as the random aggregates. It takes all the resulting tuples derived from executing the rule body, performs a sort using the `BW` attribute, and then returns the projected field `IP` as a nested tuple based on the sort order. Hence, the `SortedRelayList` attribute of `eRelayList` will include a sorted list of `IP` tuples. Rule `s2` applies Eq. 1 to generate a biased random variable which is then used to index into the list and select a relay.

4.2.2. Link-based Selection

The previous examples have focused exclusively on *node characteristics* – performance metrics (i.e., bandwidth) that may be attributed to individual relays. In *link-based path selection* [58], the e2e performance of a path is computed by aggregating the cost of all links that comprise the path, where cost is defined in terms of *link characteristics* such as latency, loss, and jitter. (While bandwidth is a node-based characteristic, it can also be represented as a link characteristic by considering the measured available bandwidth on a link connecting two nodes.) The use of link rather than node characteristics enables not only more flexible routing (since initiators can construct anonymous routes that meet more specific communication requirements), but also offers better protection of the identities of the communicating parties [58].

In these examples, node information gathered from the directory service and Information Providers is stored in `node(IP, Coord)` tuples and includes nodes' network addresses and virtual coordinates.

End-to-end Constraint-Based Selection (Constraint Policy). The Constraint policy allows the investigator to specify explicit communication requirements. In the example below, anonymous paths are constructed such that the e2e latency is below some user-specified threshold.

```
c1 eCandidatePath(Src,Dst,Limit,RAND(3)<IP,Coord>) :-
     ePathRequest(Src, Dst, Limit),
     node(IP, Coord), Src!=IP, Dst!=IP.

c2 ePathCost(Src, Dst, Limit, P, Cost) :-
     eCandidatePath(Src, Dst, Limit, P),
     Cost = f_coorddist(Src.Coord, P[0].Coord) +
       f_coorddist(P[0].Coord, P[1].Coord) +
       f_coorddist(P[1].Coord, P[2].Coord) +
       f_coorddist(P[2].Coord, Dst.Coord).

c3 ePathRequest(Src, Dst, Path) :-
     ePathCost(Src,Dst,Limit,Path,Cost), Cost > Limit.

c4 ePathResult(Src, Dst, Path) :-
     ePathCost(Src,Dst,Limit,Path,Cost), Cost <= Limit.
```

Rule `c1` is similar to rule `r1` of the Random selection policy. Here, however, the `Coord` field is also projected for use in rule `c2`. Based on the three selected relays, `c2` computes the e2e path cost as the sum of the Euclidean distances of the coordinates. The process repeats (rule `c3`) until a path whose overall cost is less than `Limit` (an input variable) is selected (rule `c4`).

Tunable Performance/Anonymity Selection (Weighted Policy). The Weighted link-based path selection algorithm provides tunable performance and anonymity. The algorithm operates in two phases:

In the first phase, the initiator rapidly generates (but does not instantiate) candidate paths consisting of three relays chosen uniformly at random without replacement. The initiator computes the e2e cost of each generated candidate path. In the second phase, the initiator sorts the candidate paths by their cost estimates. Using the family of functions introduced by Snader and Borisov [63] (see Eq. 1), the initiator instantiates the candidate path with index $\lfloor n \cdot f_s(x) \rfloor$, where x is chosen uniformly at random from $[0, 1)$, and n is the number of nodes. As with Snader and Borisov's algorithm, a larger value of s more heavily weighs path selection in favor of performance. The s parameter is denoted by the s attribute in the initial path request.

WEIGHTED is represented in A^{3}LOG as follows:

```
w1 eCandidatePaths(Src, Dst, S, RAND(3, 100)<IP,Coord>, PathCosts) :-
     ePathRequest(Src, Dest, S), node(IP, Coord),
     Src != IP, Dst != IP, PathCosts = {}.

w2 eCandidatePaths(Src, Dst, S, PathList, PathCosts) :-
     eCandidatePaths(Src, Dst, S, PathList, PathCosts), f_size(PathList) > 0,
     P=f_popfront(PathList),
     PathCost=f_coorddist(Src.Coord,P[0].Coord) +
              f_coorddist(P[0].Coord,P[1].Coord) +
              f_coorddist(P[1].Coord,P[2].Coord) +
              f_coorddist(P[2].Coord,Dst.Coord),
     PathCosts.append([P, PathCost]).

w3 ePathResult(Src, Dest, Path) :-
     eCandidatePaths(Src, Dst, S, PathList, PathCosts), f_size(PathList)=0,
     SortedPathCosts=f_sortByField(PathCosts, "PathCost", "desc"),
     sbRand=(1 - 2^(S*f_rand01())) / (1-2^S)),
     Path=f_selectIndex(SortedPathCost,sbRand).
```

Rule `w1` first generates 100 random permutations of three elements each from the node table. Then, rule `w2` repeatedly converts these list elements into pairs with the path's e2e cost, based on the embedded coordinates. Finally, rule `w3` sorts this list and selects an index using the Snader-Borisov random variable described in Eq. 1, with a tunable performance parameter s. Note that in this case, we sort in reverse order since lower latency is preferred to higher latency. The above rule assumes a left-to-right execution ordering of predicates. This assumption can be avoided with a more verbose version of the above program using some additional rules.

Hybrid Selection (HYBRID Policy). Although the above rules use a single metric when selecting a path, it is easy to combine multiple factors for relay selection.

The following rule (`h1`) selects a path whose minimum bandwidth is above `Thres` by a conditional join on nodes in the table, and then uses the coordinate embedding system to select a path with e2e latency less than `Limit`. In this example, `node(IP, Coord, BW)` tuples store the virtual latency coordinate as well as bandwidth for each node. Interestingly, we need only make one change to replace `c1` (from the CONSTRAINT strategy) with `h1`:

```
h1 eCandidatePath(Src, Dst, Limit, RAND(3)<IP,Coord,BW>) :-
     ePathRequest(Src, Dst, Limit, Thres), node(IP,Coord,BW),
     Src!=IP, Dst!=IP, BW>Thres.
```

Other hybrid policies using multiple metrics can be similarly constructed (for example, using the WEIGHTED policy only on nodes whose bandwidth is above a threshold).

5. Path Instantiation Policies

A^3's forwarding engine performs *path instantiation*, a process that establishes necessary network state at each selected relay to enable bidirectional data flow over an anonymous circuit between a given initiator and any destination. Unlike relay selection, which happens locally at the initiator, path instantiation is an inherently distributed operation, and thus exercises the distributed execution features of A^{3}Log.

We begin with a brief overview of the path instantiation scheme used by Onion Routing (see Footnote 2). After selecting a path consisting of one or more relays – called *onion routers* – the initiator sends a recursively encrypted message called an *onion* to the first hop of the selected path. Each *layer* of the onion contains the address of the next desired hop in the path, and seed material to generate symmetric keys shared with the initiator[3]. Public key cryptography ensures that every node can interpret exactly one layer of the onion. Each node removes its layer, generates keys from the seed material, and – if it is not the endpoint – forwards the remainder of the onion on to the next hop. The endpoint sends a confirmation message to the initiator backward along the newly-instantiated path.

More precisely, if the relays in the anonymous path are $R_1, ..., R_n$ and $M_1, ..., M_n$ are the relays' corresponding onion layers, then the onion is encrypted as $E_{R_1}(M_1, E_{R_2}(M_2, ..., E_{R_N}(M_N)))$, where $E_X(W)$ denotes the encryption of message W using the public key belonging to X. In practice, only the key seed material is encrypted with the public key. The remaining data is encrypted using a symmetric key derived from the key seed material. Finally, onion routing specifies an additional link-layer protocol that governs how messages are exchanged between onion routers.

5.1. Onion Routing in A^{3}Log

Our A^{3}Log implementation of Onion Routing requires 12 rules to specify path instantiation. These rules consist of three recursive computations: building the onion, relaying the onion along the path to establish state at each node, and forwarding a confirmation back along the path. We extended our implementation to support forwarding data along an instantiated path at a cost of five additional rules.

We briefly summarize the format of the relations at each node. All relations are indexed by a locally unique `CID` (circuit identifier). An initiator stores a `circuitPath(CID,Path)` fact that associates a circuit with a path representing the chosen relay nodes. The `Path` variable represents the result of the relay selection phase and is populated based on the `ePathResult` tuple. In addition, the initiator stores the current state of the circuit in the `circuitStatus(CID,Status)` relation. The value for `Status` may be either `BUILDING` or `ESTABLISHED`. As the path is being instantiated, the initiator and each intermediate relay creates a link-local identifier (`ACI`) for the circuit, stored along with the circuit's next relay in a `circuitForward(CID,ACI,Node)` fact. Similarly, the final relay and each intermediate relay stores the `ACI` generated by the previous `Node` in the `circuitReverse(CID,ACI,Node)` relation. At each relay, symmetric encryption keys

[3]In practice, each layer also contains information about which cryptographic algorithm to use in each direction of the circuit, a timestamp, and a version identifier.

(shared with the initiator) for forward and reverse cryptographic operations are stored in the `circuitKeys(CID,ForwardKey,ReverseKey)` relation. For each relay node, the initiator maintains these keys in the `circuitInitiator(CID, Relay, ForwardKeys, ReverseKeys)` relation.

It is worth noting that many of the relations used by the Onion Routing rules can also be used by Tor and Crowds. For example, all of these systems involve multiplexing traffic from multiple anonymous circuits over a single link, necessitating the use of per-circuit link-local unique identifiers. Also, in each system, paths are bidirectional, requiring intermediate nodes to store the next node in each of the forward and backward directions.

Below, we highlight the use of A^{3}Log via the following three rules (`oc1-oc3`) that express the local recursive computation of generating an Onion at the initiator:

```
oc1 circuitPath(CID, Path),
    circuitStatus(CID, "BUILDING"),
    circuitForward(CID, ACIForward, FirstRelay),
    eCreateOnion(CID,LastRelay,RemainingPath,FirstLayer) :-
       ePathResult(_,_,Path), FirstRelay=f_first(Path).IP,
       LastRelay=f_last(Path).IP, ACIForward=f_gen_aci(),
       RemainingPath=f_removeLast(Path), CID=f_gen_cid(), FirstLayer={}.
```

Rule `oc1` is triggered upon insertion of a new path. It generates state at the initiator for the new circuit, including the local `CID` and link-local `ACI`. These are respectively used to differentiate between circuits at a given node and circuits on a given link. In addition, `oc1` associates the new circuit with its path representation, and a status (i.e., `BUILDING`) indicating that the circuit is currently being instantiated and is not yet ready for use. Rule `oc1` triggers the recursive rule, `oc2`, through the `eCreateOnion` event:

```
oc2 eCreateOnion(CID,NextRelay,RemainingPath,NextLayer) :-
       eCreateOnion(CID, CurrentRelay,Path, PrevLayer),
       f_size(RemainingPath) != 0,
       NextRelay = f_last(Path).IP, RemainingPath = f_removeLast(Path),
       encryptOnion(CID, CurrentRelay, PrevLayer, &EncryptedLayer),
       NextLayer={NextRelay, EncryptedLayer}
```

The `eCreateOnion` event represents an intermediate step of circuit instantiation. Its first argument references the `CID` of the circuit being created, its second notes the most recently added relay, and its third contains the intermediate representation of the onion. Note that onions are built outwards from the innermost layer. We denote the innermost layer as an empty list, as this layer will be interpreted by the ultimate relay in the circuit, who does not extend the path any further.

Rule `oc2` calls the `encryptOnion` *Composable View* (CView, described in Section 5.2), which encrypts the previous layer of the onion. Rule `oc2` is linearly recursive and will continue to trigger itself and derive new facts as long as `RemainingPath` is non-empty. Each invocation of the rule removes a relay node from `RemainingPath` as it adds a layer of encryption. Upon reaching the terminating condition – when `RemainingPath` is empty – rule `oc3` is triggered:

```
oc3 eOnionMessage(@FirstRelay, ACI, CompleteOnion) :-
       eCreateOnion(CID, CurrentRelay, RemainingPath, PrevLayer),
       f_size(RemainingPath) = 0, circuitForward(CID, ACI, FirstRelay),
       encryptOnion(CID, CurrentRelay, PrevLayer, &CompleteOnion).
```

Rule `oc3` calls `encryptOnion` to encrypt the final layer of the onion, and sends the completed onion to the first relay node (via the location specifier `@FirstRelay`) read from the `circuitForward` relation. Upon receiving the onion, each intermediate relay will peel off and decrypt a layer of the onion (using the `decryptOnion` CView), extract the location of the subsequent hop, and recursively forward the onion. The `decryptOnion` CView is similar to `encryptOnion` CView in structure, except for the use of decryption functions instead of encryption functions.

The above rules do not implement Onion Routing's link-layer protocol. One may easily specify this protocol in A^{3}Log by adding a layer of indirection to any rule that sends a high-level anonymous message. We omit the specification here, as it involves relatively mundane serialization, encapsulation, and encryption.

5.2. *Composable Virtual View for Onion Encryption*

In order to maximize reusability between different path instantiation protocols and to enable re-configurable encryption, we leverage *Composable Virtual Views* (CViews) [35] to express high-level cryptographic primitives. A CView is a user-defined function implemented in A^{3}Log. A call to a CView may only occur in the body of a rule, and has the following syntax:

```
viewName(K1,K2,...,Kn, &R1,&R2,...,&Rm)
```

Each CView has a set of input attributes – shown above as `K1,K2,...Kn` – which must be bound at the beginning of the call to the CView, and a set of attributes, `&R1,&R2,...,&Rm`, that are returned by the call. Note that CViews do not augment the expressive power of the A^{3}Log language but rather provide modularity. In fact, any rule that uses CViews can be rewritten as a series of regular A^{3}Log rules using a rewrite [35].

We illustrate the `encryptOnion` CView used by the above rule:

```
def encryptOnion(CID, Node, Data_in, &Data_E) {
   eo1 circuitInitiatorKeys(CID, Node, Key_forw, Key_back)
       this.return(Data_E) :-
          this.init(CID, Node, Data_in), KeySeed = f_genKeySeed(),
          Key_onion = f_sha1(KeySeed), Key_forw = f_sha1(Key_Onion),
          Key_back = f_sha1(Key_forw),
          Payload_E = f_symEncrypt(Key_onion,Data_in),
          publickey(Node,PubKey), KeySeed_E = f_asymEncrypt(PubKey,KeySeed),
          Data_E=[KeySeed_E, Payload_E].
}
```

The built-in predicates `this.init` and `this.return` respectively specify the input values and return values to/from the CView. Rule `eo1` generates key seed material and iteratively applies the SHA-1 hash function to derive three symmetric keys to be shared between the initiator and a given `Node` in the circuit: (i) `Key_onion`, used for encrypting the layer of the onion (except for `KeySeed`) destined for `Node`; (ii) `Key_forw`, used for cryptographic operations on data sent forward from the initiator; and (iii) `Key_back`, used for cryptographic operations on data sent backward to the initiator. These keys are stored at the initiator in the `circuitInitiatorKeys` relation[4]. Rule `eo0` then employs

[4]Note that `circuit_initiator` does not contain `Key_onion` – this key is only used for cryptographic operations on the onion sent for path instantiation, and need not be persisted.

`Key_onion` to encrypt `Data_in`, which consists of the next node in the circuit, and the previous layer of the onion. The `KeySeed` is then encrypted with the public key of `Node`.

Since CViews can separate the cryptographic operations from the specification of the protocol, one can easily tune the encryption by customizing the above `encryptOnion` (and the corresponding `decryptOnion`) CView. Furthermore, CViews can facilitate reusability of these high-level cryptographic primitives across different path instantiation protocols.

5.3. Path Instantiation Policies for Tor and Crowds

We briefly outline how the above path instantiation policies may be modified to support the techniques used by Tor and Crowds.

Tor: Unlike Onion Routing in which the initiator recursively builds a single onion that is relayed along the entire path, Tor uses an incremental telescoping path instantiation strategy. At a high level, a circuit initiator sends a *CREATE* message to the first *Tor router* in the desired circuit. The Tor router establishes local state and replies to the CREATE message, resulting in a path of length one. Should the initiator choose to add another hop to the end of path, he relays an *EXTEND* message to the current endpoint. The current endpoint translates the EXTEND into a CREATE message and sends it to the desired new endpoint. The new endpoint of the circuit replies with a confirmation message, which is forwarded back to the initiator. The initiator may continue to extend the path if desired.

Both CREATE and EXTEND messages can be encoded as A^{3}Log message tuples, each containing half of a Diffie-Hellman handshake that has been encrypted with the public key of the desired new endpoint. The new endpoint completes the handshake with the initiator, resulting in symmetric keys shared with the initiator. The encryption/decryption modules can be implemented as a CView module with the corresponding cryptographic functions, similar to the technique used for Onion Routing (Section 5.2). For sending messages between Tor routers, Tor specifies a link-layer protocol similar to that of Onion Routing.

Crowds: Path instantiation in Crowds begins when an initiator starts an anonymous relay on his machine called a *jondo* and contacts a server to obtain membership in a *crowd* – a collection of anonymous users. To build a path, the initiator forwards a request to a jondo chosen uniformly at random – possibly his own. Upon receiving a request to create a path, a jondo chooses to extend the path to another jondo (again chosen uniformly at random) with probability p_f, or ceases path creation with probability $1 - p_f$. Typically, an initiator will use a single bidirectional path for all anonymous communication. The forward half of the path instantiation scheme used in Crowds is implemented using the following rules:

```
c0 circuitStatus(CID, "BUILDING"), circuitForward(CID, ACI_out, Node_out),
   extend(@Node_out, ACI_out, Me) :-
       establish_path(), ACI_out=f_gen_aci(),
       CID = f_gen_cid(), random_jondo(&Node_out).
```

Rule `c0` begins the process of building a new path of jondos in response to an `establish_path` event. Such an event is triggered when a node retrieves a new list of jondos, for example. `c0` generates a CID and ACI for the new circuit, and selects a jondo uniformly at random (using the `random_jondo` CView) to receive the path extension request, `extend`. Upon receipt of an `extend` request, rule `c1` is triggered:

```
c1 circuitReverse(CID, ACI_in, Node_in), incoming(CID, X) :-
   extend(@Me, ACI_in, Node_in), X = f_rand01(), CID = f_gen_cid().
```

Rule `c1` generates a random number in the range $[0, 1]$, as well as a CID for the circuit. `c1` also derives a local `incoming` event, containing the local CID of the new circuit, and the previously generated random number. The `incoming` event triggers rule `c2`:

```
c2 circuitForward(CID, ACI_out, Node_out), extend(@Node_out, ACI_out, me) :-
     incoming(CID, X), p_forward(P), X <= P, ACI_out = f_gen_aci(),
     random_jondo(&Node_out).
```

Rule `c2` compares the random number (`X`) against the probability of extending the path forward, (`P`). If $X \leq P$, then rule `c2` generates an outgoing ACI, and selects a jondo uniformly at random to serve as the next relay in the circuit. Alternatively, if $X > P$, another set of rules relays a confirmation back to the initiator informing him that the newly instantiated path is ready for use.

6. Implementation

The declarative features of A^3 enable rapid prototyping and development of anonymity protocols. To maximize the usefulness of an extensible anonymity architecture, we designed our A^3 implementation to serve as a testbed for anonymity researchers. Towards that goal, in addition to the use of declarative techniques, our A^3 implementation achieves the following key functionalities:

- **Network simulation mode.** To enable researchers to scale their experiments, our platform provides a simulation mode in which multiple simulated A^3 instances communicate using a virtual network layer. Our A^3 implementation uses ns-3 [43], an emerging discrete-event network simulator aimed to replace the popular ns-2 simulator. ns-3 emulates diverse network effects (e.g., congestion, loss, jitter) and implements virtualized versions of physical, link, network, and transport layer protocols. The simulation mode allows investigators to measure the performance and security properties of anonymity protocols under different network topologies, sizes, and configurations.

- **Implementation mode.** Alternatively, the A^3 client may run in implementation mode, enabling experimentation on the actual Internet. Note that an identical codebase is used for both simulation and implementation modes. The only distinction between the two modes is the use of network simulation in ns-3 versus our own custom sockets implementation in implementation mode.

- **Transparent tunneling.** When run in implementation mode, A^3 instantiates a virtual network interface using Linux's tun/tap capabilities; IP packets transmitted through this interface are transparently tunneled to the A^3 executable and subsequently redirected through the A^3 network. Legacy applications can be configured to use A^3 by causing them to bind to the virtual network address (if such functionality is supported by the application) or by configuring host-based redirection rules (for example, via `iptables`).

We utilize a MySQL database as our Directory Service. We constructed adapters to interface with the CoMon monitoring service [47] on PlanetLab as well as the Vivaldi virtual coordinate embedding system. The former enables node-based relay selection policies that prioritize relays based on their bandwidth utilization, available memory, CPU load, or any other feature monitored by CoMon. Vivaldi supports link-based policies (in particular, those that consider latency). We built our own implementation of Vivaldi, which we integrated into ns-3.

The source code of our implementation (released under the GPLv2 license) may be downloaded from `http://a3anonymity.com`.

RapidNet Declarative Networking Engine. The Relay Selection Engine and Forwarding Engine are constructed using RapidNet [30, 40], an open-source declarative networking engine that is integrated with ns-3. The RapidNet system is a significant improvement over its predecessor P2 system in the following ways:

First, we have significantly improved the performance of the declarative runtime system via a careful reengineering effort. This has resulted in higher throughput in messaging and rule firing, enabling us to send actual packets via transparent tunneling through A^3. (This is in contrast to our previous implementation in which packet forwarding was handled by an external utility program.) The new unified architecture provides enhanced customizability in the data plane.

Second, we have also enhanced the RapidNet system to address execution ambiguities [34] that arise in the P2 system due to the inter-leaving of local rule executions and incoming network events. Our new system provides a clear operational semantics for A^{3}Log that enables one to directly reason about correctness relative to centralized Datalog programs [25]. At a high level, RapidNet provides *atomic rule execution* in which a network event will trigger the execution of local rules at a node until a *fixpoint* is reached (i.e. no new facts are generated) before the next network event is processed.

Third, RapidNet adds several language constructs into A^{3}Log that are required for path instantiation – in particular, encryption/decryption modules, secure communication [76], and CViews.

Finally, via the integration with ns-3, one can incorporate actual network effects in simulation-based experiments. This is an improvement over our prior simulation results [61] which did not simulate congestion, packet queueing, or other network events that are supported by ns-3.

7. Evaluation

In this section, we present measurement studies that demonstrate A^3's ability to implement diverse relay selection and path instantiation strategies. We show how A^3 permits investigators to study performance and anonymity under both simulation (Section 7.1) and actual deployments (Section 7.2). Additionally, we describe a case study that highlights A^3's ability to instantiate high-performance relay selection strategies, permitting the "anonymization" of legacy VoIP softphones (Section 7.3).

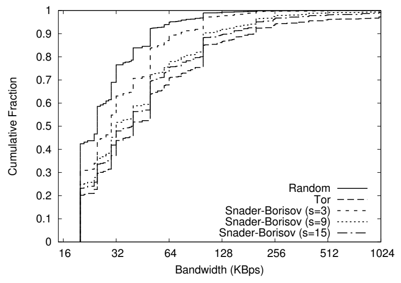


Figure 2: Achieved performance (log scale) as measured by e2e bandwidth under simulation using bandwidths from the Tor directory server for various node-based relay selection policies.

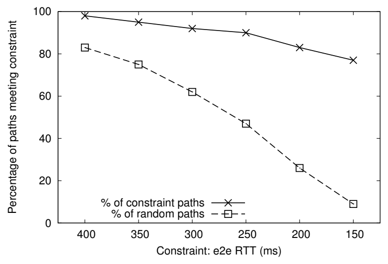


Figure 3: Achieved performance as measured by e2e RTT under simulation using the King dataset with the Constraint relay selection policy.

7.1. Simulation Results

To demonstrate A^3's ability to implement diverse relay selection policies, we use ns-3 to simulate pairwise latencies using latency data from real-world network traces. All relay selection policies were written in A^{3}Log. Unless otherwise indicated, all experiments involving setting up three-relay circuits.

Bandwidth. Figure 2 shows the achieved e2e bandwidth for the Random, Tor, and Snader-Borisov node-based relay selection strategies. Paths were constructed using 500 simulated nodes whose bandwidth capacities were set to the first 500 bandwidths reported by Tor's directory servers. As expected, the Tor routing policy produces paths with significantly greater bandwidths than random selection. The Snader-Borisov algorithm achieves tunable performance results – as the value of *s* increases, the effective e2e bandwidth of anonymous paths also increases.

Latency. The efficacy of our Constraint algorithm for producing low-latency paths can be observed from Figure 3. Here, Internet latencies are based on data from the King dataset [19] – a collection of pairwise latencies computed using the "King" method [18] between Internet DNS servers. Our network consists of the first 500 nodes from the King dataset. Each node was configured to have a bandwidth capacity of 100Mbps. The Figure plots the percentage of anonymous paths whose e2e latency met the constraint for both the Random and Constraint policies. The results from the uniform selection policy serve as an approximation for the percentage of possible paths that meet the constraint, and therefore indicate the difficulty of finding conforming paths. Failure to meet the requirements specified by the Constraint strategy are due to embedding errors in the Vivaldi virtual coordinate system. That is, underestimations of network distances occasionally cause the Relay Selection Engine to incorrectly believe that a nonconforming path met the requirements of the policy.

When the constraint is relaxed to permit paths with e2e latencies of up to 350ms, 75% and 95% of the paths generated using uniform and Constraint, respectively, adhere to the requirement. Even for very stringent requirements – e2e latencies of 150ms

Dataset / Metric	**Relay selection strategy**	**Min-entropy (bits)**	**Probability of presence**
King (Latency)	Random	6.3808	0.012
	Constraint ($<$ 400ms)	5.632	0.020
	Constraint ($<$ 350ms)	5.632	0.020
	Constraint ($<$ 300ms)	5.632	0.020
	Constraint ($<$ 250ms)	5.358	0.024
	Constraint ($<$ 200ms)	5.552	0.022
	Constraint ($<$ 150ms)	5.340	0.024
	Weighted (s=3)	6.2695	0.013
	Weighted (s=9)	5.8898	0.017
	Weighted (s=15)	5.6696	0.020
Tor Directory (Bandwidth)	Random	6.6775	0.010
	Tor	1.4218	0.373

Table 3: Min-entropy for various relay selection algorithms using latency and bandwidth datasets. The *probability of presence* (rightmost column) reflects the probability that a given path contains the most frequently chosen relay, and is equivalent to $1/2^{\text{min-entropy}}$.

or less – 77% of paths produced for the Constraint policy met the requirement. In contrast, less than 9% of random paths had latencies below the threshold.

Anonymity. In previous work, we introduced the *node prevalence* metric for quantifying the anonymity offered by a relay selection strategy [57, 58]. Conceptually, the node prevalence of a node is the likelihood that it will be chosen in an anonymous path, given the current network state and chosen relay selection policy. Ideally, the node prevalences should form a uniform distribution – i.e., all nodes should be equally likely to appear in an anonymous path. This denies an attacker the abilities to (i) intelligently predict which relays will be chosen and (ii) strategically position rogue nodes to attract a disproportionate share of anonymous traffic.

In this work, we conservatively focus on worst-case analysis and use *min-entropy* to capture the uncertainty of identifying any relay in an initiator's anonymous path. If p_i is the node prevalence for node i, then the min-entropy is defined as

$$H_{\min(X)} = -\lg \max_{i \in \mathcal{N}} p_i$$

where $\mathcal{N}$ is the set of nodes in the network. Strategies with greater min-entropy – and therefore more uncertainty – offer better anonymity.

Table 3 lists the min-entropies achieved in our bandwidth (using the Tor dataset) and latency (using the King dataset) networks. Tor's relay selection strategy exhibits very low min-entropy, indicating that a single high-bandwidth relay is present in approximately 37% of anonymous paths. In contrast, our results show that link-based relay selection strategies (i.e., Weighted and Constraint) produce greater min-entropy than node-based (that is, bandwidth) techniques. This is consistent with earlier results [58] that found that link-based relay selection policies often offer stronger anonymity.

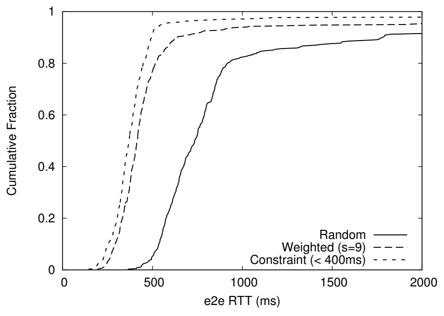


Figure 4: Achieved e2e RTT on PlanetLab for the Random, Weighted, and Constraint relay selection strategies.

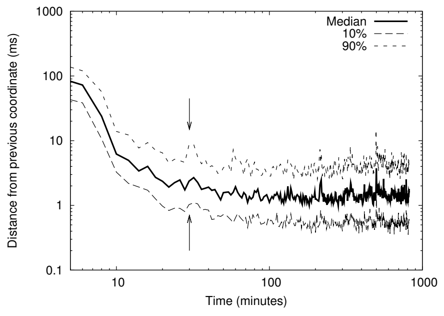


Figure 5: Median, 10th, and 90th percentile distances between coordinate updates on PlanetLab (log scale). Initially, 90% of all relays join the network at approximately the same time. Arrows indicate the point at which the remaining relays join.

We also note that A^3 is able to achieve a tradeoff between anonymity and performance. For instance, in the Weighted scheme, as the value of s increases from 3 to 15, the median combined RTT across three relay nodes decreases from 217 ms to 157 ms. Correspondingly, the min-entropy decreases as A^3 sacrifices a modicum of anonymity for higher-performing routes with lower latency.

It is worth emphasizing that investigators may use min-entropy as a method of quantifying the *selectivity* of a relay selection strategy. Strategies that incur too high a min-entropy may not be suitable when strong anonymity is desired.

7.2. PlanetLab Results

To evaluate our platform's utility on real-world networks, we installed A^3 on approximately 75 hosts on the PlanetLab testbed.

Path Performance. Figure 4 shows the e2e path performance results on PlanetLab for the Random, Weighted, and Constraint strategies. Weighted (with $s = 9$) reduced the median RTT of paths by 319ms (43%) as compared to random selection. 63% of paths met the fairly stringent 400ms requirement using the Constraint policy. By comparison, only 0.2% of random paths had e2e RTTs of less than 400ms.

Information Provider Polling Frequency. In order to produce paths that adhere to application policies, the Routing Engine must rely on the data stored in the Local Directory Cache. If the data are stale, then routing decisions will be based on outdated information. However, frequent polling of the Information Providers consumes bandwidth both at relay nodes (whose resources may already be overburdened from forwarding traffic) and at the Providers. The rate at which information should be refreshed is highly dependent upon the particular metric. For example, bandwidth capacities may be fairly static, whereas bandwidth utilization varies significantly over time.

To understand this tradeoff for our Network Coordinate Information Provider, we examined the rate at which coordinates changed under high degrees of churn. Figure 5 (log scale on both axes) plots the rate of change (as measured by the distance between

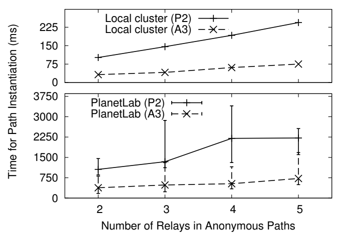


Figure 6: Time for path instantiation on a local cluster *(top)* and on PlanetLab *(bottom)*.

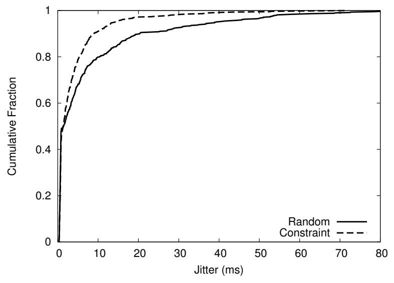


Figure 7: Cumulative distribution of jitter using the `SIPp` SIP test suite on PlanetLab.

successive coordinate updates) on PlanetLab. Since relays operate independently and conduct coordinate updates at varying times, results are grouped at one minute intervals, with the 10th, 50th (median), and 90th percentiles plotted on the graph. Initially, 90% of all relays join the network at approximately the same time, resulting in substantial coordinate movement early in the experiment. However, the system quickly stabilizes–the median rate of change decreases to less than 2ms within 20 minutes.

To model a more realistic scenario, the remaining 10% of PlanetLab nodes join the network after approximately 30 minutes (indicated by arrows on the graph). Immediately following the introduction of the new participants, the median difference between coordinate updates experiences a minor jump, but remains below 3ms.

Our results indicate that latency is fairly stable (at least on PlanetLab), requiring infrequent coordinate updates. Even when members of a large coalition of relays join the network simultaneously, the effect on coordinate stability is minor.

Path Instantiation. Our previous evaluation validates A^3's ability to support a wide range of relay selection policies. Next, we examine the performance of path *instantiation*, using onion routing as our benchmark. As described in Section 5, the A^{3}Log implementation of onion routing requires layered public-key cryptography for onion assembly. To isolate the effects of CPU and communication overhead, we conducted our evaluation in a local cluster in addition to the PlanetLab testbed.

Our local cluster consists of quad-core machines with Intel Xeon 2.4GHz CPUs and 4GB RAM running Fedora 10 with kernel version 2.6, which are interconnected by Gigabit Ethernet. The local cluster configuration allows us to isolate the computation overhead of onion routing.

Figure 6 *(top)* shows the path instantiation times (measured from the initiator creating the onion to the establishment of the bidirectional onion path) as the number of relays increases. For each relay size, we measured approximately 40 path instantiations and computed the median. We make the following observations: First, as expected, the path instantiation time increases linearly with the number of relays. Second, our RapidNet-based implementation (A3) significantly outperforms our previous P2-based system (P2) [61] – for example, the cost of constructing the path decreases by more than half when using the new implementation. Finally, even for large five-relay paths, the overhead of our system (approximately 75ms) is low.

Figure 6 *(bottom)* shows a similar experimental evaluation on the PlanetLab testbed, where we again measure the median path instantiation times as the number of relays increases. For each path size, we produce 75 paths with different sender and receiver pairs. Error bars indicate the 10^{th} and 90^{th} percentiles. As before, the RapidNet-based implementation (A3) offers a significantly reduced overhead when compared against our existing P2-based system (P2). We also observe that path instantiation times are significantly higher on PlanetLab than in our local cluster, which is expected given the high loads and congestion present on the network. Nevertheless, the majority of path instantiations complete within one second, even for five-relay anonymous circuits.

7.3. Case Study: High-Performance Anonymous VoIP Communication

To demonstrate how investigators can use A^3 to customize relay selection policies for applications with real-time communication requirements, we present a case study of anonymous voice-over-IP (VoIP) communication. To maintain the communicants' sense of transparent interactivity, VoIP communication channels must exhibit sufficient bandwidth to support the voice codec and low latency and jitter to eliminate the perception of delay and echo. (For example, the ITU-T requires latencies less than 400ms and recommends delays less than 150ms for telephony [68].)

SIP Benchmarks. We first conducted VoIP benchmarks using the `SIPp` traffic generator [17] for the SIP protocol [54]. As a testbed, we constructed an A^3 network using 75 geographically diverse nodes on PlanetLab. We selected a fixed sender and receiver, and configured the sender to place 300 calls to the receiver using `SIPp` with the Random and Constraint relay selection strategies. The latter strategy imposed a maximum estimated e2e path latency of 120ms. Each "call" required the selection, construction, and teardown of an anonymous A^3 channel. During a call, the sender transmitted dummy packets at a rate of 64 kbps (the required bandwidth for the G.711 voice codec).

Figure 7 shows the cumulative distribution of jitter rates measured by `SIPp` for the two relay selection strategies. Here, the x-axis plots the average jitter rate experienced over the duration of a call. Indicating that latency is a good predictor of jitter, the graph shows a decrease in jitter when confining relay selection to low-latency paths. For example, 20% of the anonymized calls that used Random experienced average jitter above 10ms, while less than 9% of calls established using Constraint exhibited such average jitter.

SIP Softphone. To highlight A^3's practicality, we additionally placed live and anonymous VoIP calls using the `Linphone` [24] SIP softphone. We did not modify `Linphone`, and all application traffic was transparently tunneled through anonymous A^3 paths via our implementation's tunneling service. Our network consisted of 75 PlanetLab nodes and two desktops running in our lab. The desktops served as the caller and callee, and calls persisted for approximately 10 minutes. Additionally, we recorded all packets on the receiver node and applied the `Wireshark` protocol analyzer to the packet dump to measure the jitter of the calls.

Figure 8 *(left)* shows the measured jitter over the duration of a call when the Random relay selection algorithm was applied. (We conducted several such calls, all of which

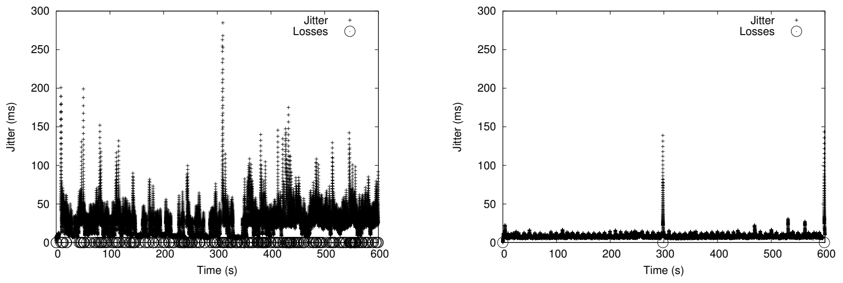


Figure 8: Observed jitter during VoIP calls when Random *(left)* and Weighted *(right)* relay selection strategies are applied. Circles along the x-axis indicate lost VoIP frames.

exhibited similar performance.) Circles along the x-axis of the graph indicate VoIP packets that were either lost or delivered too late to be played. Using the Random strategy, the call experienced frequent and substantial jitter and had a loss rate of 5%. Additionally, there was a noticeable degradation in voice quality, making it difficult to carry a conversation using the softphone.

In contrast, Figure 8 *(right)* plots the resulting jitter when we applied the Weighted (with $s = 9$) strategy. When paths are weighted in favor of low latency, the `linphone` call quality is substantially improved: non-negligible jitter is infrequent and the loss rate is a tolerable 0.3%. With Weighted, we were able to converse with little discernible lag or sound loss using the softphones on our two communication endpoints.

8. Integrating A^3 into Tor

The flexibility to rapidly prototype and evaluate relay selection and path instantiation policies, both under simulation and deployment modes, makes A^3 a useful research tool. As a practical demonstration of A^3 "in the wild", we describe a modified version of Tor which we call *RapTor* that interprets A^{3}Log *relay selection policies*, enabling flexible path selection while inheriting Tor's maturity and userbase. RapTor is thus useful for transitioning relay selection policies that have been tested under A^3 to deployable policies on the live Tor network. RapTor has the additional advantage of benefiting from Tor's strong anonymity mechanisms, and performance optimizations (e.g. flow and congestion control, although such features could in be encoded in declarative policies [36]).

Figure 9 shows the architecture of our RapTor system [65]. As with A^3, relay selection strategies are specified in A^{3}Log (as `.olg` files). A RapidNet *toolkit* translates the policies into C++ source files. At runtime, Tor passes *router descriptors* to a Tor-RapidNet interface, which in turn translates this information into tuples for use with the RapidNet engine. These router descriptors describe the state of each router (for example, their observed bandwidth capacity) as reported by the Tor directory servers. The

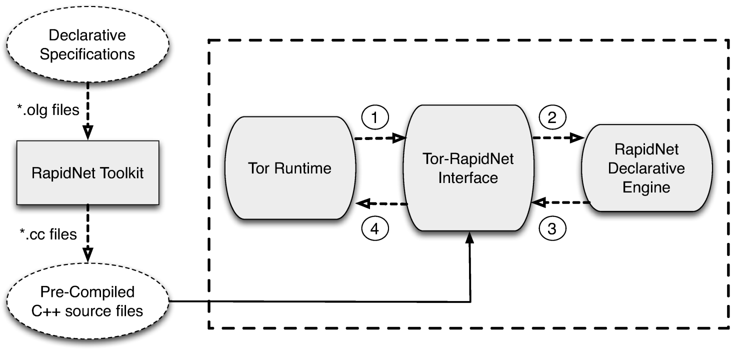

Declarative Specifications
*.olg files
RapidNet Toolkit
*.cc files
Pre-Compiled C++ source files
Tor Runtime
1
2
3
4
Tor-RapidNet Interface
RapidNet Declarative Engine

Figure 9: RapTor architecture. (1) The modified Tor binary sends router descriptors to Tor-RapidNet interface. (2) The interface converts router information to RapidNet tuples. (3) The RapidNet engine generates a path based on the compiled selection strategy. (4) The interface transmits the chosen path to Tor, where it is instantiated.

engine applies the (now compiled) relay selection policy and outputs a path of Tor relays that conforms to the policy. Finally, the Tor-RapidNet interface communicates this path to the Tor runtime, where it is instantiated using Tor's standard path instantiation mechanism.

RapTor policies may utilize any field available in the relay descriptors. For example, the following short (and perhaps ill-advised) Euro relay selection strategy selects a path of three relays such that each relay is constrained to be in Europe (specified using the country code ("CO") parameter).[5]

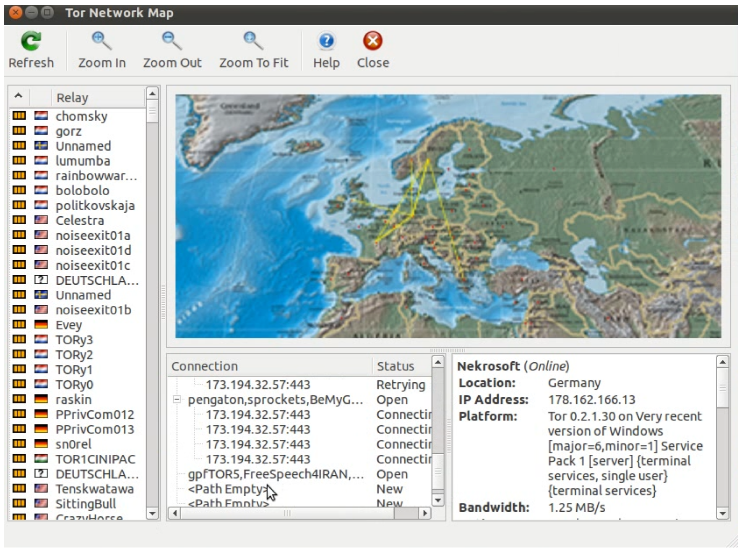
```
euro path(@N,c_RANDK(3)<OR>) :-
     newPath(@N), link(@N, Dst, OR, BW, CO), (CO >= 12 && CO <= 21).
```

We implemented such a policy on the live Tor network. Since this is running on the actual Tor network, this precludes the use of A^3's coordinate systems (which must be deployed globally on all participating Tor nodes). However, RapTor can still leverage A^3's declarative interface to select along Tor's existing relay descriptor fields. Figure 10 shows a visualization of a constructed path that used this strategy, using Tor's Vidalia visualizer [69]. Refer to our website [51] for more videos on additional policy examples showcasing RapTor running on the actual live Tor network.

Overhead. Relative to unmodified Tor, RapTor incurs a modest overhead due to the inclusion of the RapidNet declarative engine. To minimize measurement artifacts (for example, traffic spikes), we measure overheads using private and isolated deployments of RapTor and Tor within our local cluster environment.

Table 4 summarizes our results obtained from one RapTor client running on a Dell Optiplex machine with 8GB memory, Intel quad-core processors (3.40GHz), and Linux version 2.6. We observe that RapTor consumes a modest 20MB of additional memory and incurs a 65ms average increase in path selection time as compared to Tor. Most

[5]Tor provides coarse-grained location data for routers, resolved using the GeoIP [37] mapping service.



Relay
chomsky
gorz
Unnamed
lumumba
rainbowwar...
bolobolo
politkovskaja
Celestra
noiseexit01a
noiseexit01d
noiseexit01c
DEUTSCHLA...
Unnamed
noiseexit01b
Evey
TORy3
TORy2
TORy1
TORy0
raskin
PPrivCom012
PPrivCom013
sn0rel
TOR1CINIPAC
DEUTSCHLA...
Tenskwatawa
SittingBull
CrazyHorse

Connection	Status
173.194.32.57:443	Retrying
pengaton,sprockets,BeMyG...	Open
173.194.32.57:443	Connecti
173.194.32.57:443	Connecti
173.194.32.57:443	Connecti
173.194.32.57:443	Connecti
gpfTOR5,FreeSpeech4IRAN,...	Open
<Path Empty>	New
<Path Empty>	New

Nekrosoft *(Online)*

Location:	Germany
IP Address:	178.162.166.13
Platform:	Tor 0.2.1.30 on Very recent version of Windows [major=6,minor=1] Service Pack 1 [server] {terminal services, single user} {terminal services}
Bandwidth:	1.25 MB/s

Figure 10: A Vidalia screenshot of relays selected using the EURO selection strategy with RapTor. The instantiated paths are shown as yellow lines.

	Unmodified Tor	**RapTor**
Resident memory usage	15MB	35.5MB
Path selection time	4ms	69ms

Table 4: RapTor performance overheads.

of the additional overhead is due to the use of A^3's runtime engine for executing the policy rules. We note that the additional memory usage is modest for a commodity PC, and the increase in latency is small relative to the actual latency for the constructed path itself.

Importantly, path selection is a relatively rare event: by default, Tor paths remain in effect for several minutes even if they are not used. As a straightforward optimization, we can entirely eliminate the latency overhead due to A^{3}LOG-based relay selection by periodically running relay selection in the background and caching the result until new paths are needed. (Such an approach could also be applied to the vanilla version of Tor.) We anticipate adopting this strategy for future releases of A^3 and RapTor.

Transition plan. RapTor enhances Tor by providing flexible and customizable relay selection policies. However, because RapTor relies on Tor for all other functionality, it is less suitable for developing, prototyping, and evaluating novel anonymity protocols. In particular, RapTor lacks the ability to experiment with different path instantiation policies (Section 5). Since Tor is a deployed system, changes to the path instantiation logic would make the modified version incompatible with the existing live network.

Similarly, to maintain interoperability with the live network, RapTor cannot add additional fields to Tor relay descriptors (for example, to support virtual coordinates).

Conversely, however, if new fields are added by the operators of the network, then RapTor can use these new fields in customizable relay selection policies.

Finally, RapTor cannot operate in simulation mode, unless combined with a network emulator [3, 71]. As noted in existing work [3, 64], conducting experiments on the live Tor network is discouraged as it could inadvertently degrade the anonymity of the network's users. Nonetheless, our RapTor validates the effective use of A^3's declarative policy engine for selecting paths on Tor.

Moving forward, we envision a workflow in which researchers prototype and evaluate anonymity protocols using A^3, either under simulation (to evaluate the protocol against a large number of possible configurations) or an isolated deployment (to assess performance in the presence of real-network effects). If the tested policies become widely accepted by the Tor user community, RapTor can instantiate these new policies on the live Tor network.

9. Conclusion

This paper presents the design and implementation of A^3, an anonymity platform that enables researchers to rapidly prototype and deploy relay selection and path instantiation strategies. We demonstrate that A^3 provides sufficient flexibility to encode the relay selection algorithms used by Tor, Snader and Borisov's refinement to Tor, and link-based approaches in only a few lines of A^3Log. Results from simulations over trace-driven datasets and our deployment on PlanetLab show that A^3 produces paths that conform to the specified policies with little computational overhead.

In addition to providing flexible relay selection, A^3 also enables initiators to customize both the manner in which anonymous paths are constructed as well as the mechanisms used to transport data over such paths. For example, we show how the setup and data transmission phases of Onion Routing can be compactly specified in the A^3Log policy language.

A^3's flexibility makes it well-suited for research in anonymity networks. The platform permits experimentation using a wide variety of policies, enabling investigators to construct application-specific anonymous routing strategies. Additionally, A^3's modular design and declarative techniques permit the platform to be easily extended to support additional metrics. By constructing small adapters that interface with Information Providers, A^3 can be adapted to support policies that reference a diverse set of routing criteria.

Acknowledgments

This work is partially supported by NSF Grants CNS-1149832, CNS-0831376, IIS-0812770, CNS-0627579, CNS-0524047, and CNS-1064986; DARPA ONR-N00014-09-1-0770; and OSD/AFOSR MURI FA9550-08-1-0352. Additionally, this material is based upon work supported by the Defense Advanced Research Project Agency (DARPA) and Space and Naval Warfare Systems Center Pacific under Contract No. N66001-11-C-4020. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Project Agency and Space and Naval Warfare Systems Center Pacific.

References

[1] M. Akhoondi, C. Yu, and H. V. Madhyastha. LASTor: A Low-Latency AS-Aware Tor Client. In *IEEE Symposium on Security and Privacy (Oakland)*, 2012.

[2] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker. Low-resource Routing Attacks Against Tor. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2007.

[3] K. Bauer, M. Sherr, D. McCoy, and D. Grunwald. ExperimenTor: A Testbed for Safe and Realistic Tor Experimentation. In *USENIX Workshop on Cyber Security Experimentation and Test (CSET)*, 2011.

[4] S. Becker, J. Seibert, C. Nita-Rotaru, and R. State. Securing Application-Level Topology Estimation Networks: Facing the Frog-Boiling Attack. In *International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2011.

[5] O. Berthold, H. Federrath, and M. Köhntopp. Project "Anonymity and Unobservability in the Internet". In *Workshop on Freedom and Privacy by Design (CFP)*, 2000.

[6] N. Borisov, G. Danezis, P. Mittal, and P. Tabriz. Denial of Service or Denial of Security? How Attacks on Reliability can Compromise Anonymity. In *ACM Conference on Computer and Communications Security (CCS)*, 2007.

[7] E. Chan-Tin, D. Feldman, Y. Kim, and N. Hopper. The Frog-Boiling Attack: Limitations of Anomaly Detection for Secure Network Coordinates. In *ICST Conference on Security and Privacy in Communication Networks (SecureComm)*, 2009.

[8] M. Costa, M. Castro, R. Rowstron, and P. Key. PIC: Practical Internet Coordinates for Distance Estimation. In *International Conference on Distributed Computing Systems (ICDCS)*, 2004.

[9] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A Decentralized Network Coordinate System. *SIGCOMM Comput. Commun. Rev.*, 34(4):15–26, 2004.

[10] F. Dabek, J. Li, E. Sit, F. Kaashoek, R. Morris, and C. Blake. Designing a DHT for Low Latency and High Throughput. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[11] R. Dingledine and N. Mathewson. Tor Path Specification. `http://www.torproject.org/svn/trunk/doc/spec/path-spec.txt`, January 2008.

[12] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium (USENIX)*, 2004.

[13] H. Federrath. JAP: Anonymity & Privacy, 2010. http://anon.inf.tu-dresden.de/.

[14] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. IDMaps: A Global Internet Host Distance Estimation Service. *IEEE/ACM Transactions on Networking*, 9(5):525–540, 2001.

[15] M. J. Freedman, K. Lakshminarayanan, and D. Mazières. OASIS: Anycast for Any Service. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2006.

[16] M. J. Freedman and R. Morris. Tarzan: A Peer-to-Peer Anonymizing Network Layer. In *ACM Conference on Computer and Communications Security (CCS)*, 2002.

[17] R. Gayraud and O. Jacques. SIPp, 2010. http://sipp.sourceforge.net/.

[18] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating Latency Between Arbitrary Internet End Hosts. In *ACM SIGCOMM Workshop on Internet Measurment (IMW)*, 2002.

[19] K. P. Gummadi, S. Saroiu, and S. D. Gribble. "King" Data Set, 2010. http://pdos.csail.mit.edu/p2psim/kingdata/.

[20] R. Jansen and N. Hopper. Shadow: Running Tor in a Box for Accurate and Efficient Experimentation. In *Network and Distributed System Security Symposium (NDSS)*, 2012.

[21] R. Jansen, N. Hopper, and Y. Kim. Recruiting New Tor Relays with BRAIDS. In *ACM Conference on Computer and Communications Security (CCS)*, 2010.

[22] M. A. Kaafar, L. Mathy, C. Barakat, K. Salamatian, T. Turletti, and W. Dabbous. Securing Internet Coordinate Embedding Systems. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, August 2007.

[23] M. A. Kaafar, L. Mathy, T. Turletti, and W. Dabbous. Real Attacks on Virtual Networks: Vivaldi out of Tune. In *SIGCOMM Workshop on Large-Scale Attack Defense (LSAD)*, 2006.

[24] Linphone. Linphone: Open Source Video SIP Phone for Desktop & Mobile, 2010. http://www.linphone.org/.

[25] B. T. Loo, T. Condie, M. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative Networking: Language, Execution and Optimization. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2006.

[26] B. T. Loo, T. Condie, M. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative Networking. In *Communications of the ACM*, 2009.

[27] B. T. Loo, T. Condie, J. M. Hellerstein, P. Maniatis, T. Roscoe, and I. Stoica. Implementing Declarative Overlays. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2005.

[28] B. T. Loo, H. Gill, C. Liu, Y. Mao, W. R. Marczak, M. Sherr, A. Wang, and W. Zhou. Recent advances in declarative networking. In *International Symposium on Practical Aspects of Declarative Languages (PADL)*, 2012.

[29] B. T. Loo, J. M. Hellerstein, I. Stoica, and R. Ramakrishnan. Declarative Routing: Extensible Routing with Declarative Queries. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2005.

[30] B. T. Loo, S. C. Muthukumar, X. Li, C. Liu, J. B. Kopena, M. Oprea, and T. Tao. RapidNet, 2010. `http://netdb.cis.upenn.edu/rapidnet`.

[31] C. Lumezanu, R. Baden, D. Levin, N. Spring, and B. Bhattacharjee. Symbiotic Relationships in Internet Routing Overlays. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2009.

[32] H. Madhyastha, E. Katz-Bassett, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane Nano: Path Prediction for Peer-to-Peer Applications. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2009.

[33] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane: An Information Plane for Distributed Services. In *USENIX Symposium on Operating System Design and Implementation (OSDI)*, 2006.

[34] Y. Mao. On the Declarativity of Declarative Networking. In *ACM International Workshop on Networking Meets Databases (NetDB)*, 2009.

[35] Y. Mao, B. T. Loo, Z. Ives, and J. M. Smith. MOSAIC: Unified Platform for Dynamic Overlay Selection and Composition. In *ACM International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*, 2008.

[36] K. Mattar, I. Matta, J. Day, V. Ishakian, and G. Gursun. Declarative transport: A customizable transport service for the future internet. In *5th International Workshop on Networking Meets Databases*, 2009.

[37] MaxMind: GeoIP — IP Address Location Technology. `http://www.maxmind.com/app/ip-location`.

[38] A. Mislove, G. Oberoi, A. Post, C. Reis, P. Druschel, and D. S. Wallach. AP3: Cooperative, Decentralized Anonymous Communication. In *ACM SIGOPS European Workshop: Beyond the PC*, 2004.

[39] S. J. Murdoch and R. N. M. Watson. Metrics for Security and Performance in Low-Latency Anonymity Systems. In *Privacy Enhancing Technologies Symposium (PETS)*, 2008.

[40] S. C. Muthukumar, X. Li, C. Liu, J. B. Kopena, M. Oprea, and B. T. Loo. Declarative Toolkit for Rapid Network Protocol Simulation and Experimentation. In *ACM SIGCOMM Conference on Data Communication (demo)*, 2009.

[41] T. S. E. Ng and H. Zhang. A Network Positioning System for the Internet. In *USENIX Annual Technical Conference (USENIX-ATC)*, 2004.

[42] V. Nigam, L. Jia, B. T. Loo, and A. Scedrov. Maintaining distributed logic programs incrementally. In *International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP)*, 2011.

[43] Network Simulator 3. http://www.nsnam.org/.

[44] G. O'Gorman and S. Blott. Large Scale Simulation of Tor: Modelling a Global Passive Adversary. In *Asian Computing Science Conference on Advances in Computer Science: Computer and Network Security (ASIAN)*, 2007.

[45] L. Øverlier and P. Syverson. Locating Hidden Servers. In *IEEE Symposium on Security and Privacy (Oakland)*, 2006.

[46] P2 Declarative Networking System, 2010. http://p2.cs.berkeley.edu.

[47] K. Park and V. Pai. CoMon: A Monitoring Infrastructure for PlanetLab, 2010. http://comon.cs.princeton.edu.

[48] PlanetLab, 2010. http://www.planet-lab.org.

[49] R. Ramakrishnan and J. D. Ullman. A Survey of Research on Deductive Database Systems. *Journal of Logic Programming*, 23(2), 1993.

[50] V. Ramasubramanian, D. Malkhi, F. Kuhn, M. Balakrishnan, A. Gupta, and A. Akella. On the Treeness of Internet Latency and Bandwidth. In *Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS/Performance)*, June 2009.

[51] Raptor: Integrated Tor and A3 System. http://netdb.cis.upenn.edu/a3/video.html.

[52] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[53] M. Rennhard and B. Plattner. Introducing MorphMix: Peer-to-Peer Based Anonymous Internet Usage with Collusion Detection. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2002.

[54] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261, Internet Engineering Task Force, 2002.

[55] D. Saucez, B. Donnet, and O. Bonaventure. A Reputation-Based Approach for Securing Vivaldi Embedding System. In *Dependable and Adaptable Networks and Services*, 2007.

[56] Y. Shavitt and T. Tankel. Big-bang Simulation for Embedding Network Distances in Euclidean Space. In *IEEE International Conference on Computer Communications (INFOCOM)*, 2003.

[57] M. Sherr. *Coordinate-Based Routing for High Performance Anonymity*. PhD thesis, University of Pennsylvania, 2009.

[58] M. Sherr, M. Blaze, and B. T. Loo. Scalable Link-Based Relay Selection for Anonymous Routing. In *Privacy Enhancing Technologies Symposium (PETS)*, August 2009.

[59] M. Sherr, M. Blaze, and B. T. Loo. Veracity: Practical Secure Network Coordinates Via Vote-Based Agreements. In *USENIX Annual Technical Conference (USENIX-ATC)*, June 2009.

[60] M. Sherr, B. T. Loo, and M. Blaze. Towards Application-Aware Anonymous Routing. In *USENIX Workshop on Hot Topics in Security (HotSec)*, August 2007.

[61] M. Sherr, A. Mao, W. R. Marczak, W. Zhou, B. T. Loo, and M. Blaze. A3: An Extensible Platform for Application-Aware Anonymity. In *Network and Distributed System Security Symposium (NDSS)*, 2010.

[62] C. Shields and B. N. Levine. A Protocol for Anonymous Communication over the Internet. In *ACM Conference on Computer and Communications Security (CCS)*, 2000.

[63] R. Snader and N. Borisov. A Tune-up for Tor: Improving Security and Performance in the Tor Network. In *Network and Distributed System Security Symposium (NDSS)*, 2008.

[64] C. Soghoian. Enforced Community Standards For Research on Users of the Tor Anonymity Network. In *Workshop on Ethics in Computer Security Research (WECSR)*, 2011.

[65] S. Soundararajan. RapidToR: A Declarative Path Selection Engine for Tor. Master's thesis, University of Pennsylvania, 2012.

[66] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2001.

[67] P. F. Syverson, D. M. Goldschlag, and M. G. Reed. Anonymous Connections and Onion Routing. In *IEEE Symposium on Security and Privacy (Oakland)*, 1997.

[68] Telecommunication Standardization Sector of ITU. Series G: Transmission Systems and Media, Digital Systems and Networks - International Telephone Connections and Circuits – General Recommendations on the Transmission Quality for an Entire International Telephone Connection. ITU G.114, International Telecommunications Union, 2003.

[69] Tor Project: Vidalia. https://www.torproject.org/projects/vidalia.html.en.

[70] TorCtl: Python Tor Control. http://fscked.org/projects/torctl.

[71] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker. Scalability and Accuracy in a Large-scale Network Emulator. *SIGOPS Oper. Syst. Rev.*, 36:271–284, December 2002.

[72] A. Wang, P. Basu, B. T. Loo, and O. Sokolsky. Declarative network verification. In *International Symposium on Practical Aspects of Declarative Languages (PADL)*, 2009.

[73] A. Wang, L. Jia, W. Zhou, Y. Ren, B. T. Loo, J. Rexford, V. Nigam, A. Scedrov, and C. L. Talcott. FSR: Formal analysis and implementation toolket for safe inter-domain routing. In *IEEE/ACM Transaction on Networking (ToN)*, 2012.

[74] B. Wong, A. Slivkins, and E. G. Sirer. Meridian: A Lightweight Network Location Service Without Virtual Coordinates. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2005.

[75] D. J. Zage and C. Nita-Rotaru. On the Accuracy of Decentralized Virtual Coordinate Systems in Adversarial Networks. In *ACM Conference on Computer and Communications Security (CCS)*, 2007.

[76] W. Zhou, Y. Mao, B. T. Loo, and M. Abadi. Unified Declarative Platform for Secure Networked Information Systems. In *International Conference on Data Engineering (ICDE)*, 2009.

[77] W. Zhou, M. Sherr, T. Tao, X. Li, B. T. Loo, and Y. Mao. Efficient querying and maintenance of network provenance at internet-scale. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2010.

[78] W. Zhou, O. Sokolsky, B. T. Loo, and I. Lee. DMaC: Distributed monitoring and checking. In *International Workshop on Runtime Verification (RV)*, 2009.