AC^{3}N: An API and Service for Anonymous Communication in Content-Centric Networking

Gene Tsudik
University of California Irvine

Ersin Uzun
PARC

Christopher A. Wood
University of California Irvine, PARC

ABSTRACT

Privacy problems that stem from a lack of truly anonymous communication in today's Internet are exacerbated by growing evidence of large-scale network packet interception and eavesdropping [12]. Internet users have a limited set of tools available at their disposal to enable better communication, including Tor [3] and new protocols such as tcpcrypt [8, 9] and DNS-over-TLS [32]. Content-Centric Networking (CCN) is an emerging (inter-)networking architecture with the goal of becoming an alternative to the IP-based Internet. Such an architecture must at least have parity with IP based solutions for anonymous communication. Thus, there is a clear demand for a CCN analog to Tor and related protocols. ANDāNA (Anonymous Named Data Networking Application) was the initial attempt to satisfy this demand for CCNs in the context of the legacy Named Data Networking (NDN) architecture – an instance of CCN. However, its elementary design and hasty implementation led to performance and usability issues that hinder practical use. In this paper, we introduce AC^{3}N: Anonymous Communication for Content-Centric Networking, a substantially evolved and improved incarnation of ANDāNA. AC^{3}N supports high-throughput and low-latency anonymous content retrieval in modern CCNs, for both unidirectional and bidirectional settings. We discuss the design, implementation, performance, and anonymity properties of AC^{3}N. Our experimental results indicate that AC^{3}N incurs very low overhead while providing anonymity features analogous to Tor.

1. INTRODUCTION

Network services and applications have undergone a tremendous transformation since their inception in the 1970s. Content distribution, instead of email and remote access to shared resources, has become the leading source of Internet traffic. For example, Netflix alone accounted for nearly 30% of all downstream Internet traffic in 2012 [5]. The number and popularity of such content-centric services are only expected to increase with the growth of data-intensive consumer applications and devices, leading to added pressure on network resources, increased congestion, and wasted bandwidth in today's Internet. Furthermore, with growing evidence of large-scale network packet interception and eavesdropping [12], consumer anonymity and privacy is quickly becoming a desired feature of networking technologies and services.

Content Centric Networking (CCN) [21, 23] is one of today's leading candidates as a viable replacement for the IP-based Internet. Two of its primary characteristics are that (1) content is named, addressable and routable in the network, and (2) all content is signed by its producer. By the first property, a consumer who wishes to obtain content first issues a request (interest) for said content by name, which is then routed to the producer or a network entity (i.e., router) that is capable of satisfying the request. The corresponding content carrying the same name is then sent to the consumer along the reverse path. The second property enables content integrity and authenticity to be decoupled from where it is stored (cached) and how it is delivered to consumers. These two fundamental design characteristics permit content to be opportunistically cached throughout the network, thereby lowering congestion and improving overall bandwidth utilization.

The CCN architecture enables content access control via producer-specified forms of encryption or content name obfuscation. However, support for consumer and producer anonymity is not a standard feature. Furthermore, some features of CCN make anonymity difficult to attain. For example, although neither source or destination addresses are associated with CCN traffic, other sources of information can be used to identify consumers or producers, e.g., content names, router cache contents, and content digital signatures. See [4] for a thorough discussion of how network caches enable consumer privacy and anonymity violations.

ANDāNA (Anonymous Named-Data Networking Application) [14] is the initial attempt to support anonymous communication in Named Data Networking (NDN)[21, 2], an early instantiation of CCN. Inspired by Tor [15, 3], it uses onion-like concentric encryption to wrap interests for content that are gradually decrypted and forwarded by participating *anonymizing routers*. Along the return path, content is wrapped in layers of encryption as it flows from the satisfying entity to the consumer. Unlike Tor, which is a mature tool with well over a decade of deployment experience, ANDāNA was a proof-of-concept prototype application-layer add-on for NDN; its main purpose was to demonstrate the feasibility of anonymous content retrieval over NDN. Supporting high-throughput, low-latency, unidirectional, and bidirectional traffic for voice, video, and media streaming applications was not ANDāNA's goal.

Motivated by these shortcomings, we present an improved design for anonymous communication in CCN. Our approach, henceforth referred to as AC^3N (Anonymous Communication for Content-Centric Networking), addresses many performance, anonymity, and usability pitfalls of ANDāNA. The design of AC^3N relies only on the underlying network's ability to pull uniquely named content by name (via interests) and does not depend on any other design features, e.g., in-network caching. This makes AC^3N applicable in any CCN incarnation, such as CCNx [1] and NDN [2].

In the remainder of this paper, we discuss the design of AC^3N and report on its implementation as an application over CCNx 1.0 [1]. We follow this discussion with a set of network software stack modifications and corresponding APIs that would enable anonymity *within* the stack. These changes would make anonymity an "out of the box" feature of CCN and ICNs with a similarly designed stack. We also present performance results from testing AC^3N in numerous environments with various types of uni- and bi-directional traffic. To illustrate its effectiveness, we compare these performance results to past incarnations of ANDāNA. The results indicate that our design leads to noticeable performance gains compared to ANDāNA, with no reduction in consumer or producer anonymity. We conclude that AC^3N provides a usable interface for efficient anonymous communication in CCN. Additionally, with a clear integration path into the core of the network stack, the likelihood of its adoption and use will likely increase, which is crucial in an emerging technology and landscape where anonymity is a pervasive concern.

2. PRELIMINARIES

This section overviews the properties of the CCN architectures[1] that are relevant to anonymous communication. We then present the adversarial model under which we consider anonymity. We then use it to assess the ANDāNA design and identify certain engineering shortcomings as well as anonymity flaws that are remedied by AC^3N.

2.1 CCN Communication Overview

Content distribution in CCN follows a *pull model*, whereby content is requested by consumers by name instead of by location (e.g., an IP address). These requests are called *interests*, and they contain the name of the desired content rather than its location. Although an *interest* is intended to carry a meaningful (human-readable) URI-like name, it can in fact carry an arbitrary string corresponding to any data type, such as encoded binary data.

An interest is routed based on the specified content name over a sequence of routers, each of which keeps state of the forwarded interest. An interest might get routed to a producer who would reply with the requested content. Alternatively, the requested content might be found in a cache of an intermediate router along the consumer-to-producer path. The latter can occur because each router is expected, though not mandated, to opportunistically cache every content object it forwards to consumers. The bidirectional flow of interests and content in the network guarantees that all traffic flow is symmetric; a single content object is always returned in response to an interest along the same consumer-to-producer path.

To support this communication model, a router processes an interest as follows. Upon receipt of an interest, a router looks up the content by name in its *content store* (CS), i.e., cache. If a matching entry is found then it is subsequently forwarded downstream over the same interface upon which the interest arrived. Interests that do not match any cached content are stored in a *Pending Interest Table* (PIT) together with their arrival interfaces. Multiple interests with the same name are collapsed into a single PIT entry to prevent redundant interests being sent upstream. The interest is then forwarded to the outgoing interface(s) indicated by the local *Forwarding Information Base* (FIB), i.e., routing table. In CCN, each FIB entry contains a name prefix and outgoing interface identifier. In CCNx, the FIB is searched using longest-prefix match on the interest name, similar to IP routing table indexing.

Upon receipt of a content object that matches a PIT entry, the router forwards this content object to all interfaces associated with the PIT entry and caches it for a period of time (typically set by the producer). The router then deletes the PIT entry after the content object forwarded. PIT entries may also be deleted if no matching content object is received within a predefined time-out period.

3. SECURITY AND ANONYMITY

Though appealing from a performance perspective, decoupling content from its producer and addressing content by name introduces many unique security and anonymity issues. For example, security is inherently tied to content, as opposed to the channel between two communicating hosts. Thus, for sensitive content, the producer and its consumer must take steps to restrict access to the plaintext data in the content object. One way to accomplish this is by encrypting the content. This makes confidentiality primarily an application-layer concern.

Conversely, content integrity and origin authentication are inherent features of CCN that are are enforced at the network layer. These properties are facilitated by producer-generated content object digital signatures and consumer-specified cryptographic hash digests of the desired content. Thus, to verify the authenticity of a content object, consumers must either (a) verify the digital signature or (b) verify that the hash digest of said content object matches their expected value. See [19] for a discussion of issues regard-

[1]Both CCNx and NDN are instances of CCN.

ing both signature and hash digest verification, as well as the topic of trust management in CCN.

Anonymity is highly influenced by many features of CCN instantiations. For example, interest and content object names may reveal information about the producer and, potentially, the consumer. Anonymity may also be compromised by (a) the contents of router caches and (b) content object digital signatures. In the former case, curious users can explore the contents of a router's cache through cleverly constructed interests, thereby learning what content was requested by their neighbors; see [4] for more details about this type of attack. In the latter case, digital signatures necessarily expose information about the signer that is revealed during the verification process, e.g., by the certificate containing the public verification key.

In order to fully understand the extent of these anonymity shortcomings, we introduce an adversarial model used in the design and development of $\mathsf{AC^3N}$. In this model, we assume an adversary who is capable of performing the following actions[2]:

- Deploy compromised routers,
- Compromise existing routers,
- Control content producers,
- Deploy compromised caches, and
- Observe and replay traffic

To keep this model realistic, we assume that the time to mount any one of these attacks is non-negligibly longer than the average RTT for an interest-content exchange. Formally, we define an adversary $\mathcal{A}$ as a 3-tuple: $(\mathsf{P}_{\mathcal{A}}, \mathsf{C}_{\mathcal{A}}, \mathsf{R}_{\mathcal{A}})$ where the components denote the set of compromised producers, consumers, and routers, respectively. See Table 1 for a complete list of notation used in this work. Following [14], if $\mathcal{A}$ controls a producer or a consumer then it is assumed to have complete and adaptive control over how they behave in an application session. In other words, $\mathcal{A}$ can control all of the timing, format, and actual information of each content through comprised nodes and links.

We define a *configuration* as a snapshot in time of the current activity associated with a consumer. In other words, each configuration is a relation that maps consumers to the state of a subset of the network. Let $c, r_1, \ldots, r_n, p$ be a consumer-to-producer path of length $(n+1)$ from $c \in \mathsf{C}$ to $p \in \mathsf{P}$. Furthermore, let $\overline{\mathsf{int}}_1^n$ be an interest sent from c to p that traverses the route $r_1, \ldots, r_n$. A configuration CF is then defined as:

$$\mathsf{CF} : \mathsf{C} \rightarrow \{(r_1, \ldots, r_n, p, \overline{\mathsf{int}}_1^n)\}.$$

This relation can be viewed as a map from $c \in \mathsf{C}$ to a set of routers defining a path, or *circuit*, from c to all $p \in \mathsf{P}$ that interests $\overline{\mathsf{int}}_1^n$ traverse.

Following [14], we define anonymity in the context of indistinguishable consumer configurations. Specifically, two configurations CF and CF' are said to be *indistinguishable with respect to* $\mathcal{A}$, denoted $\mathsf{CF} \equiv_{\mathcal{A}} \mathsf{CF}'$, if, for all such polynomial-time adversaries $\mathcal{A}$ there exists a negligible function ϵ such that:

$$\left|\Pr[\mathcal{A}(1^\kappa, \mathsf{CF}) = 1] - \Pr[\mathcal{A}(1^\kappa, \mathsf{CF}') = 1]\right| \leq \epsilon(\kappa),$$

for global security parameter κ. This means that the probability that, given two configurations, the likelihood that $\mathcal{A}$ can correctly differentiate one from the other is no better than a random guess. Simply put, if $\mathcal{A}$ was able distinguish between two separate configurations, $\mathcal{A}$ would then also be able to determine, at a minimum, that either (a) *some* interest was sent by two different consumers, or (b) two different interests emanated from the *same* consumer. Since this is the minimum amount of information that can be revealed to $\mathcal{A}$, we use this as our basis for defining consumer, producer, and session anonymity, as well as producer and consumer linkability and interest linkability.

Definition 1. *[14] For $u \in (\mathsf{C} \backslash \mathsf{C}_{\mathcal{A}})$, u has* consumer anonymity *in* CF *with respect to $\mathcal{A}$ if $\exists$ $\mathsf{CF}' \equiv_{\mathcal{A}} \mathsf{CF}$; such that $\mathsf{CF}'(u') = \mathsf{CF}(u)$ and $u' \neq u$.*

Definition 2. *[14] Given $\overline{\mathsf{int}}_1^n$ and $p \in \mathsf{P}$, $u \in \mathsf{C}$ has* producer anonymity *in* CF *with respect to p and $\mathcal{A}$ if $\exists$ $\mathsf{CF}' \equiv_{\mathcal{A}} \mathsf{CF}$such that $\overline{\mathsf{int}}_1^n$ is sent by a non-compromised consumer to $p' \neq p$.*

Definition 3. *Two entities p and c serving as producer and consumer in an application session are said to have* session anonymity *in* CF *with respect to $\mathcal{A}$ if both c and p have producer and consumer anonymity in* CF *with respect to $\mathcal{A}$.*

There are two types of linkability that are important in this work: producer and consumer linkability, and interest linkability. Both of these are defined with respect to consumers, producers, interests, and content objects. Informally, two or more of these "entities" are *unlinkable* with respect to $\mathcal{A}$ if $\mathcal{A}$ cannot determine if they are related in any meaningful way. As an example, such a meaningful relation might be that content object C corresponds to the Interest I.

Since packet (message) arrivals are discrete events observed at consumers, routers, and producers, we refer to distinct messages based on the order in which they arrive. Specifically, let $\mathsf{int}{:}i^e$ be the ith interest received or processed by entity e in the network, e.g., a consumer, router, or producer. With this notation in place, we formally define interest unlinkability below. Content object unlinkability has an analogous definition.

Definition 4. *Two interests $\mathsf{int}{:}i^e$ and $\mathsf{int}{:}j^e$ that arrive at entity e are* unlinkable *with respect to $\mathcal{A}$ in configuration* CF *if*

$$|\Pr[\mathcal{A}(1^\kappa, \mathsf{CF}, \mathsf{int}{:}i^e) = 1] - \Pr[\mathcal{A}(1^\kappa, \mathsf{CF}, \mathsf{int}{:}j^e) = 1]| \leq \epsilon(\kappa).$$

Producers and consumers may also be linkable with respect to a particular configuration CF and $\mathcal{A}$. Intuitively, this

[2]Any one of these actions can be performed adaptively, i.e., in response to status updates or based on observations.

means that interests issued by a consumer and those received by a producer can be *paired*. We formally define the inverse of this idea below.

DEFINITION 5. *A producer $p \in \mathsf{P}$ and consumer $c \in \mathsf{C} \setminus \mathsf{C}_{\mathcal{A}}$ are* unlinkable *in* CF *with respect to* $\mathcal{A}$ *if there exists* $\mathsf{CF}' \equiv_{\mathcal{A}} \mathsf{CF}$ *where interests generated from* c *are sent to a producer* $p' \neq p$.

It may be easier to consider the notion of linkability instead. Specifically, a producer $p \in \mathsf{P}$ and consumer $c \in \mathsf{C} \setminus \mathsf{C}_{\mathcal{A}}$ are linkable if in CF with respect to $\mathcal{A}$ if *all interests* sent from c are sent to p.

Linkability and anonymity are closely related. In particular, consider the following corollaries, which are proved in [14].

COROLLARY 1. *[14] Producer $p \in \mathsf{P}$ and consumer $c \in \mathsf{C} \setminus \mathsf{C}_{\mathcal{A}}$ are unlinkable in configuration* CF *with respect to* $\mathcal{A}$ *if* p *has producer anonymity with respect to* c*'s interests or* c *has consumer anonymity and* $\exists\ \mathsf{CF}' \equiv_{\mathcal{A}} \mathsf{CF}$ *where* $\mathsf{CF}'(c') = \mathsf{CF}(c) = c$ *with* $c' \neq c$ *and* c'*'s interests are routed to a producer* $p' \neq p$.

COROLLARY 2. *[14] Producer $p \in \mathsf{P}$ and consumer $c \in \mathsf{C} \setminus \mathsf{C}_{\mathcal{A}}$ are unlinkable in configuration* CF *with respect to* $\mathcal{A}$ *if both* p *and* c *have producer and consumer anonymity, respectively.*

Our primary goal is to achieve consumer and producer anonymity and unlinkability with minimal overhead. We describe key design elements and show that AC^{3}N achieves this goal in Section 4. For brevity, we analyze claims of anonymity and unlinkability in Appendix A.

3.1 ANDāNA Highlights

To motivate AC^{3}N, we first re-examine ANDāNA, the first anonymous communication tool designed for CCN designs. In ANDāNA, just as in Tor, interests and content objects traverse *circuits* (paths) of anonymizing routers (ARs). The ARs in each circuit are chosen by the consumer. Before interests are issued by a consumer, their name is first *wrapped* in concentric layers of encryption. Each "layer" contains a routable name prefix for the next hop (AR) in the circuit and the underlying encrypted layers. Each AR decrypts their layer of the name to obtain the next routable prefix in the circuit and corresponding layer (i.e., it "unwraps" its layer of encryption), and then forwards the interest with the new name accordingly. Upon the receipt of content objects in the reverse path, each AR will encrypt the entire content object and forward the "wrapped" result to the next downstream hop. The consumer then recovers the content object by iteratively decrypting each layer of encryption surrounding the content object. This linear wrapping and unwrapping behavior is illustrated in Figure 1.

Unlike Tor, ANDāNA does not support persistent anonymous circuits between consumers and producers. Rather,

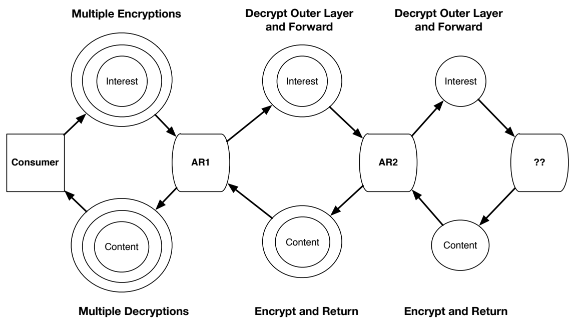

Figure 1: Interest and content concentric encryption and decryption in ANDāNA. The right-most entity is labeled with "??" because the plaintext interest may traverse through more than a single hop before reaching the final producer.

ephemeral (one-time) circuits are created as the interest is sequentially decrypted and forwarded. State information in each AR is only maintained in the symmetric (session-based) variant ANDāNA.

In the symmetric variant of ANDāNA, state information, consisting of a unique session identifier and symmetric key used for interest and content decryption and encryption, respectively, is established in each anonymizing router using a standard three-way handshake protocol. The use of symmetric encryption removes the computational burden of public key encryption. However, the ANDāNAdesign requires that the session identifier be sent in the clear for every interest, which allows $\mathcal{A}$ to link interests and content packets, thus enabling deanonymization attacks against consumers (see below for details). Furthermore, the handshake procedure wastes consumer bandwidth and time, especially in the case of short-term communication.

3.2 Identified Issues

The primary motivation for AC^{3}N is to attain the same anonymity guarantees as the public key variant of ANDāNA with *better* versatility and performance. Although ANDāNA includes a symmetric (session-based) variant as a more efficient alternative, it does not provide unlinkability. Generally, unlinkability is a sufficient, rather than a necessary, condition for anonymity. However, in ANDāNA interest-content correlation can lead to consumer and producer linkability, which can immediately violate anonymity.

For example, suppose that $\mathcal{A}$ eavesdrops on incoming and outgoing interests for a particular AR. By looking at the traffic patterns, $\mathcal{A}$ can link incoming and outgoing session IDs. In fact, a variant of this adversary was studied in the context of Tor by Murdoch and Danezis in [25] and was shown to be quite successful. We believe that the same attack could be augmented to apply to ANDāNA. In particular, repeating this attack at each AR in a circuit can result in deanonymization of both the consumer and producer.

The use of application and environment contextual information has been investigated in [16], where side-channel and environment information (e.g., deterministic behavior of an AR always forwarding a packet after unwrapping an interest received from a downstream neighbor) is used to quantify the *degree of unlinkability*. Furthermore, regardless of how linkability information is acquired, it has been shown that it can degrade consumer and producer anonymity beyond that attainable by general traffic analysis [28].

Since most relevant literature focuses on mix-based anonymizing services akin to Tor, upon which ANDāNA was designed, it is clear that all linkability problems studied in the context of Tor are also applicable to symmetric variant of ANDāNA. This is why one of the key goals of AC^{3}N is to attain the same anonymity guarantees as the public key variant of ANDāNA, which has no linkability issues, while still providing more efficient support for low-latency, high-throughput and bidirectional traffic, as compared to the symmetric variant of ANDāNA.

4. AC^{3}N DESIGN

This section describes the design of AC^{3}N. All relevant notation is presented in Table 1.

4.1 Circuit and Session Establishment

Similar to Tor[15], anonymizing routers (ARs) and circuits are at the core of AC^{3}N. As previously mentioned, a *circuit* is a sequence of ARs through which upstream interests and downstream content objects flow. Circuits are established for long-term sessions, i.e., they are not ephemeral. ARs in a circuit serve two purposes: (1) decapsulate (decrypt) and forward encrypted interests, and (2) encapsulate (encrypt) content objects using previously acquired or agreed upon keys and forward them downstream.

Consumers generate interests wrapped in several layers of encryption and receive content objects also wrapped in several layers of encryption that it can decrypt. Each AR is an *application* running on router, and therefore technically serves as the producer for each downstream AR in the AC^{3}N circuit. The standard CCN communication model suggests that such content *must be signed*. However, AC^{3}Nstrays from this requirement and uses MACs for more efficient authenticity checks.

To increase interest and content throughput, circuit sessions are established and initialized with long-term symmetric keys used for both content encryption and MAC tag generation and verification. The complete set of session state information, which is established for n ARs $r_1, \ldots, r_n$ in a circuit, is as follows:

- Session IDs $\mathsf{Session}_i$ and session initialization vectors (IVs) $\mathsf{SessionIV}_i^0$,
- Content encryption keys E_{k_i} and initial counter values $\mathsf{EncryptionIV}_i$, and
- Pairwise MAC keys M_{k_i} between adjacent ARs and the consumer (used to tag and verify content).

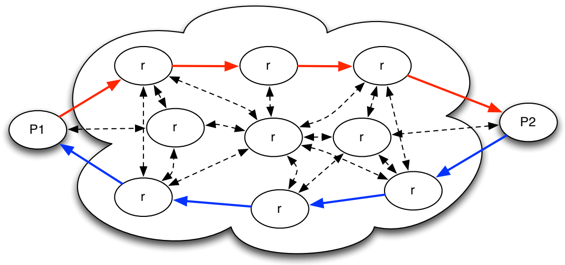

Figure 2: A bidirectional circuit in AC^{3}N.

Algorithm 1 describes the circuit establishment procedure in more detail. Consumers execute the $\mathsf{EstablishCircuit}$ function, which invokes the Init function to establish state with AR r_i, $i = 1, \ldots, n$. The ARs accept session establishment via the $\mathsf{InitHandler}$ functions, which store session information in local memory and respond with an appropriate acknowledgment. Note that no two routers will share the same session identifier (with non-negligible probability) even though they are part of the same circuit. This is because consumers generate session identifiers independently and uniformly at random from $\{0, 1\}^\kappa$.

After a circuit and its session information have been established, all subsequent traffic is protected via a CCA-secure symmetric scheme [22]. The encryption and MAC key for router r_i are indexed via $\mathsf{SessionIndex}_i^j$, the dynamic session index (identifier) sent in the cleartext along with the encrypted interest. To provide unlinkability, the session index is "advanced" from $\mathsf{SessionIndex}_i^j$ to $\mathsf{SessionIndex}_i^{j+1}$, after each new interest is received and forwarded, using a one-way and strongly collision-resistant hash function $H(\cdot)$. Specifically, the transfer functions are:

$$\mathsf{SessionIndex}_i^{j+1} = H(\mathsf{SessionIV}_i^j + \mathsf{Session}_i)$$
$$\mathsf{SessionIV}_i^{j+1} = 1 + \mathsf{SessionIV}_i^j \mod (2^\kappa).$$

Note that $\mathsf{SessionIV}_i^j$ is kept private. AC^{3}N sessions are unidirectional, which means that bidirectional traffic requires two sessions (see Figure 2). This allows each party to choose its own set of ARs. Besides promoting better privacy, this can improve QoS by distributing computational load among multiple, and possibly distinct, sets of ARs.

Once established, a circuit from c to p resembles that of Figure 1. State initialization in AC^{3}N is separate in time from circuit usage, i.e., it uses a handshake routine to initialize state. However, our design does not preclude on-line state establishment. For example, the first wrapped interest issued by a consumer for a new circuit could be overloaded to include all of the state establishment information in addition to the associated interest information.

4.2 AC^{3}N Circuit Usage

Table 1: Relevant notation.

C	Set of all consumers
P	Set of all producers
R	Set of all routers
κ	Global security parameter
$\mathcal{A}$	Adversary
c, p	Consumer and producer, respectively
$r_i \in \mathsf{R}$	The i-th anonymizing router (AR) in an $\mathsf{AC^3N}$ circuit
Circ_i	A set of session information corresponding to the i-th circuit at an $\mathsf{AC^3N}$ consumer
$\overline{\mathsf{int}}_i^j$	Encrypted interest wrapped from r_i to r_j $(i \leq j)$
(pk_i, sk_i)	Public and private key pair of router r_i
$\mathcal{E}_{pk_i}(\cdot)$	Public key encryption using pk_i
$\mathcal{D}_{pk_i}(\cdot)$	Public key decryption using pk_i
$\mathsf{Encrypt}_{k_i}(\cdot)$	XOR-based symmetric key encryption using key k_i
$\mathsf{Decrypt}_{k_i}(\cdot)$	XOR-based symmetric key decryption using key k_i
ST_i	AR r_i session table used to store session ID and digest tuples
E_{k_i}	Interest and content encryption key for AR r_i
M_{k_i}	Shared MAC key for AR r_i and r_{i+1}
$\mathsf{EncryptionIV}_i$	Encryption initialization vector used for r_i
$\mathsf{SessionIV}_i^j$	j-th value of the dynamic session initialization vector for r_i
$\mathsf{Session}_i$	Session ID for r_i
$\mathsf{SessionIndex}_i^j$	j-th dynamic session index (identifier) shared between the consumer and AR r_i
$H(\cdot)$	Collision-resistant hash function with domain $\{0,1\}^*$ and range $\{0,1\}^\kappa$

Generally, $\mathsf{AC^3N}$ circuits are used the same way as in ANDāNA. The encrypted interest generation procedure is shown in Algorithm 2. In it, a consumer c wraps an interest for a sequence of ARs and forwards it towards the first AR.

Note that each encrypted interest also includes a timestamp to mitigate replay attacks. The interest and content forwarding procedures are shown in Algorithms 3 and 4, respectively. Superscripts for session IVs and indexes are omitted for presentation clarity.

Content encryption at r_i uses XOR-based symmetric encryption with a key stream generated by a cryptographically secure pseudorandom generator with input $\mathsf{EncryptionIV}_i$. As presented in the content forwarding routine, $\mathsf{EncryptionIV}_i$ is advanced similarly to the $\mathsf{SessionIV}_i$ so that the key stream is a fresh pseudorandom bit string for each content object. One additional benefit of this form of encryption is that it permits the key stream to be precomputed offline. It does, however, introduce the probability of improperly computed key streams, which will result in corrupt ciphertext. Appendix B discusses this issue in more detail.

After issuing an interest using the encrypted interest generation procedure, an encrypted content object and MAC tag tuple $\overline{data_i^n} = (data_i^n, \sigma_1)$ is returned. The consumer then verifies the MAC tag σ_1 and then decrypts $data_i^n$. The commutative property of XOR allows the consumer to decrypt each layer of the content in any arbitrary order.

5. IMPLEMENTATION AND INTEGRATION

In this section we describe the $\mathsf{AC^3N}$ implementation, API, local service daemon, and underlying AR discovery service and protocol that can be used to simplify the usage of $\mathsf{AC^3N}$. Our discussion does not preclude other implementations or designs. However, we hope that it sheds light on the importance that, with all of these pieces, the bar to usage will be significantly lower, and thus, the likelihood of adoption among users and application developers will increase quite significantly when compared to ANDāNA and Tor (both of which are external software applications).

5.1 $\mathsf{AC^3N}$ Implementation

$\mathsf{AC^3N}$ is written entirely in Python on top of CCNx 1.0 and implements all functionality described in Section 4. AR applications are initialized with a single name prefix, which is the prefix that is used to route all interests to that router. The consumer application creates a list (circuit) of proxies to communicate with and store state for each of the routers in the circuit. Upon instantiation, each proxy establishes session state and sends it to a corresponding AR application. For example, the i-th consumer proxy instance in the consumer application shares its session state information with the i-th AR application instance in the list, which corresponds to the i-th prefix, or hop, in the $\mathsf{AC^3N}$ circuit.

5.2 An $\mathsf{AC^3N}$ API

The application-layer design of $\mathsf{AC^3N}$ is simple and straightforward. However, it still requires users to install separate software and run these applications both locally and on desired anonymous routers. This usability gap may hinder widespread adoption and acceptance. A better approach would be to provide the functionality of $\mathsf{AC^3N}$ "out of the box" with installations of CCNx. To do this, we need to integrate the $\mathsf{AC^3N}$ functionality into the CCNx software suite and provide (1) an API to use $\mathsf{AC^3N}$ and (2) a service to discover anonymous routers for creating circuits.

The design of $\mathsf{AC^3N}$ is simple enough that the only input necessary to create a circuit is n the desired circuit length.

Algorithm 1 Circuit Session Establishment Protocol

```
Require: Anonymous routers $r_1, r_2, \ldots, r_n$ ($n \geq 1$) with public keys $pk_1, pk_2, \ldots, pk_n$.
 1: function InitHandler(int)
 2:     $(E_{k_i}, M_{k_i}, M_{k_{i+1}}, \mathsf{EncryptionIV}_i, \mathsf{SessionIV}_i^1, \mathsf{Session}_i) := \mathcal{D}_{sk_i}(\mathsf{int})$
 3:     $\mathsf{SessionIndex}_i^1 := H(\mathsf{Session}_i + \mathsf{SessionIV}_i^1)$
 4:     Store $(\mathsf{Session}_i, E_{k_i}, M_{k_i}, M_{k_{i+1}}, \mathsf{EncryptionIV}_i, \mathsf{SessionIV}_i)$
 5:     Insert $(\mathsf{SessionIndex}_i^1, \mathsf{Session}_i, \mathsf{SessionIV}_i^1)$ into the session table $\mathsf{ST}_i$
 6:     $\mathsf{resp} \leftarrow \mathsf{Encrypt}_{E_{k_i}}(\mathsf{SessionIndex}_i^1)$
 7:     return resp

 8: function Init($r_i, M_{k_{i+1}}$)
 9:     $E_{k_i} \leftarrow \{0,1\}^\kappa, M_{k_i} \leftarrow \{0,1\}^\kappa$
10:     $\mathsf{EncryptionIV}_i \leftarrow \{0,1\}^\kappa, \mathsf{SessionIV}_i^1 \leftarrow \{0,1\}^\kappa$
11:     $x_i \leftarrow \{0,1\}^\kappa, \mathsf{Session}_i := H(x_i)$
12:     $\mathsf{SessionIndex}_i^1 := H(\mathsf{Session}_i + \mathsf{SessionIV}_i^1)$
13:     $\mathsf{Payload} := \mathcal{E}_{pk_i}(E_{k_i}, M_{k_i}, M_{k_{i+1}}, \mathsf{EncryptionIV}_i, \mathsf{SessionIV}_i, \mathsf{Session}_i)$
14:     $\mathsf{int} := \mathsf{namespace}_i/\mathsf{CREATESESSION}/\mathsf{Payload}$
15:     $\mathsf{resp} := \mathsf{GetContent}(\mathsf{int})$
16:     $(\mathsf{AckSessionIndex}_i^1) := \mathsf{Decrypt}_{E_{k_i}}(resp)$
17:     if $\mathsf{SessionIndex}_i^1 = \mathsf{AckSessionIndex}_i^1$ then
18:         return $(\mathsf{Session}_i, E_{k_i}, M_{k_i}, x_i, \mathsf{EncryptionIV}_i^1, \mathsf{SessionIV}_i^1)$
19:     else
20:         return Error

21: function EstablishCircuit($j, r_1, \ldots, r_n$)
22:     $(\mathsf{Session}_n, E_{k_n}, M_{k_n}, \mathsf{EncryptionIV}_n^1, \mathsf{SessionIV}_n^1) := \mathsf{Init}(r_n)$
23:     $\mathsf{Circ}_j = \{\}$
24:     $\mathsf{Circ}_j[n] = [(\mathsf{Session}_n, E_{k_n}, M_{k_n}, \mathsf{EncryptionIV}_n^1, \mathsf{SessionIV}_n^1)]$
25:     for $i = n-1$ downto 1 do
26:         if $i = n-1$ then
27:             $(\mathsf{Session}_i, E_{k_i}, M_{k_i}, \mathsf{EncryptionIV}_i^1, \mathsf{SessionIV}_i^1) := \mathsf{Init}(r_i, \perp)$
28:         else
29:             $(\mathsf{Session}_i, E_{k_i}, M_{k_i}, \mathsf{EncryptionIV}_i^1, \mathsf{SessionIV}_i^1) := \mathsf{Init}(r_i, M_{k_{i+1}})$
30:         $\mathsf{Circ}_j[i] = (\mathsf{Session}_i, E_{k_i}, M_{k_i}, \mathsf{EncryptionIV}_i^1, \mathsf{SessionIV}_i^1)$
```

Algorithm 2 Encrypted Interest Generation(Circ_j)

```
Require: Interest int, $r_1, r_2, \ldots, r_n$ circuit length $n$
Ensure: Encrypted interest $\overline{\mathsf{int}}_1^n$
 1: $\overline{\mathsf{int}} = \mathsf{int}$
 2: for $i = n$ downto 1 do
 3:     $\{\mathsf{Session}_i, E_{k_i}, M_{k_i}, \mathsf{EncryptionIV}_i^k, \mathsf{SessionIV}_i^k\} := \mathsf{Circ}_j[i]$
 4:     $\mathsf{SessionIndex}_i^k := H(\mathsf{Session}_i + \mathsf{SessionIV}_i^k)$
 5:     $\mathsf{SessionIV}_i^{k+1} = \mathsf{SessionIV}_i^k + 1 \pmod{2^\kappa}$
 6:     $\overline{\mathsf{int}}_i^n = r_i/\mathsf{SessionIndex}_i^k/\mathsf{Encrypt}_{E_{k_i}}(\overline{\mathsf{int}}, \mathsf{timestamp})$
 7: return $\overline{\mathsf{int}}_1^n$
```

Algorithm 3 AR Encrypted Interest Forwarding

```
Require: $\overline{\mathsf{int}}_i^n$
Ensure: $(\overline{\mathsf{int}}_{i+1}^n, \mathsf{Session}_i)$ or discarded packet
 1: $r_i/\mathsf{SessionIndex}_i^k/\mathsf{Encrypt}_{E_{k_i}}(\overline{\mathsf{int}}, \mathsf{timestamp}) := \overline{\mathsf{int}}_i^n$
 2: if $\mathsf{SessionIndex}_i^k \in \mathsf{ST}_i$ then
 3:     $(\mathsf{Session}_i, E_{k_i}, M_{k_i}, \mathsf{EncryptionIV}_i, \mathsf{SessionIV}_i^j) := \mathsf{Lookup}(ST_i, \mathsf{SessionIndex}_i$
 4:     $\mathsf{SessionIV}_i^{k+1} := \mathsf{SessionIV}_i^k + 1 \pmod{2^\kappa}$
 5:     $\mathsf{SessionIndex}_i^{k+1} := H(\mathsf{Session}_i + \mathsf{SessionIV}_i^{k+1})$
 6:     Update $(\mathsf{SessionIndex}_i^{k+1}, \mathsf{Session}_i, \mathsf{SessionIV}_i^{k+1})$ in $\mathsf{ST}_i$
 7:     $(\overline{\mathsf{int}}_{i+1}^n, timestamp) := \mathsf{Decrypt}_{E_{k_i}}(\overline{\mathsf{int}}_i^n)$
 8:     if decryption fails or timestamp is not stale then
 9:         Discard $\overline{\mathsf{int}}_i^j$
10:     else
11:         Persist tuple $T_i = (\overline{\mathsf{int}}_i^n, \overline{\mathsf{int}}_{i+1}^n, \mathsf{Session}_i)$ to pending interest table $\mathsf{PT}_i$
12:         return $(\overline{\mathsf{int}}_{i+1}^n, \mathsf{Session}_i)$
13: else
14:     Discard $\overline{\mathsf{int}}_i^n$
```

Algorithm 4 AR Content Forwarding

```
Require: Content $\overline{data_{i+1}^n}$ in response to interest $\overline{\mathsf{int}}_{i+1}^n$
Ensure: Encrypted data packet $\overline{data_i^n}$
 1: Recover tuple $T_i = (\overline{\mathsf{int}}_i^n, \overline{\mathsf{int}}_{i+1}^n, \mathsf{Session}_i)$ based on $data_{i+1}^n$
 2: Parse $\overline{data_{i+1}^n}$ as a tuple $(data_{i+1}^n, \sigma_{i+1})$
 3: if $\sigma_{i+1} = \perp$ and $M_{k_{i+1}} = \perp$ then
 4:     Verify the signature of $\overline{\mathsf{int}}_{i+1}^n$.
 5:     if The signature passed verification then
 6:         Pass
 7:     else
 8:         return Error
 9: else if $\sigma_{i+1} \neq \perp$ and $M_{k_{i+1}} \neq \perp$ then
10:     if $\sigma_{i+1} = \mathsf{Verify}_{M_{k_{i+1}}}(data_{i+1}^j)$ then
11:         Pass
12:     else
13:         return Error
14: else
15:     return Error
16: Remove signature or MAC tag and name from $data_{i+1}^n$
17: Create new empty data packet $data_i^n$
18: Set name for $data_i^n$ as the name for $\overline{\mathsf{int}}_i^n$
19: $data_i^n := \mathsf{Encrypt}_{E_{k_i}}(\mathsf{EncryptionIV}_i^k, data_{i+1}^n)$
20: $\mathsf{EncryptionIV}_i^{k+1} := \mathsf{EncryptionIV}_i^k + 1 \pmod{2^\kappa}$
21: $\sigma_i := \mathsf{MAC}(data_i^n)$
22: $\overline{data_i^n} = (data_i^n, \sigma_i)$
23: return $\overline{data_i^n}$
```

After a circuit is created, the only input for actually using it is the interest name to be wrapped during the initial encryption phase. When complete, the application then needs to close the circuit, which destroys local session state and issues interests to destroy remote session state at each anonymous router. The AC^{3}N API shown in Figure 3 collates these requirements into a single, simplistic API that can be used by any application developer. The type `CCNxAnonymousCircuit` contains all of the location state and session information necessary to use the circuit, such as the list of anonymous routers, encryption and MAC keys, and initialization vectors. The type `CCNxName` is a Label Content Information (LCI) encoded string that encapsulates the interest name (see [24]). Finally, the `CCNxContentObject` type encapsulates a content object and is returned as the result of issuing an interest to the API through `ccnxAnonymousCircuit_GetConten`

A local service running underneath the AC^{3}N API is used to identify anonymous routers for creating circuits, registering ARs for usage in such circuits, and unregistering ARs when their duties are complete. We call this service ac3nd, as shown in Figure 5.2. Usage of the API and ac3nd service differ between consumers and ARs. Consumers use them to identify sets of candidate ARs for circuit creation, and then use the circuit for anonymous communication. ARs are obtained via either a local catalog of known ARs or the service discovery protocol outlined in the following section. ARs use this API and local service to register themselves as candidates for circuit creation. Currently, ARs are not able to indicate if they wish to be an entry, middle, or exit node in

`CCNxAnonymousCircuit *ccnxAnonymousCircuit_Register(CCNxName *prefix)` : Create an anonymous circuit endpoint for an AR that processes interests and content under the provided name prefix. This function is nonblocking.

`void ccnxAnonymousCircuit_Unregister(CCNxAnonymousCircuit * circuit)` : Unregister the local system as a hop for AR circuits and release any acquired resources.

`CCNxAnonymousCircuit *ccnxAnonymousCircuit_Create(size_t n)` : Create and initialize CCNxAnonymousCircuit containing n routers. This function is nonblocking.

`CCNxContentObject *ccnxAnonymousCircuit_GetContent(CCNxAnonymousCircuit *circuit, CCNxName *name)` : Issue an encrypted interest for the content with the provided name and return the resulting CCNxContentObject. This function blocks until the desired content or an appropriate NACK is returned.

`void *ccnxAnonymousCircuit_Close(CCNxAnonymousCircuit *circuit)` : Close the circuit and perform all of the required state cleanup.

Figure 3: The AC^{3}N API.

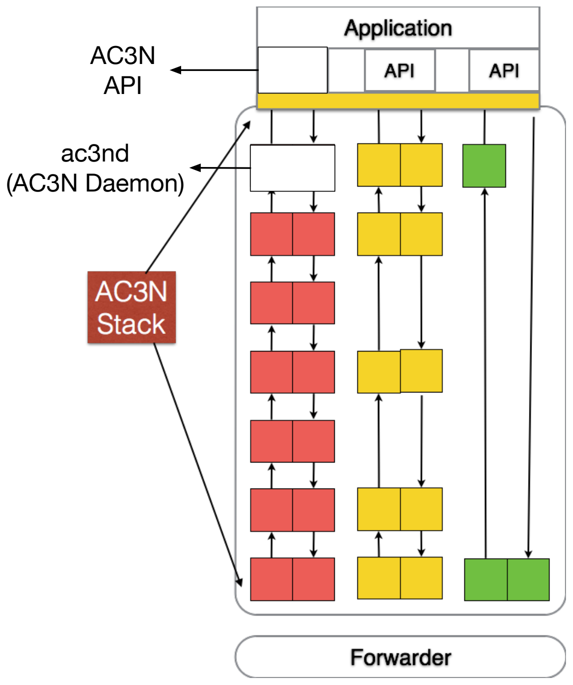

Figure 4: CCNx transport stack with AC^{3}N stack to support the AC^{3}N daemon.

an AC^{3}N circuit, though this feature may easily be incorporated. ARs also use this API to unregister themselves from the service when they no longer wish to participate as a node in AC^{3}N circuits.

5.3 Discovery Service

An important element of AC3N is the AR (relay) discovery service. While it is true that applications can be given a *fixed set* of nodes from which AC^{3}N circuits can be constructed, this solution suffers from poor usability and a lack of scalability. Clearly, a discovery service is needed to facilitate widespread adoption of the protocol. In this section, we present the design of such a service.

As with any service discovery protocol, there are (at least) three primary components: (1) service clients, (2) service providers, and (3) service brokers. The service clients are those who wish to use the resources of service providers. Service brokers facilitate the registration of service providers and help redirect clients to the appropriate providers for their needs. Fortunately, there is only a single service provided by AC^{3}N participants: to be an AR node for circuit establishment. Clients communicate with the service broker(s) to identify a random set of ARs to use for circuit establishment.[3] Providers (candidate ARs) register with the broker(s), who store a set of candidate ARs to satisfy future client requests.

This simple type of interaction is shown in Figure 5.3. All requests (interests) share the same prefix – `/ac3n/services/` – and carry a unique nonce so that they are always directed to the service broker for processing. Also, observe that providers specify their desired AR hosting `prefix` in their request, along with a signature computed over this prefix and their public key. Standard public-key signatures may be used for this registration step. For privacy reasons, the AR is not expected to provide a certificate to the broker for providing its public key. Certificates would reveal information about each AR. Thus, when generating registration interests and signatures, each AR simply needs to use a new and sufficiently random public and private key pair. The signature serves to prove ownership of the private key and that the announced prefix is valid.

After several ARs have registered with the broker and joined the service, clients then request random sets of ARs from the broker for their own personal use. Clients can specify the exact number of ARs they require or leave the set size empty. The broker will generate a completely random set of ARs of the specified size, or of size 3 (as per the Tor default

[3]It is important that the broker provide a completely random set of ARs to satisfy client requests.

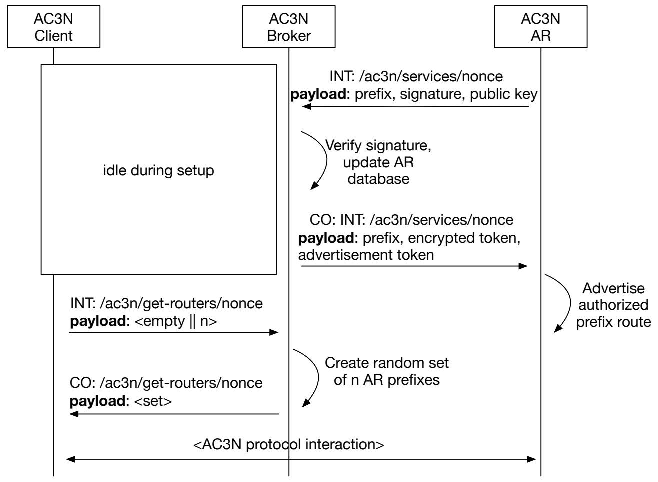
AC3N Client
AC3N Broker
AC3N AR
INT: /ac3n/services/nonce
payload: prefix, signature, public key
idle during setup
Verify signature, update AR database
CO: INT: /ac3n/services/nonce
payload: prefix, encrypted token, advertisement token
Advertise authorized prefix route
INT: /ac3n/get-routers/nonce
payload: <empty || n>
Create random set of n AR prefixes
CO: /ac3n/get-routers/nonce
payload: <set>
<AC3N protocol interaction>

Figure 5: Sample AC^{3}N service interaction between a single client, broker, and provider.

circuit size) if the set size value in the client interest is empty.

One problem with the service as given is that it entrusts all AR selection information to the broker. For complete privacy, we require a solution akin to private information retrieval (PIR) [11], wherein the broker learns nothing about the ARs requested by a particular client. One naive way to implement a PIR protocol for service discovery is to replace the broker AR database amongst k different brokers. The client could then ask for AR sets from each of the k brokers and derive a subset of ARs from the union of the responses. This trivial implementation of PIR is very bandwidth and memory expensive, since it leads to a multiplicative factor of k increase in memory and messages. However, developing a CCN-compliant PIR protocol for content lookup is outside the scope of this paper, so we do not address this issue further.

With the AC^{3}N service discovery protocol, the ac3nd can on clients can then transparently identify potential ARs to use when establishing new `CCNxAnonymousCircuit` instances. The AC^{3}N API implementation would interface with this local service to (a) retrieve a list of possible anonymous name prefixes and (b) perform anonymous router behavior (e.g., interest decryption and content encryption) on behalf of the application.

6. PERFORMANCE ASSESSMENT

In this section we provide an assess the performance of AC^{3}N against ANDāNA. ANDāNA was originally implemented in C using the CCNx 0.8x library. As outlined in the section 2, the CCNx protocol and implementation has changed significantly since this original work. Thus, to bring the evaluation up to speed with existing technology, both ANDāNA and AC^{3}N were implemented using the CCNx 1.0 library from PARC. All experiments were conducted on Vagrant VMs running Ubuntu 14.04 LTS. Each host was equipped with an Intel(R) Core(TM) i5-3427U CPU at 1.80GHz with 8GB of main memory. Also, we use the public key variant of ANDāNA, since it provides the same functionality as AC^{3}N.

Note that we do not include Tor in our performance comparison. We argue that such a comparison would be ineffective and misleading. Tor is designed to run over TCP/IP network architectures, whereas AC^{3}N is designed for CCN architectures. Current NDN and CCNx implementations run as overlays upon TCP/IP. Thus, there is an unavoidable amount of overhead incurred by running AC^{3}N. Put another way, TCP/IP and the NDN and CCNx architectures differ fundamentally in that there is virtually no transport and network layer in the latter. Thus, AC^{3}N performance results would need to take this overhead into consideration when compared against Tor over TCP/IP. For this reason, we only compare ANDāNA and AC^{3}N.

6.1 Unidirectional Assessment

Perhaps the most standard use case for both ANDāNA and AC^{3}N is as a unidirectional anonymizing circuit. Thus, it is important to assess the performance of both tools for this particular use case. In this work, we consider the most important metrics for performance to be (a) interest-content latency L and (b) and throughput S. To assess these metrics, we consider the following simple experiment. Let $\mathsf{Circ} = R_1, \ldots, R_n$ be a circuit of length $n-1$ with n AR nodes. A client C is connected to R_1, and C wishes to retrieve content from producer P connected to R_n. Thus, the complete path is $C, R_1, \ldots, R_n, P$. To obtain content, C issues a random interest that can be satisfied by P through Circ. Such an interest is issued once every t seconds and the RTT is recorded. To compute the average latency, the average of all observed RTTs is computed. To compute the maximum throughput, the total number of content bytes received is divided by the total time to send a large amount of back-to-back interests without delay. The RTT and throughput measurements as a function of the circuit length are shown in Figures 6 and 7.

6.2 Bidirectional Comparison

In order to justify AC^{3}N as a viable choice for anonymous communication suitable for low-latency, bidirectional traffic over CCN, we first establish a baseline of performance measurements. Since ANDāNA targeted circuits of length 2 (i.e., two AR hopes), we consider circuits of the same length. To set a baseline of performance measurements, we instantiated two entities C_1 and C_2 (both acting as a producer and consumer) that interact by requesting moderate-size content in short, frequent intervals. In order ensure that such content is never satisfied by the cache of any intervening AR, which is a reasonable assumption for real-time voice and video applications that always want fresh content, instead of stale cached content, each content is requested from an

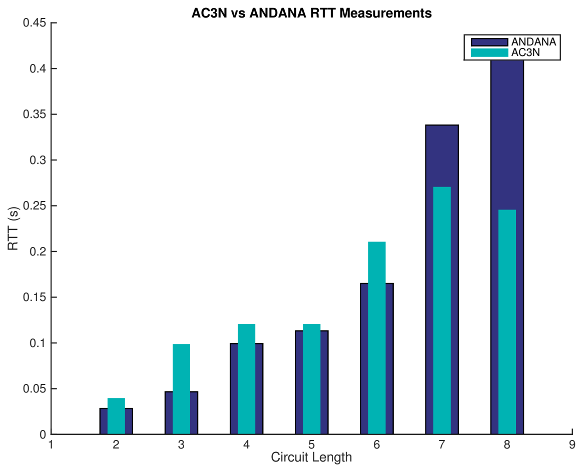

Figure 6: Unidirectional RTT measurements.

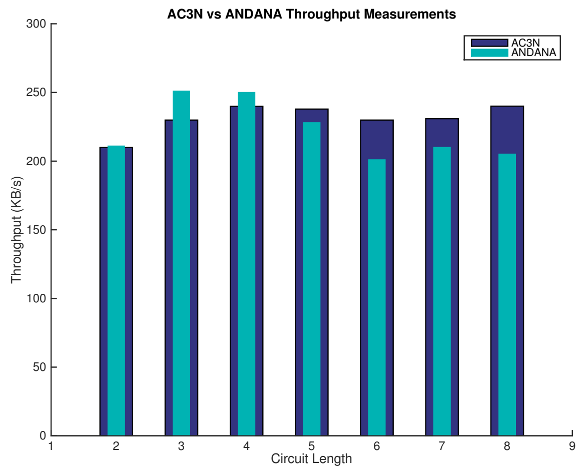

Figure 7: Unidirectional throughput measurements.

anonymized namespace and indexed by a sequence number. For example, to request the latest content from C_2, C_1 issues an (encrypted) interest for `lci:/$C_2$/$S$`, where S is the sequence number, incremented as soon as the interest is issued. This ensures that such interests are never satisfied by any router-cached content for the duration of the session.

With this experimental setup, we tested AC^{3}N under the following scenarios:

1. C_1 and C_2 connected point-to-point (i.e., no intermediate hops).
2. C_1 and C_2 connected via two "insecure" ARs that perform no interest or content encryption (i.e., each AR just serves as an application-level proxy to forward interests and content along through the circuit).
3. C_1 and C_2 connected via two "secure" ARs that perform interest and content encryption.

Table 2: ANDāNA baseline bidirectional performance metrics.

Test Scenario	L_1 (s)	σ_1 (s)	L_2 (s)	σ_2 (s)
1	0.00735	0.00798	0.00340	0.00053
2	0.01321	0.00257	0.01359	0.00567
3	0.03905	0.13791	0.04248	0.13923

Table 3: AC^{3}N bidirectional performance metrics.

Test Scenario	L_1 (s)	σ_1 (s)	L_2 (s)	σ_2 (s)
1	0.00517	0.00813	0.00541	0.00049
2	0.01176	0.00371	0.01521	0.00611
3	0.07805	0.12865	0.05321	0.11691

Table 3 shows performance results from scenarios 1, 2, and 3 for ANDāNA. We characterize performance with respect to the user-perceived values of these latency L. We denote L_i as the perceived (RTT) latency of party $C_i \in \{1, 2\}$. We also assess the standard deviation for each of these measurements. All experiments were performed using the following procedure:

- Each party generates $1,000$ sequential messages, at a rate chosen uniformly from the range $[1, 100]$s.
- Each party responds to all interests with a random content object of 150B.

According to our preliminary experimental evaluation, low-latency, bidirectional communication is feasible using ANDāNA.

7. RELATED WORK

Beyond ANDāNA , there has not been much prior work on anonymous communication in content-centric networks. One approach to consumer privacy is proposed in [7]. Rather than using encryption, it requires content producers to mix sensitive information with so-called "cover" content. In addition to forcing producers to participate in consumer anonymity, all cover content must also be stored for a finite length of time. Furthermore, it does not provide protection against malicious producers and does not offer consumer-producer unlinkability.

Anonymous communication over TCP/IP has a long and extensive history, mostly centered around two fundamental techniques: centralized anonymity proxies and distributed anonymity layering services. The Lucent Personalized Web Assistant [18] is one example of a centralized proxy interposed between communicating end-points. Unfortunately, such techniques are susceptible to passive eavesdropping attacks that monitor proxy activity. Their centralized nature also leads to a single point of trust and failure.

For the most part, distributed approaches are based on Chaum's mix network approach to secure (anonymous) email [10]. The main idea is that several layers of concentric public-key encryption are applied to outgoing messages to traverse a specified set of mixes, each of which iteratively unwraps one layer of encryption and forwards the resulting payload to the next mix hop. By design, each mix buffers incoming messages that are decrypted, shuffles the buffer when a certain threshold is reached, and sequentially forwards each message after a random delay. This "mixing" strategy serves to thwart passive eavesdropping attacks since (given sufficient volumes of incoming messages) an adversary cannot correlate outgoing and incoming messages.

Other low-latency solutions based on mix networks are Babel [20] and Mixminion [13]. The main difference is that their goal is to provide anonymity with respect to a *global eavesdropper* adversary. To do so, each mix generates cover (chaff) traffic in addition to randomized delays. However, such unpredictable traffic delivery patterns make these solutions unsuitable for applications with low-latency requirements.

Low-latency anonymous communication techniques aim to minimize latency by avoiding batching (delaying) and reordering of messages as well as chaffing. As shown by [29], traffic patterns in low-latency anonymity systems can be used to deanonymize clients. Notable examples include: Crowds [26] (vulnerable to local eavesdroppers and predecessor attacks [30]), Morphmix [27], Tarzan [17], and Tor [15] as well as its variant I2P [31]. Crowds is unique in that each mix probabilistically chooses whether to forward a decrypted message or send it to its final destination. Morphmix is distinct from Crowds in that it does not use a lookup service to track all participating nodes. It is also unique in that circuits are dynamically created by each mix. In particular, clients first pick a mix (entry point) for a circuit, which then randomly selects the next hop in the circuit along which messages are forwarded. Tarzan's unique property is that it uses globally verifiable *mimics* for each node to immitate bidirectional cover traffic, thus further obfuscating the message flow between senders and receiver.

As the most popular modern means of anonymous communication, Tor uses a centralized directory to locate and establish circuits through nodes (circuits of length three are sufficient for the required anonymity claims). Furthermore, these circuits are short-lived and typically last no longer than ten minutes. The amount of available bandwidth at each node is taken into account during circuit establishment and multiple TCP connections are multiplexed over one circuit so as to achieve better performance. To improve throughput and mask traffic, communication between adjacent Tor nodes (in a given circuit) is secured via SSL. Finally, Tor does not introduce any decoy traffic or randomization to hide traffic patterns.

8. CONCLUSIONS AND FUTURE WORK

This paper presented $\mathsf{AC}^3\mathsf{N}$ – an application-layer anonymizing service for ICNs. Its simple design relies only upon the fundamental interest-content request paradigm of information retrieval. Experiments indicate that $\mathsf{AC}^3\mathsf{N}$ can support high-throughput, low-latency, unidirectional and bidirectional traffic over ICNs with anonymity guarantees equivalent to Tor.

There are several directions for further optimization and development. For instance, is possible to avoid the handshake stage of circuit establishment entirely. To do so, the circuit state can be embedded in the first interest issued by the consumer, i.e., the interest could be crafted as the concatenation of state information and the normal encrypted interest payload. Since the prepended session index $\mathsf{SessionIndex}$ of this initial interest will not correspond to an entry in the state table, the AR can deduce that it must be the "state establishment interest." The AR would then decrypt and decompose the rest of the interest to recover the initial state information and encrypted interest payload. The state table would be updated to reflect this new circuit, and the encrypted interest would be decrypted and issued to the network. One caveat of this "piggybacking" technique is that each layer of all interests issued in the circuit must be the same length.

Another optimization would involve using the interest-content "piggybacking" technique proposed in [6] for *bidirectional* traffic. Using this technique, two communicating parties would share the same set of anonymizing routers with different state information for each direction (e.g., the same MAC keys will not be used to tag content in both directions). Content objects flowing downstream in a circuit will also contain the next encrypted interest to be issued by the producer of said content objects. Therefore, the AR can remove and decrypt the interest from the content, prepend the resulting interest to the content object, and then encrypt and tag the result before sending it downstream. Although interest decryption and content encryption steps must be performed sequentially, this design would only incur the cost of a single interest once, since all subsequent interests would traverse ARs in conjunction with the content requested by the preceding interest.

9. REFERENCES

[1] CCNx. `http://ccnx.org/`.

[2] Named-data networking. `http://named-data.net/`.

[3] Tor project website. `https://www.torproject.org/`.

[4] G. Acs, M. Conti, P. Gasti, C. Ghali, and G. Tsudik. Cache privacy in named-data networking. In *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*, pages 41–51. IEEE, 2013.

[5] V. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner, and Z.-L. Zhang. Unreeling netflix: Understanding and improving multi-cdn movie delivery. In *INFOCOM, 2012 Proceedings IEEE*, pages 1620–1628, March 2012.

[6] M. Almishari, P. Gasti, N. Nathan, and G. Tsudik. Optimizing bi-directional low-latency communication in named data networking. *ACM SIGCOMM Computer Communication Review*, 44(1):13–19, 2013.

[7] S. Arianfar, T. Koponen, B. Raghavan, and S. Shenker. On preserving privacy in content-oriented networks. In *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, pages 19–24. ACM, 2011.

[8] A. Bittau, M. Hamburg, M. Handley, D. Mazieres, and D. Boneh. The case for ubiquitous transport-level encryption. In *USENIX Security Symposium*, pages 403–418, 2010.

[9] A. Bittau, M. Hamburg, M. Handley, D. Mazieres, and D. Boneh. Simple opportunistic encryption. 2014.

[10] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90,

1981.

[11] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981, 1998.

[12] CNN. Latest nsa leaks point finger at high-tech eavesdropping hub in uk, 2013. http://www.cnn.com/2013/12/20/world/europe/nsa-leaks-uk/.

[13] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a type iii anonymous remailer protocol. In *Security and Privacy, 2003. Proceedings. 2003 Symposium on*, pages 2–15. IEEE, 2003.

[14] S. DiBenedetto, P. Gasti, G. Tsudik, and E. Uzun. Andana: Anonymous named data networking application. In *Network and Distributed System Security - NDSS*. The Internet Society, 2012.

[15] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[16] M. Franz, B. Meyer, and A. Pashalidis. Attacking unlinkability: The importance of context. In N. Borisov and P. Golle, editors, *Privacy Enhancing Technologies*, volume 4776 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin Heidelberg, 2007.

[17] M. J. Freedman and R. Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 193–206. ACM, 2002.

[18] E. Gabber, P. B. Gibbons, D. M. Kristol, Y. Matias, and A. Mayer. Consistent, yet anonymous, web access with lpwa. *Communications of the ACM*, 42(2):42–47, 1999.

[19] C. Ghali, G. Tsudik, and E. Uzun. Elements of trust in named-data networking. *arXiv preprint arXiv:1402.3332*, 2014.

[20] C. Gulcu and G. Tsudik. Mixing e-mail with babel. In *Network and Distributed System Security, 1996., Proceedings of the Symposium on*, pages 2–16. IEEE, 1996.

[21] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *Proc. ACM CoNEXT 2009*, pages 1–12, Dec. 2009.

[22] J. Katz and Y. Lindell. *Introduction to modern cryptography: principles and protocols*. CRC Press, 2007.

[23] M. Mosko. Ccnx 1.0 protocol specification roadmap. 2013.

[24] M. Mosko. Labeled content information, 2015. http://tools.ietf.org/html/draft-mosko-icnrg-ccnxlabeledcontent-00.

[25] S. Murdoch and G. Danezis. Low-cost traffic analysis of tor. In *Security and Privacy, 2005 IEEE Symposium on*, pages 183–195, May 2005.

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

[27] M. Rennhard and B. Plattner. Introducing morphmix: peer-to-peer based anonymous internet usage with collusion detection. In *Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*, pages 91–102. ACM, 2002.

[28] S. Schiffner and S. Clauß. Using linkability information to attack mix-based anonymity services. In *Proceedings of the 9th International Symposium on Privacy Enhancing Technologies*, PETS '09, pages 94–107, Berlin, Heidelberg, 2009. Springer-Verlag.

[29] A. Serjantov. On the anonymity of anonymity systems. *University of Cambridge, Computer Laboratory, Technical Report*, (UCAM-CL-TR-604), 2004.

[30] M. K. Wright, M. Adler, B. N. Levine, and C. Shields. The predecessor attack: An analysis of a threat to anonymous communications systems. *ACM Transactions on Information and System Security (TISSEC)*, 7(4):489–522, 2004.

[31] B. Zantout and R. Haraty. I2p data communication system. In *ICN 2011, The Tenth International Conference on Networks*, pages 401–409, 2011.

[32] L. Zhu, Z. Hu, J. Heidemann, D. Wessels, A. Mankin, and N. Somaiya. Connection-oriented dns to improve privacy and security (extended). *USC/Information Sciences Institute, Tech. Rep. ISI-TR-2015-695, Feb*, 2015.

APPENDIX

A. ANONYMITY ANALYSIS

To assess the anonymity claims of $\mathsf{AC^3N}$ we start by adopting the adversarial model of $\mathsf{AND\bar{a}NA}$ [14] and previous described in Section 3. The fundamental differences between $\mathsf{AND\bar{a}NA}$ and $\mathsf{AC^3N}$ are that, in the latter, (1) each pair of adjacent routers share a distinct MAC key used for efficient content authenticity checks and (2) sessions are identified by the output of $H(\cdot)$, rather than encrypting and decrypting interests using expensive asymmetric procedures. Accordingly, proofs of anonymity need to be augmented to take this into account. The remainder of the design is syntactically equivalent to that of $\mathsf{AND\bar{a}NA}$. Thus, we simply re-state relevant theorems without proof. In doing so, however, we generalize them to circuits of length $n \geq 2$.

Theorem 1. *[14] Consumer $c \in (\mathsf{C} \setminus \mathsf{C}_{\mathcal{A}})$ has consumer anonymity in configuration CF with respect to adversary $\mathcal{A}$ if there exists $c \neq c'$ such that any of the following conditions hold:*

1. *c, c' are in the same anonymity set with respect to the adversary $\mathcal{A}$ (see [14]).*
2. *There exist ARs r_i and r'_i such that $r_i, r'_i \notin \mathsf{R}_{\mathcal{A}}$, both r_i and r'_i are on the circuit traversed by $\overline{\mathsf{int}}_1^n$.*

Theorem 2. *[14] Consumer c has producer anonymity in configuration CF with respect to producer $p \in \mathsf{P}$ and adversary $\mathcal{A}$ if there exists a pair of ARs r_i and r'_i such that r_i and r'_i (for some uncompromised entity $c \notin \mathsf{C}_{\mathcal{A}}$) are on the path traversed by $\overline{\mathsf{int}}_1^n$, $p \neq p'$, and all routers $r_1, \ldots, r_n$ are equal in $\mathsf{CF}(c)$ and $\mathsf{CF}(c')$.*

Unlike the $\mathsf{AND\bar{a}NA}$session-based design, $\mathsf{AC^3N}$ does not suffer from interest linkability. This result is captured in the following theorem.

Theorem 3. *Independent interests corresponding to the same circuit session are not linkable.*

Proof. Let $\mathsf{int}{:}i$ and $\mathsf{int}{:}(i+1)$ be two subsequent interests issued by a consumer using the same circuit and received at router r_j. By the Encrypted Interest Generation procedure, after $\mathsf{int} : i$ is issued, $\mathsf{SessionIndex}_j^{i+1}$becomes $H(\mathsf{Session}_j^i + (\mathsf{SessionIV}_j^i + 1))$ and $\mathsf{SessionIV}_j^{i+1}$ becomes $\mathsf{SessionIV}_j^i + 1 (\bmod 2^{\kappa})$. Thus, the session index included in $\mathsf{int}{:}(i+1)$ is $\mathsf{SessionIndex}_j^{i+1}$. By the properties of H, $\mathsf{SessionIndex}_j^{i+1}$ is indistinguishable from $\mathsf{SessionIndex}_j^i$ since the inputs are different. Therefore, without knowledge of $\mathsf{SessionIV}_j^i$, an adversary cannot link $\mathsf{int}{:}i$ and $\mathsf{int}{:}(i+1)$ to the same session. This property thus holds recursively for any two interests $\mathsf{int}{:}i$ and $\mathsf{int}{:}(i+k)$, $(k > 2)$. Therefore any two independent interests corresponding to the same circuit session are unlinkable. □

B. OPERATION ANALYSIS

In addition to anonymity properties, we are also concerned with the correct operation of each AR supporting a session between two parties. In this context, we define session correctness as the ability of a consumer to correctly decrypt content that is generated *in response to* its original interest. That is, if a consumer issues an interest, it should be able to correctly decrypt the content that it receives. The following factors impact the correctness of the session:

1. Each AR $r_1, \ldots, r_n$ on the consumer-to-producer circuit should correctly recover the session identifier associated with the current session.
2. The session key streams should only be advanced upon the receipt of an interest corresponding to the consumer who initiated the session or content that is generated from the upstream router (potentially the producer) in the circuit.

The first item is necessary in order for each AR to correctly decrypt interests, encrypt content, and perform content signature generation and verification. The second item is necessary so that all content can be correctly decrypted by the consumer. We claim that, given a CCA-secure public key encryption scheme, the probability that either one of these factors being violated by an adversary $\mathcal{A}$ is negligible. Let $\mathsf{ForgeSession}$ and $\mathsf{KeyJump}$ denote the events corresponding to instances where an adversary creates a ciphertext that maps to a valid session identifier for *some* session currently supported by an AR (i.e., the forged session belongs to the routers session table ST), and the event that an adversary causes the key stream for *some* AR in a consumer-to-producer circuit to fall out of sync with the consumer.

By the design of $\mathsf{AC^3N}$, it should be clear that $\mathsf{KeyJump}$ occurs when $\mathsf{ForgeSession}$ occurs, since the key stream is only advanced upon receipt of an interest, but may also occur when an adversary successfully forges a MAC tag corresponding to the signature of a piece of content from the upstream router (or producer). We denote this latter event as $\mathsf{ContentMacForge}$. With the motivation in place, we now formally analyze the probabilities of these events occurring below. For notational convenience, we assume that each event only occurs as a result of some adversarial action, so we omit this relation in what follows.

Lemma 1. *For all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists some negligible function* negl *such that*

$$\Pr[\mathsf{ForgeSession}] \leq \mathsf{negl}(\kappa).$$

Proof. By the design of $\mathsf{AC^3N}$, we know that session identifiers are computed as the output of a collision resistant hash function $H : \{0,1\}^* \rightarrow \{0,1\}^m$, where $m = \mathsf{poly}(\kappa)$ (i.e., polynomial in the global security parameter). Consequently, forging a session identifier *without* the input to H implies that a collision was found, thus violating collision resistance of H. Thus, forging a session is equally hard as finding a collision in H, or more formally, $\Pr[\mathsf{Collision}(H) = 1] = \Pr[\mathsf{ForgeSession}]$. By the properties of collision resistance of H which states that $\Pr[\mathsf{Collision}(H) = 1] \leq \mathsf{negl}(\kappa)$ for some negligible function negl, it follows that $\Pr[\mathsf{ForgeSession}] \leq \mathsf{negl}(\kappa)$.

□

Lemma 2. *For all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists some negligible function* negl *such that*

$$\Pr[\mathsf{ContentMacForge}] \leq \mathsf{negl}(\kappa).$$

Proof. By the design of $\mathsf{AC^3N}$, the MAC scheme used for content symmetric content signature generation and verification is defined as the algorithms $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Verify})$, where Gen generates the secret key k used in the scheme, $\mathsf{Mac}_k(m)$ outputs the MAC tag $t := F_k(m)$ for some pseudo-random function F, and $\mathsf{Verify}_k(m, t)$ outputs 1 if $t = \mathsf{Mac}_k(m)$ and 0 otherwise. This is known and proven to be a secure MAC scheme, meaning that for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function negl such that $\Pr[\mathsf{MacForge}_{\mathcal{A},\Pi}(1^\kappa) = 1] \leq \mathsf{negl}(\kappa)$, and since $\mathsf{ContentMacForge}$ occurs exactly when the even $\mathsf{MacForge}$ occurs, we have that $\Pr[\mathsf{ContentMacForge}] \leq \mathsf{negl}(\kappa)$. □

Lemma 3. *For all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists some negligible function* negl *such that:*

$$\Pr[\mathsf{KeyJump}] \leq \mathsf{negl}(\kappa).$$

Proof. By the design of $\mathsf{AC^3N}$, it follows that $\Pr[\mathsf{KeyJump}] = \Pr[\mathsf{ForgeSession}] + \Pr[\mathsf{ContentMacForge}]$, and since the sum of two negligible functions is also negligible, it follows that there exists some negligible function negl such that $\Pr[\mathsf{KeyJump}] \leq \mathsf{negl}(\kappa)$. □

Theorem 4. *Session correctness of* $\mathsf{AC^3N}$ *is only violated with negligible probability.*

Proof. This follows immediately from Lemmas 1, 2, and 3 and the fact that the sum of two negligible functions is also negligible.[4] □

[4]This sum comes from the fact that probability of "failure" events must be taken into account in both directions of the session.