# Secure Off-Path Replication in Content-Centric Networks

Marc Mosko and Christopher A. Wood*
Computer Science Laboratory, Palo Alto Research Center
Email: mmosko@parc.com, woodc1@uci.edu

***Abstract*—We present SCR, a secure content replication protocol for the Content-Centric Networking (CCN) architecture. The goal of SCR is to allow a data producer to cache protected content in off-path semi-trusted caches or replicas. In contrast to the standard "take what you want" model of CCN, SCR ensures that no unauthorized, off-path entity can obtain data from these replicas, even if the content is encrypted. SCR allows a producer to encrypt data under any viable access control scheme, such as group-based access backed by broadcast encryption, and delegate the delivery of said content to distributed replicas in the network. SCR is analogous to "blind caching" in IP-based networks, which aim to provide caching as a service in the presence of end-to-end encryption via TLS. We discuss the design details and security features, e.g., revocation, of SCR. We then compare SCR to the HTTP(S)-based blind caching model. We show that our scheme can outperform blind caching due to (1) less protocol complexity and message overhead, (2) faster session establishment, and (3) the ability to obtain data in parallel from multiple, independent replicas.**

## I. Introduction

Information-Centric Networking (ICN) architectures such as Content-Centric Networking [1] and Named-Data Networking [2] emphasize the transfer of discrete named data packets from producers to consumers. The latter issue requests for data by name and the network is responsible for routing these towards the intended producer(s). Once found, a response is then forwarded back to the consumer along the same reverse path. Each content object typically carries some authenticator, e.g., a digital signature and public verification key. This allows consumers and routers to verify the integrity and authenticity of the content object. Moreover, depending on the sensitivity of the data, the payload may be optionally encrypted such that only authorized consumers can decrypt and use it. While forwarding this content object, each router may blindly and optionally cache it to satisfy future requests for the same data. In effect, this helps reduce unnecessary congestion and latency in the network due to flash floods for popular content.

Despite these benefits, untrusted caches are problematic from a security perspective. **First**, since producers have no knowledge about the use or location of caches, there is no architectural support for dynamically flushing content from the network. This is particularly problematic in the context of encryption-based access control. If a consumer's access to an encrypted content object needs to be revoked but that content still exists in the network, then it is possible for the consumer retrieve and decrypt the content outside of its window of access.

**Second**, untrusted caches support the "take what you want" access model. Any consumer with knowledge of the name of sensitive data, even if it's encrypted, may request the content from the network. By design, caching routers perform no authorization checks for requests. It is therefore trivial for an attacker to access sensitive data in its encrypted form.

**Third**, since producers and routers do not have any relationship, it is possible for one producer to maliciously *starve* competing producers out of cache resources. This can place an undue burden on legitimate consumers and producers who would normally benefit from the cache. Moreover, since routers have no trust relationship with producers, they are susceptible to content poisoning attacks. This is when malicious producers inject fake content, i.e., content with an incorrect payload but valid authenticator and verification key, into the network. CCN addresses this problem by allowing consumers to specify the identity of the verification key in their requests [3]. Routers must then check (a) that this identity matches that which is provided in the content and (b) that the authenticator is valid. However, this enforcement places yet another undue computational burden on caching routers. This could be abused as a DoS attack.

These problems stem from the simple fact that the network is untrusted and yet required, by design, to offer many services to applications. However, we claim that the benefits of CCN – name-based requests and responses with object security – without any of the aforementioned problems can be realized with semi-trusted routers or content replicas. To illustrate this point, we present SCR a secure content replication protocol for CCN. SCR enables producers to offload the distribution of access-controlled data to semi-trusted caches, or replicas. This is analogous to the recent "blind-caching" approach in IP-based networks (IPBC), e.g., [4], [5]. Our main contributions are as follows: First, we present the design and an analysis of SCR with respect to its relevant security properties. Second, we offer a detailed comparison of SCR to the recent IPBC strategies and discuss where SCR can out-perform these technologies.

## II. CCN Overview

### A. Architecture

Content Centric Networking (CCN) is one of the main ICN architectures. Named Data Networking (NDN) [2] is its academic dual with minor protocol and packet format differences.[1] This section overviews CCN with respect to the latest specifications [1] and the CCNx reference implementation. Given familiarity with either CCN or NDN, it can be skipped without loss of continuity.

*Work done while at PARC. This author is supported by the NSF Graduate Research Fellowship DGE-1321846.

[1]Therefore, we focus primarily on CCN in the remainder of this paper.

In contrast to IP, which focuses on inter-process communication and data transfer via addresses and ports, CCN [6], [1] focuses on *content* by making it named, addressable, and routable in the network. A content name is a URI-like [7] string composed of one or more variable-length segments. To obtain content, a user (consumer) issues a request, called an *interest* message, with the name of the desired content. This interest can be *satisfied* by either (1) a router storing the content in its cache or (2) the content producer. A *content object* message with the corresponding data is returned to the consumer upon satisfaction of the interest. Name matching in CCN is exact, e.g., an interest for /foo/bar/baz can only be satisfied by a content object named /foo/bar/baz.

Aside from the name, interest messages may include the following fields:

- `Payload` – a field that lets consumers push data to producers along with the request.
- `ContentObjectHashRestriction` – an optional hash value of the content being requested. If this field exists, the network guarantees the delivery of the exact content that consumer requests. From here forward, we refer to the `ContentObjectHashRestriction` as the ContentId.

Similar to interests, content objects also carry a payload and some additional metadata. However, unlike interests, they also usually carry an authenticator (digital signature or MAC) used to assert the correctness of the name-to-data binding. This authenticator allows consumers and routers to verify the authenticity and integrity of content. Content objects do not need to carry a name if the corresponding interest carried a ContentId. This is because the content can be matched to the interest by computing and checking its hash for equality with the ContentId from the interest. (This equality check is also used to verify the response.)

There are three types of entities in CCN:[2] (1) consumer, which issues interests for content, (2) producer, which generates and publishes content to the network, and (3) routers, which forward interest messages and content between consumers and producers. Each CCN entity maintains two components:

- *Forwarding Interest Base* (FIB) – a table of name prefixes and corresponding outgoing interfaces. The FIB is used to route interests based on longest-prefix-matching of their names.
- *Pending Interest Table* (PIT) – a table of outstanding (pending) interests and a set of corresponding incoming interfaces.

An entity may also maintain an optional *Content Store* (CS) used for caching. From here on, we use the terms *CS* and *cache* interchangeably. Routers use the FIB to forward interests towards producers and the PIT to forward content object messages along the reverse path to consumers. Specifically, upon receiving an interest, a router $R$ first checks its cache (if present) to see if it can satisfy this interest locally. If the content is not available locally and there are no pending interests for the same name in its PIT, $R$ forwards the interest to the next hop according to its FIB. For each forwarded interest, $R$ stores some state information in the PIT, including the name of the interest and the interface from which it arrived, so that content may be sent back to the consumer. If an interest for $N$ arrives while there is already an entry for the same content name in the PIT, $R$ only updates the arriving interface list of the matching PIT entry. When content is returned, $R$ forwards it to all of the corresponding interfaces listed in the PIT entry and then deletes said entry from the PIT. If a router receives a content object without a matching PIT entry, the message is silently discarded.

[2]A physical entity, or host, can be both a consumer and producer of content.

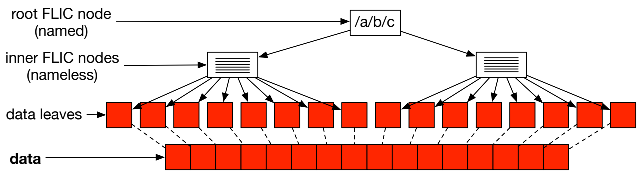

Fig. 1. Sample FLIC tree.

## B. Protocol Mechanics

Interests and content objects are the base for different types of messages in CCN. LINKs [1] are a type of content object whose body (payload) contains a CCN name and an optional ContentId. As content objects, LINKs may carry a signature so that the consumer or recipient can verify their correctness. LINKs therefore serve as a cryptographic binding of a (Name, ContentId) tuple.

FLICs are another type of content object message described in [8]. A FLIC message is part of a network-level collection of content objects. Each FLIC node in a collection contains a list of hash digests. A consumer uses a FLIC node's contents and their knowledge of a *locator* (or name prefix) to construct a sequence of interests with the name as the locator and hash digest as the ContentId.[3] Each of these pairs is called a *pointer*. FLIC nodes may contain pointers to normal content objects (data leaves) or other FLIC nodes, thus creating a DAG structure with a single root and normal content objects as the leaves. Typically, inner-nodes in this DAG are nameless and only the root node has a name. Consumers resolve FLIC root nodes to data leaves by performing an in-order depth-first-search of the pointers in a FLIC node, as shown in Figure 1.

Also, FLICs may contain optional metadata fields such as a locator to override the consumer's known locator, `DataDigest` to indicate the cryptographic hash digest of the data contained in *all* of the children pointers[4], and `DataSize` to indicate how many data bytes are contained in all of the children pointers. With knowledge of the total number of bytes in a FLIC node, as well as the size of data contained in each node, a consumer can randomly seek to different offsets in the DAG.

## C. Object Security and Access Control

Data integrity and authenticity are core properties of the CCN protocol. Security (confidentiality) of the content payload

[3]The purpose of FLICs is to bootstrap hash-based verification in contrast to digital signature verification, which is much more expensive.

[4]This is similar to a Merkle tree digest.

is delegated to the application. In general, there are two ways to protect this data: (1) by encrypting the response or (2) obfuscating the request. The first option restricts access to data by ensuring that only authorized consumers with the correct decryption keys (or a means of obtaining them) can access the content. Many variations of this approach have been proposed based on general group-based encryption [9], broadcast encryption [10], attribute-based encryption [11], and proxy re-encryption [12]. Kurihara et al. [13] generalized these specialized approaches in a framework called CCN-AC, an encryption-based access control framework to implement, specify, and enforce access policies. It uses CCN manifests (similar to FLICs) to encode access control specification information for a particular set of content objects. In contrast, the NDN-NBAC [14] uses namespace conventions to enable consumers to derive the names of decryption key(s) necessary to decrypt content.

As an alternate to content encryption under long-lived keys, session-based encapsulation can be used. To the best of our knowledge, the CCNx-KE protocol is the only one of its kind [15]. CCNx-KE allows a consumer to establish an ephemeral, forward-secure session with a *namespace*, e.g., /netflix, that can be used to request data packets *within the session*. Both the requests and responses are encapsulated by encryption with the session key. CCNx-KE is simple: the consumer needs to only know (a) the public key of the authentication producer and (b) their minimal routable prefix. CCNx-KE allows the authentication endpoint (producer) to *migrate* a session to a trusted service that can procure content (replica). (We will use this feature in our later designs.) To do so, the authenticator provides the consumer with a so-called $\mathsf{MoveToken}$ that is later relayed to the content provider. This token is constructed such that the content provider can extract the traffic secret associated with the previously verified session and bootstrap a new session. This exchange is shown in Figure 2. From here on, we will use the term $\mathsf{SessionInit}$ to refer to these steps in the CCNx-KE protocol.

The second data protection mechanism is known as interest-based access control (IBAC) [16], and it protects requests instead of responses. (However, it does not preclude responses being encrypted as well.) The main idea is that the content name can only be derived by authorized consumers. Using the standard IBAC scheme, two consumers in the same "access control group" issue identical requests for the same protected content if they are expected to benefit from a cached version. However, if replay attacks are a concern, then these IBAC-protected requests must also carry a unique nonce and timestamp in a signed envelope. The producer is responsible for providing the correct signature verification key(s) in the response so that this signature can be verified by any router or replica. Moreover, the replica must also check to see that the nonce and timestamp are not replays of a previous packet. Lastly, if revocation is necessary, then the consumer must provide an additional signature over its nonce and timestamp and provide its public key in the request. The verifying entity must verify this signature and the outer signature before responding with content.

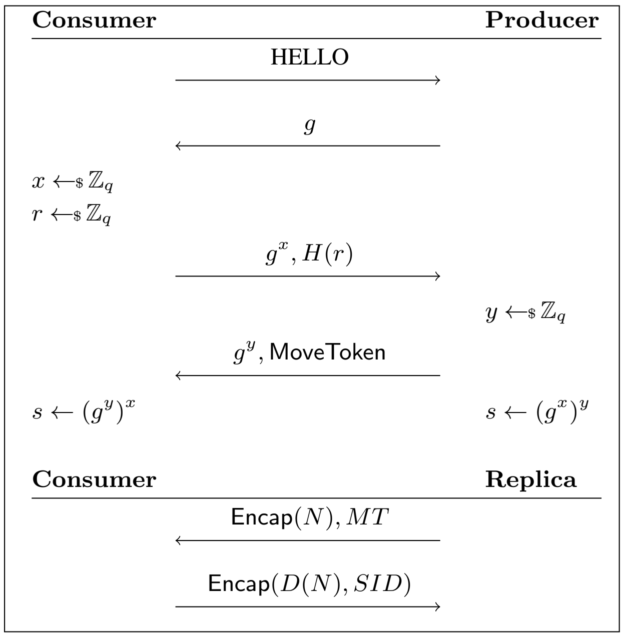

Fig. 2. The abbreviated CCNx-KE protocol illustrating usage of the $\mathsf{MoveToken}$ for 0-RTT data transfer from a replica.

## III. System and Security Model

Let $P$ be a producer that wishes to publish content under a namespace $N_P$. (Here, $N_P$ is the *minimal routable prefix* of $P$.) Let $\mathcal{R}$ be a set of replicas with whom $P$ has a trust relationship. $P$ trusts each replica $R_i \in \mathcal{R}$ to serve content under $N_P$. Each $R_i \in \mathcal{R}$ has a unique name prefix $N_{R_i}$ that can be used as a *locator* for accessing its stored content. $N_P$ and $N_{R_i}$, for each $R_i \in \mathcal{R}$, are propagated to the network using a routing protocol such as DCR [17] or NLSR [18].

We assume $P$ can pre-populate content it owns to each $R_i \in \mathcal{R}$. Consumers must then (a) obtain the locator for one or more replicas from which to fetch the content and (b) use the appropriate identifier(s) to indicate which content is desired. Content pre-population is handled in an *offline* process, whereas content retrieval is inherently *online*. In the remainder of this section we discuss designs for each of these tasks. Before doing so, we first describe the security model for the system and SCR protocol.

Let $\mathbb{U}(N)$ denote the set of authorized consumers for a content object with name $N$ generated and controlled by a producer $P$, and let $\bar{\mathbb{U}}(N)$ be its complement, i.e., the set of all unauthorized consumers. Let $\mathsf{Path}(Cr, P)$ be the set of all routers on the path between all consumers $Cr \in \mathbb{U}(N)$ and $P$. We assume the existence of an adversary Adv who can deploy and compromise any router $R \notin \mathsf{Path}(Cr, P)$.[5] Formally, we define Adv as a 3-tuple: $(\mathcal{P}_{\text{Adv}} \setminus \{P\}, \mathcal{C}_{\text{Adv}} \setminus \mathbb{U}(N), R_{\text{Adv}} \setminus \mathsf{Path}(Cr, P))$ where the components denote the set of compromised producers, consumers, and routers, respectively. If Adv controls a producer or a consumer then it is assumed to have complete and adaptive control over how they behave in an application session. Moreover, Adv can

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

control all of the timing, format, and actual information of each content through compromised nodes and links.

Let $\mathsf{Guess}$ denote the event where Adv successfully guesses the unique locator and identifier tuple necessary to fetch a content object from a replica. Let $\mathsf{Bypass}$ denote the event where Adv successfully bypasses the security protections in place at the replica and retrieves a protected content object. We define the security of SCR with respect to these two events as follows.

*Definition 1:* SCR is *secure* if $\Pr[\mathsf{Guess} + \mathsf{Bypass}] \leq \epsilon(\kappa)$ for any negligible function $\epsilon$ and a security parameter $\kappa$.

To justify our adversarial limitation to off-path routers, consider the following. If Adv can compromise a router $R \in \mathsf{Path}(Cr, P)$, then Adv is able to observe *all* content that flows along this path. Therefore, Adv can trivially observe the content in transit from the replica to an authorized consumer. Our goal here is to prevent an *off-path* Adv from ever getting access to an item in the replica, encrypted or not.

## IV. Secure Content Replication Protocol

SCR requires $C$ to learn the prefix of at least one replica in $\mathcal{R}$. To do so, we use nameless FLICs to create large collections of data that can be *moved* from $P$ to various $R_i \in \mathcal{R}$. There are generally three steps to the replication process: (1) creating the encrypted content that is to be distributed, (2) building the nameless FLIC on top of this content for efficient retrieval and verification, and (3) building a so-called *root manifest* (ROM) that contains pointers to duplicated manifest trees. In this section we describe each of these steps in detail. In the following text, we use $D(N)$ to denote the *application data* associated with the name $N$.

### A. Replication

**Step 1: Content Encryption**: The first step is to encrypt the data to be replicated under the desired authorization or access control group $\mathbb{U}(N)$. To do so, $P$ first generates a random symmetric key $DEK$ of length $\kappa$. It then encrypts $D(N)$ with $DEK$ using a mode of symmetric-key encryption that allows random access, e.g., AES-CTR, to produce encrypted content $C(N)$. In this case, since $DEK$ is randomly generated and never re-used, the IV for the CTR mode can be fixed and thus not encoded in the data.

**Step 2: Manifest Creation**: The goal of this step is to create a nameless FLIC for $C(N)$. Let $k_{\mathbb{U}(N)}$ be the group key associated with the authorization group $\mathbb{U}(N)$. $P$ first encrypts $DEK$ with $k_{\mathbb{U}(N)}$ to create $\bar{DEK}$ Then, $P$ segments $C(N)$ into chunks of the appropriate MTU size such that the data can fit within a single content object packet. The FLIC manifest $T(N)$, also called a *transport manifest*, is then constructed on top of this sequence of $C(N)$ chunks. Then, *for each replica $R_i$*, $P$ produces a signed LINK $L_i(N)$ that binds $N_{R_i}$ to $H(T_{root}(N))$, where $T_{root}(N)$ is the root of $T(N)$. Finally, $P$ creates the ROM manifest $ROM(N)$ which contains each link $\{L_i(N)\}$ and $\bar{DEK}$. When complete, $ROM(N)$, $T(N)$, $\bar{DEK}$, and $\{L_i(N)\}$ are produced as output.

**Step 3: Data Placement**: This step serves to upload the encrypted transport manifest to each replica. Specifically, for each $R_i \in \mathbb{R}$, $P$ sends $T(N)$ to $R_i$. If IBAC is to be used when resolving $T(N)$, then $P$ also uploads the group verification key $k^v_{\mathbb{U}(N)}$ associated with $\mathbb{U}(N)$.

### B. Retrieval

Once the data replication process is complete, consumers can then begin fetching the data from the replicas. This is done with IBAC or (CCNx-KE) sessions. By definition 1, both approaches are secure. Specifically, in search of $N$ and $D(N)$, $C$ first contacts $P$ to obtain $ROM(N)$ (with an interest for $N$). Once obtained, $C$ then uses (at least one of) the links $L_i(N)$ to request $T_{root}(N)$ from (at least one of) the replicas. $C$ then uses this to resolve $T(N)$ and obtain $C(N)$. After $C$ decrypts $\bar{DEK}$ from $ROM(N)$, it can use $DEK$ to decrypt $C(N)$ and obtain $D(N)$, thus completing the process.

One major benefit of this replication method is that the consumer has complete control over how it stripes its requests across the available replicas. Traditionally, CDN POPs use their own server-side load balancers or rely on DNS load balancing to manage flash connections from clients. Consistent hashing schemes ensure that client loads are evenly distributed among a set of available servers (replicas). However, this requires that all traffic go through the same load balancer in the network. With the clients in control, they can more evenly distribute their traffic using a similar consistent hashing scheme. (And it does not preclude server-side load balancing.) Specifically, let $R_1, \ldots, R_l$ be the set of replicas available to the consumer in nearby POPs. (A consumer would not use a replica that is out of reach.) Let $S$ be the total size of the data contained in the FLIC DAG leaves (specified in $T_{root}(N)$). To reduce its time to fetch all of $T(N)$, the consumer would request at most $\lceil S/l \rceil$ bytes of data from each replica. This optimally divides the traffic amongst the available replicas. The client could go further and use RTT measurements from each replica to adjust and prioritize non-congested sources to enable faster downloads. The details of such a transport protocol are outside the scope of this work.

### C. Traffic Protection

In SCR, $C$'s goal is to obtain replica prefixes and then, from those replicas, fetch data. To meet our security definition, $ROM(N)$ must be unavailable to Adv. This is because $ROM(N)$ contains links to the root transport manifests stored at each replica. Therefore, it is sufficient to protect $ROM(N)$ with any suitable form of access control. However, we recommend IBAC since it requires producers to perform consumer authorization checks before supplying $ROM(N)$. In contrast, protecting the traffic between consumers and replicas is more nuanced since we must protect against on-path attackers. Thus, there are (at least) two ways by which a consumer could fetch this information:

1) Request with IBAC protection.
2) Request over a session.

Figure 3 shows these traffic protection variants for the replication and fetch parts of SCR. Standard content-based encryption for the consumer-to-replica traffic is not suitable here since it does not prevent replay attacks and can be intercepted by on-path attackers.

Option (1) has the desired properties (a) that the requests are obfuscated and therefore reveal nothing about the original name and (b) replay attacks are not possible. However, this requires the requests to the replicas to be bound to a *specific* replica, rather than a general name prefix for a set of replicas. For example, imagine there exists two replicas $R_1$ and $R_2$ under the /foo/bar namespace, where the specific locators are $N_{R_1}$ =/foo/bar/$R_1$ and $N_{R_2}$ =/foo/bar/$R_2$, respectively. Now assume $C$ issues an IBAC-protected interest to /foo/bar that is serviced by $R_1$. If Adv captured this request he could then replay it to $R_2$. (This is of course not possible if $R_1$ and $R_2$ synchronized their previously observed nonce and timestamp pairs, but that can become quite expensive especially the number of replicas increases under a namespace.) Therefore, in this case, $C$ would need to send its request to $N_{R_1}$ or $N_{R_2}$ to bind the request to a specific replica. Another aspect to consider is that one standard router between $C$ and $R_i$ may cache the content and *not* enforce IBAC rules. This cache could be queried by Adv to obtain the IBAC-protected content. Therefore, as a preventative measure, the replica could set the `ExpiryTime` of the IBAC-protected response to zero to prevent it from being cached and served from a standard CCN router.

Assuming the requests are properly pinned so as to avoid replay attacks, there is still the problem of the computational overhead on the replica. IBAC with replay prevention requires the request to carry a digital signature that is verified by the replica with a known producer-provided key. [16] showed that the computational overhead can easily be exploited to attack the replica router. Therefore, this technique should be used sparingly.

Option (2) is a more efficient alternative to IBAC-protected request-response pair. $C$ establishes a session with $R_i$ and, within that session, all content is privately fetched. By the virtues of the key exchange protocol, Adv can replay a previous request for content but (a) will not receive a response since it would be detected by $R_i$ and (b) it would not be unable to decrypt the CCNx-KE encapsulation layer to obtain the protected content. Moreover, since $R_i$ will only perform symmetric-key cryptographic operations for in-window requests, the computational burden is eased.

Either of these options can be used to securely request data from $P$ or $R_i$. However, as we discussed, there are tradeoffs for each approach, which are summarized in Table II. In general, we recommend using IBAC to fetch $ROM(N)$ from $P$ assuming this is not abused as a DoS. Otherwise, we recommend using sessions for all traffic protection.

## V. Discussion

### A. Comparison to HTTP-Based Blind Caching

In TCP/IP, secure off-path replication is handled at the level of HTTP via "blind caches" or "out-of-band caches" [4], [5], or IPBC. In these systems, clients specify their willingness to use a cache in an initial request to the server. If the server approves of the cache, it provides a response whose body contains (a) the key that is used to decrypt the desired data and (b) a list of URLs from which the content can be fetched. The client then parses this lists and connects to one cache to obtain the resource. If the cache is not primed, then the cache reaches out to the origin to obtain the content. Once received in full,

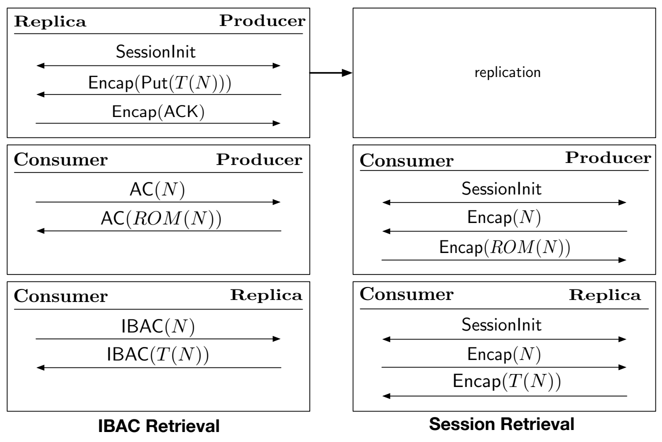

Fig. 3. Left: IBAC $C \leftrightarrow P$; Session $C \leftrightarrow R$. Right: Session $C \leftrightarrow P$; Session $C \leftrightarrow R$.

the client decrypts the data using the provided key and verifies its authenticity. Traffic between all three parties (client, server, and replica cache) is protected by TLS. [4] describes a solution to support a "cache cloud," wherein a top-level administrative cache can redirect requests to other caches under its control. This effectively acts as a load balancer.

Our secure replication protocol differs from these approaches in several key ways. First, the consumer is not required to obtain the desired resource from a single replica. It can stripe its requests across any number of replicas, thereby reducing the total fetch time. Although HTTP supports range requests [19], those are currently not used to fetch single resources. Were this technique to be used then SCR and IPBC would be comparable in this respect. Second, the data encryption key is *not* encrypted *within* an IBAC-protected response or session. It is encrypted end-to-end for the consumer (or its larger access control group). This prevents the key from being leaked due to (a) problems in TLS and (b) TLS termination middleboxes and proxies. Third, our CCN-based design is much simpler and more lightweight than the IP+TCP+TLS+HTTP approach. By collapsing this cross-layer functionality into a single protocol without losing any of the benefits of the independent layers, our design is simpler and therefore less likely to be implemented incorrectly. Fourthly, our design does not require sessions everywhere to operate correctly. The use of IBAC-protected requests to obtain root manifests from the producer is *simpler and faster* than a TCP and TLS handshake followed by a HTTP GET. Lastly, since its based on CCN, SCR benefits from receiver-based congestion control (see e.g., [20]) rather than sender-driven congestion control a la TCP.

### B. SCR Revocation

Revocation has long been a difficult problem to solve in standard CCN since (a) producers are unaware of where their content is cached and (b) they want to maximize cache utility while minimizing the window of possible exposure to revoked consumers. (A trivial approach to revoke content would be to set the `ExpiryTime` of all protected content to zero so that it is never cached and the producer has complete control over which consumers get which data.) However, with

TABLE I
COMPARISON OF IBAC AND SESSION-BASED COMMUNICATION

| **Comm.** | **IBAC** | | **Session-Based** | |
|---|---|---|---|---|
| | Pros | Cons | Pros | Cons |
| $C \leftrightarrow P$ | - One RTT to obtain replica information<br>- Replica information can be cached | - Computational bottleneck for (a single) $P$ | - Efficient response processing at $P$<br>- Session resumption for repeat consumers<br>- Provide MoveToken for fast session bootstrapping with replicas | - Session state storage consumption<br>- Multiple RTTs to obtain replica information |
| $C \leftrightarrow R$ | - Minimal number of packets sent to fetch data | - **Larger** computational bottleneck for content retrieval | - Efficient data transfer once session is bootstrapped | - Sessions are pinned to a specific replica |

trusted replicas, the problem is simplified, assuming there is no collusion between the replica(s) and consumer(s). In particular, since all requests for $ROM(N)$ go to $P$, the latter can simply not respond with $ROM(N)$ to revoked or unauthorized consumers. The replicas must also not allow session resumption to be used to prevent revoked consumers to fetch data beyond its accessibility window. If IBAC is used instead of sessions to fetch data, then the problem is still trivial to solve. When data is uploaded to the replica, the producer can supply an empty CRL. When a consumer's access is later revoked, the producer appends this consumer's public key digest to each replica's CRL. The replica must check for the existence of a request's public key digest in the CRL before responding with IBAC-protected content.

## VI. ANALYSIS

In this section we will compare SCR to IPBC. The key performance indicators we will study are the signaling overhead and message complexity. For both settings, we will assume caches are primed. For all packet size details, we will use what is currently outlined in the latest CCN [21] and FLIC [8] specifications. We will assume a standard Ethernet, IP, and TCP header size of 24B, 20B, and 20B, respectively. We will also assume a fixed and bounded name (URI) size of 256B to allow for equal comparison. For CCN, we will also assume that the packets do not contain any optional headers, which means that the packet header is a fixed size of 8B. Finally, we will also assume a standard link MTU of 1500B.

### *A. Signalling Overhead*

To assess the signaling overhead for the two schemes, we consider the more data-intensive part of the protocol – fetching data from the replicas. We will begin with the data retrieval process. In our approach, the data that's transferred from the replica to the consumer is all of $T(N)$. According to the CCN packet format, the FLIC data leaves will be of the size 1204B. Inside a FLIC node, there is in the worst case, 72B of encoding and metadata overhead. (This is assuming a FLIC with a single `BlockHashGroup` and metadata consisting of the `DataSize` and `DataDigest` fields. The nodes in $T(N)$ do not contain a locator since that is inherited from $ROM(N)$.) That means there is 1396B left over for storing the pointers, each of which is 40B. Thus, each node can safely store 34 pointers. This determines the fanout of $T(N)$, and from there we can determine the total number of nodes in $T(N)$ and its total size.

Now consider the blind cache approach. Under the previously mentioned constraints, and assuming the TCP MSS is set

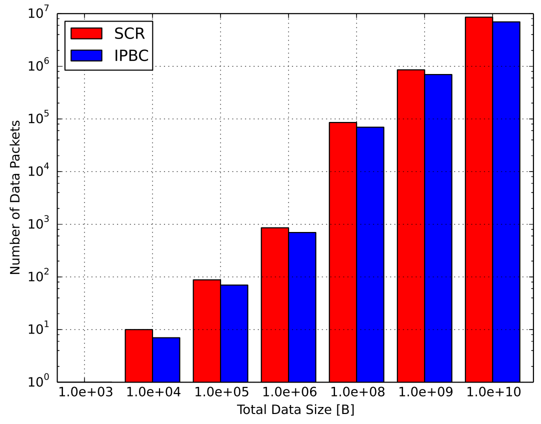

Fig. 4. Packet count difference between SCR and IPBC.

such that a packet will never exceed the link MTU, the HTTP response will have a header size of approximately 40B. (The URI is not included in the response, and we assume there are no optional headers, such as cookies. However, in practice, this is a very liberal lower bound and would likely be higher.) The rest of the response contains the data. The header and data are then segmented into an appropriate number of TCP packets based on the MSS. Inside each of these TCP packets, the framing overhead is 64B (for Ethernet, IPv4, and TCP), which leaves 1436B for data.

Using this signaling overhead, we plotted (a) the total number of frames needed to transfer the data from the producer (server) to the consumer (client). The results are shown in Figures 4 and 5. The IPBC approach yields fewer data packets overall, but not by a significant amount. However, recall that one differentiating feature between our scheme and IPBC is that IPBC assumes data will be transferred in full from a single cache. In contrast, SCR allows for clients to easily fetch data in parallel from multiple replicas.

In most cases, the size of $ROM(N)$ is negligible compared to $T(N)$ and can be fit within a single CCN or HTTP response (TCP data packet). Therefore, we consider these two cases equivalent.

### *B. Message Complexity*

We now compare the message complexity of SCR and IPBC with respect to the total number of end-to-end messages necessary to exchange data. Once data begins transferring,

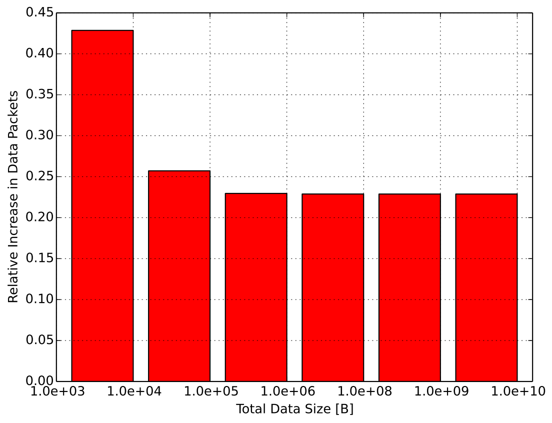

Fig. 5. Relative increase in the packet count from SCR over IPBC.

both CCN and TCP are similar with respect to the number of messages sent since CCN involves an explicit request and response whereas TCP requires a data packet and ACK. Therefore, we only consider the message complexity in bootstrapping this data transfer.

The IPBC scheme uses TLS and HTTPS for communication between all parties. Fresh TLS connections require a total of 7 end-to-end messages to create a TLS (1.2) session: 3 for the TCP handshake and 4 for the TLS handshake. CCNxKE requires only 4 end-to-end messages to establish the session. And finally, an IBAC-protected exchange is a single request and response with 2 messages. [6] Both IBAC and CCNxKE/TLS require signature verification (the former when verifying the request and the latter when validating message signatures), so this additional complexity is not a factor to consider. Therefore, SCR requires less end-to-end (EtE) messages compared to IPBC when initializing a data transfer. These numbers are summarized in Table II.

TABLE II
COMPARISON OF SCR AND IPBC MESSAGE COMPLEXITY

| Communication | #EtE Messages |
|---|---|
| SCR-IBAC | 2 |
| SCR-Session | 4 |
| IPBC-TLS | 7 |

## VII. CONCLUSION

We presented SCR, a secure content replication protocol. We discussed the design of SCR and its relevant security properties against off-path entities and honest-but-curious replicas. We also compared its performance to the HTTP(S)-based blind caching designs being pushed today. Our results indicate that SCR has the potential to outperform blind caching with respect to total data retrieval latency. For future work, we plan to implement our framework in both CCNx [22] and as an IP-based protocol built on top of QUIC [23] to conduct experiments in real-world systems.

[6]However, since IBAC can be abused by an intelligent adversary to induce computational DoS attacks, it should be used with caution.

## REFERENCES

[1] M. Mosko, I. Solis, and C. Wood, "CCNx Semantics," Internet Engineering Task Force, Internet-Draft draft-irtf-icnrg-ccnxsemantics-03, Jun. 2016, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-irtf-icnrg-ccnxsemantics-03

[2] L. Zhang, A. Afanasyev, J. Burke *et al.*, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

[3] C. Ghali, G. Tsudik, and E. Uzun, "Network-layer trust in named-data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 12–19, 2014.

[4] Eriksson, Göran A.P and Mattsson, John and Mitra, Nilo and Sarker, Zaheduzzaman, "Blind cache: a solution to content delivery challenges in an all-encrypted web," 2016. [Online]. Available: https://www.ericsson.com/res/thecompany/docs/publications/ericsson_review/2016/etr-secure-ott.pdf

[5] G. Eriksson, M. Thomson, and C. Holmberg, "An Architecture for Secure Content Delegation using HTTP," Internet Engineering Task Force, Internet-Draft draft-thomson-http-scd-01, Jun. 2016, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-thomson-http-scd-01

[6] V. Jacobson, D. K. Smetters, J. D. Thornton *et al.*, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 2009, pp. 1–12.

[7] T. Berners-Lee, R. Fielding, and L. Masinter, "RFC 2396: Uniform resource identifiers (URI): generic syntax," 1998.

[8] C. Tschudin and C. Wood, "File-Like ICN Collection (FLIC)," Internet Engineering Task Force, Internet-Draft draft-tschudin-icnrg-flic-01, Jul. 2016, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-tschudin-icnrg-flic-01

[9] D. K. Smetters, P. Golle, and J. D. Thornton, "CCNx access control specifications," PARC, Tech. Rep., Jul. 2010.

[10] S. Misra, R. Tourani, and N. E. Majd, "Secure content delivery in information-centric networks: Design, implementation, and analyses," in *ICN*, 2013.

[11] M. Ion, J. Zhang, and E. M. Schooler, "Toward content-centric privacy in icn: Attribute-based encryption and routing," in *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*. ACM, 2013, pp. 39–40.

[12] C. A. Wood and E. Uzun, "Flexible end-to-end content security in CCN," in *CCNC*, 2014.

[13] J. Kurihara, C. Wood, and E. Uzuin, "An encryption-based access control framework for content-centric networking," *IFIP*, 2015.

[14] Y. Yu, A. Afanasyev, and L. Zhang, "Name-based access control," *Named Data Networking Project, Technical Report NDN-0034*, 2015.

[15] M. Mosko, E. Uzun, and C. A. Wood, "Ccnx key exchange protocol version 1.0," 2016.

[16] C. Ghali, M. A. Schlosberg, G. Tsudik *et al.*, "Interest-based access control for content centric networks," in *Proceedings of the 2nd International Conference on Information-Centric Networking*. ACM, 2015, pp. 147–156.

[17] J. J. Garcia-Luna-Aceves, "Name-based content routing in information centric networks using distance information," in *Proceedings of the 1st international conference on Information-centric networking*. ACM, 2014, pp. 7–16.

[18] A. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang, "Nlsr: named-data link state routing protocol," in *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*. ACM, 2013, pp. 15–20.

[19] R. Fielding, Y. Lafon, and J. Reschke, "Hypertext transfer protocol (http/1.1): Range requests," Tech. Rep., 2014.

[20] K. Schneider, C. Yi, B. Zhang, and L. Zhang, "A practical congestion control scheme for named data networking," in *Proceedings of the 2016 conference on 3rd ACM Conference on Information-Centric Networking*. ACM, 2016, pp. 21–30.

[21] M. Mosko, I. Solis, and C. Wood, "CCNx Messages in TLV Format," Internet Engineering Task Force, Internet-Draft draft-irtf-icnrg-ccnxmessages-03, Jun. 2016, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-irtf-icnrg-ccnxmessages-03

[22] "CCNx distillery," https://github.com/parc/CCNx_Distillery, accessed: May 14, 2016.

[23] R. Hamilton, J. Iyengar, I. Swett, and A. Wilk, "Quic: A udp-based secure and reliable transport for http/2," *IETF, draft-tsvwg-quic-protocol-02*, 2016.