BEAD: Best Effort Autonomous Deletion in Content-Centric Networking

Cesar Ghali* Gene Tsudik* Christopher A. Wood+
University of California, Irvine
Email: {cghali, gene.tsudik, woodc1}@uci.edu

***Abstract*—A core feature of Content-Centric Networking (CCN) is opportunistic content caching in routers. It enables routers to satisfy content requests with in-network cached copies, thereby reducing bandwidth utilization, decreasing congestion, and improving overall content retrieval latency.**

One major drawback of in-network caching is that content producers have no knowledge about where their content is stored. This is problematic if a producer wishes to delete its content. In this paper, we show how to address this problem with a protocol called BEAD (Best-Effort Autonomous Deletion). It performs content deletion via small and secure packets that resemble current CCN messages. We discuss several methods of routing BEAD packets from producers to caching routers with varying levels of network overhead and efficacy. We assess BEAD's performance via simulations and provide a detailed analysis of its properties.

***Keywords*—Content-Centric Networking, Information-Centric Networking, caching, best-effort content deletion, controlled flooding, forwarding histories, accounting.**

I. INTRODUCTION

Content-Centric Networking (CCN) is a relatively recent internetworking paradigm touted as an alternative current IP-based Internet architecture. While IP traffic consists of packets between communicating end-points, CCN traffic is comprised of explicit requests for, and responses to, named content objects.

An important feature of name-based content retrieval is decoupling of content from its producer. This enables more natural content distribution by allowing routers to opportunistically cache content *within the network*. Cached content can be returned in response to future requests, called *interests*. This reduces the need to forward interests to content producers, thus lowering network congestion and content retrieval latency.

However, router caches are not mandatory in CCN. In some cases, caching content might not be beneficial, e.g., for routers with high content processing speeds, since high arrival rates translate to less time spent in cache. If the content's cache lifetime is very short, the probability of cache misses increases and the cache's utility decreases commensurately. Indeed, some prior literature shows (via simulations and experiments) that caching at the edges of the internetwork, i.e., at consumer-facing routers, is most beneficial and more cost-effective than doing so in the core, i.e., in transit routers [1].

*Supported by NSF award: "CNS-1040802: FIA: Collaborative Research: Named Data Networking (NDN)".
+Supported by NSF Graduate Research Fellowship DGE-1321846.

ISBN 978-3-901882-83-8

To help caching routers determine the lifetime of cached content, the latter includes an optional `ExpiryTime` field. Routers are expected to flush content once this time elapses. However, a router can choose to keep content cached beyond its lifetime. Lifetime of content in a particular router's cache depends entirely upon that router's implementation and policy. This uncertainty (or freedom) means that content may linger in the network for a very long time.

One notable drawback of this libertarian approach to caching is that some content may need to be deleted *before* `ExpiryTime` elapses. Consider content that frequently (yet sporadically) evolves over time, e.g., news articles. The appearance of breaking-news articles is unscheduled. As situations develop, updates and corrections to the content occur at unpredictable times. Such updates supersede previously distributed content by rendering it stale. Thus, in this case, producers need a way to remove old content. Another example is content (that has released and subsequently cached) which contains erroneous information. As errors are detected and corrected, a producer needs to flush the incorrect older version.

The deletion problem occurs because `ExpiryTime` is the only way for a producer to communicate *anticipated* content lifetime to the network. However, a producer can not change its mind after content has been published and distributed. Thus, there is a need for a safety mechanism for in-network content deletion. For this reason, we design such a technique called BEAD: Best-Effort and Autonomous Deletion. In the process, we encounter and address several challenges, including efficacy, performance, and security. We also experimentally assess the proposed technique.

The rest of this paper is organized as follows. Section II overviews CCN. Related work is summarized in Section III. Section IV presents minimal requirements for content deletion. Sections V and VI describe authentication and routing of deletion requests in BEAD, respectively. The BEAD technique is analyzed in Section VII and its performance is assessed in Section VIII. The paper ends with a discussion of BEAD optimizations and practical factors in Section IX. Future work is summarized in Section X.

II. CCN OVERVIEW

We now summarize the current CCN architecture [2]. Given familiarity with CCN, it can be skipped without loss of continuity.

Unlike IP, which focuses on addressable end-hosts, CCN emphasizes named and addressable content. A consumer issues a request, called an *interest*, specifying the name of desired content. CCN names are structured similar to URIs. For example, a content produced by the NSA might be named: `ccnx:/us/gov/DoD/NSA/Snowden-Diary`. An interest for a particular content named N is routed towards an authoritative producer for that content, based on N itself. In CCN, both interest and content messages have general-purpose `Payload` fields. Consumers can use an interest's `Payload` field to *push* information to producers, while producers use a content's `Payload` field to carry actual application data.

As an interest traverses the network, each router determines if a copy of requested content is cached in its Content Store (CS). If a cache hit occurs, the router satisfies the interest by sending the matching content on the interface on which the interest arrived. Otherwise, the router (1) records some state derived from the interest in its Pending Interest Table (PIT) in order to provide a backwards path for the future content, and (2) forwards the interest to the next hop(s) specified in its Forwarding Information Base (FIB). State retained in the PIT contains the content name and the interface(s) on which interests for that name have been received. A FIB is a routing table that maps hierarchical name prefixes to outbound interfaces. Longest-Prefix Matching (LPM) is used to determine the matching FIB entry.

A router R can collapse multiple interests into the same PIT entry whenever all of the following holds:

1) R receives an interest for name N
2) R does not have content N in its cache
3) R's PIT already contains an entry for N

When interest collapsing occurs, R only records the interface on which the new interest arrived and drops that interest. Whenever requested content arrives, R forwards it on all interfaces listed in the corresponding PIT entry. Afterwards, the PIT entry is flushed.

If no router can find a locally cached copy of requested content, the interest eventually reaches the producer that responds with the matching content, if possible. If the producer can not provide it (e.g., content does not exist) a NACK is generated [2], [3]. As content traverses the reverse path to the consumer, routers may choose to cache it in anticipation of future requests. As mentioned earlier, each content includes a producer-set `ExpiryTime` field. This value is content- and application-specific. However, each router can use any cache management algorithm, e.g., LRU or LFU.

III. Related Work

Lack of on-demand content deletion is a well-known problem in CCN [4]–[9]. The problem of *unsafe replicas* or stale content in CCN was first considered in [10]. Analytical and experimental assessment showed that: *"...the more frequently content is requested the higher is the chance of one request ending up in between a revocation and the eviction [of the stale key]."* The proposed method relies on a monotonically decreasing cache lifetime enforced by cooperating routers. This does not allow a producer to change the lifetime after content is published; it only seeks to minimize the time window when stale or unsafe replicas can be accessed.

[4] proposed a mechanism to implement revocation of content without input from the consumer. The proposed approach uses the ccnx-sync protocol to perform OCSP-like [11] synchronization of key data, i.e., determine content that has been revoked. This requires proactive behavior by each participating repository. [5] suggests using ChronoSync [12] to synchronize revoked key endorsements among group members. Revocation, however, is not the same as cache deletion. Revoked content, if still cached, can be inadvertently accessed by malicious or benign consumers.

[13] discussed a new caching technique allowing routers to proactively share content with downstream peers which did explicitly request that content. The suggested multicast forwarding strategy serves to increase the number of replicas in the network. However, unsolicited content objects can be seen as a form of attack similar to cache poisoning [7].

The concept of cost-aware caching in CCN was introduced in [14]–[18]. Various economic incentives for ISPs and ASs to cache content on behalf of producers have been explored. Cost-aware routers that cache based on popularity and economic incentives are studied in [19]. In general, the economic problem of supporting prioritized caching in the network is addressed without any attention to the inverse problem: how is content removed from caches?

IV. BEAD Requirements

Our motivation stems from the need to remove stale or erroneous content from the network, i.e., from routers' caches. One intuitive way of doing this is through the use of versioning, whereby the content naming format includes a component that explicitly reflects the current version. For example, the content of BBC's World News web-page could be named: `ccnx:/bbc/news/world/v2.4`. Alternatively, timestamps could be used. In that case, the same BBC page could be named `ccnx:/bbc/news/world/1449187200`.[1] However, in either case, is unclear how a consumer would determine (in advance) the current timestamp or version number, without which an interest can not be formed.[2]

The main problem with versioning and timestamps is that they can not handle unpredictable content updates. In current CCN design, producers are oblivious to where and for how long their content is stored in the network. Although this opportunistic caching is one of the biggest CCN advantages, it greatly complicates deletion of stale content. We believe that, in order to address the problem, producers need:

1) A way to communicate a single deletion request to all routers that might have cached offending content.
2) A way to efficiently secure deletion requests (allowing routers to quickly authenticate them) while avoiding trivial Denial of Service (DoS) attacks.

The first requirement is reminiscent of IP traceback – a class of techniques for identifying the original source of a (usually malicious) packets. In the context of IP, this is often framed as a mechanism to mitigate Denial of Service (DoS) attacks. In this paper, the goal is to learn *where* content was previously forwarded so that deletion requests can be routed along the same paths. These paths correspond to the original sources

[1] 1449187200 is 12/04/2015 at 12:00am UTC.

[2] There is one trivial way: a consumer contacts the producer directly and asks for the most recent version number or timestamp. However, this would incur an extra round-trip delay per content retrieval.

of interests for that content. Thus, ideas from IP traceback based on packet logging (e.g., [20]) and (deterministic or probabilistic) packet marking (e.g., [21], [22]) influence the design and forwarding strategies of BEAD messages.

We now show how to address these requirements with the BEAD technique.

V. Authenticating Deletion Requests

Producers must prove content ownership to routers that receive deletion requests. Otherwise, an adversary can impersonate a producer and induce content deletion, resulting in another form of DoS. One way to attain authentication is by a producer-generated digital signature on each deletion request. However, besides being inefficient, forcing routers to verify signatures on deletion requests can be itself parlayed into DoS attacks [7], [23]. Moreover, it involves public key retrieval, certificate handling and other messy (for routers) issues.

Our approach uses a light-weight token that proves content ownership. When a producer P creates a content object C, it generates a random λ-bit string x_C, called the *deletion token*. P then computes the digest of this token using a suitable cryptographic hash function[3], $y_C = H(x_C)$, and includes y_C in C. Later, if and when P wishes to delete C from the network, it includes x_C in the deletion request. (We assume that P can route these requests to any router caching C.) Upon receipt, each R verifies that y_C (cached alongside the content) matches $H(x_C)$. If so, R knows that P must have issued the request and deletes C from the cache.[4]

VI. Routing Deletion Requests

The remaining (though major) issue is how to route deletion requests from the producer to each caching router. This can be viewed as a multicast problem where producers must distribute a message (deletion request) to only a subset of nodes which could have cached the content.

Let $Int[N]$ and $C[N]$ be the interest and content messages referring to name N. The hash of $C[N]$ is a λ-bit string d, i.e., $d = H(C[N])$. Let $E[N, d]$ be a deletion request for content named N and hash digest d. Let $\mathbb{R}_N$ be the set of routers which cached $C[N]$. Finally, let the FIB of router $R \in \mathbb{R}_N$ be FIB^R.

From here on, we use the term erase to refer to deletion requests. Also, we assume that erase messages are authenticated using the method described in Section V.

A. Flooding

We begin by considering the simplest approach: reverse-path controlled flooding [24] of deletion requests. When $R \in \mathbb{R}_N$ receives $E[N, d]$, it forwards it on all interfaces except those which have a matching FIB entry.

Flooding offers some advantages, the most important of which is the ability to reach network edges even if routers on the producer-to-consumers paths no longer cache the content to be deleted. This is important since routers do not cache content uniformly and some may not even have caches. On the negative side, the volume of traffic generated from a single deletion request is very high and most deletion requests would be forwarded to routers that never even cached the target content.

B. Forwarder Histories for Content Traceback

In the optimal case, routers would only forward erase messages on interfaces on which the referenced content had been previously forwarded. In other words, erase messages should only be forwarded along the content distribution spanning tree where the producer is the root and leaves are the consumers who requested the content. One way to forward erase messages along the edges of this tree is for each router $R \in \mathbb{R}_N$ to maintain a forwarding history of $C[N]$. There are several places where this history can be kept, including: (1) in the cache where $C[N]$ is stored, (2) in a forwarding log (similar to [20], as a form of IP traceback) at each router, and (3) in the packets themselves. In each case, historical information constitutes a form of traceback that allows routers to identify where content was previously forwarded. We now describe each approach in more detail.

1) In-Cache Forwarding Histories: When a router caches $C[N]$ it can also remember the downstream interfaces where the cached copy was forwarded. We denote the set of these interfaces as $\mathbb{F}_N$. When a router receives an interest $Int[N]$ on interface F_i, it responds with $C[N]$ and adds F_i to $\mathbb{F}_N$. For a router with K interfaces, this additional state costs $\mathcal{O}(K)$ bits per cache entry. When a router caching $C[N]$ receives $E[N, d]$, it forwards it on all interfaces in $\mathbb{F}_N$.

In-cache forwarding histories are only effective for routers with large caches, since the lifetime of forwarding information is bound to the lifetime of cache entries, which can be small or even zero (if a router has no cache at all). Since a forwarding history $\mathbb{F}_N$ is deleted whenever $C[N]$ is flushed from the cache, this can lead to a future $E[N, d]$ not being forwarded to downstream routers which might still cache $C[N]$.

2) Local Forwarding Logs: Long-term packet logs have their roots in IP traceback techniques from the early 2000-s, e.g., [20], [25]. The problem here is similar: routers need long-term histories of packets (content) that were previously processed and forwarded. In this context, a history is a set-like data structure that allows content objects to be inserted and then later queried for membership. There are two types of histories: *lossless* and *lossy*. The former always return "yes" for content objects that have previously been inserted. In contrast, a lossy history might return false positives or negatives. Routers use these structures by associating one history to each interface. When a router receives $E[N, d]$ and $C[N]$ is not cached, it forwards $E[N, d]$ on each interface for which the corresponding forwarding interface history has a record of $C[N]$, i.e., all histories for which membership query returns "yes". This procedure is outlined in Figure 1.[5]

We now describe some ways of implement lossless and lossy histories that vary in their computation and memory requirements.

Lossless Forwarder Histories require a unique identifier to be kept after a content object has been forwarded. We

[3] Suitable hash functions include those with pre-image resistance, which means that, given y, it is difficult to find an x such that $y = H(x)$.

[4] This is due to the randomness of x_C and the collision-resistance of $H(\cdot)$.

[5] Similar to the flooding algorithm, this check is not performed for interfaces via which the content producer can be reached.

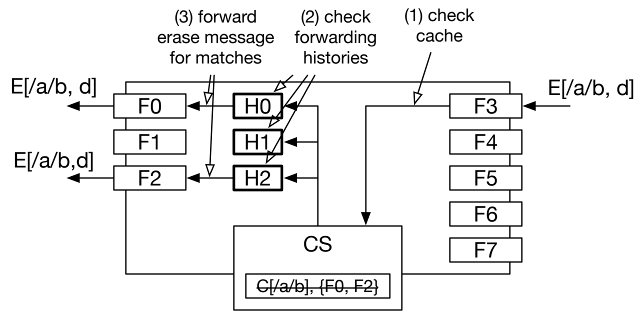


Fig. 1. $E[N, d]$ forwarding strategy based on per-interface forwarding histories. Upon receipt of an erase message a router searches the cache for the respective content. If the content is not present then the histories are examined and the history is forwarded as needed.

assume that content hash d serves as such an identifier (with collision probability negligible in λ). Implementing this type of forwarder history can be done trivially with a hash set HS_R as follows: to insert a content object into the history, compute and store d in HS_R. To query the history, return "yes" if $d \in HS_R$ and "no" otherwise. Insertion and lookup each require constant time.

Lossy Forwarder Histories are intended to store historical information in memory-constrained systems at the cost of false positives and false negatives. Similar to SPIE traceback [20], we use Bloom Filters (BFs) [26] to implement lossy forwarder histories. BFs enable probabilistic set membership queries.

The choices of BF properties, e.g., size and hash functions, impact efficacy of this technique. Filters that saturate too quickly result in high false positive rates. If all interface filters become saturated then erase is effectively broadcast. Therefore, it is important to eventually remove stale elements from filters. Unfortunately, a regular BF does not provide element removal. However, so-called Counting Bloom Filters (CBFs) [27] support set membership queries with removal. Instead of using bits to indicate set membership, CBFs use counters. When loading an element into CBF, the counters corresponding to the output of the hash functions are increased by one. Consequently, removing an element is done by decrementing the same counters. The problem with CBFs is that one must know the element to delete. Since routers would discard content after inserting them into these filters[6], they have no way of knowing what content is in the filter, and thus what elements to eventually delete. Their only recourse is to remove elements by decrementing counters at random. Intuitively, a router would delete random elements from the filter (the history) at a frequency which reflects the average `ExpiryTime` of received content. This can increase the false negative probability and reduce the possibility of delivering erase messages to their corresponding destination.

Variants of the CBF, such as Time-Decaying (TDBFs) [28], [29] and Stable (SBFs) [30] BFs can also be used. TDBFs have the property that elements are slowly removed from the filter over time, thereby keeping the rate of false positives minimized. However, the natural decay property may lead to false negatives. SBFs on the other hand are dynamically self-resized to keep the false probabilities minimized. Similar to CBFs and TDBFs, SBFs also suffer from false negatives.

[6]This is because content is only added to histories upon its removal from the cache.

3) Interest Marking for Content Traceback: Packet marking is a standard technique for IP traceback [21]. In the context of this work, marking is performed on interests to indicate sources of content requests. This information can be later used to learn the interface to which an erase needs to be sent. Specifically, erase messages can carry this marking information in order for routers to identify the appropriate downstream interfaces without storing any local state.

One trivial marking method is to append the arrival interface to each interest. Specifically, when R receives $Int[N]$ on face F_i, R prepends (R, F_i) to a list contained in the header of the interest. Producers record these traces upon receipt. In the event that an erase needs to be generated, P includes the trace in the erase and forwards it on the appropriate downstream interface. When R receives an erase with a trace it pops the last element (R, F_i) off the trace list and forwards it on the specified interface F_i.

This technique distributes the forwarding history among messages in the network. Therefore, this information must be secure. To illustrate this requirement, assume router R_i receives $E[N, d]$ with the sequence of hops

$$[(R_i, F_i), (R_{i-1}, F_{i-1}), \ldots, (R_2, F_2), (R_1, F_1)]$$

from interface F_{i+1}. R_i needs a way to securely guarantee that (R_i, F_i) was previously prepended, by itself, to the subsequence:

$$[(R_{i-1}, F_{i-1}), \ldots, (R_2, F_2), (R_1, F_1)].$$

Otherwise, the adversary can forge unsolicited erase messages with apparently correct routing sequences. Alternatively, one can modify existing sequences in erase messages to prevent them from being routed towards their destination.

One way of authenticating hop-sequence traces is for R_i to compute a Message Authentication Code (MAC) [31], [32] tag t_i over the (relevant) interest details, e.g., the name and previously present traces in the hop-sequence. R_i then adds the tuple (R_i, F_i, t_i) to the interest before forwarding it. Since erase messages carry the name of the content to be deleted, each router will be able to verify its pre-computed tag before forwarding erase messages downstream. Since routers compute and verify tags locally, a key management and distribution protocol is not required. We do, however, assume that routers are able to generate and maintain cryptographic keys of sufficient length necessary for MAC computation. As an added feature, hop-sequence information can also be used for detecting both interest and erase loops [33].

Although this technique of marking interest is effective to deliver erase messages to all routers on the path between consumers and producers, it has several drawbacks. One of this is that interest traces received by producers need to be stored so that they can be included in erase messages. This is due to the fact that (1) each trace corresponds to only one path in the network, and (2) interests issued by multiple consumers are most likely to traverse different paths to the producer. Producers can attempt to compile all collected traces in a data structure forming a spanning tree. This structure would be included in erase message headers, allowing routers to forwarder erase messages correctly. The main disadvantage of this approach is that the size of the data structure grows linearly with the number of consumers and is most likely to be greater than average allowed MTU. This means that erase messages will be fragmented (and possibly re-fragmented), and hop-by-hop reassembly is not avoidable. Another alternative is for producers to send multiple erase messages one for each set of traces correlated to a hop-sequence. In Section VII, we compare and evaluate the performance and resource consumption of these two techniques.

VII. Analysis

We now assess some routing strategies for erase messages. Let n_t^R be the total number of content objects forwarded by R at time t and let μ_F^R be R's content forwarding rate. Note that n_t^R grows monotonically as a function of μ_F^R.

A. Flooding Analysis

Recall that the reverse path flooding algorithm works by only sending broadcast messages to interfaces through which the *producer* is not reachable. Though very effective, this is highly unscalable. If all routers flooded erase messages then they would certainly be delivered to every $R \in \mathbb{R}_N$. However, the number of routers receiving a specific erase message is much larger than $|\mathbb{R}_N|$. Therefore, flooding should always be the last resort for erase messages. We assess the actual overhead of this technique in Section VIII.

B. Forwarding History Analysis

We now analyze performance of lossless and lossy forwarding histories described in Section VI-B.

1) Lossless Histories: The memory (and possibly computational) cost of a lossless forwarder history grows as a function of t. Thus, history collection will inevitably saturate memory at some point. Let n_{max}^R be the total size (in entries) of the history memory for R. Saturation is reached at time t such that $n_t^R \geq n_{max}^R$. We compute the time required to saturate a lossless forwarder history in two scenarios. We assume that each content object is $4,096$ bytes and hash digests are 32 bytes.

- **Consumer-facing router:** We assume a caching consumer-facing router (e.g., an access point) with 4GB of history storage and data rate of 100 Mbps. This data rate is equivalent to a content forwarding rate of $\mu_F^R = 3'200$ Cps (content packets per second). If R operates at full capacity with a full cache – i.e., storing every forwarded content requires eviction of an already cached one – it will take $41,943$ secs. for history storage to be saturated. This is roughly 12 hours. This window of time might be longer than the `ExpiryTime` of content objects that are subject to be erased. For instance, news feed pages are likely to be updated with a frequency faster than $1/12$ hours.
- **Core router:** We assume a non-caching CCN core router with 1TB of flash history storage and data rate of 10 Tbps, i.e., equivalent to $\mu_F^R = 335$ MCps. If R always operates at full capacity (i.e., forwards at 10 Tbps), lossless forwarder history can be saturated in 102 secs. In this case producers have a time window of less than 2 minutes to issue an erase message for content C after it was last served.

R's saturation time can be lengthened by increasing the size of the forwarder history. However, at this rate, the cost of adding more memory to make saturation time useful is far too expensive: 1TB for 2 minutes of history.

A very natural question arises: what happens when R's history storage is saturated? R can evict old history entries randomly, or according to some policy, e.g., LRU. However, keeping track of history entries' ages might lead to reduced performance. Another alternative is to divide history storage into smaller chunks, each corresponding to a set time window of history entries. Once history storage is saturated, the oldest chunk is erased to provide space for new entries. Using the consuming-facing router example above, 4GB of history storage can be divided into 12 chunks, each corresponding to one hour. The router could then erase the history recorded 12 hours ago in order to store history entries for the coming hour.

2) Lossy Histories: Lossy histories are useful when lossless ones are too expensive, e.g., in core network routers. Our approach to lossy forwarder history is based on Bloom Filters (BFs) – probabilistic data structures with tunable performance. Given an m-bit BF that stores n elements, the number of input hash functions k can be optimized and false positive probability can be estimated using Equation 1 [34]. Note that optimal value of k is also given as a function of m an n.

$$f(m, \cdot, n) \approx (0.6185)^{\frac{m}{n}}, \quad k = \ln(2) \cdot \frac{m}{n} \tag{1}$$

In practice, a router can optimize the number of hash functions in order to lower false positive probability. An upper bound of k can be set to limit hashing overhead.

As mentioned above, standard BFs do not support entry deletion, which is necessary to deal with the saturation problem. As indicated in [20], historical information for Internet-scale traffic (IP packets) can not last beyond a few minutes, which might still be less than what we needed for BEAD.

We now analyze lossy forwarder histories in the context of two scenarios mentioned above with the same history storage and data rates. We also assume that each content object added to a BF changes the value of new distinct k bits from 0 to 1. Clearly, this is unrealistic, since we do not consider the possibility of overlapping of hash function outputs for different input elements. However, this assumption captures the worst-case scenario.

- **Consumer-facing router:** To maintain a maximum false positive probability of 10^{-32}, a BF of size 4 GB can fit $n \leq 2 \times 10^8$ elements. Based on Equation 1, it requires $k = 120$ hash functions. Thus, it will take $89'478$ secs. (a little over one day) for the forwarder history to be saturated.
- **Core router:** To maintain the same false positive probability, a BF of size 1 TB can accommodate $n \leq 5.7 \times 10^8$ elements, which corresponds to $k = 107$ hashes. The forwarder history will be saturated in 245 secs.

One major drawback of using BFs for lossy forwarding histories is that history saturation is more difficult to resolve. Recall that, with lossless histories, a router can remove old entries in order to add new ones. A router could also delete the oldest chunk of the history once it is saturated. However, with lossy histories, a router can either: (1) flush the entire lossy history and start over, or (2) use CBFs which support element deletion with the use of counters. Unfortunately, this introduces false negative probabilities.

3) Packet Marking Analysis: Packet marking is computationally inexpensive since it requires a single MAC computation per (either interest or erase) packet. However, its drawback is increased memory footprint of the interest along every hop. Recall that traces in the hop-sequence consist of: (1) router identifier, (2) interface identifier, and (3) tag. Assuming a 2-byte interface identifier and a SHA-256-based MAC, the total size of each trace is 38 bytes. This corresponds to extra 608 bytes for each interest, assuming a 16-hop router-level path.[7]

We now compare two hop-sequence techniques described in Section VI-B3. Assume a tree topology with: (1) producer P at the root with height h, (2) 2^h consumers at the leaves with height 0, and (3) $2^h - 2$ routers. We assume all consumers request content C and all routers append hop-sequence traces to the corresponding interests. In this case, P receives 2^h interests, each with $h-1$ traces. If P includes all these traces in a single erase message, its size would grow by $\big(2^h \cdot (h-1)\big) \times 38$ bytes. This grows to 35 MB for $h = 16$, which is clearly impractical.[8] On the other hand, if P decides to send a separate erase to each consumer it would generate 2^h erase messages. The same overall volume of traces (35 MB) will be sent from P to consumers. However, it would be split into numerous erase messages. One advantage is that erase messages size will likely not exceed the path MTU and therefore not require fragmentation.

C. Summary of BEAD

As follows from the above, BEAD is not a single protocol. It is a set of techniques for generating erase messages and distributing them to routers which may have cached offending content. We presented several alternatives, each of which are practical in different network locations. For instance, consumer-facing (caching) routers can keep lossless or lossy histories for at least a day. Meanwhile, interest marking is better suited for core network routers. Therefore, we believe that all aforementioned techniques can be used, in combination, for routing erase messages. Our specific recommendations are as follows:

1) If R supports interest marking, the first tuple in the hop-sequence traces is valid and appended by the router itself, then information in the tuple is used to route the erase downstream.
2) If the content is in R's cache, then in-cache history is used to route the erase.
3) If the content is not in R's cache, but R keeps lossless or lossy histories, then they are used for erase message routing.
4) Otherwise, R floods received erase messages.

[7]The average Internet hop-count is currently 16 [35].

[8]We defer designing a more efficient scheme for combining hop-sequence traces to future work.

Recommendation 1 is most appropriate for core network routers, 2 and 3 for less busy edge network routers, and 4 as a failover mechanism. Most routers would likely prefer to drop erase messages instead of flooding them. This is why BEAD is *best-effort*: it does not guarantee that each erase message will be delivered to all entities caching the target content.

As mentioned before, not all published content is subject to future deletion. If routers can make this distinction, there is no need to record history entries about content that will not be deleted. Such distinction can be achieved by adding an optional `CanERASE` flag to content object headers. If this flag is not present, the default behavior is to assume that no erase messages will ever be sent for the corresponding content. Moreover, interests requesting content that will not be deleted are not required to be marked by routers. Producers could tell consumers what content is subject to deletion (i.e., an erase) by overloading catalogs or manifests. As described in [7] and [36], catalogs and manifests contain lists of Self-Certifying Names (SCNs) of content to be requested. This list is provided by the producer and can contain the `CanERASE` flag alongside each SCN. In this case, the interest header format should be modified to include this optional flag. Moreover, since it is not guaranteed that all content objects will be requested using SCNs, the default behavior of (core) routers should be to append hop-sequence traces to interests if the `CanERASE` flag is missing.

VIII. Simulation Results

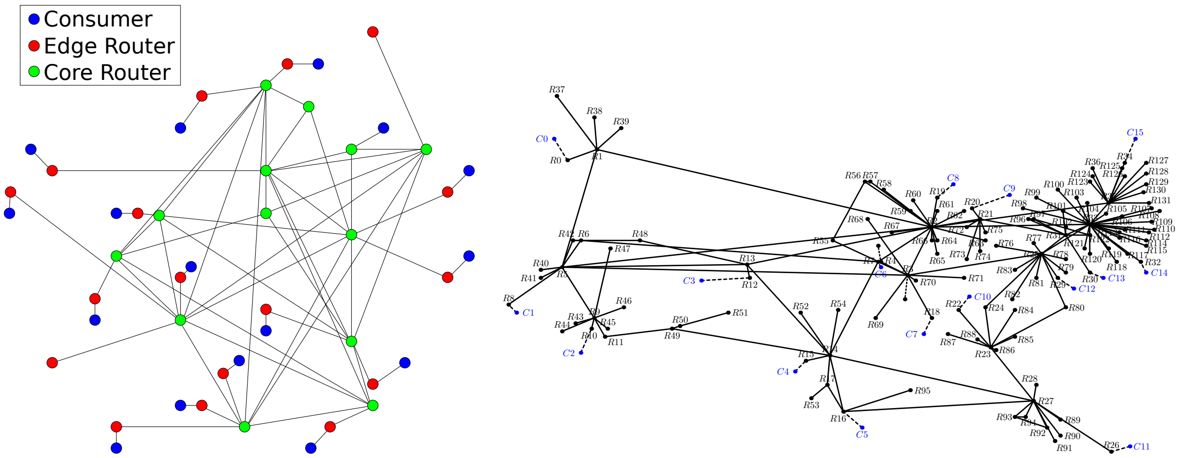
Our simulations focused on two properties of BEAD: network overhead (in terms of additional bytes added for erase messages) and forwarder overhead for processing erase messages, i.e., the average amount of time it takes to process each erase.

A. Network Overhead

To assess network overhead due to generating and forwarding erase messages we study the most costly scenario next to broadcasting: BEAD with lossless histories and routers with lossless links. To do so, we extended ndnSIM 2.0 [37] – an implementation of NDN architecture as a NS-3 [38] module for simulation purposes – to support erase messages. With this modified architecture, we ran two sets of experiments using the following topologies (shown in Figure 2):

- The DFN network, Deutsches ForschungsNetz (German Research Network) [39], [40]: a German network developed for research and education purposes which consists of 30 connected routers positioned in different areas of Germany. The blue dots in the figure represent group of consumers (10 consumers per blue dot) connected to edge routers (red dots), while the green dots represent core network routers.
- The AT&T backbone network [41]. This consists of over 130 routers. Each logical consumer in the figure represents multiple (5) physical consumers connected to an edge router.

In all experiments, consumers issue requests at a rate of 10 interests per second for content with the name prefix `/prefix/A` and monotonically increasing sequence number suffix. Every router uses a lossless history to record previously



(a) The DFN topology.

(b) The AT&T topology. Blue nodes represent clusters of consumers and black nodes represent routers.

Fig. 2. The DFN and AT&T topologies.

forwarded content objects for erase forwarding. Routers communicate over lossless links. Lastly, producers issue erase messages for 50% of their content every 1 second. (This may cause a producer to send a BEAD more than once.) Under these conditions, we measure router packet processing overhead with respect to content objects and erase messages. Figures 3(a) and 3(b) compare the overhead of processing content objects and erase messages in the DFN topology with 160 consumers. Similarly, Figures 3(c) and 3(d) show the same type of overhead in the AT&T topology with the same number of consumers. Comparatively, we find that erase messages contribute very little overhead to the network with respect to the bandwidth consumed by content objects. Specifically, the total amount of erase message traffic in the DFN topology is 1.8% of the total content objects traffic, whereas it is only 0.09% in the AT&T topology. To understand these differences, consider Figures 3(c) and 3(d). In Figure 3(c), core routers receive and forward more content packets than those not in the core. In Figure 3(d), those same core routers receive erase messages but do not forward all of them since they have were not in the history. This means that the content had previously been deleted. This is why the amount of egress traffic is less than the amount of ingress traffic.

We also assessed the actual computational overhead incurred by each router in these scenarios. The average time to process a single erase message for the DFN and AT&T scenarios are shown in Figures 4(a) and 4(b). We see that only a subset of the routers incur greater than 1.0ms to process an erase. These are the routers closest to the producer since they almost always receive, store, and forward erase messages.

IX. Monetizing Content Deletion

We now discuss potential economic incentives for routers and ISPs to support content deletion and implement BEAD.

A. BEAD & Accounting

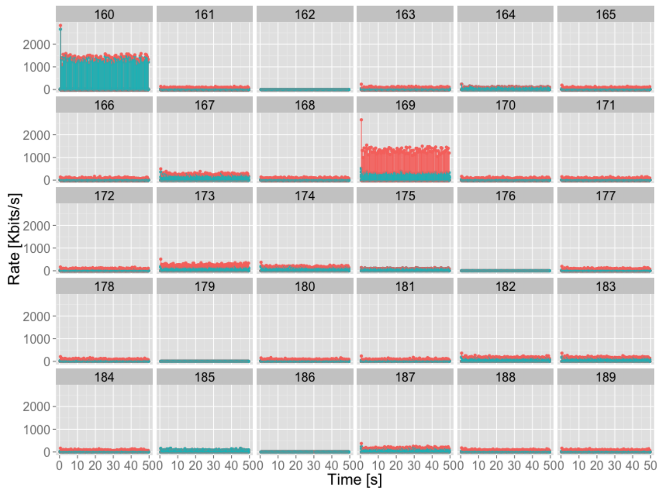
So far, we discussed how the network routes erase messages towards routers that possibly cache corresponding content. The main challenge is that producers do not know where such content is cached. We also acknowledge that BEAD is best-effort, unless flooding is used, which is undesirable.

However, if producers knew exactly where content is cached, then erase messages could be routed efficiently. For example, if a producer knew that a particular AS had a copy of the content cached *by some node in the system*, then the producer could specifically ask the AS to distribute an erase internally. This is far superior to routing erase messages in the core of the network in hopes that they *might* reach this AS (and any others with a cached copy).

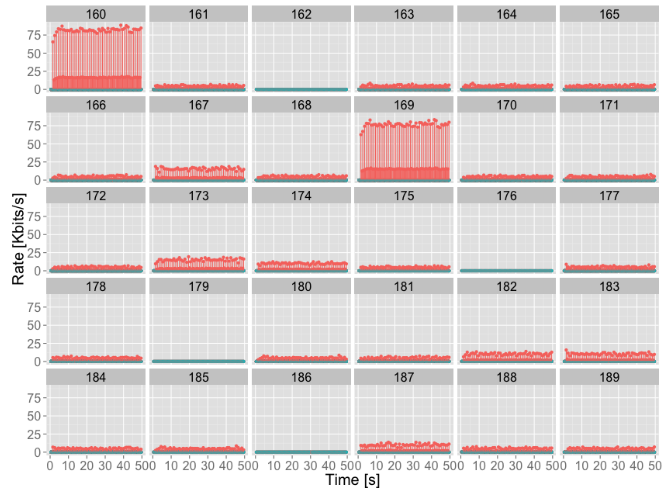
We believe that it is possible to distribute content caching location information along with accounting information. A scheme for secure accounting in CCN [42], suggests that routers should notify producers of content they serve from caches by sending a so-called "push interest" or *pInt*. This approach can be modified such that: (1) AS gateways send *pInt* messages when content is cached in their domain and (2) *pInt* messages carry the prefix of an AS accounting management server within the AS.[9] Whenever a producer wants to delete certain content, it sends an erase message to each accounting management server (one per AS) that previously reported caching corresponding content. Then, the latter distribute the erase message within their ASs. Intra-AS distribution can be achieved via techniques described in Section VI. In fact, flooding might well be appropriate for that purpose since erase messages would not traverse AS boundaries.

The relationship between accounting and BEAD is natural. This is because one of the important applications of accounting is to bill for cache space. From an economic perspective, it would not be surprising for in-network caching to become a paid service. Routers and ASs could offer caching services for producers. A reasonable extension to this service would be to also offer a deletion service via BEAD.

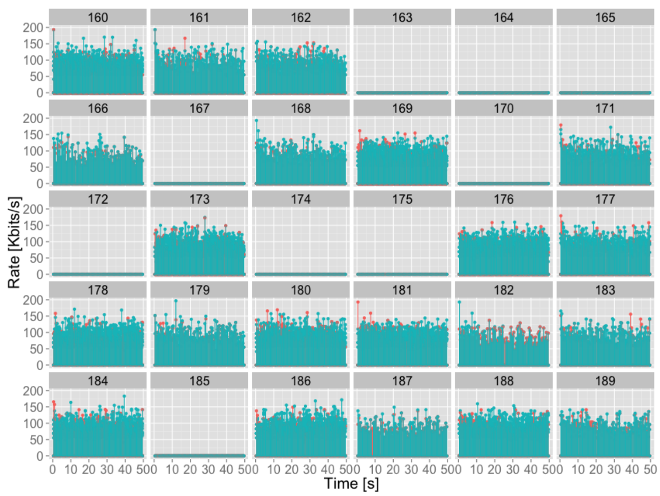
[9] Accounting management servers are centralized entities that manage accounting activities inside the AS.



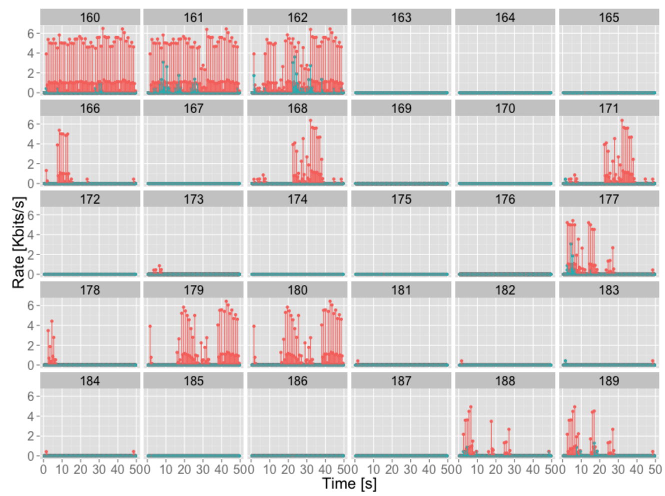
(a) Data processing overhead in the DFN topology with 160 consumers.



(b) erase message processing overhead in the DFN topology with 160 consumers.



(c) Data processing overhead in the AT&T topology with 160 consumers. Not all routers are present in the image.



(d) erase message processing overhead in the AT&T topology with 160 consumers. Not all routers are present in the image.

Fig. 3. Network overhead from processing erase messages. Routers are identified by integers in the range [160..189]. InData (OutData) and InErase (OutErase) correspond to the amount of content object and erase traffic received from (sent to) an upstream (downstream) node, respectively. Ingress data is shown in red and egress data is shown in blue.

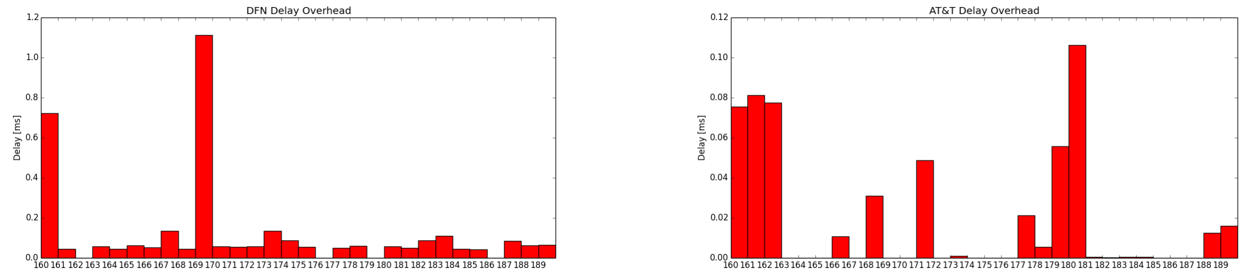
B. BEAD in the Core

Flooding in the network core is not viable as a means of distributing erase messages. Moreover, forwarder histories and packet marking are (relatively) expensive operations and too costly for the fast path in the core. ISPs will likely just drop these messages due to a lack of economic incentive to forward them. Thus, in any plausible CCN network – where producers and consumers are at the edges of a network, while most traffic is routed through the core – erase messages are most likely to be propagated along only half of producer-to-consumer path(s). This is troublesome since content is most likely to be cached near consumers in edge (or near-edge) routers, and erase messages might never reach these routers.

To address this issue, core routers must be incentivized to carry and forward erase messages from producers to consumers. Since erase messages will typically amplify traffic, producers should be expected to pay for this increase. As before, this effectively turns BEAD into a service provided by ISPs that complements monetized caching; producers who pay for cache space may also have the choice to pay for on-demand deletion via BEAD.

X. Conclusion

We proposed BEAD – a technique for best-effort autonomous deletion in CCN. BEAD is designed to solve the problem of stale or unsafe content in CCN. We described an efficient and lightweight form of authenticator for BEAD deletion requests and discussed several ways in which they could be routed from producers to consumers. We assessed the performance of each technique and verified the network overhead using simulations. For future work, we will expand the set of experiments to study the penetration impact due to erase message forwarding based on lossy histories. We will



(a) DFN topology with 160 consumers.

(b) AT&T topology with 160 consumers.

Fig. 4. Forward erase processing overhead in the DFN and AT&T topologies. The results are captured for each of the routes assessed in the bandwidth overhead experiments. Routers are identified by integers in the range [160..189] and correspond to the routers in Figure 3.

also study this metric in the presence of lossy links. Finally, we will formalize the integration of accounting and BEAD to form a comprehensive platform for premium caching in CCN.

REFERENCES

[1] A. Dabirmoghaddam *et al.*, "Understanding optimal caching and opportunistic caching at the edge of information-centric networks," in *ICN*, 2014.

[2] M. Mosko and I. Solis, "CCNx semantics," 2015, https://www.ietf.org/id/draft-irtf-icnrg-ccnxsemantics-00.txt.

[3] A. Compagno *et al.*, "To NACK or not to NACK? negative acknowledgments in information-centric networking," in *ICCCN*, 2015.

[4] G. Mauri and G. Verticale, "Distributing key revocation status in named data networking," in *Advances in Communication Networking*, 2013.

[5] Y. Yu *et al.*, "An endorsement-based key management system for decentralized NDN chat application," *Technical Report NDN-0023*, 2014.

[6] C. Ghali *et al.*, "Interest-based access control for content centric networks," in *ICN*, 2015.

[7] C. Ghali *et al.*, "Network-layer trust in named-data networking," *ACM CCR*, vol. 44, no. 5, 2014.

[8] Y. Yu *et al.*, "Schematizing trust in named data networking," in *ICN*, 2015.

[9] C. Wood *et al.*, "Flexible end-to-end content security in ccn," in *CCNC*, 2014.

[10] F. Angius *et al.*, "Drop dead data," https://users.soe.ucsc.edu/~cedric/papers/angius2015drop.pdf.

[11] M. Myers *et al.*, "RFC 2560: X.509 internet public key infrastructure online certificate status protocol-ocsp," *Internet Engineering Task Force*, 1999.

[12] Z. Zhu and A. Afanasyev, "Let's ChronoSync: Decentralized dataset state synchronization in named data networking," in *ICNP)*, 2013.

[13] R. Ishiyama *et al.*, "On the effectiveness of diffusive content caching in content-centric networking," in *APSITT*, 2012.

[14] P. K. Agyapong and M. Sirbu, "Economic incentives in information-centric networking: implications for protocol design and public policy," *IEEE Communications Magazine*, vol. 50, no. 12, 2012.

[15] A. Araldo *et al.*, "Cost-aware caching: optimizing cache provisioning and object placement in ICN," in *GLOBECOM*, 2014.

[16] C. Wang and J. W. Byers, "Incentivizing efficient content placement in a global content oriented network," Technical Report BUCS-TR-2012-012, Boston University, Tech. Rep., 2012.

[17] A. Araldo *et al.*, "Cost-aware caching: Caching more (costly items) for less (ISPs operational expenditures)," *TPDS*, 2015.

[18] K. Suksomboon *et al.*, "On incentive-based inter-domain caching for content delivery in future internet architectures," in *AINTEC*, 2012.

[19] A. Araldo *et al.*, "Design and evaluation of cost-aware information centric routers," in *ICN*, 2014.

[20] A. C. Snoeren *et al.*, "Hash-based IP traceback," in *ACM CCR*, vol. 31, no. 4, 2001.

[21] M. T. Goodrich, "Efficient packet marking for large-scale IP traceback," in *CCS*, 2002.

[22] A. Belenky and N. Ansari, "IP traceback with deterministic packet marking," *IEEE communications letters*, vol. 7, no. 4, 2003.

[23] P. Gasti *et al.*, "Dos and ddos in named data networking," in *ICCCN*, 2013.

[24] F. Baker and P. Savola, "RFC 3704: Ingress filtering for multihomed networks," Tech. Rep., 2004.

[25] R. Stone *et al.*, "CenterTrack: An IP overlay network for tracking DoS floods." in *USENIX*, 2000.

[26] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, 1970.

[27] L. Fan *et al.*, "Summary cache: a scalable wide-area web cache sharing protocol," *TON*, vol. 8, no. 3, 2000.

[28] L. Zhang and Y. Guan, "Detecting click fraud in pay-per-click streams of online advertising networks," in *ICDCS*, 2008.

[29] G. Koloniari *et al.*, "One is enough: distributed filtering for duplicate elimination," in *CIKM*, 2011.

[30] F. Deng and D. Rafiei, "Approximately detecting duplicates for streaming data using stable bloom filters," in *SIGMOD/PODS*, 2006.

[31] H. Krawczyk *et al.*, "RFC 2104: HMAC: Keyed-hashing for message authentication," 1997.

[32] P. Gutmann, "RFC 6476: Using message authentication code (MAC) encryption in the cryptographic message syntax (CMS)," 2012.

[33] J. Garcia-Luna-Aceves and M. Mirzazad-Barijough, "Enabling correct interest forwarding and retransmissions in a content centric network," in *ANCS*, 2015.

[34] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet mathematics*, vol. 1, no. 4, 2004.

[35] F. Begtasevic and P. Van Mieghem, "Measurements of the hopcount in internet," in *PAM*, 2001.

[36] J. Kurihara *et al.*, "An encryption-based access control framework for content-centric networking," in *IFIP Networking*, 2015.

[37] S. Mastorakis *et al.*, "ndnSIM 2.0: A new version of the NDN simulator for NS-3," Technical Report, 2015.

[38] "Network simulator 3 (NS-3)," http://www.nsnam.org/.

[39] "DFN-Verein," http://www.dfn.de/.

[40] "DFN-Verein: DFN-NOC," http://www.dfn.de/dienstleistungen/dfninternet/noc/.

[41] A. Compagno *et al.*, "Poseidon: Mitigating interest flooding DDoS attacks in named data networking," in *LCN*, 2013.

[42] C. Ghali *et al.*, "Practical accounting in content-centric networking," in *NOMS*, 2016.