Privacy-aware Multipath Video Caching for Content-Centric Networks

Qinghua Wu, Zhenyu Li, Gareth Tyson, Steve Uhlig, Mohamed Ali Kaafar, Gaogang Xie

***Abstract*—The prevalence of Internet video streaming challenges the design and operation of modern networks. Content Centric Networking (CCN) has been proposed to address the challenges through ubiquitous in-network caching. While the expected benefits include higher performance and lower bandwidth consumption, CCN introduces new privacy issues at layer 3. This is because adversaries could infer the content consumed by others by checking cached data in routers. In this paper, we first analyse the design space to improve both caching performance and cache privacy for video delivery in CCN. In light of the observation that these two metrics need to be balanced, we propose *CodingCache*. It adopts network coding and random forwarding to exploit the potentials of multipath routing in CCN to improve both the diversity of cached content along different paths and the anonymity set for consumers. We evaluate CodingCache through extensive experiments based on a real-world topology and a unique dataset of video access logs from a large-scale commercial video service. Our results demonstrate that, compared with existing CCN strategies, CodingCache is able to increase the cache hit rate while also improve the use of the caches across the network, together with reasonable cache privacy. Furthermore, we show that CodingCache is complementary to existing caching strategies, providing the combined benefits of coding and advanced caching.**

I. Introduction

Internet video services have become extremely popular in recent years [1]. As such, competitive video providers have been striving to provide users with higher performance streaming (i.e. low buffering ratio, high bitrate [2]), while Internet Service Providers (ISPs) have been simultaneously trying to mitigate the load on their network.

An emerging technology promising to address these needs is *Content Centric Networking* (CCN). CCN is a network architecture intended to facilitate large-scale content delivery by shifting the communication paradigm from *where* to *what*. CCN is built on named content. Content consumers issue *interest* packets, that are forwarded hop-by-hop (at layer-3) to sources based on the name of the requested content. Each CCN router is capable of caching data packets, serving future requests through its cache, rather than forwarding the requests to the origin of the data. The voluminous nature of video means that this architecture is particularly well suited for serving such content. Hence, we argue that video delivery could be a powerful usecase for CCN's approach of ubiquitous caching [3].

Q. Wu, Z. Li (corresponding author) and G. Xie are with Institute of Computing Technology, Chinese Academy of Sciences (CAS). Q. Wu is also with University of CAS. E-mail: {wuqinghua, zyli, xie}@ict.ac.cn

G. Tyson and S. Uhlig are with Queen Mary University of London, UK. E-mail: {g.tyson, steve.uhlig}@qmul.ac.uk.

M. A. Kaafar is with NICTA, Australia. E-mail: dali.kaafar@nicta.com.au.

Despite the benefits of caching, there remain a number of challenges. Of particular interest to us is the problem of *user privacy* [4], [5]. By explicitly naming data it becomes possible for third parties to potentially monitor user activity. For example, if two users, Bob and Alice, share a wireless access point and Alice obtains a cache hit at the first hop, she can infer the content that has previously been requested by Bob. We find this type of privacy intrusion particularly worrisome for video content as there are many intuitive scenarios that are inherently private in nature. Beyond this, video consumption patterns can also be used to infer a wealth of personal information, e.g. gender, age and relationship status [6], [7]. Hence, we argue that privacy considerations should be explicitly built into any future CCN deployment while, importantly, not undermining the performance advantages gained through ubiquitous caching. Failing to fulfill *both* requirements will be unlikely to motivate deployment. Consequently, we ask how can a *privacy-aware* caching algorithm be designed that also achieves *high performance* in CCN for video delivery?

To answer this question, we begin by analysing the trade-offs between caching performance and privacy in CCN, and observe that caching performance and privacy, indeed, need to be balanced. On the one hand, improving privacy requires that routers along different paths cache identical contents to hide user activity; while, on the other hand, improving performance requires caches along different paths to store a diversity of objects to better utilise storage capacity. As a key contribution, we focus on these questions in multipath environments.

In light of the above observations, we propose *CodingCache*, aimed at running in conjunction with CCN [8], [9]. CodingCache exploits multipath routing to increase the number of potential users that an individual cache serves. Through this, we obscure individuals' activities by increasing the number of potential users who may have left an object in the cache. Or, from a user's perspective, CodingCache increases the number of routers that can potentially be responsible for the content requests. We formally denote such routers as the anonymity set for a user and use this metric to measure the privacy throughout the paper. Although CodingCache could be used for any content, we particularly focus on video delivery due to its prominence and scale [1].

To enable high performance multipath routing, we use Random Linear Network Coding (RLNC) [10] to encode several video content chunks into one block. A coded block could serve the requests for any of the original chunks that the coded block is made of. In CodingCache, routers on different paths between the server and the consumers cache linearly independent coded chunks of the video content. In this way, the diversity of cached content along different paths improves, which can increase caching performance as well as obscure the original requester.

We evaluate CodingCache using trace-driven simulations based on two real-life datasets: (*i*) a unique dataset of video request logs from one of the largest Internet video providers in China, PPTV; and (*ii*) a large network topology of a Chinese ISP (China Telecom [11]). The results show that CodingCache achieves impressive performance in terms of both performance and privacy. In particular, compared with the native CCN caching strategy, CodingCache nearly doubles the cache hit rate and makes use of all available next-hop routers as the anonymity set. To summarize, we make the following contributions on video delivery in CCN:

1) We explore the design space of CCN caching from the perspective of balancing caching performance and privacy, particularly for sensitive video content.
2) We detail the design of CodingCache, a multipath CCN cache scheme that uses network coding to improve both performance and cache privacy.
3) We evaluate the performance of CodingCache for video delivery and compare it with existing schemes through extensive trace-based experiments. We show the benefits of CodingCache in terms of caching performance as well as privacy, and highlight the potential of network coding in CCN.

The remainder of this paper is organised as follows. In Section II, we briefly describe the adversary model and survey the related work. Section III explores the design space of caching performance and cache privacy. Section IV presents CodingCache and analyses its performance. In Section V, we evaluate CodingCache through experiments. Section VI concludes the work.

II. Background

A. Adversary Model

Before continuing, it is important to formalise the adversary model for attackers attempting to undermine user privacy. We assume a single attacker, who wishes to discover if a user (i.e. victim) has accessed a set of video objects. The attacker attempts to attach themselves topologically near to their victim; in the worst case, they would share the same access router (e.g. WiFi in a cafe). The attacker has no privileges beyond other users.

To perform an attack, the adversary sends interest requests for videos they are curious about. By inspecting the hop count (or timing information) on the returned data packet, the attacker can check if a nearby cached copy was returned. The closer the cache is, the smaller the possible set of consumers who requested it. This can be used to infer if a nearby party requested a specific video object. This is a particularly powerful attack when dealing with unpopular videos (i.e. rare content), which often carry more information for personalized user interests than popular ones [4], [12]. Note that this attack is not possible in an IP architecture as it does not perform local caching.

B. Related Work

1) Caching Performance: There has been a long-standing interest in analysing CCN caching performance through both analytical models [13], [14], [15], and simulations [16], [17]. These works show that caching performance is affected by both caching decisions and forwarding strategy.

Existing studies on CCN caching primarily improve performance by investigating caching decision strategies [18], [19], [20], [21], [22], [17]. There are several decision strategies, including the native CCN caching strategy (denoted as CEE in the rest of the paper) [8], LCD [18] and ProbCache [22]; these all decide which router(s) cache the content along the transmission path. Other strategies preferentially cache content on routers according to where routers are located in the topology [19], [21], or the content popularity [20]. The specific benefits for video distribution have also been explored, e.g. [23], [24], [3]. While these works consider the performance issue of caches in CCN, most overlook the privacy aspect.

Another trend is exploring the joint design of caching and forwarding strategies. Both of these affect the cache performance in CCN; further, they have an impact on each other [25], [26]. Rossini et al. [16] evaluate multipath forwarding strategies on caching performance. In [27], interest packets are forwarded through routes with the highest priority among the multiple available interfaces. This has been shown not to result in considerable performance benefits when there are multiple available paths [16]. Instead, it highlights that the key to caching performance is to enlarge the diversity of cached content and to dispatch requests correspondingly to exploit this diversity. We exploit this observation to design CodingCache, which precisely aims to improve the diversity of cached video content.

2) Cache Privacy: Privacy in CCN is less well studied. A few works have investigated cache privacy threats in CCN [4], [5], [28], [29]. For example, Acs et al. [28] propose a random delaying strategy to defend against timing attacks. Chaabane et al. [5] adopt probabilistic caching [30] to undermine the accuracy of privacy attacks. Compared to previous work that improves cache privacy at the cost of performance, we aim to improve the cache performance while preserving reasonable levels of cache privacy. Unlike [5], we also aim to achieve this without the need for explicit collaborative algorithms.

3) Network Coding: Network coding [10], [31] has been used to improve network throughput and scalability. For example, it has been used in wireless networks [32] and video systems [33]. To the best of our knowledge, the only other work to explore this in the field of CCN is Montpetit et al. [34]. However, it no longer follows the one-interest-one-data design principle of CCN [9]. Further, they do not explore the potential of using network coding for privacy enhancing purposes. A key contribution of our work is expanding upon this initial work to highlight the great potential of network coding in CCN.

III. Design Space of Caching Performance and Privacy with Multiple Paths

CodingCache is specifically designed for operation in multi-path environments. As such, we first analyse the design space of caching performance and privacy when using *multiple* paths between video servers and consumers. As of yet, this is a poorly understood topic. As previously stated, caching behaviour is determined by both caching and forwarding strategies, which include whether to cache an object, which object in the cache to replace, and to which next hop router a request is forwarded to. As our design and analysis are not tied to a special caching strategy, we use the native CCN caching strategy (i.e. CEE) here for illustration. We consider three standard forwarding strategies for deciding which path a request is forwarded along: (*i*) native CCN [27], which is single path; (*ii*) hash-based, which deterministically selects between paths; and (*iii*) random, which

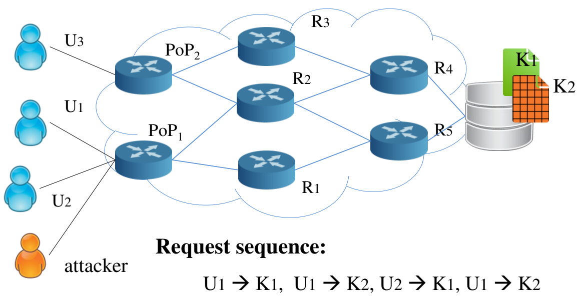


Fig. 1: An example to show the trade-off between caching performance and cache privacy. The content server provides an object consisting of 2 chunks, K_1 and K_2. Consumers U_1, U_2 and the attacker access the content through PoP_1. Consumer U_3 is connected to the network through PoP_2. Each router on the path can only cache at most one chunk (i.e. cache capacity $C = 1$). The request sequence is shown in the bottom of the figure.

randomly selects between multiple possible paths. In the following, we use the example in Figure 1, as well as formal analysis, to explore the caching performance and privacy under various strategies. We later use this analysis to drive our own design process.

A. Exploring Caching performance

Caching performance is measured by the cache hit rate and the average transmission path length. Ideally, hit rates should be kept high, and path lengths kept low. The fundamental idea of improving performance when multiple paths are available is to increase the diversity of the cached content along available paths. Theoretically, doing this would improve both metrics.

We use the example in Figure 1 to explore the cache performance under various strategies. Table I lists each forwarding decision of the three strategies (native, hash, random) in Figure 1 as well as the corresponding caching results. From the table, both the native CCN strategy (noted as *native*) and *random* strategy achieve no cache hits. While the hash-based strategy (noted as *hash*) has with 1 cache hit, higher than that of the other two strategies.

Next, we formally analyse the reasoning behind the performance of the above three forwarding strategies. Assume that there are n ($n > 1$) next-hop routers (namely $R_1, \cdots, R_n$) to which PoP_1 could forward interest packets. Each router has cache capacity C. A video content chunk K will be cached in R_i with probability $g(p_r, C)$, where *request rate* p_r for an object at a router is defined as the ratio of requests on this content chunk to all requests received by that router. Function $g(\cdot, \cdot)$ is an increasing function of the request rate and cache capacity, as higher request rate or a larger

TABLE I: Trade-off between caching performance and cache privacy under various strategies.

	Seq	Fwd to	R1 Cache	R2 Cache	Cache Hit?	| Anony Set |
	1	R1	K1	NIL	×	
(a)	2	R1	K2	NIL	×	1
native CCN	3	R1	K1	NIL	×	
	4	R1	K2	NIL	×	
	1	R1	K1	NIL	×	
(b)	2	R2	K1	K2	×	1
hash strategy	3	R1	K1	K2	√	
	4	R2	K1	K2	√	
	1	R2	NIL	K1	×	
(c)	2	R1	K2	K1	×	2
random strategy	3	R1	K1	K1	×	
	4	R2	K1	K2	×	

cache capacity will make the content more available.

Under the native forwarding strategy, suppose that a router R_1 has the highest priority in PoP_1's FIB and there is no congestion between PoP_1 and R_1. All the requests will be forwarded through R_1 along a single path. For content K, the probability that content is cached is $G(p_r, C)$, where p_r is the request rate of the content at PoP_1.

With the *hash strategy*, the requests for content K is deterministically forwarding to router R_i, therefore the request ratio p_r' of chunk K at R_i in hash forwarding is larger than p_r in native CCN forwarding, which implies that the probability $g(p_r', C)$ that the content chunk K is cached in R_i is larger than $g(p_r, C)$. In other words, *hash* centralises all requests for an object along a single path, increasing the probability of a hit.

In random forwarding, requests for content K is forwarded to router R_i with probability p_i, where $1 \le i \le n$. After a random split across paths, the requests rate on each path still follows the same distribution. For content K, the request rate at router R_i is still p_r. Therefore, content K will be cached in routers $R_1, \cdots, R_n$ with probability $\sum_{i=1}^{n} p_i \cdot g(p_r, C) = g(p_r, C)$, which is equal to that of native strategy.

In summary, we obtain the following observation about the comparison of caching performance under various strategies: $native = random < hash$.

B. Exploring Cache privacy

The cache privacy issue in CCN lies in the capability of an adversary to use the router's cache to infer whether the content has been recently fetched by another user (whether in the vicinity of the adversary or not). For example, imagine two users, Alice and Bob, share an access point; if Alice generates a request for Content X, which is then returned directly from the access point's cache, Alice can infer that Bob has also previously accessed X. Unfortunately, if we desire high performance it becomes necessary to allow this attack to take place so to provide nearby cached replicas to other legitimate user.

Intuitively, privacy could be improved by routers on multiple paths collaborating to create a distributed cache that serves a bigger set of users. This could obscure whether the target victim has requested the content, e.g. if a cache serves 1 million users, it is harder to attribute behaviour compared to a cache serving 2 users. We thus use the size of the *anonymity set* [35] to measure cache privacy, where the anonymity set for a consumer at the l-th hop is the set of l-hop away routers that could potentially serve a video request from the consumer.

For user U_1 in Figure 1, the size of its anonymity set at the first hop is 1, as there is only PoP_1 that U_1 could reach in 1 hop. If a request from U_1 could reach either R_1 or R_2, then the size of anonymity set at the second hop is 2. If a set of routers serve a large number of users, it clearly makes it more difficult to infer which one requested a given item of content. Our foundation of strengthening cache privacy when multiple paths are available is therefore to increase the number of caches along different paths that could serve the same request. Again, we use Figure 1 to explore the privacy attained by the different strategies.

In the *native* CCN strategy, PoP_1 and PoP_2 preferentially forward requests to router R_1 and R_3 respectively. They only forward to other routers when there is congestion or failure in the corresponding link. Therefore, when the content is returned from routers with hop count 2, the attacker can determine that the content is stored in R_1; thus, the attacker could infer that either user U_1 or U_2 has accessed the content. As shown in Table I(a), the native CCN strategy achieves an anonymity set of 1, as only a single router that is two hops away from the consumers can serve the request.

As shown in Table I(b), the *hash* strategy also yields an anonymity set of size 1, i.e. equal to the native strategy. Since the hash function is deterministic, the forwarding path is set before the consumer sends the request. An adversary can therefore known which router the request is forwarded to by iteratively requesting chunks of the same content. Hash achieves high performance, but poor privacy.

Finally, we inspect the *random strategy*. If the interest packets for K_1 and K_2 are forwarded to routers randomly, any chunk corresponding to the request from the attacker could be cached in either R_1 or R_2. When the content is returned from a router, the attacker cannot distinguish whether it's from R_1 or R_2. Therefore, the attacker could not determine whether the content is accessed by U_1, U_2 or U_3. As shown in Table I(c), the

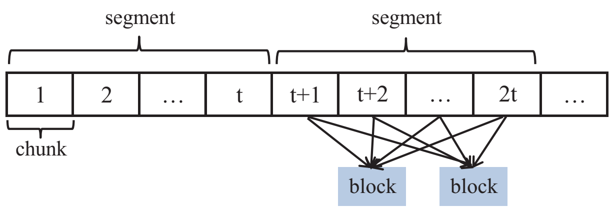


Fig. 2: A segment contains t video content chunks. Chunks in a segment are encoded into blocks, using RLNC.

size of anonymity set under random forwarding strategy is increased to 2.

To summarize, in terms of privacy: $native = hash < random$. In other words, an architecture optimized for performance should use native or hash, whereas an architecture optimized for privacy should used random. This creates a direct conflict between these two goals that must be balanced. We next explore this balance to design CodingCache.

IV. Design of CodingCache

The previous section has shown that algorithms optimised for privacy tend to reduce performance, and vice versa. As such, it is necessary to design dedicated algorithms to achieve these joint goals. We next detail the design of CodingCache, a caching and forwarding mechanism to address both performance and privacy needs. In essence, CodingCache aims to obfuscate user requests by spreading them across multiple distinct paths (using principles taken from network coding). Through this, each router starts to cache video content for a much larger set of consumers, thereby dramatically increasing the size of the anonymity sets.

A. Overview of CodingCache

To spread requests across multiple paths, we use principles taken from network coding. Video content chunks are grouped into segments as shown in Figure 2. The number of chunks in a segment is t, which is a design parameter. Chunks in a segment are encoded into one *block* through Random Linear Network Coding (RLNC) [10], where a coded block has the same size as a chunk. The coded block is the unit for data reply and caching. Consumers get the original chunks in a segment by decoding t linearly independent coded blocks. Intuitively, a larger segment size will lead to better caching performance, as it broadens the diversity of cached blocks in the network. However, a larger segment size also requires more coding information piggybacked in each interest packet, as well as more time to obtain the original content chunks. We will analyse the relationship between coding overhead and segment size in Section IV-E and evaluate its impact on caching performance by varying segment size in Section V.

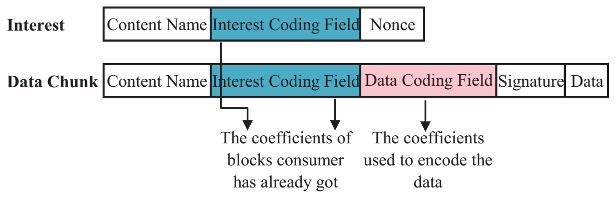


Fig. 3: Coding fields in Interest and Data.

CodingCache introduces coding fields into native CCN interest and data packets, as shown in Figure 3. In both type of packets, the *interest coding field* indicates the blocks the consumer has already obtained within the same segment. Only blocks that the user has not previously acquired are returned. This field in the data packet is copied from the corresponding interest packet. A data packet also has a *data coding field*, which is the Random Linear Network Coding(RLNC) coefficients that are used to encode the data. Note that a coded block is linearly independent of a request if their coefficients are linearly independent. When a request is matched with a linearly independent block in a cache, it corresponds to a *cache hit*.

The origin video content server encodes the chunks of a segment using RLNC. For a segment K that contains t chunks, the coded block is $B = a_1K_1 + \cdots + a_iK_i + \cdots + a_tK_t$, where the addition and multiplication operations are over finite field, and $< a_1 \dots a_i \dots a_t >$ are the coefficients generated using RLNC. Due to the randomization of RLNC, the coefficients of two blocks corresponding to the same segment are likely to be linearly independent.

B. CodingCache in action

Content servers, intermediate routers as well as consumers are involved in implementing CodingCache. As changes in these network components are required in deploying CCN anyway, we believe that it would be perfectly feasible to integrate network coding at this stage. Here, we describe the coding and forwarding actions of each type of node when receiving interest and data separately. We detail the operations of CodingCache in each entity below.

Content server: The process of video content servers providing content is shown in Algorithm 1. When a video content server receives an interest for any chunk of segment K, it generates a vector of t random coefficients, which is linearly independent of the ones in the interest coding field (i.e. the ones the consumer has previously downloaded). It then uses the generated coefficients to encode the t chunks of K to a new block and forwards the block downstream.

Algorithm 1 Process of content server handling requests.

```
1: procedure SERVERPROCESS($req$)
2:     $K, codings \leftarrow$ unpack($req$)
3:     while true do
4:         $\overrightarrow{cf} \leftarrow$ random()
5:         if linear_independent($codings, \overrightarrow{cf}$) = true then
6:             break
7:         end if
8:     end while
9:     $block \leftarrow \sum_{i=1}^{t} cf_i K_i$
10:    forward_back($block$)
11: end procedure
```

For each segment, the video server prepares (upon content creation) t random coefficients that are linearly independent, and generates t corresponding blocks using these coefficients before the interests arrives. When receiving an interest, the server should select one block that is linearly independent of those that the consumer already has. This reduces response time.

Consumer: Consumers send interest packets to retrieve chunks. Suppose a consumer requests the video chunk K_i of segment K. The interest coding field of a new interest packet is filled with coefficients of all blocks the consumer has already obtained corresponding to K.

The process of the consumer retrieving content is shown in Algorithm 2. As the unit of data transfer is a segment, to retrieve all the chunks of a segment K, the consumer needs to express the interest t times, where t is the segment size. Each time the consumer updates the coding field in the new interests by appending the coefficients of the blocks the consumer has already received. After obtaining t linearly independent blocks, the consumer decodes them to get the original content chunks.

Algorithm 2 Process of consumer requesting content.

```
1: procedure CONSUMERPROCESS($K$)
2:     $blocks \leftarrow \{\}$, $codings \leftarrow \{\}$
3:     for each $i$ in range($1, t$) do
4:         $req \leftarrow$ pack($K, codings$)
5:         $B \leftarrow$ request_block($req$)
6:         append($blocks, B$)
7:         append($codings, B.cf$)
8:     end for
9:     decode($blocks$)
10: end procedure
```

Intermediate router: Intermediate routers handle interest forwarding and data caching. In what follows, we show how the diversity of cached content is increased by coding and how the increased diversity improves caching performance and privacy. Note that *both* constitute our primary goals in CodingCache.

Upon receiving an interest for a video chunk of a segment K from a downstream router (i.e. a router that is closer to the consumer), a router first checks whether a coded block of K (that is linearly independent of the coefficients contained in the interest) exists in its cache. If there is such a block, the router responds with the coded block to the downstream node, which means a cache hit. Otherwise, the router *uniformly and randomly* forwards the interest to one of the upstream nodes that will deterministically reach the content object. In other words, we use the *random strategy* for interest forwarding, which achieves good cache privacy (as shown in Section III). However, we have also found that such a strategy negatively impacts performance. We therefore use network coding to address this limitation.

Upon receiving a coded data block response from its upstream node, the router forwards the block to the downstream router based on the PIT table as in native CCN. The router might also cache the coded block according to the router's cache decision strategy (we later evaluate several strategies). For a given segment K consisting of t chunks, there might be more than one coded block in the cache of a router. Let s represent the number of blocks that a segment can have in a cache at most. Obviously, s is no greater than t. A larger s results in a higher possibility of finding a linearly independent block for the request. However, caching more blocks of a segment consumes space in the cache, leaving less space to cache blocks of other segments. We will study the impact of s on the caching performance and cache privacy in Section V.

When a router receives a coded block of K, if the number of cached blocks of K in its cache is less than s, then the received block is added to the cache and an existing block in for the segment other than K is ejected if the cache is full. Otherwise, each of the cached blocks of K is recoded with the received block in order to keep the information of the newly received block. That said, for each cached block B of K, it is replaced with a new block $B^{'} = cB + c_1B_1$, where c and c_1 are non-zero random coefficients and B_1 is the received block. Note that the received block is linearly independent of the cached blocks, since otherwise the interest would be satisfied by the cached blocks and not be further forwarded. The new block $B^{'}$ is linearly independent of either B or B_1. By recoding in intermediate routers, the information of K contained by B_1 is kept in the network (rather than dropped), and therefore improved caching performance is achieved. For example, the requests of any chunk in K from the consumers who obtained B or B_1 can still be satisfied with $B^{'}$.

The recoding also improves the diversity of coded blocks among a set of nodes. Let's again use Figure 1 as an example. Without recoding, PoP_1 would cache a coded block that is also cached in either R_1 or R_2. As such, a request that is not satisfied by PoP_1 might also not be satisfied by the two-hops away routers. On the other hand, with recoding, PoP_1 can cache a recoded block of two blocks B_1 and B_2 belonging to the same segment that are cached on R_1 and R_2 respectively. As the recoded block is linearly independent of B_1 and B_2, a request that cannot be satisfied by the PoP_1 might be satisfied by the two-hop away routers. It is worth noting that recoding is an optional function, which might not be performed in some routers (e.g. core routers).

Algorithm 3 Caching process of block B using recoding in routers.

```
 1: procedure RANDOMCODING($B$)
 2:     if cache_decision($B$) $= true$ then
 3:         check if $\exists K$ in cache $s.t.$ $B \in K$
 4:         if $K \neq \emptyset$ then
 5:             for each $B' \in K$ do
 6:                 $B^{'} \leftarrow$ random_recode($B, B^{'}$)
 7:             end for
 8:             if sizeof($K$) $< s$ then
 9:                 insert($B$)
10:             end if
11:         else
12:             insert($B$)
13:         end if
14:     end if
15: end procedure
```

The procedure of caching a block using recoding is shown in Algorithm 3. When a new block B is returned from an upstream router, the *cache_decision* function in Line 2 decides whether or not to cache it. Available cache decision strategies include the native CCN caching strategy CEE, LCD [18] and ProbCache [30]. If there are blocks in the cache that belong to the same segment as B, each block in the segment is recoded with B, using non-zero random coefficients. In Line 9, if the number of blocks of this segment has not reached the limit s, B is inserted according to the replacement policy.

Since the strategies in CodingCache are orthogonal to the caching decision and replacement policy, existing caching strategies (e.g. LCD [18] or ProbCache [22]) can be easily coupled with it to combine their respective benefits. Another advantage of CodingCache is that the uniformly random forwarding strategy leads to load balancing among the links attached to individual routers.

C. An example of CodingCache

We illustrate the process of retrieving and caching video content with CodingCache in Figure 4. In this scenario, consumer U_1 has already obtained block $3K_1 + 2K_2$. He therefore generates an interest with

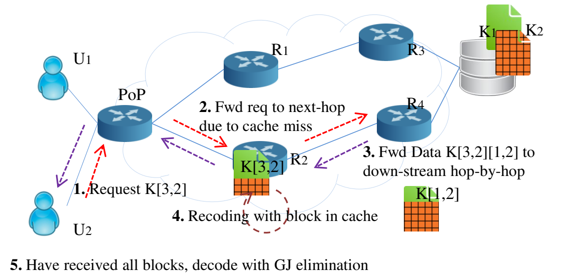

U1
R1
R3
K1
K2
PoP
R4
2. Fwd req to next-hop due to cache miss
K[3,2]
R2
3. Fwd Data K[3,2][1,2] to down-stream hop-by-hop
1. Request K[3,2]
K[1,2]
4. Recoding with block in cache
U2
5. Have received all blocks, decode with GJ elimination

Fig. 4: The process of retrieving and caching data.

TABLE II: Caching performance and cache privacy of CodingCache.

Time	Fwd to	R1 Cache	R2 Cache	Hit?	| Anoy Set |
1	R1	K1+2*K2	NIL	×	
2	R2	K1+2*K2	3*K1+2*K2	×	2
3	R2	K1+2*K2	3*K1+2*K2	√	
4	R1	K1+2*K2	3*K1+2*K2	√	

$[3, 2]$ in the *interest coding field*, which is forwarded to router R_2. Since the cached block in R_2 is linearly dependent with the request, R_2 forwards the request to the next-hop router. When R_4 receives the request, it performs a lookup in the cache. As R_4 stores a linearly independent block $K_1 + 2K_2$, it returns the block to the downstream router. When R_2 receives the block, it decides whether to cache the block according to the caching policy. If the decision is to cache the block, since $K_1 + 2K_2$ and $3K_1 + 2K_2$ are in the same segment and linearly independent, router R_2 recodes them with random coefficients to get a new block, named $5K_1 + 6K_2$, and stores the new block. Finally, U_2 receives all the necessary blocks, and decodes them with Gauss-Jordan elimination to get the original chunks K_1 and K_2.

Table II lists the cache hits and the size of the anonymity set of CodingCache by taking the request sequence in Figure 1 as an example. CodingCache achieves 2 cache hits and has anonymity set of size 2. Compared with the strategies listed in Table I, CodingCache improves both caching performance and cache privacy.

D. Performance Analysis of CodingCache

Caching Performance: We illustrate through Theorem 1 that CodingCache can achieve the optimal caching performance by uniformly random forwarding when each path from consumer to video content server has the same cache capacity.

Theorem 1. *The cache hit rate is maximised in CodingCache with the uniform random forwarding strategy.*

Proof: We define the *request rate* of a chunk at a node (e.g. intermediate router, consumer) as the ratio of requests for the chunk to all the requests which appear in this node. Requests from consumers are randomly forwarded to next-hop routers (that each deterministically will reach the video server). Thanks to random forwarding for content chunks with request rate p_r, consumers still have the same request rate along any path. As the request rate of any content chunk does not change in any path, the request distribution across all content chunks in any path does not change either.

Assume a consumer needs to obtain a segment K consisting of t chunks. The probability that the coded block of the segment is cached on routers along a single path depends on: (1) the distribution d of the request rates for the content, (2) the probability p^* that the segment is requested, (3) the cache capacity C on a router, and (4) the number of chunks t in a segment. Therefore, the block is cached with probability $h = f(d, p^*, C, t)$.

We can express the expected number of blocks of the segment stored on routers along a single path as $h \times t$. Given n paths to the video server from the consumer, let p_i denote the probability that a request for any of the t chunks is forwarded through path i. Among the t requests, the number of cache hits for segment K along path i is $\min(h \times t, p_i \times t)$, which means it's limited by both the number of blocks cached on path i and the number of requests forwarded along that path. Given n paths to the content server, the average cache hit rate for the t requests is

$$\begin{aligned}\rho &= \frac{\sum_{i=1}^{n} \min(h \times t, p_i \times t)}{t} \\ &= \sum_{i=1}^{n} \frac{h + p_i - |h - p_i|}{2} \\ &= \frac{nh+1}{2} - \sum_{i=1}^{n} \frac{|h - p_i|}{2} \\ &\le \frac{nh+1}{2} - \frac{|nh-1|}{2} \quad .\end{aligned}$$

The equality holds when all $(h - p_i)$ have the same sign as $(nh-1)$, which is satisfied by setting all p_i's to the same value, $1/n$. ■

From Theorem 1, when routers forward requests following a uniform random selection of the interfaces, the network achieves optimal caching performance. In summary, the optimal caching performance is achieved by both network coding and uniform multipath forwarding, which therefore exploit all cache capacity in the paths.

Cache Privacy: With random coding and a uniform random forwarding, linearly independent blocks for a given segment would be cached on routers along multiple paths. Through this, CodingCache implicitly collaborates across multiple routers to create a distributed cache that serves a larger set of users. The caching of a coded block in a router can be caused by the request from any user in the set. Thus, this increases the difficulty of an adversary attributing a content item to a given user.

E. Implementation Issues

CodingCache requires the nodes to perform coding, recoding or decoding blocks. The time complexity of encoding, recoding, and decoding (either at content servers, routers or consumers) is linear with the video chunk size. For example, decoding using Gauss-Jordan elimination has a time complexity of $O(t^2 L)$, where L is the chunk size. Since t is often small, the time complexity induced by these operations primarily depends on the chunk size.

Another concern is the time taken by routers to check whether the coded blocks in its cache are linearly independent of the interest. At most $(t-1)$ coefficients are appended to an interest. One possible approach to compute whether the coefficients are linearly independent is to use SVD (Singular Value Decomposition), which has a time complexity of $O(t^3)$. Assuming a small value of t and a hardware-based acceleration of the SVD computation [36], the time taken by routers would be reasonably low. We also note that network coding has been used in the past to improve P2P Live video streaming latency [33]. This supports CodingCache's ability to support video delivery in the real world, if and when a large-scale CCN deployment is achieved.

V. Performance Evaluation

A. Experimental methodology

To explore the behaviour of CodingCache, we perform an extensive set of simulations using ndnSIM [37]. To ensure realistic results, we utilise two large-scale datasets to model both topology and video request patterns. The topology is modelled using traces taken from China Telecom, which detail the connectivity between cities in China [11]. The network graph is made of 321 nodes, 1507 links, has a diameter of 6 hops, an average path length of 2.84 hops, a maximum degree of 97, and an average degree of 9.39. We recreate this topology in ndnSim, and replay real video request traces over it. These are extracted from a unique dataset [38] provided by PPTV, a major Chinese VoD provider. The logs consist of nearly 140 million video access requests. Each entry contains the name and size (in bytes) of the requested video, and the IP address that originated the request. We use a Chinese IP geolocation database, QQwry[1], to map each IP address to the city it belongs

[1]http://www.cz88.net/

TABLE III: Experiment parameters.

Parameter	Default Value	Candidate Values
Caching decision	CEE	LCD
Replacement policy	LRU	LFU
Number of next-hop routers	No Limit	$1, 2, \cdots, 5$
Number of content servers	1	$2, 3, 4$
Chunk size	100KB	$(2 \sim 8)\cdot$ 100KB
Cache size over total content size (r)	10^{-4}	$(2^{-3} \sim 2^{4}) \cdot 10^{-4}$
Segment size (t)	4	$2, 6, 8, \cdots, 16$
Max number of coded blocks of a segment in cache (s)	N/A	$1, 2, \cdots, t$

to. Using this information, we embed the consumers in the simulated topology with all users in each city behind the corresponding city gateway in the topology. In our experiments, we follow the PPTV Content Distribution Network (CDN) configuration, and use at most 4 video servers in each simulation. 2 are placed in the cities with the largest number of requests (as is typical in CDN deployments), and the other 2 are located in cities with a smaller number of requests but a greater degree of connections.

A video object is divided into equal-sized chunks. A request for a video from the logs is simulated with q interest packets, where q is the number of chunks of that video. The default chunk size is 100KB, and varies between the default value and 0.8MB in our experiments. A segment has 4 chunks by default (i.e., $t = 4$), and this value could vary from 2 to 16. Thus, the minimum size of a segment is 200KB, while the maximum size could be 12.5MB.

The ratio r of the per router cache size to the total video size is crucial for CCN caching performance, and varies in the literature, ranging from 10^{-5} [16], 10^{-3} [39] to a maximum of 2% in [26]. We set this ratio to 10^{-4} as a default value and vary it from 10^{-5} to 10^{-3} to explore its impact. The maximum number of coded blocks of a segment cached in a single cache s varies between 1 and t. We particularly focus on the cases where $s = 1$ and $s = t$.

Last, we explore a variety of cache decision and replacement policies. In the experiments, we use LRU (Least Recently Used) as the default cache replacement policy, and CEE and LCD [18] as default cache decision policies, as they are the popular policies widely used in CCN. CEE caches every object in every router along its transmission path, and is the default caching decision in CCN. LCD [18] moves the cached chunks one hop closer to the consumer with each successive request. The main parameters and their default values are listed in Table III. We vary these parameters to evaluate their impact on CodingCache, the native CCN strategy (noted as CCN), hash strategy (Hash), and random strategy (Rand). Note that the goal of CodingCache is *both* high performance and privacy.

B. Caching performance

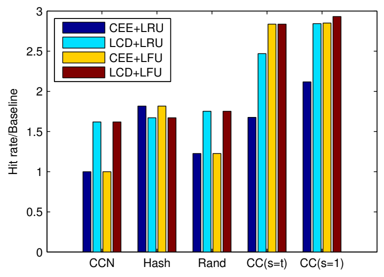
First, we evaluate the caching performance of CodingCache. We focus on two performance metrics: the cache hit rate and the average transmission path (in hops). Here, we use the performance of native CCN with default settings as the baseline (noted as *Baseline*), and evaluate other strategies by the ratio of their value to Baseline.

1) Cache hit rate: Figure 5a shows the hit rate of CCN, Hash, Rand and CodingCache (denoted as CC for short) relative to the Baseline. Higher values indicate superior performance. As expected, the hit rates of CCN and Rand are lowest. Hash gets a better cache hit rate as it increases the locality of video requests (because it forwards requests for the same videos along a single deterministic path). On the other hand, CC achieves the highest cache hit rate, as it exploits random recoding to increase the diversity of cached videos, and utilises random forwarding to obtain the whole content from implicitly collaborative routers.

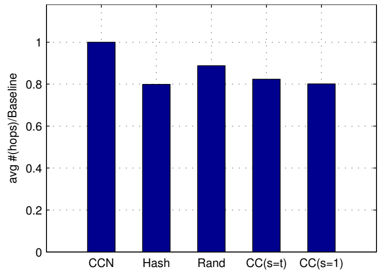
Interestingly, we observe that the cache hit rate of CC with $s = 1$ is slightly better than that with $s = t$, where s is the maximum number of coded blocks of a segment that can be cached by a single router. This results from the network topology characteristics. In the network, the average number of next-hop routers is around 5. In CC, requests are forwarded randomly to the available next-hop routers and recoding is used in intermediate routers. As such, for a segment, its corresponding blocks cached in the next-hop routers are likely to be linearly independent. Given that only $t = 4$ linearly independent blocks are required to obtain the original chunks, letting each router cache one block of each segment (i.e., $s = 1$) is sufficient to recover this segment. Increasing s in this case does not improve the hit rate but results in redundant caching.

It is also worth noting that performance varies across the different caching decision and replacement strategies used by CC. We observe that LCD outperforms CCN's default caching decision, improving the ratio of hit rate from 2.1 to 2.9 when using CC. As expected, LFU leads to better cache hit rates than LRU, mainly because LFU allows more popular videos to stay in caches for a longer time. Irrespective of the caching decision or replacement policy adopted, CC achieves the best cache hit rate among these strategies.

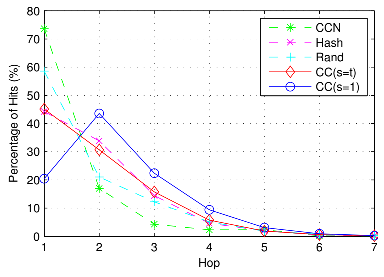
2) Average transmission path length: Figure 5b shows the average transmission path lengths compared



(a) Hit rate



(b) Avg. path length



(c) Cache hit distr.

Fig. 5: Caching performance evaluation

to the Baseline (in hops). We observe that the average number of hops to retrieve video chunks with CC is nearly the same as with hash-based forwarding. This is because cache hits with hash-based forwarding occur closer to the consumer compared to CC. CCN and Rand have the longest average transmission path, due to their poor overall cache hit rate.

To better understand where cache hits occur in the topology, we break down the hits according to the distance of the hit from the consumer, as shown in Figure 5c. CCN and Rand have a similar cache hit distribution, with 60% of their hits at the first hop (the access router). CC with $s = t$ has a similar hit rate distribution to Hash, with fewer hits at the first hop than CCN. Hits at the first hop imply a low latency, but also a poor utilisation of the cache space further into the topology, which can yield a low global cache hit rate (as shown in Figure 5a). We also observe that CC with $s = 1$ has a totally different cache hit distribution: it has only 20.4% of its hits at the first hop, and 43.5% at the 2nd hop. The reason for this behaviour is that at hop 1, each segment can have at most 1 block in the cache with CC. Requests for the t chunks result in only one cache hit at hop 1, regardless of video popularity. However, for routers further into the topology (e.g. 2 or more hops away from the consumer), the requests can be forwarded to multiple routers. The recoding at intermediate routers increases the diversity of cached blocks among these routers. The random forwarding distributes requests to the routers uniformly, which brings more benefits from linearly independent blocks along multiple paths.

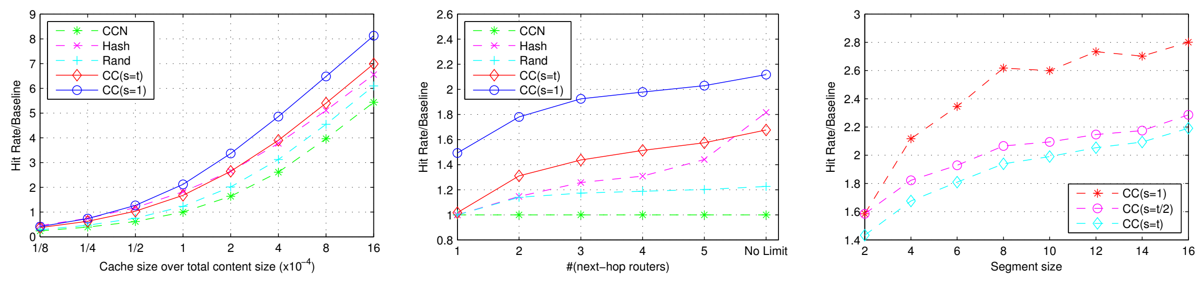
C. *Effects of design parameters on caching performance*

We next evaluate the factors that affect the caching performance. Major factors include the cache capacity C, the number of available next-hop routers, and the segment size t. Again, we use the performance of native CCN with default settings as the Baseline, and evaluate the factors by the ratio of their values to Baseline.

In Figure 6a, we show the impact of the cache size of individual routers. As expected, the hit rate of all strategies increase steadily as cache capacity grows. We can see that as r (the ratio of cache size to total video size) increases from about 10^{-5} to 10^{-3}, Hash provides the least gain with an increase by a factor of 13. Other schemes see increases by about 18 to 20 times. When r is about 10^{-3}, the hit rate of CC with $s = 1$ is 50% higher than for CCN. Regardless of the cache size, the hit rate of CC with $s = 1$ is highest among all the considered strategies.

Figure 6b shows the impact of the number of next-hop routers on the cache hit rate. Among the considered strategies, CC with $s = 1$ obtains the highest cache hit rate. The reason is that the routers along a path store identical video chunks in CCN, while in CC the caches are different thanks to random recoding. As the number of next-hop routers increases, the hit rates in CCN and Rand do not vary much, confirming our analysis in Section III. As expected, the hit rates of Hash and CC increase with the number of next-hop routers. Hash utilises the multiple paths to request different chunks, which enhances the cache hit rate by increasing the locality of video chunks in each cache. CC exploits the benefits of multiple paths by caching linearly independent blocks of individual videos along different paths. In particular, when the number of next-hops is not restricted, CC improves the ratio of hit rate to 2.2.

Figure 6c illustrates the effect of the segment size t, i.e., the number of chunks coded into one block. We compare three strategies, CC with $s = 1$, $s = t/2$ and $s = t$. We observe that CC with $s = t/2$ results in nearly the same cache hit rate as CC with $s = t$, which is less than for CC with $s = 1$. We also see that as the segment size increases, the cache hit rates increase for all strategies. For CC with $s = 1$, the hit rate grows quickly for small t (say $t < 8$). The reason is that a large t can broaden the diversity of cached blocks in the network, which leads to improved performance. However, the effect t is dependent on the the number of next-hop routers (ranging from 2 to 8 with average 5 in our network), which limits the possibility of exploring such diversity by users. When t becomes greatly larger than the number of next-hop routers, the



(a) Cache capacity (b) # of next-hops (c) Segment size

Fig. 6: Caching performance when varying impact factors

diversity brought by a larger t cannot be fully explored. As such, when $t \leq 8$, increasing t does not lead to a significant growth of hit rate.

D. Cache privacy

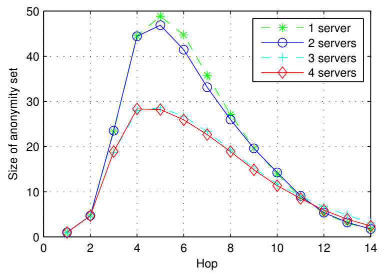
So far we have confirmed that CC attains high performance, as measured by traditional metrics. However, a key goal of our work has been to build privacy-aware caching schemes. We next measure how effectively CC is at preventing attackers from inferring user activity.

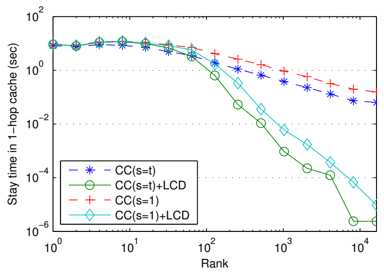
To achieve this, we use three privacy metrics. First, we measure the size of the anonymity set for each request (larger anonymity sets indicate higher privacy). Second, we measure the number of indistinguishable cache hits in the network. Indistinguishable cache hits correspond to the cache hits that happen at a distance at least equal to the distance to the video server from the perspective of an adversary. In this case, an adversary cannot infer that a cache hit happened at all. Third, we measure the stay time of content in the first hop routers. This is because the anonymity set at the first hop is always 1; thus, the shorter the stay time, the lower the probability that a video access could be detected by adversaries.

As previously mentioned, privacy is attained in CodingCache by using multipath routing to increase the number of clients each router serves. Both hash-based forwarding (Hash) and random forwarding (Rand) also exploit the availability of multiple next-hop routers, but quite differently. For a given object, Hash will determine the exact path from the consumer to the video server. Thus, the size of the anonymity set is 1 at each hop. In Figure 7a, we show the size of the anonymity set at each hop with random forwarding, when varying the number of video servers. At hop 1, the size of the anonymity set is always 1. As the number of hops increases, the size of the anonymity set first increases until hop 5, then decreases. Within the first few hops, the path diversity increases, and therefore the size of the anonymity set well. After reaching 5 hops, the loop-free requirement on forwarding decreases both the path diversity and the size of the anonymity set. We also observe that as the number of servers increase, the number of next-hop routers decrease. This is because with more servers in the network, a single router will choose fewer next-hop routers for each server, thus reducing the size of the anonymity set.

The next privacy metric considered is the number of indistinguishable cache hits. As CCN data chunks might contain hop counts, this measures the number of requests that would be impossible to identify as being cached. Such a hit would be unlikely to improve delay, but would reduce the load on the origin server. The results are in Table IV, which shows the ratio of the indistinguishable cache hits as a ratio to Baseline. CC with $s = 1$ has the highest ratio, i.e., the best cache privacy. As CC with $s = 1$ exploits the diversity of coded content (beyond 1 hop), it leads to better cache privacy at the cost of lower hit rate at 1-hop routers.

The final metric considered is the stay time for objects cached at hop 1. This is important as the size of the anonymity set at hop 1 is always 1. If the video content is returned from this location, an adversary can directly infer that the content has been requested by a neighbouring node. The probability that an adversary can detect whether a video content block is cached at hop 1 is approximately linear to the maximal stay time of the content at that hop. At hop 1, CC with $s = t$ acts similarly to strategies that do not rely on network coding, and each video object's stay time is independent of the forwarding strategy. We plot in Figure 7b the maximal stay time for each content in the caches for CC with $s = t$ and $s = 1$. To ease readability, videos are ranked by decreasing popularity (i.e., number of requests), then they are grouped using logarithmic bins $[2^i, 2^{i+1})$, where $i = 0, 1, 2, \cdots$. From Figure 7b, we observe that the stay time depends on the video popularity. The most popular videos have constant stay time. As videos becomes less popular, the stay time decays as a power-law. The stay time of each video with CC $s = t$ is always shorter than with CC $s = 1$, demonstrating the slightly better privacy of $s = t$ in 1-hop routers. It is therefore likely that the value of s would need to be varied based on the sensitivity of the content being distributed. This provides a powerful, yet


(a) Size of the anonymity set


(b) Stay time of data in 1-hop router

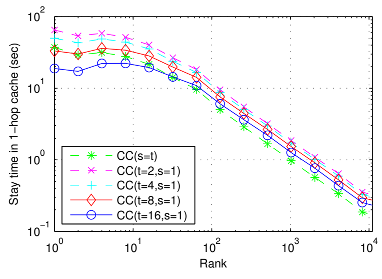

(c) Impact of segment size

Fig. 7: Cache privacy evaluation.

TABLE IV: Ratio of indistinguishable cache hits to that of CCN.

Strategy	**Hash**	**Rand**	**CC(s=t)**	**CC(s=1)**
Ratio	6.4	3.2	6.2	11.3

simple means, to improve privacy without needing to modify system-wide configurations.

It is also interesting to note that the stay time varies with different caching decision strategies. We find that LCD offers the highest privacy, as measured by stay time. LCD caches content one hop closer to the consumer on each request. As such, edge routers at hop 1 will tend to cache more popular videos, reducing the numbers of other video objects that can be cached. This is particularly desirable from a privacy perspective as unpopular videos are often more sensitive and of interest to adversaries [4]. Figure 7b shows the average stay time in routers at hop 1 for CC and LCD. We observe that compared to CC, CC+LCD significantly reduces the stay time of unpopular content, demonstrating good cache privacy. This provides an example where other caching strategies can be combined with CC to make it more powerful.

Finally, we investigate the impact of segment size t on cache privacy for routers at hop 1 (see Figure 7c). We observe that as the segment size increases, the duration that videos stay in the cache is reduced, leading to better privacy. On the other hand, as seen from Figure 6c, a larger segment size results in both better cache performance and privacy. However, a larger value of t also means that consumers need to retrieve more linearly independent blocks to recover the original chunks, implying an increased delay for the consumer. Therefore, large segment size could be less suitable for real-time applications such as streaming media. In summary, this shows that flexibility in the setting t is necessary for the practical use of CodingCache.

VI. Conclusion

CCN has widely been reported as a powerful platform for improving video delivery. In this paper we have explored two topics related to CCN caching: (1) Exploiting the multipath capabilities of modern network topologies; and (2) Enabling high performance privacy-aware video caching.

To achieve our goals, we have proposed the use of network coding to code video content chunks into multiple blocks that can follow multiple paths back to the consumer. Through this, we expand the anonymity set of each user (the set of potential routers that can serve a given user) and increase the diversity of cached content among different paths. As such, our algorithm, CodingCache, offers an attractive combination of high performance with tight privacy controls. We have evaluated CodingCache with real-life video request traces and shown that it outperforms native CCN and deterministic hash routing. Furthermore, we have shown that CodingCache can compliment existing cache decision and replacement strategies, adding benefits with little overhead. We believe this is strong evidence that CodingCache offers an attractive balance as a candidate as a core CCN caching algorithm. A number of implications arise from our work. Perhaps most notably, we have shown that network coding is an attractive technology in the context of video over CCN. We argue that, considering the need to deploy new hardware anyway, it would be sensible to natively build network coding support, in order to mitigate privacy concerns and increase performance. We next plan to implement CodingCache as a prototype to evaluate its performance when used with different ICN architectures. We also intend to evaluate CodingCache with different content types other than video, as well as expanding its configurability to target different environments.

References

[1] "Cisco visual networking index: Forecast and methodology, 2013-2018," tech. rep., Cisco, 2014.

[2] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang, "Understanding the impact of video quality on user engagement," in *Proceedings of the ACM SIGCOMM*, 2011.

[3] Y. Sun, S. K. Fayaz, Y. Guo, V. Sekar, Y. Jin, M. A. Kaafar, and S. Uhlig, "Trace-driven analysis of icn caching algorithms on video-on-demand workloads," in *Proceedings of the 10th ACM CoNEXT*, 2014.

[4] T. Lauinger, N. Laoutaris, P. Rodriguez, T. Strufe, E. Biersack, and E. Kirda, "Privacy risks in named data networking: what is the cost of performance?," *ACM SIGCOMM CCR*, 2012.

[5] A. Chaabane, E. De Cristofaro, M.-A. Kaafar, and E. Uzun, "Privacy in Content-Oriented Networking: Threats and Countermeasures," *ACM SIGCOMM CCR*, 2013.

[6] A. Chaabane, G. Acs, and M. Kaafar, "You are what you like! information leakage through users? interests," in *Proc. Annual Network and Distributed System Security Symposium*, 2012.

[7] N. Xia, H. H. Song, Y. Liao, M. Iliofotou, A. Nucci, Z.-L. Zhang, and A. Kuzmanovic, "Mosaic: Quantifying privacy leakage in mobile networks," in *Proceedings of the ACM SIGCOMM*, 2013.

[8] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *ACM CoNEXT*, pp. 1–12, 2009.

[9] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, D. Massey, C. Papadopoulos, *et al.*, "Named Data Networking (NDN) Project," tech. rep., PARC, Tech. report ndn-0001, 2010.

[10] T. Ho, M. Mdard, J. Shi, M. Effros, and D. R. Karger, "On randomized network coding," in *Proceedings of the Annual Allerton Conference on Communication Control and Computing*, vol. 41, pp. 11–20, 2003.

[11] Y. Tian, R. Dey, Y. Liu, and K. W. Ross, "Topology mapping and geolocating for China's Internet," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 9, pp. 1908–1917, 2013.

[12] Z. Li, G. Xie, J. Lin, Y. Jin, M. A. Kaafar, and S. Kave, "On the geographical patterns of a large-scale mobile video-on-demand system," in *Proceedings of the IEEE INFOCOM*, 2014.

[13] E. J. Rosensweig, J. Kurose, and D. Towsley, "Approximate models for general cache networks," in *INFOCOM, 2010 Proceedings IEEE*, pp. 1–9, IEEE, 2010.

[14] I. Psaras, R. G. Clegg, R. Landa, W. K. Chai, and G. Pavlou, "Modelling and evaluation of CCN-caching trees," *IFIP NETWORKING*, pp. 78–91, 2011.

[15] Y. Li, H. Xie, Y. Wen, and Z.-L. Zhang, "Coordinating in-network caching in content-centric networks: Model and analysis," in *IEEE ICDCS*, 2013.

[16] G. Rossini and D. Rossi, "Evaluating CCN multi-path interest forwarding strategies," *Computer Communications*, 2013.

[17] G. Tyson, S. Kaune, S. Miles, Y. El-khatib, A. Mauthe, and A. Taweel, "A trace-driven analysis of caching in content-centric networks," in *ICCCN*, pp. 1–7, IEEE, 2012.

[18] N. Laoutaris, H. Che, and I. Stavrakakis, "The LCD interconnection of LRU caches and its analysis," *Performance Evaluation*, vol. 63, no. 7, pp. 609–634, 2006.

[19] W. Chai, D. He, I. Psaras, and G. Pavlou, "Cache 'less for more' in information-centric networks," in *IFIP NETWORKING*, 2012.

[20] K. Cho, M. Lee, K. Park, T. T. Kwon, Y. Choi, and S. Pack, "WAVE: Popularity-based and collaborative in-network caching for content-oriented networks," in *IEEE INFOCOMM WKSHPS*, pp. 316–321, IEEE, 2012.

[21] Y. Wang, Z. Li, G. Tyson, S. Uhlig, and G. Xie, "Optimal cache allocation for content-centric networking," in *IEEE ICNP*, 2013.

[22] I. Psaras, W. K. Chai, and G. Pavlou, "In-network cache management and resource allocation for information-centric networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 99, no. PrePrints, 2014.

[23] B. Han, X. Wang, N. Choi, T. Kwon, and Y. Choi, "Amvs-ndn: Adaptive mobile video streaming and sharing in wireless named data networking," in *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, pp. 375–380, IEEE, 2013.

[24] S. Ando and A. Nakao, "In-network cache simulations based on a youtube traffic analysis at the edge network," in *Proceedings of The Ninth International Conference on Future Internet Technologies*, p. 10, ACM, 2014.

[25] S. Guo, H. Xie, and G. Shi, "Collaborative forwarding and caching in content centric networks," in *IFIP Networking*, 2012.

[26] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. M. Maggs, K. Ng, V. Sekar, and S. Shenker, "Less Pain, Most of the Gain: Incrementally Deployable ICN," in *ACM SIGCOMM*, 2013.

[27] C. Yi, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang, "Adaptive forwarding in named data networking," *ACM SIGCOMM CCR*, vol. 42, no. 3, pp. 62–67, 2012.

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

[29] S. Arianfar, T. Koponen, B. Raghavan, and S. Shenker, "On preserving privacy in content-oriented networks," in *ACM SIGCOMM ICN workshop*, 2011.

[30] I. Psaras, W. K. Chai, and G. Pavlou, "Probabilistic in-network caching for information-centric networks," in *ACM SIGCOMM ICN workshop*, pp. 55–60, ACM, 2012.

[31] S.-Y. Li, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Transactions on Information Theory*, vol. 49, no. 2, pp. 371–381, 2003.

[32] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft, "XORs in the air: practical wireless network coding," in *ACM SIGCOMM CCR*, 2006.

[33] M. Wang and B. Li, "R2: Random push with random network coding in live peer-to-peer streaming," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 9, pp. 1655–1666, 2007.

[34] M.-J. Montpetit, C. Westphal, and D. Trossen, "Network coding meets information-centric networking: an architectural case for information dispersion through native network coding," in *ACM NoM Workshop*, pp. 31–36, 2012.

[35] L. Sweeney, "k-anonymity: A model for protecting privacy," vol. 10, pp. 557–570, World Scientific, 2002.

[36] A. Ahmedsaid, A. Amira, and A. Bouridane, "Improved SVD systolic array and implementation on FPGA," in *FPT*, 2003.

[37] A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM: NDN simulator for NS-3," Technical Report NDN-0005, NDN, October 2012.

[38] Z. Li, J. Lin, M.-I. Akodjenou, G. Xie, M. A. Kaafar, Y. Jin, and G. Peng, "Watching videos from everywhere: a study of the PPTV mobile VoD system," in *ACM IMC*, 2012.

[39] K. Katsaros, G. Xylomenos, and G. Polyzos, "MultiCache: An overlay architecture for information-centric networking," *Computer Networks*, 2011.