Network Names in Content-Centric Networking

Cesar Ghali* Gene Tsudik* Christopher A. Wood†
Computer Science Department, University of California Irvine
{cghali, gene.tsudik, woodc1}@uci.edu

ABSTRACT

Content-centric networking (CCN) is a networking paradigm that emphasizes request-response-based data transfer. A *consumer* issues a request explicitly referencing desired data by name. A *producer* assigns a name to each data it publishes. Names are used both to identify data to and route traffic between consumers and producers. The type, format, and representation of names are fundamental to CCN. Currently, names are represented as human-readable application-layer URIs. This has several important security and performance implications for the network.

In this paper, we propose to transparently decouple application-layer names from their network-layer counterparts. We demonstrate a mapping between the two namespaces that can be deterministically computed by consumers and producers, using application names formatted according to the standard CCN URI scheme. Meanwhile, consumers and producers can continue to use application-layer names. We detail the computation and mapping function requirements and discuss their impact on consumers, producers, and routers. Finally, we comprehensively analyze several mapping functions to show their functional equivalence to standard application names and argue that they address several issues that stem from propagating application names into the network.

CCS Concepts

•**Networks** → *Naming and addressing; Cross-layer protocols;*

Keywords

content-centric networks; network name; name translation

1. INTRODUCTION

*Supported by the NSF grant: "CNS-1040802: FIA: Collaborative Research: Named Data Networking (NDN)".

†Supported by the NSF Graduate Research Fellowship DGE-1321846.

ICN'16, September 26-28, 2016, Kyoto, Japan

DOI: http://dx.doi.org/10.1145/2984356.2984373

The Internet usage model has changed considerably over the last two decades. Limitations of the current architecture became more pronounced with the emergence of mobile- and data-centric network services and applications. This profound shift in usage led to the emergence of several major efforts aiming to design a candidate next-generation Internet architecture. Content-Centric Networking (CCN), an instance of Information-Centric Networking (ICN) [1, 2], is a recent network architecture that aims to overcome some of these limitations. Unlike current IP-based networking wherein hosts are directly addressed, CCN focuses on addressing data, called *content*, using explicit (and usually human-readable) names. Consumers request desired content by issuing a so-called *interest* carrying the name of the content. The network is in charge of finding and returning the requested content.

Every interest contains a routable name, composed of one or more variable-length components that are opaque to the network [3]. For example, the name of the WSJ's news homepage content for March 9, 2016 might be: `/ccn/usa/wsj/news/03-09-2016/index.html`. The application expressiveness of names eliminates the need for name resolution services such as the Domain Name System (DNS) [4], though it does not preclude name discovery by some external service. A router uses names in conjunction with a Forwarding Information Base (FIB)[1] to forward interests towards the nearest copy of the desired content. As in IP, the router data plane is responsible for searching the FIB using longest-prefix matching to identify the appropriate interfaces (if any) on which an interest should be forwarded.

Although simple, this design has several undesirable consequences. First, it places non-deterministic computational burden on routers that must index the FIB to forward incoming interests. Modern high-speed FIB designs use data structures that range from hash tables [5] to prefix tries [6,7]. All of them must account for variable length names and name segments. Second, and perhaps more importantly, the current design forces application layer semantics (i.e., names) down to the network layer.

In this paper, we propose and evaluate a way to transparently decouple applicationlayer and network-layer names in CCN. Our approach produces network layer names that are deterministically mapped to from application names, formatted according to the current and well-known CCN URI scheme [3]. This yields several benefits: (1) less application

[1]The FIB is a data structure that maps name *prefixes* to network interfaces.

data percolates into the network layer data plane, (2) packets carry less variable-sized names and name segments, and (c) forwarding logic is simplified and therefore improved. We discuss the impact of using network names on CCN entities (consumers, producers, and routers) and related protocols, such as routing. We also address security considerations related to our network names. We then present a comprehensive analysis of their characteristics and statistical properties to show that they are functionally equivalent to standard CCN application names. Lastly, we conclude with a discussion of related work and future directions.

2. CCN BACKGROUND

We now overview some key concepts and features of CCN. Given familiarity with CCN, this entire section can be skipped without loss of continuity.

2.1 CCN Architecture Overview

Named data (content) is the focal point of CCN. A content name is composed of one or more variable-length segments opaque to the network layer [1]. Segment boundaries are explicitly delimited by "/" in the usual URI-like representation. For example, the name of a BBC's news homepage for May 9, 2016 might be: `/ccn/uk/bbc/news/05-09-2016/index.html`.

Content is obtained via explicit requests, called interests. A consumer issues an interest that specifies the requested content name. An interest is routed – based on this name – towards the nearest entity that has a copy of the referenced content: either that content's producer or a router that cached it. Thus, names are treated first and foremost as *locators* for content. Generally, an interest matches a content object if their names are equal. However, a name is not guaranteed to map to a unique content. To this end, further restrictions are possible to narrow the target of an interest. One such restriction carried in an interest is the `ContentObjectHashRestriction` field (also called *content identifier*), which specifies the cryptographic hash digest of the content object. An interest with a non-empty content identifier can only match a content object if the hash of the latter equals the content identifier. An interest might also reflect another restrictive field: `KeyIdRestriction`, or *key identifier*. If present, this value must match the eponymous field in the content.

CCN routers maintain at least two elements to route interest requests and content object responses:

- Forwarding Information Base (FIB): a table that maps name prefixes to egress link identifiers. Ingress interests are forwarded to specific egress links by locating the correct links in the FIB using longest-prefix-match (LPM) of the name.
- Pending Interest Table (PIT): a high-speed cache-like structure that stores names, content identifiers, ingress link identifiers, and other miscellaneous information for interests that have been previously forwarded. The PIT can be seen as a data structure similar to a hash table. The hash table key is the hash of a tuple containing the interest name, `KeyIdRestriction`, and `ContentObjectHash Restriction`. The hash table value is the list of ingress link identifiers along with some additional data (e.g., the hop count for the interest).

Upon receipt of an interest, a router will first examine its PIT to see if a request for the same content has been previously forwarded. If so, the interest is aggregated by appending its ingress link identifier to the list in the matching PIT entry. If not, the FIB is indexed (using LPM) to determine the next hop to which the interest should be forwarded. When a content object is received, the PIT is examined to find a matching entry. This is done by comparing (for equality) the content object name and identifiers with those contained in the PIT. If there is no matching entry, the content object is deemed unsolicited and dropped. Otherwise, the content object is forwarded to all of the downstream links contained in the matching PIT entry and the PIT entry is subsequently flushed. The FIB is not examined when forwarding content objects.

Routers may also be equipped with an optional content store (CS), or cache. If present, content objects may be stored in the CS after being forwarded downstream according to the PIT. When a router with a CS receives an interest the CS is examined for a locally cached copy of the requested content object. The CS is searched using exact match on the name and identifiers. If the content object is found in the CS, then the router will reply with the data instead of forwarding the interest further upstream.

2.2 CCN Name Semantics and Format

CCN requests are not limited to static content. For example, the latest Facebook news feed page could be named `/ccn/facebook/newsfeed.html`; this content could change over time (and therefore be cached for very little time). Also, in some cases, additional information such as user identities can be added as special-purpose name segments or in interest `Payload` fields to allow the producer to generate the appropriate content. To avoid cache hits for interests requesting the same content but with different payloads, a special name segment called the `PayloadID` is appended to the end of the interest names. The value of this segment is the cryptographic hash digest of the `Payload` field.

CCN uses a Type, Length and Value (TLV) [8] format to encode all packet fields in interests and content objects [9]. All TLVs have a 2-byte type and 2-byte length field. This places an upper bound of 64KB on the size of each TLV field and the entire packet as a whole. The type namespace is documented in [9]. For names, different types are used to determine different name segment semantics. Some distinguished types are `T_NAMESEGMENT`, `T_INTPAYLOADID`, `T_VERSION`. (The name segment type space allocates room for 4096 distinct application-specific types.) For example, a content named `/ccn/wsj/news/T_INTPAYLOADID=0xA..B/T_VERSION=0x02` can be represented using the following TLV-encoded format (we use S-expressions in the following representation):

```
T_NAME
    (T_NAMESEGMENT  3  "ccn")
    (T_NAMESEGMENT  3  "wsj")
    (T_NAMESEGMENT  4  "news")
    (T_INTPAYLOADID 4  0x36F3AB30)
    (T_VERSION      1  0x02)
```

For the rest of the paper, we omit the length (i.e., the second field) in the above TLV S-expression representation.

3. NETWORK NAMES

Let $N = [N_1, N_2, \ldots, N_k, S_1, \ldots, S_l]$ be a CCN application name as per [3] where k segments $N_1, ..., N_k$ are used for locating and identifying content and l segments $S_1, ..., S_l$

carry application-specific information. We define segments containing application data as *any segment* that is not of the type T_NAMESEGMENT. This includes segments with the type: T_PAYLOADID, T_VERSION, T_CHUNK, and any other application-specific type to be defined. For example, the name /edu/uci/ics/csdepart.html/T_VERSION=x02 can be represented as the segment vector [edu, uci, ics, csdept.html, T_VERSION = $0x02$] with $k = 4$ and $l = 1$, where the version segment is the only piece of application data. As described previously, an interest carrying this name would have the following TLV encoding:

T_NAME
- (T_NAMESEGMENT "edu")
- (T_NAMESEGMENT "uci")
- (T_NAMESEGMENT "ics")
- (T_NAMESEGMENT "csdept")
- (T_VERSION 0x02)

We propose the use of network names by modifying this current format. In particular, each network segment of the name is replaced by the fixed-size output of a mapping function $\mathsf{T}(\cdot)$ computed over *all prior segments in the name*. In other words, the i-th name segment is now a fixed-size mapping computed over the first i segments. We specifically omit segments which are *not* part of the name locator, i.e., only segments of type T_NAMESEGMENT are included in this computation. The reasons for this requirement are discussed in Section 4.1.2. The actual mapping is computed as:

$$\bar{N}_1 = \mathsf{T}(N_1), \bar{N}_2 = \mathsf{T}(N_1, N_2), \ldots, \bar{N}_k = \mathsf{T}(N_1, \ldots, N_k)$$

Using this representation, the network name $\bar{N}$ for a given application name N is represented as $\bar{N} = [\bar{N}_1, \bar{N}_2, \ldots, \bar{N}_k, S_1, \ldots, S_l]$. The network name *replaces* the regular T_NAME field in CCN packets – it is not carried in a per-hop header or additional encapsulation layer. With a suitable $\mathsf{T}(\cdot)$, the relationship between N and $\bar{N}$ forms a bijection between the application and network namespaces so long as there are no collisions in $\mathsf{T}(\cdot)$.

We also need the network name to include an additional *name fingerprint* segment N_p, computed as the mapping of the full name N with all additional identifiers, i.e., $N_p = \mathsf{T}(N||K_{ID}||C_{ID})$, where K_{ID} and C_{ID} are key and content identifiers, respectively. As described in Section 4, N_p is used for PIT and CS lookup operations. As a consequence of removing application names from packets, content digital signatures must be generated and verified using N_p. This is discussed later in Section 6. Algorithm 1 shows the deterministic procedure for mapping an application name to a network name with N_p.

Following the CCN TLV-encoding in [9] and assuming $\mathsf{T}(\cdot)$ is SHA-256, the network name and fingerprint can be represented as:

T_NAME
- (T_MAPPED_SHA256 $[\bar{N}_1, \bar{N}_2, \bar{N}_3, \ldots, \bar{N}_i, \ldots, \bar{N}_k]$)
- (T_TYPE S_1)
- ...
- (T_TYPE S_l)
- (T_FINGERPRINT N_p)

where T_TYPE is replaced with the appropriate type for the corresponding segments.

Algorithm 1 Map

Input: N, $\mathsf{T}(\cdot)$
Output: $\bar{N}$
$\bar{N} = \emptyset$
k :=Number of T_NAMESEGMENT segments in N
for $i = 1$ to k **do**
  $seq := N_1||\ldots||N_i$
  $\bar{N} = \bar{N} \cup \mathsf{T}(seq)$
end for
$\bar{N} = \bar{N} \cup [S_1, \ldots, S_l]$
$name := N_1||\ldots||N_l||S_1||\ldots||S_l$
$N_p := \mathsf{T}(name)$
$\bar{N} = \bar{N} \cup N_p$
return $\bar{N}$

We use the type T_MAPPED_SHA256 to denote a list of name segments transformed using SHA-256. We make the mapping function explicit to allow for agility. The length of the TLV-encoded segment (not shown in the above representation) is the total byte length of all $\bar{N}$ values. Given this length and output size of $\mathsf{T}(\cdot)$, the number of $\bar{N}$ values can be computed. In other words, using a fixed-size output $\mathsf{T}(\cdot)$, segments with type T_MAPPED_XXX are of fixed size. Therefore, length and individual type of each $\bar{N}$ value do not need to be encoded.

3.1 $\mathsf{T}(\cdot)$ Function Criteria

In the proposed naming scheme, $\mathsf{T}(\cdot)$ is used for two purposes:

1. Computing values in T_MAPPED_XXX segment of a network name.
2. Computing the name fingerprint N_p used by routers for PIT and CS lookups and for signature generation and verification.

Based on this, a sensible instantiation of $\mathsf{T}(\cdot)$ is a hash function. This is because the FIB lookup actually computes multiple hash values for all prefixes of a received interest name, regardless of the underlying data structure used. Currently, most CCN routers use non-cryptographic hash functions (i.e., those that are not collision-resistant) in the FIB for LPM operations. Despite the non-negligible probability of collisions in such hash functions, routers can resolve collisions since application names are explicitly included in interests. However, in the proposed scheme, application names are replaced with fixed-size segments computed using $\mathsf{T}(\cdot)$. Thus, if a collision occurs, a router cannot resolve it, which might result in an interest being forwarded on the wrong interface(s).

One way to cope is by carefully choosing names that do not cause collisions in $\mathsf{T}(\cdot)$. However, it is clearly infeasible for producers to be aware of all existing content names. Another approach is to pick a cryptographic $\mathsf{T}(\cdot)$ that offers collision-resistance. In practice, a well-known cryptographic hash function, e.g., SHA-256, can be used to provide this property, in addition to other cryptographic characteristics, such as one-wayness.

We also note that the output of the cryptographic hash function can be truncated to any output size. We denote $\mathsf{T}(\cdot)$ truncated to s most-significant bits as $\mathsf{T}_s(\cdot)$. With $\mathsf{T}_s(\cdot)$, the corresponding network name segments will carry a type of the form T_MAPPED_XXX_S where S is digest size, e.g., T_MAPPED_XXX_64. Note that output truncation should only be used when computing network name segments and **not** for generating N_p in order to provide the least collision probability for PIT and cache lookups. In Section 5, we

Table 1: Impact on CCN Entities

Entity	Impact
Consumer	**Increased** online processing to compute network names (interests)
Producer	**Increased** storage for reverse mapping (interests) **Increased** off-line processing to compute network names (content)
Router	**Faster** FIB lookup with pre-computed name prefix hashes (interests) **Faster** PIT and cache lookups (interests) **Faster** PIT lookups due to lack of name hashing (content) **Decreased** storage requirement due to fixed-size N_p, instead of arbitrarily long names (content)

Table 2: Impact on CCN Packets

Packet	Impact
Interest	**Longer** name TLV encoding; based on $\mathsf{T}(\cdot)$ size, see Section 5
Content	**Shorter** (fixed-size) N_p, instead of complete application name

analyze collision probabilities for $\mathsf{T}_s(\cdot)$ for different s values.

In the rest of the paper, we use the terms *mapping function* and *hash function* interchangeably when referring to $\mathsf{T}(\cdot)$.

4. NETWORK NAMES IN PRACTICE

The proposed naming scheme has obvious implications for network entities as well as management and control functions, such as routing. In this section, we investigate the impact on end-hosts (consumers and producers), network entities (routers and forwarders), and management functions (routing protocols). This is summarized in Table 1. Implications for CCN packets are summarized in Table 2. All of these are considered in detail in Section 5.

4.1 Name Translation at End-Hosts

End-hosts are the primary entities affected by the proposed naming scheme since they are tasked with translating between application and network names. In practice, this mapping would be performed by a code component between the application and network layers.[2]

4.1.1 Consumer Mappings

The impact on consumers is one name mapping per interest. However, since this operation is not on the fast-path, we expect no performance penalty for consumers. To justify this claim, we assessed the overhead for translating all names in the Cisco URI dataset [10] with the (unoptimized) PARC CCNx libraries [11]. This dataset consists of $13'549'122$ unique URIs. The average length of each URI is 57.4B, with a median length of 52B, and a standard deviation of 33.182B. Across all URIs, the average number of segments is 6.67 with a median length of 6, and standard deviation of 2.212 segments. The average, minimum, and maximum time to compute the mapped version of each name is $1'029.279$us, 3.812us, and $2'474.567$us, respectively. These numbers were generated on a desktop machine with an Intel 2.8 GHz Core i7 processor.[3] This is just over one millisecond to map a single name, which is still less than most network I/O overhead. Given the average size taken across all URIs, the average, minimum, and maximum cycles/byte throughput for this mapping is $1'577.688$c/b, $1'218.037$c/b, and $3'494.538$c/b, respectively. This is far from optimal given that the modern Intel Haswell chipsets can compute SHA-256 digests at a throughput of roughly 8.59c/b [12]. Therefore, we conclude that, given a proper modern implementation, the mapping process at the consumer incurs negligible overhead.

4.1.2 Producer Inversions

Network names have two implications for producers: (1) handling, and responding to, incoming interests and (2) generating content. Since our goal is to make network names transparent to applications, the producer must be able to determine the original application name from a network name contained in an incoming interest.[4] Therefore, the producer must maintain the reverse mapping from network to application name prefixes that it published.[5]

For this to work, we assume that the producer knows all content prefixes under which it publishes content. In other words, the *network locator portion* of the namespace for which a particular producer is responsible must be well-defined. Furthermore, most producers would need to maintain an index that maps application names to content objects in order to respond to interests. The cost of the network-to-application name mapping would, in the worst case, double the size of this index. Since the size of hash table key-sets in data repositories is typically negligible (compared to size of the actual data), this should not result in any significant barrier.

Assuming the producer maintains this reverse mapping, suppose it receives an interest for $\bar{N}$ that has k mapped segments, l application segments, name fingerprint N_p and optional payload. Using the inverse mapping, it can invert the first k segments, and thus recover the original full name used by the interest-issuing consumer's application. This process does not distinguish between interests for static and dynamic content, since a name for either type of content must have a prefix corresponding to at least k leading segments, i.e., the network locator portion of the name. In case of interests for dynamic content, the "dynamic" portion of the name is the suffix that has at most l segments. Recall that this portion of the name is not mapped, i.e., it remains "human-readable" as in standard CCN names.

One drawback of imposing this reverse mapping is that $\mathsf{T}(\cdot)$ must be fixed and adopted by all consumers and producers. This limits the scalability of the proposed naming scheme and does not facilitate seamless evolution, or replacement, of $\mathsf{T}(\cdot)$. One obvious way of relaxing this requirement is to support multiple $\mathsf{T}(\cdot)$ functions. To do so, producers should be able to map *any* received network name format to the requested content. This would require a separate index

[2]Inverting this process is effortless at since the same stack component that is responsible for mapping N into $\bar{N}$ would remember this relationship and perform the inverse mapping when the content object response returns. The amount of state required for this procedure is directly proportional to the amount of pending interests at a consumer.

[3]The code is available at https://github.com/chris-wood/network-names.

[4]If the network name can be used to index into a repository for statically generated content, then inversion is not necessary.

[5]Note that the producer can not compute the inverse based only on network prefixes if $\mathsf{T}(\cdot)$ if a cryptographic hash function, as we propose.

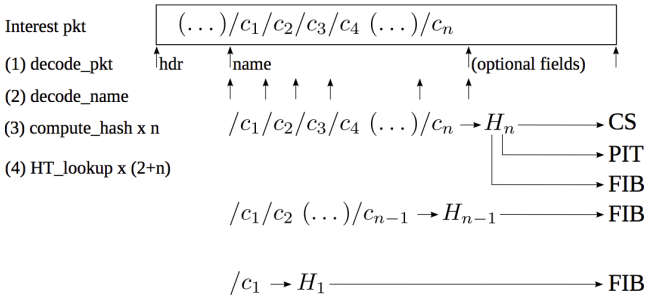


Figure 1: CCN interest forwarding using hash tables [5].

for each supported $\mathsf{T}(\cdot)$.

As far as the second implication for producers, recall that, when forwarding a content object, its name is used only for **exact matching**, i.e., there is no LPM as with interests. This means that the producer can replace the full network name with just N_p. This appreciably reduces router overhead and fixes the overall name size in the content object header. This is very beneficial when the size of a content's name exceeds that of the payload, which might be the case for small contents produced by IoT sensors [13] or NACKs [14].

4.2 Forwarding Implications

Figure 1 shows the computations needed to forward an interest per standard CCN in a forwarder that processes application names with a hash-based FIB. For an n-segment name, a forwarder must compute at most n hashes. The complete name hash is used to index the CS, PIT, and FIB, while the remaining $(n-1)$ hashes are used to index only the FIB.

Clearly, FIB lookup is the most expensive operation as it requires the most hashes. The LPM algorithm is typically implemented using Bloom filters (BFs) in hardware-based forwarders. This is because the n independent hash functions and lookups can be done in parallel. However, in software, BFs are not appropriate since the BF index operation is necessarily sequential. To the best of our knowledge, [5] is one of the most efficient software-based techniques for FIB lookup based on hash tables. Specifically, to lookup a name prefix with k segments, the technique in [5] performs a single hash (for each segment), one modular reduction (based on the number of hash table buckets), and a memory lookup. The penalty of reading memory not in the data cache is over 100 memory stall cycles.[6] For example, the cost of computing a 64-bit hash with SIPHASH [5] is 5.94 cycles/byte. In this case, the cost of hashing a name exceeds that of the penalized lookup if the name being hashed exceeds 16 bytes. Thus, for names longer than 16 bytes, hash function computation accounts for at *least half* the cycles needed for hash table lookup.

This is where the motivation for the proposed network naming scheme becomes most apparent: our proposal removes hashing overhead from: (1) FIB hash table lookup and (2) complete name hash (for CS and PIT lookups). In the former case, this reduces the fast-path overhead by at least the cost of the mapping function (depending on the name length). In the latter case, PIT and CS lookups are simplified for reasons discussed above. Since both PIT and CS are effectively indexed by N_p, this is the only information required to read and write to these data structures. Also, since N_p is always included in interest and content packets, this computation is removed from the fast path.

As with producers, routers would have to support a range of $\mathsf{T}(\cdot)$-s by accommodating different $\mathsf{T}(\cdot)$ output sizes. This can be done by (1) re-hashing network names when indexing any of these tables, or (2) maintaining a separate index for each supported size. In case (1), we claim that the router overhead is still reduced since the input to these re-hash functions is always a fixed size, not exceeding 32 bytes; application names may very well exceed this size and therefore take more time to hash. In case (2), although table management and maintenance might become more complicated, there will be no need for re-hashing. As part of future work, we plan to implement the software-based lookup in [5] using network names and analyze its performance.

4.3 Routing Protocols

Our proposal does not require any changes to CCN routing protocols. Producers announce application name prefixes for namespaces they control. Routers then hash received prefixes to populate FIB tables, as described in Section 2. To guarantee correct interest forwarding, the function used by routers to fill FIBs and $\mathsf{T}(\cdot)$ used by consumers when generating network names *must* be the same. This is the same requirement for the producer-generated reverse mappings mentioned in Section 4.1.2. Similarly, multiple $\mathsf{T}(\cdot)$ functions can be supported by routers. This would require routers to have multiple FIB entries per prefix – one for each supported hash function. Clearly, this represents a trade-off between scalability and mapping agility, versus increased FIB table sizes.

5. EXPERIMENTAL ANALYSIS

In this section, we compare our proposal to the standard CCN naming scheme, aiming to show that network names are functionally equivalent to application names, (in terms of uniqueness, size, and distribution properties) and enable correct forwarding with improved performance. We use the Unibas dataset from the The Content Name Collection [10], which contains *unique* URLs submitted by users to URL shortner websites. We convert these URLs into a CCN-compatible name format. For example, the URL `http://www.domain.com/file.html` is converted into the CCN name `/com/domain/file.html`. Table 3 shows some characteristics of the Unibas dataset.[7]

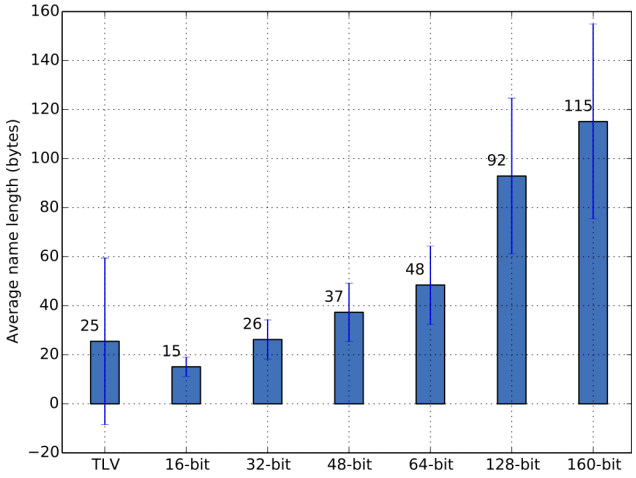
Table 4 illustrates name distribution per number of segments in each name. It shows the number of names in the dataset that contain n segments for $n \in [1, 20]$. Note that: (1) almost 30% of names have 5 segments, and (2) names of up to 20 segments account for 99.876% of the Unibas dataset. Also, we use application names that only contain locator segments, (i.e., with no identifier segments) because such identifiers are not easily distinguished from locator segments outside the application layer, and they are included in network names exactly as they appear in application ones.

[6]If the memory location is in the cache, there is no stall penalty.

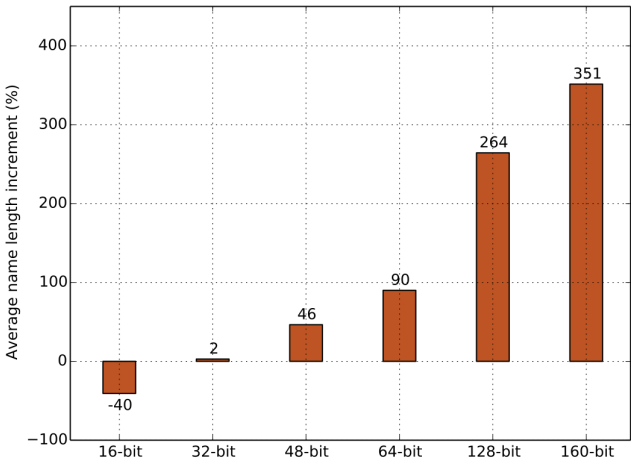
[7]For practical reasons, we truncate names beyond the maximum size of 2,000 bytes and 80 segments [15].

Table 3: Unibas Dataset Characteristics

Names		Name segments		Segments per Name	
Number of names	870′896′633	Total number of segments	4′855′203′042	Total number of segments	4′855′203′042
Average name length (bytes)	57.95	Average segment length (bytes)	10.39	Average segments per name	5.57
Name length standard deviation	77.60	Segment length standard deviation	30.02	Segments per name standard deviation	8.14
Minimum name length (bytes)	1	Minimum segments length (bytes)	1	Minimum segments per name	1
Maximum name length (bytes)	764′867	Maximum segments length (bytes)	764′867	Maximum segments per name	210′658



(a) Unibas average name length and standard deviation for various $\mathsf{T}(\cdot)$ sizes.



(b) Average name length increment for various $\mathsf{T}(\cdot)$ sizes.

Figure 2: Network overhead differences between network and application names.

5.1 Encoding Overhead

An advantage of fixed-size mapping to generate network names is that all hash values in the `T_MAPPED_XXX` segment have the same size. However, a name can still have an arbitrary number of segments, since there is no such restriction. Depending on the mapping function and whether its output is truncated, network names might be longer or shorter than their application counterparts. Also, `T_MAPPED_XXX` segments an carry extra 4 bytes: 2 for the type and another 2 for the length. Therefore, the total length of a `T_MAPPED_XXX` segment is $(k * s) + 4$, where k is the number of name segments involved in $\mathsf{T}(\cdot)$'s computation and s is the truncated size of $\mathsf{T}(\cdot)$. Whereas, the total length of a standard CCN TLV-encoded name can be computed as $\sum_{i=1}^{k}(|N[i]| + 4)$.

Figure 2(a) demonstrates the Unibas dataset average name length (in bytes) and the standard deviation for various truncated output sizes of $\mathsf{T}(\cdot)$. (TLV represents application names encoded in TLV format.) Figure 2(b) shows the average name length increment (or decrement) for various truncated output sizes of $\mathsf{T}(\cdot)$, as compared to the TLV-encoded application names. We note that for $\mathsf{T}_{16}(\cdot)$, proposed network names offer size of reduction of 40%. For $\mathsf{T}_{32}(\cdot)$, network names result in only 2% additional length over application ones. However, for $\mathsf{T}_{160}(\cdot)$, network name sizes increase by a factor of 2.5.

One advantage of network names in content objects is their fixed size. Similar to the discussion above, N_p might be larger or smaller than the original application name, depending on $\mathsf{T}(\cdot)$ output size. We explore encoding overhead of name fingerprints in content headers under three $\mathsf{T}(\cdot)$ output bit-sizes: 256, 384, and 512. We assess network overhead in two ways:

- Average overhead: overhead of using the fingerprint as compared to each name in the Unibas dataset. Results are presented as the average of all the individual overhead values.
- Overhead compared to the average name: overhead as compared to the average length of *all* names in the Unibas dataset.

Figure 3 shows results of using these methods for computing network overhead for the name fingerprint. It demonstrates that giving more weight to individual names yields higher overhead as compared to the average name length. There is a size increase for all $\mathsf{T}(\cdot)$s since names in the dataset are generally shorter than 256-512 bits. However, since the content namespace has substantial room to grow, this overhead will likely decrease as application names grow and diversify.

Table 4: Name Distribution per # of Segments

Number of segments n	Number of names	Percentage
1	13′952	0.002%
2	141′904	0.016%
3	71′327′647	8.190%
4	187′307′048	21.507%
5	253′852′565	29.148%
6	144′130′578	16.550%
7	93′837′904	10.775%
8	70′875′144	8.138%
9	25′611′959	2.941%
10	10′464′092	1.202%
11	3′973′961	0.456%
12	4′546′842	0.522%
13	1′206′905	0.139%
14	835′124	0.096%
15	844′552	0.097%
16	195′491	0.022%
17	121′486	0.014%
18	317′628	0.036%
19	168′228	0.019%
20	50′742	0.006%
Total	**869′823′752**	**99.876%**

5.2 Collision Resistance

As mentioned in Section 3.1, $\mathsf{T}(\cdot)$ must *at least* be a collision-resistant mapping function and its output can be truncated for practical purposes. However, truncation might affect collision-resistance. To this end, we now look at the collision probabilities in network names for various truncated sizes of $\mathsf{T}_s(\cdot)$, based on prefix length of names in the Unibas dataset. Specifically, we compute the collision probability for at least two *unique* application name prefixes of n segments ($n \in [1, 20]$). Table 5 shows the name distribution per number of prefix segments. Over 40% of the names have a unique prefix of 5 segments.

We evaluate the collision probabilities experimentally as well as analytically. To address the latter, we use the Binomial distribution with parameters m and p as in Equation 1.

$$f(d) = \binom{m}{d} p^d (1-p^d)^{m-d} \quad (1)$$

$f(d)$ is the probability of a unique name prefix mapped using $\mathsf{T}_s(\cdot)$ occurring at least d times among network name prefixes of length $n \in [1, 20]$, m is the size of this set, and p is the probability of a single element of that set to occur. Assuming $\mathsf{T}_s(\cdot)$ is based on a cryptographic hash function, its truncated output is guaranteed to be pseudo-random. Therefore, $p = 1/2^s$.

The probability of collision that we are trying to determine is $f(d)$ for $d \geq 2$. This is because a collision occurs whenever two or more prefixes map to the same value in the set of mapped name prefixes of length n. We denote this by $f(2^+)$, computed using Equation 2.

$$f(2^+) = 1 - f(0) - f(1) = 1 - \left(1 - \frac{1}{2^s}\right)^m - \frac{m}{2^s}\left(1 - \frac{1}{2^s}\right)^{m-1} =$$
$$1 - \left(1 - \frac{1}{2^s}\right)^m \left[1 + \frac{m}{2^s}\left(\frac{1}{1 - \frac{1}{2^s}}\right)\right] =$$
$$1 - \left(1 - \frac{1}{2^s}\right)^m \left[1 + \frac{m}{2^s - 1}\right] \quad (2)$$

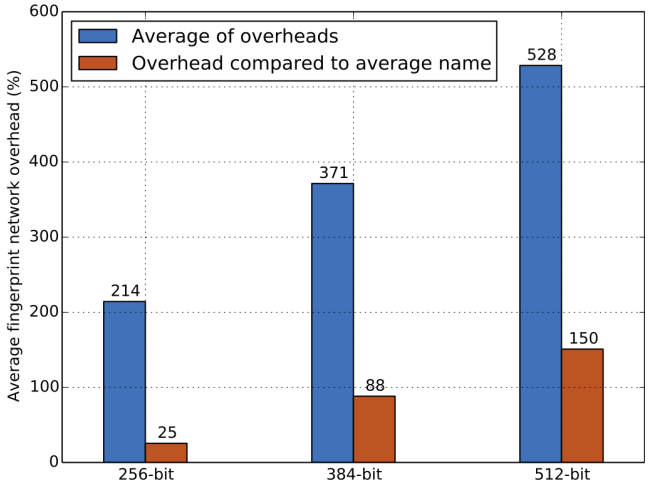


Figure 3: Average network overhead of name fingerprints in content headers with various $\mathsf{T}(\cdot)$-s.

Table 6 shows $f(2^+)$ using the given values of m in 5 (second column) and different truncated output sizes $[16, 32, 48, 64, 128, 160]$. We compared these values of the collision probability with those computed by mapping and truncating all the names in the Unibas dataset with prefixes of length $n \in [1, 20]$. From this comparison we observe the following:

- Collision probability for $\mathsf{T}_{16}(\cdot)$ is 1 because the number of names with prefix $n \in [2, 20]$ is over 2^{16}, which is the output space size of $\mathsf{T}_{16}(\cdot)$.
- Experimental collision probability for $\mathsf{T}_{64}(\cdot)$, $\mathsf{T}_{128}(\cdot)$, and $\mathsf{T}_{160}(\cdot)$ and some n values of $\mathsf{T}_{48}(\cdot)$ is 0. This is because the number of names with prefixes n in Table 5 is small compared to the output space size of the previous mapping functions. Therefore, an experimental collision cannot occur.
- Collision probability for $\mathsf{T}_{32}(\cdot)$ is nearly equal to the distribution in Table 5. Specifically, it reaches its maximum for $n = 5$, which is the most common prefix length among the names in the Unibas dataset. Figure 4 depicts this comparison.

6. SECURITY CONSIDERATIONS

From a security perspective, network names prompt several issues. Perhaps the most important is related to content signatures. Traditionally, signatures include the (application) name and payload of a content object. However, since packets no longer carry application names, the question of how one might verify content signatures in the network arises. Notice, however, that the name fingerprint N_p is included in interests and content objects. Cryptographically, there is no difference between hashing over the application name and payload vs hashing over the name fingerprint and payload. Thus, if signatures are generated using N_p instead of the application name, then routers can still perform in-network verification.

Second, since network names are computed with (possibly truncated) obfuscation functions, name prefix collision in a FIB would cause an interest to be forwarded on an incorrect interface. We believe that this is not an issue in practice because a FIB is populated by a routing protocol. Thus,

Table 5: Prefix Distribution per # of Prefix Segments

Number of prefix segments n	Number of *unique* prefixes m	Percentage
1	185′769	0.021%
2	10′159′460	1.167%
3	138′848′082	15.943%
4	305′236′992	35.049%
5	356′952′045	40.987%
6	225′092′000	25.846%
7	162′898′537	18.705%
8	98′471′166	11.307%
9	40′003′541	4.593%
10	19′165′574	2.201%
11	10′485′282	1.204%
12	7′597′933	0.872%
13	3′514′400	0.404%
14	2′540′599	0.292%
15	1′952′318	0.224%
16	1′193′997	0.138%
17	1′066′544	0.122%
18	1′044′263	0.120%
19	771′267	0.089%
20	646′401	0.074%

the routing protocol (which is exposed to application names) can explicitly prevent colliding names from being inserted into the FIB. Moreover, with cryptographic hashes, collision probability is negligible, as supported by experiments and analysis.

The third issue is that an adversary might provide fake network names that would cause interests to be forwarded towards a given router or producer as a form of denial-of-service (DoS). For example, if an adversary knows that a network name with the first component equals hash digest x, then it can generate and issue fake interests with that prefix even if it does not know the corresponding application name component. However, this is no different from currently possible interest flooding attacks [16–18]. Therefore, any solution to standard CCN interest flooding attacks would also prevent this type of DoS.

A more relevant and harmful DoS attack in the current CCN architecture is where an adversary generates interests for names that collide in the (hash-table-based) PIT and induce expensive collision-resolution steps [5]. This attack is not applicable with network names, since N_p is precomputed using a collision-resistant hash function. That is, if an adversary sends multiple interests with the same network name, then the probability that they correspond to different application names is negligibly small.

7. RELATED WORK

There is a significant body of research centered on the design and implementation of high-speed forwarders for CCN and related architectures [5, 19–21] that use different mechanisms to handle the current CCN naming scheme. For example, Quan et al. [20] propose a highly efficient name lookup engine for CCN forwarders that exploits the variability in names. Specifically, Tree-Bitmaps [22] and Bloom Filters (BFs) [23] are used in unison and in parallel to look up a name using LPM. The proposed parallel lookup algorithm involves computing BF hashes on one half of the name and then performing exact match in a Tree-Bitmap on the other half. The naming scheme we propose does not change this behavior in any way; it merely makes all inputs a uniform and predictable size. So et al. [5, 24] also proposed an efficient forwarding engine that uses a hash-based data structure for name LPM. The design uses SipHash for its low cycles/byte throughput and collision properties. The network naming scheme we propose would also benefit from the solution in [5,24] since the hashing step in the hash table lookup can either be skipped outright or, at the very least, the hash function input is reduced to a fixed size.

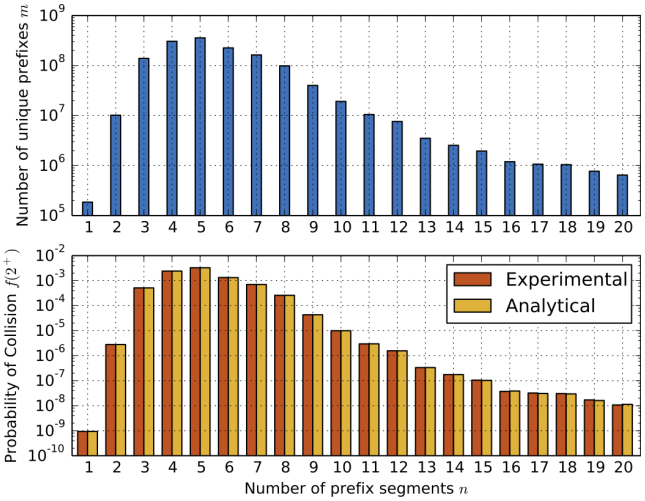


Figure 4: Collision probability for $\mathsf{T}_{32}(\cdot)$ in both experimental and analytical cases.

Perino et al. [25] propose an alternative LPM implementation that does not use successive lookups as in [5]. [25] decomposes the LPM problem into two subproblems: (1) finding the longest length of a matching prefix using a BF and (2) a hash table lookup. High-quality pre-computed hashes improve the efficacy of the latter approach while negligible effect on the former. (Other BF-based LPM solutions include [26] (and [27, 28] for IP networks).) The work of Fukushima et al. [21] also uses BFs composed with a hash table lookup to reduce the search space and size of the hash table. (This solution is an enhancement to [25].) Again, the proposed naming scheme in this paper would not negatively affect the BF lookup, though it may certainly help improve the hash table index operation.

Song et al. [29] propose a novel FIB design that uses name compression and Patricia tries [30] to reduce the footprint of FIBs. The naming scheme we propose would make this particular solution worse since cryptographic hash digests have a high amount of entropy and cannot be compressed. Wang et al. [31] proposes a LPM implementation which relies on a specific sorted order of names in the FIB data structure. As with the previous work, the encoding scheme we use removes the order property from names and therefore renders this technique ineffective. Conversely, the earlier name component encoding solution of Wang et al. [32] would work equally well with our network naming scheme since it relies on exact-match names.

Yuan et al. [33] studied the performance of existing forwarding engines to determine the pain points for forward-

Table 6: Probability of Collision Results - 'Exp.' represents collision probabilities obtained from experiments and 'Ana.' represents the same probabilities computed analytically using Equation 2. Highlighted numbers indicates anomalies between experiments and analytical results.

n	$T_{16}(\cdot)$		$T_{32}(\cdot)$		$T_{48}(\cdot)$		$T_{64}(\cdot)$		$T_{128}(\cdot)$		$T_{160}(\cdot)$	
	Exp.	**Ana.**	**Exp.**	**Ana.**	**Exp.**	**Ana.**	**Exp.**	**Ana.**	**Exp.**	**Ana.**	**Exp.**	**Ana.**
1	0.77	0.77	9.31E-10	9.35E-10	0	2.18E-19	0	5.07E-29	0	1.49E-67	0	8.08E-87
2	1	1	2.80E-6	2.79E-6	0	6.51E-16	0	1.52E-25	0	4.46E-64	0	2.42E-83
3	1	1	5.12E-4	5.12E-4	1.21E-13	1.22E-13	0	2.83E-23	0	8.32E-62	0	4.61E-81
4	1	1	2.41E-3	2.41E-3	7.14E-13	5.88E-13	0	1.37E-22	0	4.02E-61	0	2.18E-80
5	1	1	3.27E-3	3.27E-3	7.57E-13	8.04E-13	0	1.87E-22	0	5.50E-61	0	2.98E-80
6	1	1	1.33E-3	1.32E-3	2.52E-13	3.20E-13	0	7.44E-23	0	2.19E-61	0	1.19E-80
7	1	1	7.01E-4	7.01E-4	1.63E-13	1.67E-13	0	3.90E-23	0	1.15E-61	0	6.21E-81
8	1	1	2.59E-4	2.59E-4	7.82E-14	6.12E-14	0	1.42E-23	0	4.19E-62	0	2.27E-81
9	1	1	4.31E-5	4.31E-5	7.11E-15	1.01E-14	0	2.35E-24	0	6.91E-63	0	3.75E-82
10	1	1	9.93E-6	9.93E-6	0	2.32E-15	0	5.40E-25	0	1.59E-63	0	8.60E-83
11	1	1	2.96E-6	2.98E-6	0	6.94E-16	0	1.62E-25	0	4.75E-64	0	2.57E-83
12	1	1	1.57E-6	1.56E-6	0	3.64E-16	0	8.48E-26	0	2.49E-64	0	1.35E-83
13	1	1	3.34E-7	3.35E-7	0	7.79E-17	0	1.81E-26	0	5.33E-65	0	2.89E-84
14	1	1	1.75E-7	1.75E-7	0	4.07E-17	0	9.48E-27	0	2.79E-65	0	1.51E-84
15	1	1	1.05E-7	1.03E-7	0	2.41E-17	0	5.60E-27	0	1.65E-65	0	8.92E-85
16	1	1	3.73E-8	3.86E-8	0	9.00E-18	0	2.09E-27	0	6.16E-66	0	3.34E-85
17	1	1	3.21E-8	3.08E-8	0	7.18E-18	0	1.67E-27	0	4.91E-66	0	2.66E-85
18	1	1	3.05E-8	2.96E-8	0	6.88E-18	0	1.60E-27	0	4.71E-66	0	2.55E-85
19	1	1	1.70E-8	1.61E-8	0	3.75E-18	0	8.74E-28	0	2.57E-66	0	1.39E-85
20	1	1	1.07E-8	1.13E-8	0	2.64E-18	0	6.14E-28	0	1.80E-66	0	9.78E-86

ing based on CCN names at line rate. [33] concluded that the primary problems were (1) exact string matching with fast updates, (2) longest prefix matching for variable-length and unbounded names, and (3) large-scale flow maintenance. The naming scheme we propose solves problems (2) and, with the application of previously discussed techniques, may transitively help solve (1). Problem (3) remains the same when dealing with both standard CCN names and the proposed network names.

With respect to content locator and identifier separation, Adrichem et al. [34] propose an alternative naming mechanism for CCN that maps context-related application names to location-aggregated network names. The goal is to allow content to be published in a location-agnostic manner. Not only does this solution require the use of the DNS to perform this mapping, it also breaks the location-independent nature of CCN names. It may also expand the length of network names depending on the amount of location information returned from the mapping.

Ali et al. [35] examine the requirements for application naming schemes in CCN but do not discuss how it can be efficiently conveyed in the network layer.

8. CONCLUSION

This paper presented a means to transparently decouple application and network namespaces in CCN. The proposed approach is based on a deterministic mapping from application names to network names. This approach yields some important benefits, including: removal of application-specific names from the router's fast path, fixed-size name components in packets, and pre-generated name hashes for routers to use in FIB, PIT, and cache lookups. We discussed implications of the proposed naming scheme on consumers, producers and routers and also addressed some relevant security concerns. Finally, our analysis shows there is little functional difference between standard application-layer names and proposed network-layer names.

For future work we plan to implement the naming scheme in CCN-compliant stacks and (open source) high-speed forwarders to obtain quantitative measurements of its overhead. This might include different FIB data structures that exploit the fixed-length nature of network names, such as string or character tries. We also intend to further explore entropy differences between application- and network-layer namespaces.

9. REFERENCES

[1] M. Mosko *et al.*, "CCNx semantics," *IRTF Draft, Palo Alto Research Center, Inc*, 2016.

[2] B. Ahlgren *et al.*, "A survey of information-centric networking," *Communications Magazine*, 2012.

[3] marc.mosko@parc.com and C. Wood, "The CCNx URI Scheme," Internet Engineering Task Force, Internet-Draft draft-mosko-icnrg-ccnxurischeme-01, Apr. 2016, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-mosko-icnrg-ccnxurischeme-01

[4] P. Mockapetris, "RFC 1035: Domain names - implementation and specification," 1987.

[5] W. So *et al.*, "Named data networking on a router: fast and dos-resistant forwarding with hash tables," in *ANCS*, 2013.

[6] R. De La Briandais, "File searching using variable length keys," in *Western joint computer conference*,

1959.
[7] P. Brass, *Advanced data structures*. Cambridge University Press Cambridge, 2008.
[8] T. Przygienda, "RFC 3359: Reserved type, length and value (TLV) codepoints in intermediate system to intermediate system," 2002.
[9] I. Solis, marc.mosko@parc.com, and C. Wood, "CCNx Messages in TLV Format," Internet Engineering Task Force, Internet-Draft draft-irtf-icnrg-ccnxmessages-03, Jun. 2016, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-irtf-icnrg-ccnxmessages-03
[10] "The content name collection," http://www.icn-names.net/, accessed: April 8, 2016.
[11] "CCNx distillery," https://github.com/parc/CCNx_Distillery, accessed: May 14, 2016.
[12] S. Gulley and V. Gopal, "Haswell cryptographic performance," *Intel Corporation*, 2013.
[13] J. Burke *et al.*, "Secure sensing over named data networking," in *NCA*, 2014.
[14] A. Compagno *et al.*, "To NACK or not to NACK? negative acknowledgments in information-centric networking," in *ICCCN*, 2015.
[15] "WWW FAQs: What is the maximum length of a URL?" https://boutell.com/newfaq/misc/urllength.html, accessed: April 8, 2016.
[16] A. Afanasyev *et al.*, "Interest flooding attack and countermeasures in named data networking," in *IFIP Networking*, 2013.
[17] A. Compagno *et al.*, "Poseidon: Mitigating interest flooding DDoS attacks in named data networking," in *LCN*, 2013.
[18] M. Wählisch *et al.*, "Backscatter from the data plane–threats to stability and security in information-centric network infrastructure," *Computer Networks*, 2013.
[19] C. Yi *et al.*, "A case for stateful forwarding plane," *Computer Communications*, 2013.
[20] W. Quan *et al.*, "TB2F: Tree-bitmap and bloom-filter for a scalable and efficient name lookup in content-centric networking," in *IFIP Networking*, 2014.
[21] M. Fukushima *et al.*, "Efficiently looking up non-aggregatable name prefixes by reducing prefix seeking," in *INFOCOM WKSHPS*, 2013.
[22] W. Eatherton *et al.*, "Tree bitmap: hardware/software IP lookups with incremental updates," *ACM CCR*, 2004.
[23] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *ACM Communications*, 1970.
[24] W. So *et al.*, "Named data networking on a router: forwarding at 20gbps and beyond," in *ACM CCR*, 2013.
[25] D. Perino *et al.*, "Caesar: a content router for high-speed forwarding on content names," in *ANCS*, 2014.
[26] Y. Wang *et al.*, "Namefilter: Achieving fast name lookup with low memory cost via applying two-stage bloom filters," in *INFOCOM*, 2013.
[27] S. Dharmapurikar *et al.*, "Longest prefix matching using bloom filters," in *SIGCOMM*, 2003.
[28] H. Song *et al.*, "IPv6 lookups using distributed and load balanced bloom filters for 100gbps core router line cards," in *INFOCOM*, 2009.
[29] T. Song *et al.*, "Scalable name-based packet forwarding: From millions to billions," in *ICN*, 2015.
[30] W. Szpankowski, "Patricia tries again revisited," *JACM*, 1990.
[31] Y. Wang *et al.*, "Greedy name lookup for named data networking," in *ACM SIGMETRICS*, 2013.
[32] ——, "Scalable name lookup in NDN using effective name component encoding," in *ICDCS*, 2012.
[33] H. Yuan *et al.*, "Scalable NDN forwarding: Concepts, issues and principles," in *ICCCN*, 2012.
[34] N. L. Van Adrichem and F. A. Kuipers, "Globally accessible names in named data networking," in *INFOCOM WKSHPS*, 2013.
[35] A. Ghodsi *et al.*, "Naming in content-oriented architectures," in *ACM SIGCOMM workshop on Information-centric networking*. ACM, 2011.