Catch Me If You Can: A Practical Framework to Evade Censorship in Information-Centric Networks

Reza Tourani
New Mexico State University
rtourani@cs.nmsu.edu

Satyajayant Misra
New Mexico State University
misra@cs.nmsu.edu

Joerg Kliewer
New Jersey Institute of Technology
jkliewer@njit.edu

Scott Ortegel
New Mexico State University
sortegel@nmsu.edu

Travis Mick[*]
New Mexico State University
tmick@cs.nmsu.edu

ABSTRACT

Internet traffic is increasingly becoming multimedia-centric. Its growth is driven by the fast-growing mobile user base that is more interested in the content rather than its origin. These trends have motivated proposals for a new Internet networking paradigm–*information-centric networking* (ICN). This paradigm requires unique names for packets to leverage pervasive in-network caching, name-based routing, and named-data provenance. However named-data routing makes user censorship easy. Hence an anti-censorship mechanism is imperative to help users mask their named queries to prevent censorship and identification. However, this masking mechanism should not adversely affect request rates.

In this paper, we propose such an anti-censorship framework, which is lightweight and specifically targets low compute power mobile devices. We analyze our framework's information-theoretic secrecy and present *perfect secrecy* thresholds under different scenarios. We also analyze its breakability and *computational security.* Experimental results prove the framework's effectiveness: for requests it adds between 1.3–1.8 times in latency overhead over baseline ICN; significantly lesser than the overhead of the state of the art Tor (up to 38 times over TCP).

CCS Concepts

•**Networks** → **Network privacy and anonymity;** *Application layer protocols;* •**Security and privacy** → **Privacy protections;**

Keywords

NDN; information-centric networking; information theory; security; privacy

[*]This work was supported in part by the US National Science Foundation grants 1241809 and 1345232 and US DoD grant 67311RTREP. The information reported here does not reflect the position or the policy of the federal government.

ICN'15, September 30–October 2, 2015, San Francisco, CA, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3855-4/15/09…$15.00.
DOI: http://dx.doi.org/10.1145/2810156.2810171

1. INTRODUCTION

The host-centric Internet, where a node uses domain name systems to map the canonical name (e.g., www.google.com) to the IP address, and identify the data source before getting the required data, cannot scale for the traffic trends of the future. These scalability concerns have resulted in the proposal of a move to an information-centric Internet. The corresponding *information-centric networking* paradigm is based on the concept that a node interested in a content just wants the network to provide the content, it does not really care about where the content comes from; other than perhaps the content's provenance (validate creator). Invariably, this paradigm requires a node interested in a content to know the content's name and request its chunks/packets from the network using the corresponding name—the packets are routed in the network based on their name.

Named-data based networking has received significant interest recently, resulting in the proposal of several information-centric network (ICN) architectures, such as Named Data Networking (NDN) [12], NetInf [6], Publish-Subscribe Internet Routing Paradigm (PSIRP) [1], PURSUIT [16], and Data Oriented Networking (DONA) [13]. Each architecture has it's individual nuances, however all of them share three important aspects: unique name for each data unit (packet or chunk), routing based on data name, and in-network name-based data caching. The names could be either hierarchical and human-readable or machine-readable. These inherent advantages of this paradigm make it a popular candidate for the future, more scalable, Internet.

The use of data names in ICN architectures to identify content for routing and for data searches in the network brings forth some challenges as well. One important challenge is that the explicit use of names to describe the content makes it susceptible to censorship. Let's use the popular CCN/NDN architecture to illustrate this issue. In CCN, the user sends out interest requests for the named content chunks (e.g., www.google.com/movies/Arab-Spring.mpg.1) to the network. The network's built-in intelligence retrieves the chunks, either from the content provider or an intermediate router caching them. This content retrieval requires name based routing, that is, a request's name will be processed by each forwarding network node. This enables a forwarding node to easily filter and drop requests that it wants to censor. In fact, as opposed to censorship in today's Internet: implemented by blocking data based on destination IP addresses or by time-intensive deep packet inspection at proxies; *in the ICN architecture, censorship becomes trivial,* *as the content name is embedded as plaintext in the interest packets.* The filtering router can simply check the content name in each packet for filtering purposes.

Monitoring and censorship of users' traffic has been widely used in the past by regimes and countries to prevent free exchange of ideas (e.g., restricted social media access by Iran/China, data monitoring by the NSA in the US). Such acts of censorship are becoming more prevalent on the Internet today, and would become easier to orchestrate on the information-centric Internet. *Mechanisms need to be proposed to prevent such censorship acts. In this paper, motivated by these observations, we propose a* ***novel framework*** *to prevent censorship based on data names.* Our framework meets the requirements of strong security needed to prevent inspection based censorship and also fast encoding to ensure that the user request rates are not adversely affected. We select the CCN architecture for illustration on account of its popularity and mature code-base. We use a 18 node CCNx [14] testbed for our implementations and experiments.

The **contributions** of this paper are: *(i)* Design of a lightweight anti-censorship framework for the users in an ICN. Our framework leverages the computation efficient prefix-free information encoding techniques (e.g., Huffman coding) and is widely applicable for mobile devices with their low-computation capabilities. *(ii)* Discussion of the protocols that utilize the framework to enable private communication between a user and a content provider in a way that the communication cannot be censored by intermediate network entities. *(iii)* Extensive evaluation of the framework's information-theoretic and computational secrecy. *(iv)* Implementation of the framework in our CCNx testbed to validate its low communication overheads compared to the state of the art anti-censorship tools, such as Tor [19].

Section 2 presents the related work. In Section 3, we present our models and assumptions. Our framework is presented in Section 4, its information-theoretic secrecy is proved in Section 5, and analysis of its breakability and computational security is performed in Section 6. Section 7 presents the implementation and experimental results in details. Finally, we draw our conclusions and present our future work in Section 8.

2. RELATED WORK

Research in both Future Internet Architecture (FIA) security and prefix-free coding are relevant to this paper. We present relevant research in both areas starting with security in FIAs. Arianfar *et al.* proposed a steganographic approach in [4] for the censorship problem. In this scheme, the content provider generates a cover file and splits both the content and the cover into smaller blocks. The provider creates chunks by mixing blocks of the content and the cover and publishes the resulting chunks into the network. The provider sends additional information to the user, such as content hash and its length in blocks, the corresponding cover blocks, and the name generation algorithm, through a secure back channel. The use of the cover, which results in a 100% overhead and the requirement of a secure back channel for each content are the drawbacks of this scheme.

ANDaNA [7] uses two proxies, one adjacent to the requester and another closer to the destination, to create a two-layer encryption of the requests. Using ANDaNA a user decouples its identity from its requests: the first-hop proxy is only aware of the user identity but not what is requested while the second proxy's knowledge is limited to the requested content only. Despite ANDaNA's usefulness as an anti-censorship tool, it incurs significant delays (ref. results in [7]) in comparison to Tor (the Onion Routing protocol)—the popular Internet anti-censorship tool. The delays are due to the use of the CCN architecture and setting up of the secure communication channels.

Tor is one of the most widely used and effective mechanisms to ensure end-to-end data secrecy, maintain user privacy beyond the first hop, and prevent censorship by completely anonymizing the data using onion routing [19]. However, Tor also incurs significant latency and bandwidth overhead on account of the shared symmetric key operations (decryption) at each intermediate node. *The bottom line is that the state of the art approaches are expensive, requiring several specialized nodes in the network (proxies and onion routers); more importantly, they require multiple symmetric/public key encryptions, which reduces application throughput significantly.* We take a different censorship evasion approach by proposing a prefix-free Huffman encoding framework for chunk names; we prove the framework's security and demonstrate its practicality. Our framework has much lower overhead than the state of the art and has very low latency. Also the need of the secure back channel is minimal.

Encryption using Multiple Huffman Tables (MHTs) was first proposed by Wu *et al.* [20]. In this approach, n distinct Huffman tables and a vector $\mathbf{Q}$ of size m, randomly filled with integers in the range of $(0, n-1]$ are used. Each source symbol is encoded by a Huffman table, selected from the pool of n tables, based on the index of the symbol in $\mathbf{Q}$. The Huffman table selection iterates through the vector until the source message is entirely encoded. Although MHT was proved to be secure against cipher-text and the plaintext attacks, the Chosen Plain-text Attack (CPA) broke the scheme's security [21]. Updating the vector $\mathbf{Q}$ for every m look-ups, was shown to secure the MHT approach against CPAs. An alternative solution was also proposed which required the insertion of a few random bits in the encoded source sequence according to specific bit values [22].

In our scheme, we do not use MHT, instead each user has one table for encoding. The table can be changed at a user-defined rate if necessary. We prove that our framework provides higher information-theoretic security than using AES. Our approach guarantees perfect information-theoretic secrecy so long as the size of the data name is no more than a calculated threshold. We also discuss our scheme's defense against well-known cryptanalysis attacks.

3. MODELS AND CONVENTIONS

In this section, we present the system model, our assumptions, the attack model, and the convention for the generation of the key from the Huffman tree.

3.1 System Model

The network is composed of a set of users ($\mathcal{U}$), routers ($\mathcal{R}$), providers ($\mathcal{P}$), and a set of trusted proxies named *anonymizers*. A router is either a filtering router or a normal router. A filtering router (r_f) filters the incoming requests' data names and drops requests whose names appear in a blacklist of contents. The filtering process is on-line (happens on the fly), and to reduce congestion and prevent throttling of the traffic the filtering is generally performed very close to line speed (packet arrival rate). Without loss of generality, we assume that a user is directly connected to a filtering router, which is in turn connected to the rest of the network.

Assuming that the Internet Protocol is not used as the network layer, in ICN, the first hop node on the path of a request is the only one that can identify the requester (from MAC-layer header); further along the requester's identity is absent in the packet. This is the worst case scenario. If IP-based addressing is used, we assume that the requester's IP address is cloaked using some anonymization technique. With IP-cloaking the situation is the same as when IP addressing is not used. Our scheme also applies when the filtering router and the user are separated by several hops. The user requests a content by sending an *interest* packet (terminology borrowed from the CCN/NDN architecture [12]) containing the name of the content chunk. The name of an object is in a hierarchical format starting with the content provider's name, e.g., *www.youtube.com/ArabSpring.mpg.1* (the postfix number '1' is the chunk ID).

In our framework, instead of using the plaintext content name for the interest, we use an encoded name. The encoded name can only be decoded correctly by a pre-selected anonymizer which has the decoding table. No entity in the path between the anonymizer and the requester has the decoding table and hence cannot decode the name. Between the anonymizer and the data provider (source or an intermediate router) the request is transmitted as a normal request and no filtering happens. We assume that between the anonymizer and the requester the data is encrypted; otherwise the filtering entity can identify the content. Generally, a third-party entity, chosen by the user from a publicly available list, serves as the anonymizer. This is analogous to what happens today: users evade traffic censorship by choosing an anonymizing service, such as *anonymizer.com* [3], as a proxy, and bypass the censors by tunneling their traffic (e.g., Facebook, Youtube) through the anonymizer's servers. A content provider can also operate as the anonymizer.

We note that our design has a trade-off between user privacy and caching (network) efficiency. It has been shown that to guarantee strong unlinkability of users' identity with their requests in-network caching should be avoided [2, 5]. In our framework, if a user does not trust the network, it can request the content provider to be its anonymizer. This would guarantee strong unlinkability in the network, however, the corresponding cached data at intermediate routers is unusable to serve new requests. This is true with Tor and ANDaNA. The multi-layered onion-routing based encryptions render the cached private data unusable for satisfying repeat requests. On the other hand, if the user chooses an anonymizer in the network, the cached data is only unusable by the routers in the path between the user and the anonymizer. In-network caching can be leveraged in the rest of the path. *In this paper, we assume the content provider is the same as the anonymizer.*

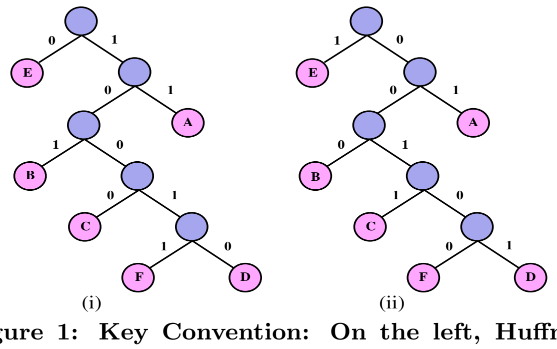


Figure 1: Key Convention: On the left, Huffman tree (i) is shown with the key $\underline{0101100110}$**; On the right, Huffman tree (ii) is presented with** $\underline{1001011001}$ **as the key. Although the structure of both trees are the same, generated codewords are different due to different keys.**

3.2 Attack Model

We assume that the attacker (censoring authority) is either an active or a passive eavesdropper. The attacker's aim is to learn how to correctly decode an interest. We also assume that the attacker has bounded capabilities, i.e., it cannot do large-scale brute force attacks. A passive eavesdropper can capture all packet transmissions to perform analysis. An active eavesdropper can capture and modify in-flight requests and also masquerade as a legitimate user to send interests.

3.3 Convention for Huffman Tree Key Generation

We consider an alphabet of size N for the source messages (chunk names). The source message is denoted by M^k, where k is the message size. The encoded message is denoted by $\underline{Z}$ and is assumed to be a binary string of size n. In our framework *three sources* of randomness exist, namely the random selections of the Huffman tree structure, the conventional key, and the source alphabet order from their respective universal sets. The user and the content provider can secretly perform one or more of these three random selections, thus making it difficult to break the system. The Huffman tree is a full binary tree with source alphabet symbols placed at the leaf nodes. Hence, for an alphabet of size N there are N leaf and $N-1$ internal nodes. However, for the same set of symbols there can be several possible trees, one of which can be chosen randomly. We will explain our convention to generate a key for a Huffman tree in detail below. The last source of randomness, is the random choice of the alphabet order, that is, the order in which the alphabet symbols are placed on the leaf nodes. Instead of the standard Huffman coding where the placement of the symbols is based on the frequency of a symbol [11], the symbol placement could follow a random distribution to further increase the framework's randomness.

In this paper, we leverage the first two methods, namely the Huffman tree structure and the conventional key. We assume that the order of the source symbols in the tree is known to everyone. Exploiting the Huffman source coding, we introduce a novel key generation mechanism from the Huffman tree. We assume that the source (user) and the destination (content provider) of a communication have secretly shared a Probability Mass Function (pmf), which represents the probability of the occurrence of the symbols of an alphabet set in a message. The pmf is selected deliberately as a part of the secrecy mechanism and does not reflect the true source distribution. We also assume that both the source and the destination have information to built the Huffman tree with the same structure. The Huffman tree is generated by assigning labels '0' and '1' to the left and the right branch respectively, at each depth of the tree.

With a little analysis one can see that for a Huffman tree with $N-1$ internal nodes there are $2^{(N-1)}$ mutation trees, where a *mutation tree* of a Huffman tree is generated by swapping the labels of the internal nodes. Each of these mutation trees is associated with a unique string of size $2(N-1)$

that is obtained by traversing the tree sequentially by levels and at each level picking up the labels from left to right—similar to breadth-first search. Assuming that we have an arbitrary pmf for a six symbols alphabet, Figure 1 illustrates the corresponding Huffman tree. According to our convention, the key for the tree (i) on the left hand side is "0101100110" while the tree (ii) on the right hand side, has the key "1001011001" despite having the same structure. The source and the destination can frequently switch to a new key from the pool of $2^{(N-1)}$ keys, corresponding to the $2^{(N-1)}$ mutation trees, to improve communication secrecy.

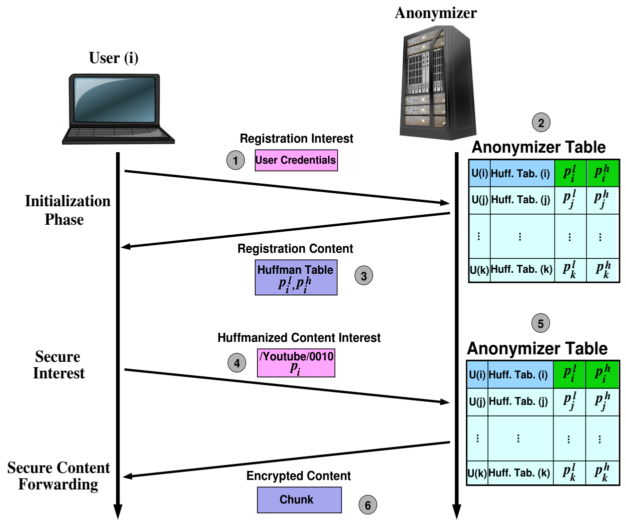
We note that this proposed convention is not restricted to the Huffman source coding. In general, every prefix-free source coding scheme with a full binary tree can use it.

4. ANTI-CENSORSHIP FRAMEWORK

Huffman coding is a promising approach to mitigate censorship in ICNs. In our framework, we encode a part of the content name, the postfix after the domain name, using the Huffman coding algorithm. The domain name is not encoded to allow for name based routing. We note that if the anonymizer is not the provider then the domain name can also be encoded. In this case, the anonymizer's domain name will be used as the prefix of the interest and used for routing. Once the interest reaches the ingress router of the domain (a CP or an anonymizer), the name in the interest is decoded into the real name. Although we use the Huffman coding technique to encode user interests other coding techniques can also be used in our framework. Our framework consists of three phases: *initialization, secure content sharing,* and *secure content response.* The initialization phase is used for sharing credentials between the user and an anonymizer to enable censorship-proof communication. For simplicity of exposition, we assume that the content provider is the anonymizer.

4.1 Initialization phase

In this phase, the user sends a membership request to the anonymizer ($\mathcal{A}$) encrypted using the anonymizer's public key and signed using the user's private key. On receiving the membership request, $\mathcal{A}$ generates a random Huffman table, by using a random pmf for the interest, and sends the membership reply secretly to the requester; it also stores these information in its table (refer Figure 2). There are two mechanisms that need to be addressed here: a) How will the membership reply be sent secretly to user u_i? and b) How can $\mathcal{A}$ store each u_i's credentials to allow quick indexing into the table to identify the corresponding Huffman table? The anonymizer $\mathcal{A}$ generates a range of pseudonyms $(p_i^l - p_i^h)$ for u_i and creates a table entry consisting of the generated Huffman table, the pseudonyms' lower (p_i^l) and upper (p_i^h) limits, and the PKI details of u_i. It then encrypts the Huffman table and the pseudonym limits using the public key of u_i (or a shared symmetric key) and signs the message using its private key before sending the reply to u_i. At this point, the initialization phase is complete and both u_i and $\mathcal{A}$ have the required information for censorship-proof communication. Steps 1 to 3 in Figure 2 illustrate this phase. After the initialization phase, the user can request privacy-sensitive content.



Anonymizer
User (i)
Registration Interest
1
User Credentials
2
Anonymizer Table
U(i) | Huff. Tab. (i) | p_i^l | p_i^h
U(j) | Huff. Tab. (j) | p_j^l | p_j^h
U(k) | Huff. Tab. (k) | p_k^l | p_k^h
Initialization Phase
Registration Content
Huffman Table p_i^l, p_i^h
3
5
Huffmanized Content Interest
4
/Youtube/0010 p_i
Secure Interest
Anonymizer Table
U(i) | Huff. Tab. (i) | p_i^l | p_i^h
U(j) | Huff. Tab. (j) | p_j^l | p_j^h
U(k) | Huff. Tab. (k) | p_k^l | p_k^h
Secure Content Forwarding
Encrypted Content
Chunk
6

Figure 2: A schematic diagram showing our framework's three phases: initialization, secure content request, and secure content response.

4.2 Secure content request

The user needs to generate an interest packet, where the interest name has to be customized to evade censorship. Essentially, the hierarchical content name is composed of the anonymizer's domain name in plaintext (to enable prefix based routing), concatenated with the Huffman encoded postfix of the name representing the exact chunk. The Huffman encoded portion of the name may vary depending on the secrecy level required by the user. For the highest level of secrecy, the complete name postfix after the anonymizer's domain name needs to be encoded. For lower levels of secrecy, the user can encode a portion of the postfix. For instance, consider a content name for a Arab Spring video: www.google.com/movies/2012/ArabSpring/HD/TahrirSquare. The interest with highest secrecy level encodes all segments except www.google.com/ (anonymizer's domain name); while the lowest level secrecy, only encodes the last segment, TahrirSquare, the prefix is plaintext. This flexibility helps the user to adjust its desired level of secrecy.

For fast indexing at $\mathcal{A}$, u_i chooses a random pseudonym $p_i \in \{p_i^l, p_i^h\}$ and adds it as an identity field in the interest packet/chunk. Anonymizer $\mathcal{A}$ uses an ordered binary tree data structure, which has the ordered pseudonym ranges of the users as leaves, to search for u_i's ID $(u(i))$ in its table. Then, given p_i, to identify u_i $\mathcal{A}$ makes $\mathcal{O}(\log |\mathcal{U}|)$ comparisons, where $\mathcal{U}$ is the set of users. We will explore faster search using collision-resistant hashed functions in the future work. Users not interested in privacy requirements can be allocated only one identifier or can use only one pseudonym from their range. After the interest packet generation, the user sends the interest for the content chunk. Step 4 of Figure 2 illustrates this.

4.3 Secure content response

When the interest packet arrives at a filtering node, the filtering node can only infer partial information from the received interest packet. The inference is only based on $\mathcal{A}$'s domain name, but is insufficient for filtering the interest to identify the content. The filtering router can only perform longest prefix matching of the interest name with entries in its forwarding information base (FIB) and forward the interest to the appropriate next hop. At $\mathcal{A}$, a table look-up on the pseudonym (Step 5 of Figure 2) in the interest packet returns the desired Huffman tree for parsing the Huffman-encoded content name. Then $\mathcal{A}$ decodes the postfix and obtains the data from the content store(s). In the reply, $\mathcal{A}$ uses the same interest name and the same pseudonym while encrypting the content (symmetric/public key).

Our framework is also amenable to the other proposed FIA named-data architectures. Note that naming of data in all these architectures fall into one of two categories: hierarchical, human readable naming and flat machine-readable naming (hashed names). In both cases, the name in the interest can be treated as a source message and encoded using the shared Huffman table. For instance, in NetInf, the anonymizer can operate as a local or global name resolution server (outside the filtering region) and do the transformation between the encoded and the real-name of the private data. In DONA, the anonymizer can operate as a high-level resource handler beyond the filtering routers. In PSIRP, a subset of the nodes in the rendezvous system can be chosen as the anonymizer(s). These nodes do the mapping between the encoded name and the real name of the data. The anonymizers can send the forwarding identifier to the source (provider) to send the data to the requester and ask the provider to use the encoded name.

5. INFORMATION-THEORETIC SECURITY ANALYSES

In this section, we investigate the information-theoretic security of our framework and the protocols under different assumptions on the eavesdropper's knowledge of the system. As mentioned in Section 3, we use the structure of the Huffman tree and the key as the sources of randomness. We omit the third source of randomness: the ordering of the alphabet in the Huffman tree. Hence, an attacker with the knowledge of the tree structure and the shared key can reconstruct the coding table, which breaks the system. However, as these information are encrypted using PKI or symmetric keys before transmission, they are secure.

Thus there are three remaining information-theoretic attack scenarios, which we will study: (i) the eavesdropper has no parsing information, that is, tree structure and the key are unknown; (ii) the eavesdropper has key information only; and (iii) the eavesdropper has tree structure information only. Throughout this section we use the term key and the mutation tree alternatively where both refer to the key in our framework obtained using the BFS-like traversal. We also note that the source message is essentially the hierarchical data name postfix that needs to be encoded.

First, we derive basic entropy terms that we will need in the rest of the section. As mentioned before, given a tree T for alphabet size N, we have $2^{(N-1)}$ mutually independent mutation trees for T, that is, $2^{(N-1)}$ keys for T. The selection of a mutation tree (i.e., key K's selection) uniformly at random from the set of mutation trees results in K's entropy to become:

$$\begin{aligned} H(K) &= -\sum_{i=1}^{2^{N-1}} p(i)\log\big(p(i)\big) && (1)\\ &= -\frac{1}{2^{N-1}}\sum_{i=1}^{2^{N-1}} \log\big(p(i)\big) && (2)\\ &= \log\big(2^{N-1}\big) && (3)\\ &= N-1. && (4) \end{aligned}$$

Besides the selection of a mutation tree of T, the random choice of the tree structure (a uniform random distribution) is also another source of randomness. The number of mutually independent full binary trees with N leaves is given by the $(N-1)^{th}$ Catalan number (C_{N-1}) (refer Catalan Problem [8]). The N^{th} Catalan number (C_N) with increasing N is given by $C_N \approx \Omega(\frac{4^N}{N^{3/2}})$, thus $C_{N-1} \approx C_N$ for large N. Consequently, the entropy of using a random and secret tree structure (T_R) can be written as:

$$\begin{aligned} H(T_R) &= -\sum_{j=1}^{(\frac{4^N}{N^{3/2}})} p(j)\log\big(p(j)\big) && (5)\\ &= -\frac{1}{\frac{4^N}{N^{3/2}}}\sum_{j=1}^{(\frac{4^N}{N^{3/2}})} \log\big(p(j)\big) && (6)\\ &\geq \log\big(\frac{4^N}{N^{3/2}}\big) && (7)\\ &= \log 2^{2N} - \log N^{3/2} && (8)\\ &= 2N - 3/2\log N. && (9) \end{aligned}$$

Also, considering the source alphabet with N symbols (e.g., $N = 512$ for Unicode or 256 for ASCII), which are uniformly randomly distributed, the source symbol entropy is given in Equation (12):

$$\begin{aligned} H(X) &= -\sum_{k=1}^{N} p(x_k)\log(p(x_k)) && (10)\\ &= -\frac{1}{N}\sum_{k=1}^{N}\log\big(p(x_k)\big) && (11)\\ &= \log N. && (12) \end{aligned}$$

Now we look at different eavesdropper attack scenarios.

5.1 Scenario 1: Both tree structure and the key are unknown

In this scenario, the eavesdropper has no knowledge of the tree structure or the key. Let M^k be the sequence of k symbols to be encoded and $\underline{Z}$ be the encoded binary sequence with length n symbols. The evaluation of the mutual information between the source message and the encoded sequence is provided by Equation (15) along with (4), (9), (12):

$$\begin{aligned} I(M^k;\underline{Z}) &= H(M^k) - H(M^k \mid \underline{Z}) && (13)\\ &\leq \max\Big(kH(X) - \big(H(T_R)+H(K)\big), 0\Big) && (14)\\ &= k\log(N) - 3N + 3/2\log(N) + 1. && (15) \end{aligned}$$

Equation (13) is obtained from the definition of the mutual information between the source message and its corresponding encoded sequence. The r.h.s. of Equation (14) is obtained from the fact that in this scenario the entropy of the message, given its encoded sequence, equals the entropies of the key and tree structure choices, and that the mutual information is always non-negative. The outcome of Equation (15) is the conditional entropy of the source sequence, given the encoded sequence, which is equal to the total randomness for both the structure and the key.

5.2 Scenario 2: Tree structure known, but not the key

In this scenario, the eavesdropper has complete knowledge of the tree structure and consequently can build the Huffman tree, and the key is the only secret. Hence, the mutual information between the source message and its encoded binary string is:

$$\begin{aligned} I(M^k;\underline{Z}) &= H(M^k) - H(M^k \mid \underline{Z}) && (16)\\ &\leq \max\Big(kH(X) - H(K), 0\Big) && (17)\\ &= \max\Big(k\log N - (N-1), 0\Big) && (18)\\ &= k\log N - N + 1. && (19)\end{aligned}$$

Equation (19) presents the entropy of the source message assuming each symbol of this message is an i.i.d. random variable. In practice, the dependency between the letters in a word in the English alphabet reduces the entropy of the upcoming symbol (letter) given the prior symbols. We will discuss this in Subsection 5.4. Equating the r.h.s. of Equation (19) to zero, we conclude that the amount of information leakage as k becomes greater than $\left(\frac{N-1}{\log N}\right)$ is proportional to the value of k. Although the leakage increases linearly with k, it must be investigated whether the eavesdropper can leverage this leakage or not. We will investigate this in the next section.

5.3 Scenario 3: Key known, but not the tree structure

Now we investigate the opposite scenario: the eavesdropper knows the key, but does not have access to the tree structure. This can happen when the eavesdropper intercepts the key sharing communication phase between the anonymizer and the user and somehow identifies the encrypted key. Equation (23) returns the entropy of the source message under this condition:

$$\begin{aligned} I(M^k;\underline{Z}) &= H(M^k) - H(M^k \mid \underline{Z}) && (20)\\ &\leq \max\Big(kH(X) - H(T_R), 0\Big) && (21)\\ &= \max\Big(k\log N - (2N - 3/2\log N), 0\Big) && (22)\\ &= (k + 3/2)\log N - 2N. && (23)\end{aligned}$$

Equating the r.h.s. of Equation (23) to zero, we have the threshold for the information leakage to be $k = \left(\frac{2N}{\log N} - \frac{3}{2}\right)$, which has the same growth rate as the previous scenario, explained in Subsection 5.2.

Table 1 illustrates the thresholds of source message lengths (in symbols) for perfect secrecy for the three scenarios, evaluated in Equations (15), (19), (23) respectively (i.i.d. symbols in the messages). *Note that messages longer than the threshold lead to leakage; leakage is defined as the difference between the message length and the length of the threshold.*

Table 1: Maximum possible source message length k (in symbols) for perfect secrecy in i.i.d. messages.

Scenario	N=32	N=64	N=128	N=256	N=512
Scenario 1	17.5	30.3	53.2	94.3	169.1
Scenario 2	6.2	10.5	18.1	31.8	56.7
Scenario 3	11.3	19.8	35.07	62.5	112.2

5.4 Dependent source scenario's information leakage

So far, we have assumed that the source message is composed of i.i.d. random variables. Although this assumption is valid in most cases (for URL names), instances exist where there is a dependency between the source message symbols. For example, if the source message uses English words, then this changes the distribution of the source symbols and they no longer follow an i.i.d. uniform distribution. Hence, we also investigate the amount of information leakage when the symbols are dependent. Now, the probability of choosing a symbol is conditioned on the previously selected symbols in the same message, which decreases the source message's rate (i.e., the average entropy per symbol in the message).

According to Shannon [17], the $\mathcal{N}$-gram (sequence of any $|\mathcal{N}|$ adjacent symbols) entropy per symbol (F_n) is bounded as

$$\sum_{i=1}^{N} i\log i(p_i^a - p_{i+1}^a) \leq F_n \leq \sum_{i=1}^{N} p_i^a \log p_{i+1}^a, \qquad (24)$$

in a way that given the previous $a-1$ symbols, there is a partial ordering of the symbols in the source alphabet corresponding to their probability of appearing as the next symbol. This can be discerned as a mapping between the symbols and integers such that the most probable next symbol (the a^{th} symbol) conditioned on the $a-1$ previous symbols, maps to $i=1$, the second probable symbol maps to $i=2$, and so on. Hence, p_i^a represents the probability of the i^{th} most probable symbol (among N symbols) to be placed at the a^{th} position in the message, conditioned on the known $a-1$ previous symbols. Clearly, p_1^a is the most probable next symbol for the a^{th} position in the message and p_N^a is the least probable symbol for the same position in the message. Therefore,

$$p_N^a \leq p_i^a \leq p_1^a, \qquad (25)$$

which can be inferred from [17]. The overall probability of source symbols for the $(a+1)^{th}$ position in the source message is at least equal to the overall probability of source symbols for the a^{th} position, that is,

$$\sum_{i=1}^{N} p_i^a \quad \leq \quad \sum_{i=1}^{N} p_i^{a+1}. \qquad (26)$$

In other words, the probability of guessing the correct source symbols increases with the size of the source message. For instance, for the word "the", the probability of guessing "e" after guessing "t" and "h" is higher than the probability of guessing "h" after guessing "t."

The general lower bound of the entropy of a source message with k symbols is given as [17]

$$\Gamma = \sum_{j=1}^{k}\sum_{i=1}^{N} i\log i(p_i^j - p_{i+1}^j). \qquad (27)$$

However, calculating Γ is not easy because of the dependence of a symbol on previous symbols. Consequently, for ease of calculation we try to obtain a bound that approaches Γ from below. To obtain this bound we first derive the following equation:

$$k\sum_{i=1}^{N} i\log i(p_i^k - p_{i+1}^k) \leq \Gamma \leq k\sum_{i=1}^{N} i\log i(p_i^1 - p_{i+1}^1). \qquad (28)$$

Equation (28) is obtained from the fact that Γ cannot be smaller than the entropy calculated for the source message by substituting the entropy of the last symbol (the last symbol's entropy, given the knowledge of the previous symbols is very low) in place of every source symbol; Γ also cannot be larger than the entropy calculated by substituting all symbols with the first symbol in the source.

It is easy to see that in Scenario 1 (Subsection 5.1), the lower bound entropy of the source message is at least as high as the l.h.s. of (28). This is especially true as URL addresses tend to also have symbols other than the English alphabet and sometimes contain incomplete words or meaningless strings, which would increase their randomness. Hence, we use the l.h.s. of Equation (28) to approximate the entropy of the source message, hence Equation (15) now becomes,

$$\begin{aligned} I(M^k;\underline{Z}) &\leq k\sum_{i=1}^{N} i\log i(p_i^k - p_{i+1}^k) \\ &\quad -(N-1+2N-3/2\log N) \qquad (29) \\ &\leq \left(kN^2\log N\sum_{i=1}^{N}(p_i^k - p_{i+1}^k)\right) \\ &\quad -(3N-3/2\log N-1). \qquad (30) \end{aligned}$$

Equating the r.h.s. of inequality (30) to zero, Equation (31) presents the condition for perfect secrecy,

$$k \leq \frac{3N-3/2\log N-1}{N^2\log N\sum\limits_{i=1}^{N}(p_i^k - p_{i+1}^k)}. \qquad (31)$$

Similarly, the perfect secrecy threshold for Scenario 2 is:

$$k \leq \frac{N-1}{N^2\log N\sum\limits_{i=1}^{N}(p_i^k - p_{i+1}^k)} \qquad (32)$$

and Scenario 3 is

$$k \leq \frac{2N-3/2\log N}{N^2\log N\sum\limits_{i=1}^{N}(p_i^k - p_{i+1}^k)}. \qquad (33)$$

We note that in the dependent source case, the bound for k is dependent on the inter-symbols dependency, which is intrinsic to each message. Hence it is difficult to derive something similar to Table 1. However, in both set-ups (independent/dependent sources) an important follow-up question is, what happens when k is greater than the corresponding bounds? Then the secrecy is no longer information-theoretically perfect. Two choices exist at that point.

When k equals the bound the user and the anonymizer can use another Huffman table to continue perfectly secure communication. They can use a synchronized protocol where before k reaches the bound the anonymizer can piggyback a new encrypted Huffman table (a small overhead) with the data. Or, in the interest of speed and low overhead, the user can choose to keep using the current table and risk leaking information. In this latter case, an eavesdropper can utilize the information leakage to mount efficient brute-force or cryptanalysis attacks to break the framework. In the next section, we analyze the feasibility of such attacks.

6. COMPUTATIONAL SECRECY AND BREAKABILITY ANALYSES

In this section, we investigate and analyze the security of our framework from the perspective of well-known attacks. As per proofs in the literature [9, 10], our framework is secure against known plaintext attacks. Also, it is secure against the chosen plaintext attack as the eavesdropper cannot get the user/anonymizer to encrypt a chosen plaintext using the corresponding Huffman table. In the chosen ciphertext attack, the attacker needs to obtain the decryption of its selected ciphertext. This is not possible in our framework as the anonymizer is the only entity with decoding capability. But, the anonymizer does not publish the decoded interest. The use of independent Huffman tables for users, selected uniformly at random, prevents the information leakage of one user from affecting others. This uniform selection of coding tables also prevents users to be able to correlate their coding tables with those of others to decode their encoded interests. Ciphertext-only attack can be mounted as the attacker has access to a set of ciphertexts, (encoded interests). If the user continues to use the same Huffman table, then the repeated interests will have the same encoded names, leaking information that the eavesdropper can use to make the cipher-text attack more potent. This leakage can be prevented by XOR-ing the postfix with a nonce and sending the nonce appended with the encoded URL.

This leaves two attacks that can be orchestrated by an attacker (active/passive): Correctly guessing the source message from the encoded sequence, i.e., ciphertext-only attack, or using brute-force to identify the correct key and the tree structure.

Correctly guessing the source message: We use *guessing entropy* [15], which is the expected number of guesses required by an attacker to ascertain the correct source message, to calculate the ease of guessing the source message. Let $G(M^k|\underline{Z})$ be the number of guesses required to identify M^k given $\underline{Z}$, in a way that $E[G(M^k|\underline{Z})]$, the expected number of successive attempts, is minimized. Equation (34) evaluates the corresponding $E[G(M^k|\underline{Z})]$:

$$E[G(M^k|\underline{Z})] = \sum_{z\in\underline{Z}} P_{\underline{Z}}(z)E[G(M^k|z)], \qquad (34)$$

where $P_{\underline{Z}}(z)$ is the probability of selecting an encoded binary sequence from the pool of all possible binary sequences. Using the results in [18], the guessing entropy is lower bounded by the conditional entropy as

$$E[G(M^k|\underline{Z})] \geq 2^{H(M^k|\underline{Z})-2}+1. \qquad (35)$$

Now we evaluate the lower bound on the guessing entropy in the three scenarios. Substituting Equation (15) in Equation (35), we have the lower bound guessing entropy for Scenario 1 in Section 5 as:

$$E[G(M^k|\underline{Z})] \geq 2^{\left(3N-3/2\log(N)-3\right)}+1. \qquad (36)$$

Similarly, by substituting Equations (19) and (23) in Equation (35), we have $E[G(M^k|\underline{Z})] \geq 2^{\left(2N-\frac{3}{2}\log(N)\right)-2}+1$ and $E[G(M^k|\underline{Z})] \geq 2^{(N-1)-2}+1$ for Scenarios 2 and 3 respectively.

Figure 3 illustrates the lower bound of the guessing entropy under the three scenarios described above, with different source alphabet size. As the conditional entropy of an unknown structure is higher than that of an unknown key, it is obvious that the attacker can extract the source message with a fewer number of guesses in Scenario 3 compared to Scenario 2.

Using a brute-force approach to identify the key and the tree structure: As mentioned in Section 3, there are $2^{(N-1)}$ mutation trees for each distribution of the Huffman code. Considering an N symbols alphabet, there exists

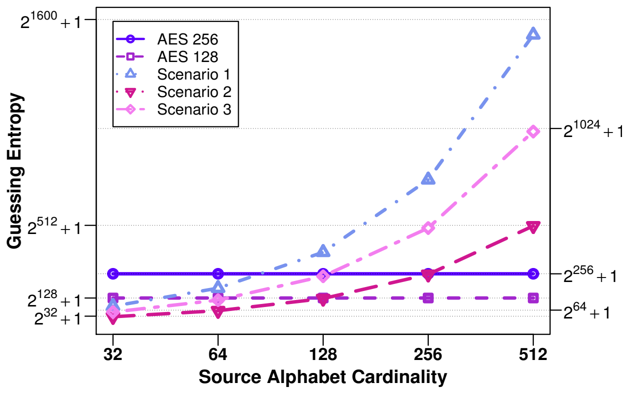


Figure 3: Guessing entropy for the three scenarios.

$C_{N'} = \frac{(2N'!)}{(N'+1)!\times(N'!)}$ different Huffman trees where $N' = N-1$. Each of these Huffman trees has $2^{(N-1)}$ mutation trees. For brute-force attack to identify the mutation tree, an attacker needs to compute on average $\frac{2^{(N-1)}}{2} = 2^{(N-2)}$ different mutation trees–exponential in N. Given this, the attacker has to use $\frac{2(N-1)!}{N!\times(N-1)!} \times 2^{(N-2)}$ different Huffman coding tables on average to decode the encoded message when attempting the brute force attack. Even with $N = 256$ (extended ASCII) it is computationally difficult to examine this search space at a filtering router; even when it is performed offline.

So far, we have proved the information-theoretic secrecy and the computational security of our framework. In the next section, we present our experimental results, which **answer the next question**: *how efficient, applicable, and scalable is the framework for real-world mobile devices?*

7. IMPLEMENTATION AND PERFORMANCE ANALYSES

For our experimental evaluation, we have clients requesting content over the network to a CCN media server (content provider), which is also the anonymizer. Our testbed consists of 18 nodes, eight desktops, six laptops, and four smartphones (3 Nexus 4 and one Nexus 5). We have created a 4-tiered line topology network connected using switches and IPv4 routers. For the experiments our clients and the anonymizer are placed on either ends of the line—requests travel over five hops. The client, server, and the nodes in the network employ the CCNx-0.7 [14] code base; our framework is written in C and is integrated into CCNx. The nodes route packets using longest prefix matching.

For fair comparison, we have disabled caching, so an interest passes through all 4 tiers. We compare latency and content retrieval time over four different scenarios, namely the vanilla (Baseline) CCN implementation, CCN with our anti-censorship framework (CCN+Huffman), data retrieval using FTP, and using Tor, the state of the art Internet anti-censorship tool. We also compare the overhead of our framework and Tor over their respective baselines, Baseline CCN and FTP, respectively. For testing Tor, we setup our testbed as a Tor network where the first three network gateways (from client towards the server) are provisioned as Tor proxies—three onion layers of symmetric encryption for the client. All our results were averaged over 100 runs. The size of the contents in our experiments were chosen from the set {1 MB, 10 MB, 100 MB, 500 MB}.

One option for encoding the data name (or the postfix after the domain name) is to use a strong symmetric key algorithm, such as AES. In Table 2, we compare the time taken for encryption/decryption by two widely used AES versions and for encoding/decoding using our framework. As an alternative solution, the client can hash the content name with a salt given by the anonymizer. The anonymizer needs to pre-hash all the content names with each salt corresponding to each client. Upon receiving an interest from a particular client, the anonymizer does a look-up on the hashed content name to find the requested content. Though the storage requirement for these hashes grows infeasible with a large number of clients and/or contents, we nonetheless evaluate the performance of the Openssl SHA1 digest for the sake of comparison.

We measured the timings on a wired laptop (AMD Turion, 2.4 GHz, dual core, 2.7 GB RAM) and on two wireless Nexus 5 smartphones (Krait 2.3 GHz, quad core, 2 GB RAM). For the Huffman operations, indicated by "Huffman coding" in Table 2, the time includes reading the source symbol frequencies, building the tree, and encoding/decoding the codewords. While the "Huffman*" represents the elapsed time only for the encoding/decoding operations. The represented time for AES accounts for the encryption and the decryption operations only. The data name in our test contained 75 characters.

The optimized OpenSSL AES version is almost four times as fast as the aescrypt version; we note that this is the version we use in our Tor experiments. Encoding/decoding in our framework (Huffman*) is three orders of magnitude faster than OpenSSL (0.000034 vs. 0.010).

Table 2: Running time comparison between the AES symmetric key cryptography and Huffman encoding.

Encoding Scheme	Encoding (s)	Decoding (s)
aescrypt in unix (laptop)	0.050	0.021
AES openssl (laptop)	0.010	0.008
Huffman coding (laptop)	0.004	0.004
AES openssl (Nexus 5)	0.041	0.023
Huffman coding (Nexus 5)	0.006	0.005
Huffman* (laptop)	0.000034	0.000027
SHA1 hashing (laptop)	0.000093	

Figure 4 shows the download times on the laptop client for different content sizes and compares Baseline CCN (*denoted as C*), CCN+Huffman (*denoted as H*), FTP (*denoted as F*), and Tor (*denoted as T*); the Y-axis is in **log scale** for clarity. It is easy to understand that the download time increases for all approaches with increase in content size. CCN performs worse than FTP on account of the overheads of multiple searches on each forwarding node: searching the cache (content store) and searching the pending interests table (PIT) for the interest. This is true even if the data item is not in the cache. Having said that, *our framework on top of CCN does not add* any appreciable extra delay (less than 1.8 times). In comparison, **Tor results in significant additional overhead** in comparison to FTP, (between 2 and 38 times). Of course, the overhead for Tor is on account of the multi-layer encryption. We have not compared our approach to ANDaNA [7] as for content sizes 10 MB and greater Tor performs better than ANDaNA; most of our tests are for such larger contents. For completeness, in the future we will compare our approach with ANDaNA. We will implement the standard ANDaNA protocol on our testbed for compar-

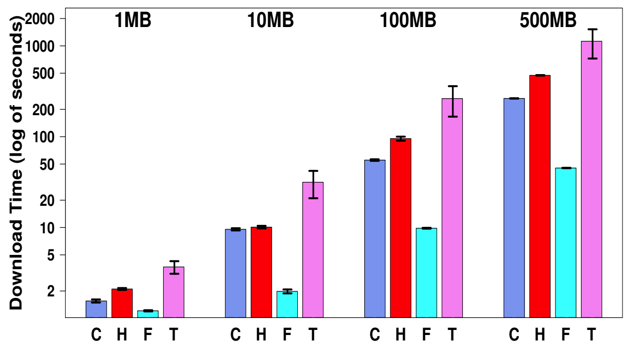


Figure 4: Average download time comparisons between Baseline CCN (C), Huffman (H), FTP (F), and Tor (T).

ison.

Figure 5 presents the results of a detailed comparison of the overhead ratio of our framework over the Baseline CCN with that of Tor over FTP. An interesting observation is that the error-bars for the overhead for Tor are much higher compared to those of our framework. Given that our tests were the only applications running in our testbed, it shows that the multiple levels of encryptions in Tor results in erratic behavior (Tor does not induce inter-packet delays for anonymity), which is undesirable in terms of user experience.

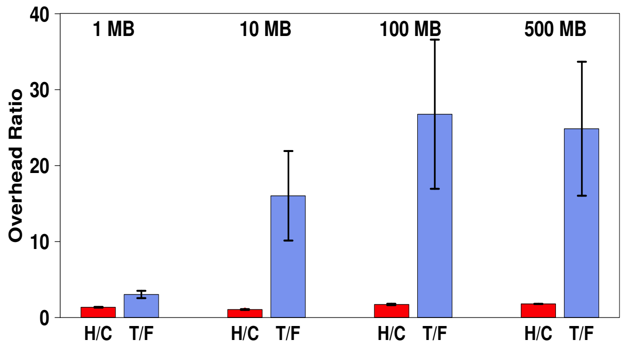


Figure 5: Protocol overhead comparison between Huffman (H/C) and Tor (T/F).

This fact is also highlighted by the estimated round trip time (RTT) results in Figure 6. *We have not shown the error-bars for the RTT because Tor's RTT varies a lot and undermines the graph's readability.* Baseline CCN and our approach have almost the same RTT, with very small deviations, however Tor has significant deviations. Note that the longer RTT values for the CCN and our approach in comparison to FTP are due to the extra lookups.

Figures 7, 8, and 9 present averaged results for the smartphone clients (Nexus 5) connected over WiFi. For conciseness, we only present the CCN and our CCN+Huffman results. Figure 7 presents the download times for the smartphones. Due to the wireless connections, which suffer from interference from other communications in the building, the smartphone clients require proportionally larger download times for all contents than the wired laptops.

Figure 8 presents the overhead of the Huffman framework over Baseline CCN. Again the overheads are minimal and with very low variance, which is excellent. Figure 9 presents the comparison of the estimated RTTs between the Baseline CCN and our framework for the smartphone client. The RTT values are very consistent for all runs, with our framework having a very small increase in RTT in comparison to the baseline. The smartphone client results conform with those of the laptop, the variance is a bit higher because of the wireless medium and interference. The average RTT stabilizes for large contents, hence the RTT values tend to become lower. We believe that the smartphone Tor RTTs will also have commensurate trends as the laptop results.

These results show that our framework is much more efficient and scalable than Tor, the state of the art, as a mechanism to mitigate censorship of user communications.

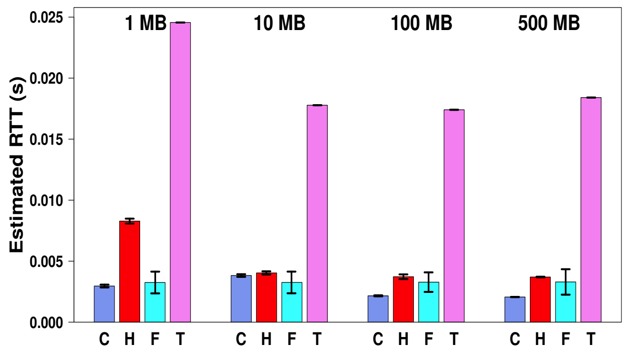


Figure 6: Estimated average round trip time for C, H, F, and T.

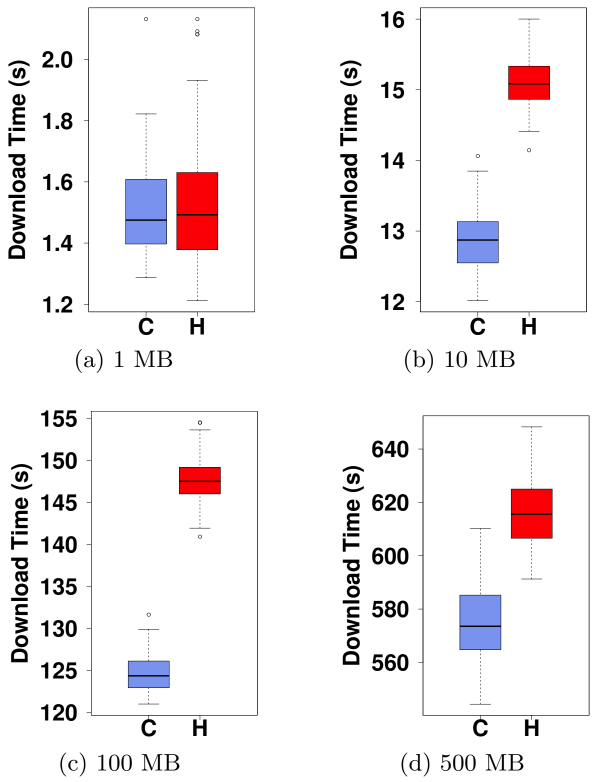


Figure 7: Download time comparison between Baseline CCN (C) and Huffman (H) for the smartphone clients.

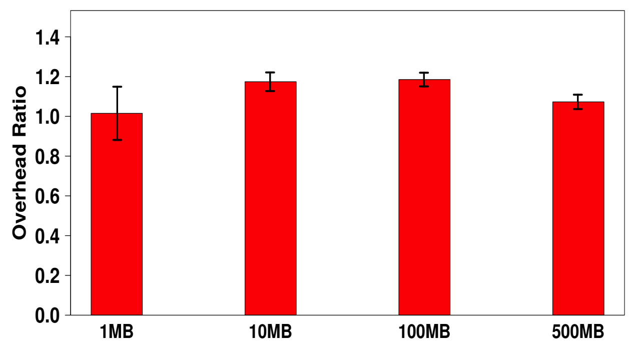


Figure 8: Protocol overhead of Huffman for the smartphone clients.

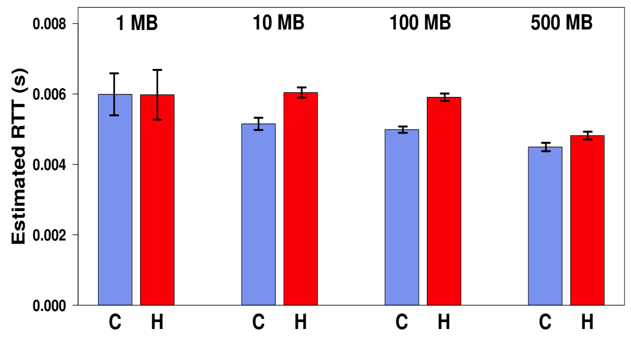


Figure 9: Estimated average RTT for baseline CCN (C) and Huffman (H) on the smartphone clients.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a lightweight anti-censorship framework for ICN users, specifically for mobile users. We proved conditions and thresholds for perfect secrecy as well as analyzed the computational complexity of the framework. The framework's breakability study showed the advantages of Huffman coding over AES, and the extensive experimental results demonstrated the efficiency of the framework in comparison to other frameworks, such as Tor.

In the future, we plan to implement our framework in other ICN architectures. We will analyze the trade-off between the users' privacy and caching by using an intermediate anonymizer. We will also investigate the use of hash functions based lookup at the anonymizer.

9. REFERENCES

[1] PSIRP Project, Deliverable D4.6: Final Evaluation Report on Deployment Incentives and Business Models, http://www.psirp.org/publications/.

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

[3] Anonymizer Universal. https://anonymizer.com/.

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

[5] A. Chaabane, E. De Cristofaro, M. Kaafar, and E. Uzun. Privacy in content-oriented networking: Threats and countermeasures. *ACM SIGCOMM Computer Communication Review*, 43(3):25–33, 2013.

[6] C. Dannewitz. NetInf: An information-centric design for the future Internet. In *3rd GI/ITG KuVS Workshop on The Future Internet*, 2009.

[7] S. DiBenedetto, P. Gasti, G. Tsudik, and E. Uzun. Andana: Anonymous named data networking application. *Arxiv preprint arXiv:1112.2205*, 2011.

[8] H. Dorrie. *100 Great problems of elementary mathematics*. Dover Publications, 1965.

[9] A. S. Fraenkel and S. T. Klein. Complexity aspects of guessing prefix codes. *Algorithmica*, 12(4-5):409–419, 1994.

[10] D. Gillman, M. Mohtashemi, and R. Rivest. On breaking a huffman code. *IEEE Transactions on Information Theory*, 42(3):972–976, 1996.

[11] D. Huffman et al. A method for the construction of minimum redundancy codes. *proc. IRE*, 40(9):1098–1101, 1952.

[12] V. Jacobson, D.K. Smetters, J.D. Thornton, M.F. Plass, N.H. Briggs, and R.L. Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 1–12. ACM, 2009.

[13] T. Koponen, M. Chawla, B.G. Chun, A. Ermolinskiy, K.H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 181–192. ACM, 2007.

[14] Palo Alto Research Lab. Ccnx. http://www.ccnx.org/.

[15] J. L. Massey. Guessing and entropy. In *Proceedings to IEEE International Symposium on Information Theory*, page 204. IEEE, 1994.

[16] PURSUIT Project. http://www.fp7-pursuit.eu/pursuitweb.

[17] C. Shannon. Prediction and entropy of printed english. *Bell system technical journal*, 30(1):50–64, 1951.

[18] G. Smith. On the foundations of quantitative information flow. In *Foundations of Software Science and Computational Structures*, pages 288–302. Springer, 2009.

[19] Tor Project: Anonymity Online. http://www.torproject.org/.

[20] C.-P. Wu and C.-C. Kuo. Design of integrated multimedia compression and encryption systems. *IEEE Transactions on Multimedia*, 7(5):828–839, 2005.

[21] D. Xie and C.-C. Kuo. nhanced multiple huffman table (mht) encryption scheme using key hopping. In *Proceedings of the 2004 International Symposium on Circuits and Systems (ISCAS)*, volume 5, pages 568–571. IEEE, 2004.

[22] J. Zhou, Z. Liang, Y. Chen, and Oscar C. Au. Security analysis of multimedia encryption schemes based on multiple huffman table. *Signal Processing Letters, IEEE*, 14(3):201–204, 2007.