Tunnels to Towers: Secure Virtual Private Networking for CCN

***Abstract*—**
***Index Terms*—**

I. Introduction

Content-centric networking (CCN) is a type of request-based information-centric networking (ICN) architecture. In CCN, all data is named. Consumers obtain data by issuing an explicit request for the content by its name. The network is responsible for forwarding this request towards producers, based on the name, who then generate and return the content response. Since a name uniquely identifies a content response, routers may cache these packets to use in response to future requests for the same name. As a consequence, all content has an (implicit or explicit) authenticator that is used to verify the name-to-data binding. In order to prevent cache poisoning attachs, wherein a malicious producer supplies fake data in a content response that is propogated in the network, a router should never serve (a) content with an invalid authenticator or (b) cached content that it cannot verify. To enable (b), content objects with a digital signature are expected to carry the public verification key or certificate. If the authenticator is a MAC, then intermediate routers cannot verify it and should therefore not cache the content.

One negative side effect of name-based requests is that any on-path or eavesdropping adversary between a consumer and producer can learn the identity and contents of all data in transit. In traditional IP-based networks, there are generally two types of mechanisms to solve this problem: (1) anonymity networks such as Tor [**?**] or (2) VPNs. As tools focused on anonymity, the former help prevent linkability of packets to their requestors without always protecting the identities or content themselves. In contrast, VPNs focus on packet confidentiality by creating a tunnel between two private networks or a consumer and single private network. All traffic over this tunnel is encrypted and thereby opaque to an eavesdropper. VPNs differ from anonymity networks such as Tor in that they are *network-layer* mechanisms that typically only introduces a single layer of encryption to protect traffic. Thus, while Tor can be used to enable VPN-like functionality, it is often far more inefficient since it operates above the network layer.

In ICN (or more specifically, CCN and NDN), ANDaNA was the first anonymity network of its kind. Similar to Tor, ANDaNA uses circuits formed from anonymizing routers (ARs) to marshall requests and responses between consumers and producers. The former onion-encrypt interests and content using the public key(s) of the target ARs. A variant of ANDaNA uses symmetric keys for packet encapsulation but suffers from linkability. Tsudik et al. [**?**] proposed an optimized version of the symmetric-key ANDaNA variant that did not permit linkability. To the best of our knowledge, there is no anonymity network variant for ICN architectures. Though tunneling is only useful for only a subset of ICN traffic, we believe it is a gap to be addressed for this emerging technology, for a variety of reasons. **First**, privacy continues to be an elusive property for CCN applications. Tunneling will help permit some degree of privacy within trusted AS domains from external passive eavesdroppers. **Second**, multi-hop circuits as used in ANDaNA are overkill when trying to retain privacy instead of anonymity. **Third**, end-to-end sessions such as those enabled by CCNxKE [1] and similar protocols only serve those engaged in the session. In contrast, since the threat model is different, tunneled traffic has the potential to serve any number of consumers within the same trusted domain. Thus, while tunneling may contrast the content-centric nature of data transmission in CCN, it fills a needed void for this architecture.

In this paper, we present CCVPN, a secure tunneling protocol and system design for CCN. Similar to ANDaNA, CCVPN encrypts interests and content objects between producers and consumers. In contrast, CCVPN

II. Preliminaries

This section presents an overview of the CCN architecture[1] and work related to confidentiality, privacy, and transport security. Those familiar with these topics can skip it without loss of continuity.

A. CCN Overview

In contrast to IP networks, which focus on end-host names and addresses, CCN [2], [3] centers on content by making it named, addressable, and routable within the network. A content name is a URI-like string composed of one or more variable-length name segments, each separated by a '/' character. To obtain content, a user (consumer) issues a request, called an *interest* message, with the name of the desired content. This interest can be *satisfied* by either (1) a router cache or (2) the content producer. A *content object* message is returned to the consumer upon satisfaction of the interest. Moreover, name matching in CCN is exact, e.g., an interest for /edu/uci/ics/cs/fileA can only be satisfied by a content object named /edu/uci/ics/cs/fileA.

[1]Named-Data Networking [2] is an ICN architecture related to CCN. However, since CCNxKE was designed for ICNs that have features which are not supported by NDN (such as exact name matching), we do not focus on NDN in this work. However, CCNx could be retrofitted to work for NDN as well.

In addition to a payload, content objects include several fields. In this work, we are only interested in the following three: `Name`, `Validation`, and `ExpiryTime`. The `Validation` field is a composite of (1) validation algorithm information (e.g., the signature algorithm used, its parameters, and a link to the public verification key), and (2) validation payload (e.g., the signature). We use the term "signature" to refer to this field. `ExpiryTime` is an optional, producer-recommended duration for the content objects to be cached. Conversely, interest messages carry a mandatory name, optional payload, and other fields that restrict the content object response. The reader is encouraged to review [3] for a complete description of all packet fields and their semantics.

Packets are moved in the network by routers or forwarders. A forwarder is composed of at least the following two components:

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

A forwarder may also maintain an optional *Content Store* (CS) used for content caching. The timeout for cached content is specified in the `ExpiryTime` field of the content header. From here on, we use the terms *CS* and *cache* interchangeably.

Forwarders use the FIB to move interests from consumers towards producers and the PIT to forward content object messages along the reverse path towards consumers. More specifically, upon receiving an interest, a router R first checks its cache (if present) to see if it can satisfy this interest locally. If the content is not in the cache, R then consults the PIT to search for an outstanding version of the same interest. If there is a PIT match, the new incoming interface is added to the PIT entry. Otherwise, R forwards the interest to the next hop according to its FIB (if possible). For each forwarded interest, R stores some amount of state information in the PIT, including the name of the interest and the interface from which it arrived, so that content may be sent back to the consumer. When content is returned, R forwards it to all interfaces listed in the matching PIT entry and said entry is removed. If a router receives a content object without a matching PIT entry, the message is deemed unsolicited and subsequently discarded.

III. Related Work

- [4]
- [5]
- ...

Content-based encryption is arguably the most popular technique for protecting CCN content from unauthorized disclosure. This technique permits content to be disseminated throughout the network since it cannot be decrypted by adversaries without the appropriate decryption key(s). Many variations of this approach have been proposed based on general group-based encryption [6], broadcast encryption [7], [8] and proxy re-encryption [9]. Kurihara et al. [10] generalized these specialized approaches in a framework called CCN-AC, an encryption-based access control framework that shows how to use manifests to explicitly specify and enforce other encryption-based access control policies. Consumers use information in the manifest to (1) request appropriate decryption keys and (2) use them to decrypt content object(s). The NDN NBAC [11] scheme is similar to [10] in that it allows decryption keys to be flexibly specified by a data owner. However, it does this based on name engineering rules instead of configuration. Interest-based access control [12] is a different type of access control scheme wherein content was optionally encrypted. Access was protected by making the names of content derivable by only authorized consumers. NDN-ACE [13] is a recent access control framework for IoT environments which includes a key exchange protocol for distributing secret keys to sensors. We revisit NDN-ACE in Section **??**.

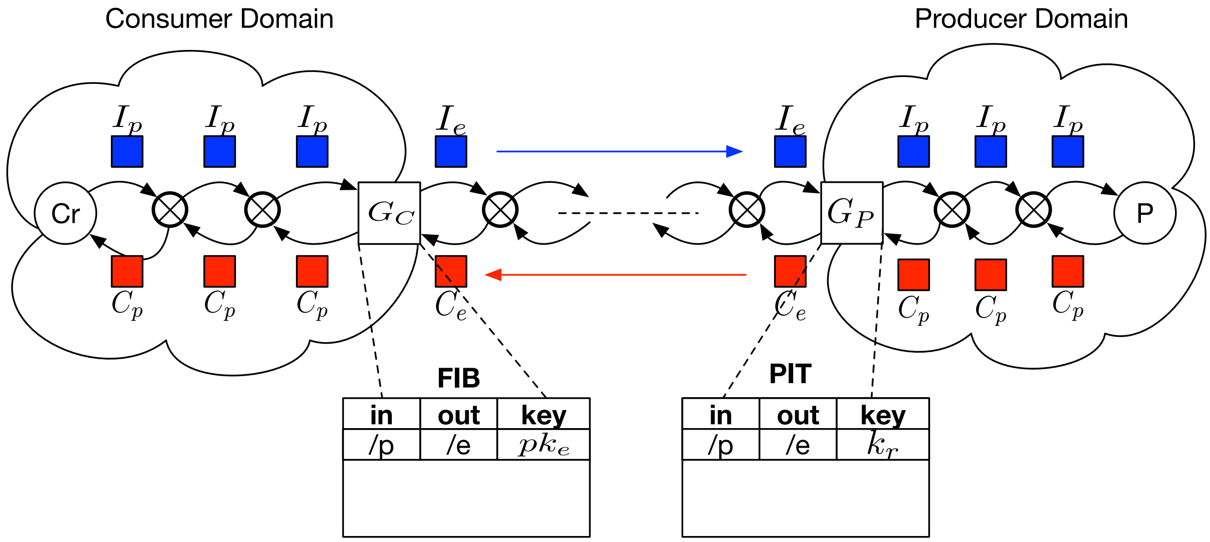
IV. CCVPN

Traditional Virtual Private Networks (VPNs) extend private networks across the Internet. They enable users to send and receive data across shared or public networks as if their computing devices were directly connected to the same private network [14]. The goal of CCVPN it to provide its users with the same functionality within the CCN Internet architecture. Therefore, the users can benefit from the features and security of a private network, even though the are not physically under the same private network.

In CCVPN, there are four main entities involved in the virtual private communication: *Consumer*, *Producer*, *Consumer Side Gateway* (G_c), and *Producer Side Gateway* (G_p). As in usual CCNs, the *Consumer* is the network node that issues an interest for a given content (e.g., file, web page, video) that it wishes to retrieve. The Producer is the network node which originally created such content. G_c and G_p are the edge gateways responsible for ensuring the private communication among distinct domains. As we discuss later on, G_c and G_p can actually be implemented as a single network device, but we firstly present them as separate entities for the purpose of clarity.

As depicted in Fig. 1, the devices inside the *Consumer* domain form a physically interconnected private network. Conversely, the devices in the *Producer* domain also form a private network. Therefore, the goal is to create an overlay virtual network that unifies the *Consumer* and the *Producer* domains in way such that the original interests and contents are only visible to the devices inside these two domains. In other words, the interest I_e and the content C_e, that are forwarded outside these domains, should give no information about the original interest I_p and the actual content C_p.

In order to achieve such anonymous communication, G_c is introduced in the *Consumer* domain to encapsulate outgoing interests. Conversely, the G_p is responsible for decapsulating the incoming encapsulated interests and forwarding them in their original form. When the content is forwarded back in response to the interest, G_p encrypts the content before



in	out	key
/p	/e	pk_e

in	out	key
/p	/e	k_r

Fig. 1. CCVPN connectivity architecture

sending it outside the *Producer* domain. Finally, G_c decrypts the received content to its original form and forwards it back towards the *Consumer*. Through the rest of this section we provide a detailed description on how such actions are implemented.

Upon the arrival of a new interest I_p, G_c checks its Forwarding Information Base (FIB) to check if the interest prefix is in the list of prefixes for VPN communication. We assume that the FIB must be pre-configured with the list of prefixes which will trigger the VPN communication. Associated with such prefixes are G_p's name and public key (pk_e). Therefore, if the prefix of I_p is in the VPN prefixes list, G_P runs Algorithm 1 to generate a new interest I_e which encapsulates the original interest I_p. Firstly, Algorithm 1 generates a random symmetric key (k_r) which will be used later on to perform the *Content* Encryption/Decryption. Next, it retrieves G_p's name and public key from the FIB. It uses the public key to encrypt the symmetric key k_r and the original interest I_p. Then it creates the new interest I_e with G_p's name as the interest name and the generated ciphertext as payload. Since I_e has the G_p's name it will be routed towards G_p and since the payload is singed with G_p's public key, only G_p can retrieve the original interest I_p and the symmetric key k_r.

input : Original interest I_p;
output : Encapsulated interest I_e;
k_r = symmKeyGen();
Gp_{name} = retrieveNameFromFIB(I_p)
pk_e = retrievePKFromFIB(I_p)
$payload$ = $Enc_{pk_e}(I_p||k_r)$
I_e = createNewInterest(Gp_{name}, $payload$)
storeToPIT(I_e,k_r)
return I_e;

Algorithm 1: Interest encapsulation (runs on G_c)

After the message is routed towards G_p, G_p verifies if the incoming interest name prefix matches its own name and, if it does, G_p runs Algorithm 2, using its own secret key (sk_e) to decrypt I_e's payload, which results in the original interest I_p and the symmetric key k_r. G_p then stores I_e's name and the symmetric key k_r in its own PIT, within the entry for the pending interest I_p, and forwards I_p. I_e's name and k_r are stored so that they can be used later on to generate the encrypted content C_e.

input : Encapsulated interest I_e;
input : Private key sk_e;
output : Original interest I_p;
Ie_{name} = getName(I_e)
$cipherText$ = getPayload(I_e)
$I_p||k_r$ = $Dec_{sk_e}(cipherText)$
storeToPIT(I_p,k_r,Ie_{name})
return I_p;

Algorithm 2: Interest decapsulation (runs on G_p)

The original interest I_p is forwarded inside the *Producer* domain until it reaches the *Producer*. The *Producer* responds with the content C_p which is forwarded back to G_p. Upon receiving C_p, G_p fetches for C_p's name (which is equal to I_p's name) on its PIT, retrieving k_r and Ie's name. Then it uses k_r to encrypt-then-MAC the real content response C_p and creates C_e which must have the same name as I_e and the encryption of C_p as payload (Algorithm 3). Since only G_c and G_p share the symmetric key k_r, only G_c will be able to decrypt C_e's payload into C_p. Therefore nobody from outside the VPN is able to access the content nor the Producer's identity.

Since C_e and I_e have the same name, C_e will be forwarded all the way back to G_c. When G_c receives C_e G_c will execute Algorithm 4. It will match C_e's name to the pending interest

input : Original content C_p;
output : Encrypted content C_e;
$name$ = getName(C_p)
k_r = retrieveKeyFromPIT($name$)
Ie_{name} = retrieveNameFromPIT($name$)
$payload$ = EncryptThenMAC(k_r,C_p)
C_e = createNewContent(Ie_{name},$payload$)
return C_e;

Algorithm 3: Content encryption (runs on G_p)

I_e in its PIT, retrieving k_r. k_r can then be used to verify the integrity of the received content and to decrypt it into the actual content C_p. After that, C_p can be forwarded back to the *Consumer*. If the MAC verification fails, it means that C_e has been forged and G_c ignores it.

input : Encrypted content C_e;
output : Original content C_p;
Ce_{name} = getName(C_e)
k_r = retrieveKeyFromPIT(Ce_{name})
$cipherText$ = getPayload(C_e)
C_p = Dec(k_r,$cipherText$)
if C_p == $\perp$ **then**
  /* MAC verification failed */
  return;
else
  return C_p;
end

Algorithm 4: Content decryption (runs on G_c)

For clarity, we have defined a *Consumer* and a *Producer* domain. However, in reality, a single gateway can implement the functions of both G_c and G_p. Therefore, consumers and producers can exist in both the domains and interests for contents can be issued from both sides. Also, it is worth to mention that, within the created CCVPNs, content caching would work just as it works in regular CCNs, i.e., routers would be able to cache contents and respond to interests that were previously requested, enabling better resource usage and lower communication delays. Finally, we emphasize that G_c encapsulation and decryption functions can also run inside the Consumer host. Conversely, G_p can be implemented within the *Producer* host. This enables its usage for one-to-one communication that would be completely anonymous to any other entity in the network.

V. Discussion and Analysis

In this section we analyze and discuss the overhead of the CCVPN design with respect to the additional processing time and state consumption needed to handle traffic.

A. State Consumption

The CCVPN design has an immediate impact on the FIB and PIT size of a gateway. (The content store size remains unaffected since only decapsulated content objects are ever cached.) Let F_S be the total size of a standard forwarder FIB in terms of bytes and N_F be the number of entries in the FIB. For simplicity, we will assume that each name prefix in the FIB has a constant size of 64B. In practice we expect this to be a comfortable upper bound. Thus, $F_S = N_F s$, where s is the size of each FIB entry. Here, s includes a name prefix (of size 64B) and a bit vector that identifies the matching links for the interface. We assume that a gateway has 128 links which, again, is a comfortable upper bound. Therefore, $F_S = 80N_F$B. Now consider the FIB size F_G for a CCVPN gateway. Some entries in these FIBs will point to "private" prefixes, i.e., other domains, and therefore have a larger size to account for the corresponding prefix and key material that must be stored. For both public- and symmetric-key encryption, the key size is the same: 32B [**?**]. Therefore, by taking into account two both the FIB entry prefix key, translation prefix, encryption key, and corresponding bit vector, the total size of one "private" FIB entry will be 176B, meaning that $F_G = 176N_F$B. By comparing F_S to F_G, we see that, in the worst case, the CCVPN FIB is at most $F_G/F_S = 176/80 = 2.2$ times larger than the standard FIB. In practice, however, we expect this to be much smaller, since the fraction of public to private FIB entries in a gateway will be non-zero.

We will now apply the same analysis to the PIT size. A standard PIT entry includes a complete name and ingress bit vector. (They may also include the optional `KeyId` and `ContentId`, but since they are included in the gateway PIT as well we omit them from this analysis.) A gateway PIT entry will contain the same elements of a standard PIT entry but also a symmetric encryption key (32B), nonce (12B), and an encapsulation name (64B + 32B). The encapsulation name is the name of an encapsulated interest and includes an additional 32B `PayloadID` segment to identify the encapsulated value in the payload. Let P_S and P_G be the sizes of the standard and gateway PIT, respectively, and let N_P be the number of PIT entries in one such table. Based on the above discussion, and assuming again that a name is at most 64B, a standard PIT entry is of the size 80B. In contrast, a gateway PIT entry is of size 204B. Therefore, in the worst case, the CCVPN PIT will be at most $P_G/P_S = 204/80 = 2.55$B larger than the standard PIT. Assuming a steady state size of approximately $1e^5$ entries [15], this means that the PIT will be 20.4MB, which is well within the capacity of modern memory systems.

B. Processing Overhead

In terms of processing overhead, the gateway adds a number of new steps to the data path of a packet. The main computational burdens are packet encapsulation and decapsulation. In the public-key variant of CCVPN, interests are processed using public-key encryption, whereas content is always processed using symmetric-key encryption. Let $T_E^P(n)$ and $T_D^P(n)$ be the time to encrypt and decrypt nB of data using a suitable public-key encryption scheme. Similarly, let $T_E^S(n)$ and $T_D^S(n)$ be the time to encrypt and decrypt nB of data using a symmetric-key encryption scheme. Then, the latency in a single interest-content exchange is increased by $T = T_E^P(n_I) + T_D^P(n_I) + T_E^S(n_C) + T_D^S(n_C)$, where n_I and

n_C are the original interest and content sizes, respectively. As a rough estimate, [16] lists the cost of AES-GCM to be $2.946\mu s$ for setup followed by 102MiB/second Intel Core 2 1.83 GHz processor under Windows Vista in 32-bit mode (with AES ISA support). For packets that are at most 1500B, the total processing time is roughly $17\mu s$. Moreover, The public-key encryption and decryption operations will always be at least as expensive, so the total latency is increased by at least $T = 4 \times 17\mu s = 68\mu s$. In comparison to the network latency for a single packet this may not be noticable, but for a steady arrival state of approximatey $1e^5$, this would lead to an instable system that would quickly overflow. (This is because $65\mu s \times 1e^5 = 6.8s$.) Therefore, there is an upper bound on the number of private packets a gateway can process per second. This bound is entirely dependent on the system configuration and network conditions.

Another performance deficiency comes from the fact that gateways cannot process packets without allocating memory. Specifically, since each packet requires either an encryption or decryption, which cannot be done entirely in-place, the gateway must allocate some amount of memory for every processed packet. This overhead can outweigh the cryptographic computations if the packet arrival rate is high enough. Therefore, when implementing CCVPN, special care must be taken to ensure that all cryptographic operations are performed in-place where possible.

VI. Experiments

A. Experimental Methodology

In this section we empirically evaluate the CCVPN design paying special attention to the metrics that were earlier discussed in Sec. V, i.e., processing overhead, network throughput, and state consumption. In our evaluation we consider the two versions of CCVPN: public key version, and symmetric key version. We here recall that the symmetric key version relies on the assumption that a secure key agreement protocol is performed between the domain's gateways prior to the CCVPN protocol execution.

Our testbed network consists of a butterfly topology, in which the consumers' side and the producers' side gateways are directly interconnected. N producers are connected to the producers' domain gateway and M consumers are connected to the consumers' domain gateway (see Fig. **??**).

To investigate the processing overhead we measure the average time demand for computing the interests' encapsulation (in public and symmetric key versions), interest decapsulation, content encryption, and content decryption for different content packet sizes (1024, 4096, 16384, and 65536 bytes). We also measure the state consumption for these same four functions.

To compute the overall network throughput we measure the average data-rate for transmissions of 1 to 1,000,000 different interests issues per consumer. We also vary the number of consumers and producers from 1 to 10 of each. Finally, in addition to the network throughput, we also exhibit the total transmission delay for each of the experiments.

B. Results

TABLE I
Interest encapsulation processing times

Encapsulation mode	Encapsulation	Decapsulation
Public Key	$444\mu s$	$449\mu s$
Symmetric Key	TODO	TODO

TABLE II
Content encryption and decryption times for different payload sizes

Packet size	Encryption	Decryption
1024B	$125\mu s$	$193\mu s$
4096B	$141\mu s$	$220\mu s$
16384B	$220\mu s$	$367\mu s$
65536B	$519\mu s$	$702\mu s$

VII. Security Analysis

TODO

VIII. Conclusion

TODO

References

[1] M. Mosko, E. Uzun, and C. Wood, "CCNx Key Exchange Protocol Version 1.0," Tech. Rep. draft-wood-ccnxkeyexchange-01, May 2016. [Online]. Available: https://raw.githubusercontent.com/PARC/ccnx-keyexchange-rfc/master/draft-wood-icnrg-ccnxkeyexchange-01.txt

[2] V. Jacobson *et al.*, "Networking named content," in *CoNext*, 2009.

[3] M. Mosko, I. Solis, and C. Wood, "CCNx semantics," *IRTF Draft, Palo Alto Research Center, Inc*, 2016.

[4] G. Tsudik, E. Uzun, and C. A. Wood, "Ac3n: Anonymous communication in content-centric networking," in *2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2016, pp. 988–991.

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

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

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

[8] M. Ion, J. Zhang, and E. M. Schooler, "Toward content-centric privacy in ICN: Attribute-based encryption and routing," in *ICN*, 2013.

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

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

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

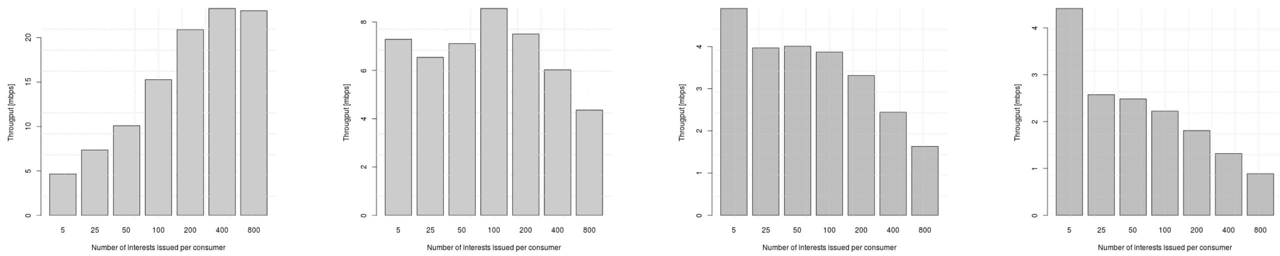
[12] C. Ghali, M. A. Schlosberg, G. Tsudik, and C. A. Wood, "Interest-based access control for content centric networks," in *International Conference on Information-Centric Networking*. ACM, 2015.

[13] W. Shang *et al.*, "Ndn-ace: Access control for constrained environments over named data networking."

[14] S. Khanvilkar and A. Khokhar, "Virtual private networks: an overview with performance evaluation," *IEEE Communications Magazine*, vol. 42, no. 10, pp. 146–154, 2004.

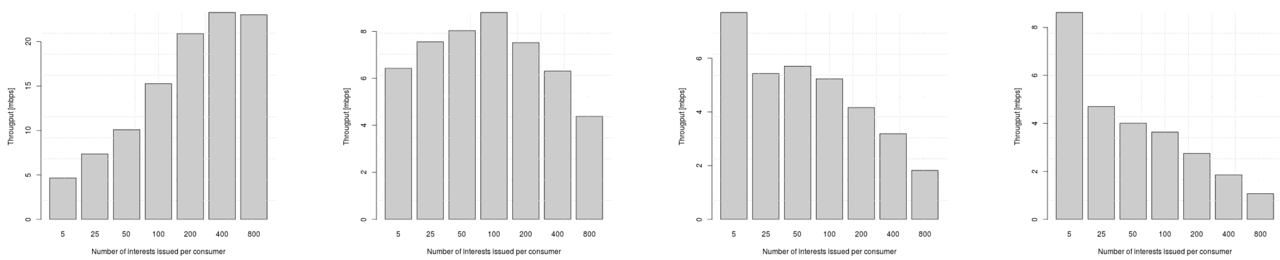
[15] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino, "Pending interest table sizing in named data networking," in *Proceedings of the 2nd International Conference on Information-Centric Networking*. ACM, 2015, pp. 49–58.

[16] "Crypto++ 5.6.0 Benchmarks," https://www.cryptopp.com/benchmarks.html, accessed: 2016-11-21.



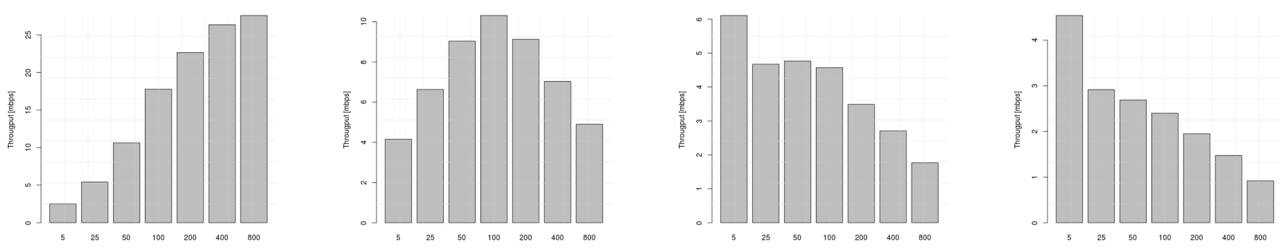
(a) 1 Consumer x 1 Producer (b) 2 Consumers x 1 Producer (c) 3 Consumers x 1 Producer (d) 4 Consumers x 1 Producer

Fig. 2. Throughput per consumer in public key mode. 1 Producer N consumers



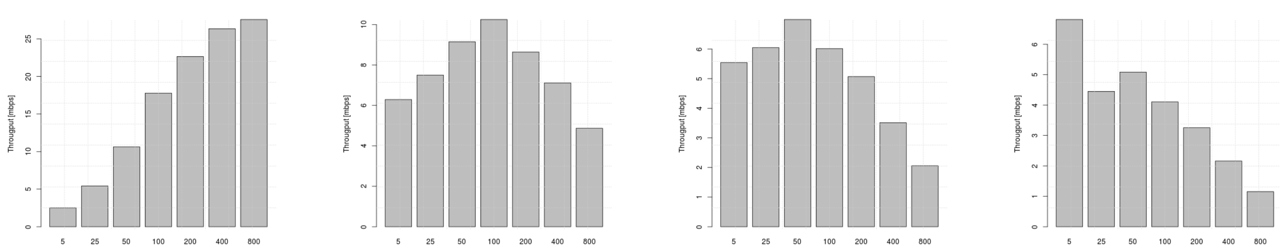
(a) 1 Consumer x 1 Producer (b) 2 Consumers x 2 Producers (c) 3 Consumers x 3 Producers (d) 4 Consumers x 4 Producers

Fig. 3. Throughput per consumer in public key mode. N Producers N consumers



(a) 1 Consumer x 1 Producer (b) 2 Consumers x 1 Producer (c) 3 Consumers x 1 Producer (d) 4 Consumers x 1 Producer

Fig. 4. Throughput per consumer in symmetric key mode. 1 Producer N consumers



(a) 1 Consumer x 1 Producer (b) 2 Consumers x 2 Producers (c) 3 Consumers x 3 Producers (d) 4 Consumers x 4 Producers

Fig. 5. Throughput per consumer in symmetric key mode. N Producers N consumers