Namespace Tunnels in Content-Centric Networks

Ivan O. Nunes, Gene Tsudik, Christopher A. Wood[+]
University of California Irvine
{ivanoliv, gene.tsudik, woodc1}@uci.edu

***Abstract*—Content-Centric Networking (CCN) is a candidate next-generation Internet architecture that offers an alternative to the current IP-based model. CCN emphasizes scalable and efficient content distribution by making content explicitly named and addressable. It also offers some appealing privacy features, such as lack of source and destination addresses in packets. However, to be considered a fully viable Internet architecture, CCN must support private and anonymous communication that is at least at parity with IP. Within this space, VPNs are a very popular tool that enable users to communicate across insecure public networks as if they were connected over a private network. They are also absent from the repertoire of CCN research. To fill this void, we design, implement, and evaluate CCVPN, a content-centric analog to IP-based VPNs in the current Internet architecture. To the best of our knowledge, CCVPN is the first such CCN-based design. Our design is functionally equivalent to IP-based VPNs while gaining better privacy due to the non-linkability of encapsulated packets to their originating network. We analyze the security of CCVPN and experimentally assess its performance.**

***Index Terms*—Content-Centric Networking, Network Security, Privacy, VPN, Tunneling**

I. Introduction

Content-Centric Networking (CCN) is a particular type of request-based Information-Centric Networking (ICN) architecture. In CCN, all content is explicitly named. A consumer obtains content by issuing an explicit request (called an *interest*) referencing the content name. The network is responsible for forwarding this interest towards the content producer, based on the name. By design, requests do not carry any originating source address. Therefore, each router must leave some per-packet state before forwarding an interest, so that responses can be returned correctly. Once the target content is found, it is forwarded along the same reverse-path towards the consumer, removing the corresponding state in each router along the way. Simultaneously, each router may opportunistically cache the content to satisfy future requests for the same data.

By default, CCN offers no privacy protection beyond the lack of source addresses; all packet data is in cleartext, unless encrypted at the application layer. Therefore, an adversary on the path between a consumer and the nearest location of content (producer or router) easily learns the requested name and the corresponding content. In the current IP-based Internet, there are generally two ways of attaining private communication: (1) anonymity networks and HTTP proxies, such as Tor [1] and [2], and (2) VPNs. The former offer anonymity, which is more than communication privacy – they inhibit linkability of multiple related packets and conceal identities of communicating end-points. In contrast, a VPN focuses on *confidentiality* by creating a secure tunnel between two private networks, one or both of which might be a single device. All traffic over this tunnel is encrypted and opaque to an eavesdropper. Unlike anonymity networks, such as Tor, VPN is a *network-layer* mechanism that typically introduces only a single layer of encryption to protect traffic. Thus, while Tor can be used to attain VPN-like functionality, it is often far less efficient since it operates above the network layer. Simply put: Tor is overkill if linkability is not a concern.

ANDaNA [3] was the first attempt to support anonymous communication in CCN. ANDaNA is basically a CCN-based Tor analog that uses circuits composed of anonymizing routers (ARs) to forward CCN packets between consumers and producers. A consumer picks a set of (at least two) ARs and concentrically encrypts an interest using the public key(s) of selected ARs. An encrypted interest traverses the sequence of ARs, shedding one layer of encryption at each AR hop. The last-hop AR forwards (actually, issues) the cleartext interest which is routed towards the producer. If and when this content is located, it traverses, in reverse, the same sequence of ARs, each of which adds a layer of encryption using a key selected by the consumer and distributed as part of the original interest processing. A more efficient variant of ANDaNA uses only symmetric keys; however, it does not offer unlinkability. Subsequently, an optimized version of the symmetric-key ANDaNA [4] without linkability was proposed.

ANDaNA was not designed for the simple VPN use-case of protecting traffic contents without hiding identities of communicating parties. Multihop circuits, as used in ANDaNA, are unnecessary when the goal is communication privacy, instead of anonymity. Furthermore, similar to Tor, ANDaNA is an application-layer tool, which means that it is not suitable for supporting high-volume low-latency communication. Moreover, since it operates at the network layer, a single VPN tunnel can serve any number of consumers within the same trusted domain. Judging from the popularity and utility of VPNs in today's Internet, we conclude that a VPN-like technology is a much-needed tool for CCN.

In this paper, we present CCVPN, the first CCN-based VPN design. Similar to ANDaNA, CCVPN encrypts interest and content packets between two end-points. However, these end-points are network gateways instead of ANDaNA's ARs. In the standard configuration, both tunnel end-points are gateways between trusted domains. Tunnels may also be nested, similar to IPSec [5]. One of the end-points (the source) can be an

[+]Supported by the NSF Graduate Research Fellowship DGE-1321846.

individual consumer. In fact, the standard two-hop ANDaNA circuit is identical to a nested tunnel with the same source. Though designed to use efficient symmetric-key cryptographic primitives, CCVPN also works with public-key cryptography. This makes it easy to deploy in real-world CCN networks.

We implemented CCVPN and experimentally assessed its performance. Our results indicate that collective throughput across multiple consumers sharing a tunnel remains stable, up to a modest bound of 60 consumers, each requesting content at the rate of 1 Mbps. Moreover, as expected, the average round-trip time (RTT) of consumer requests decreases proportionally to collective throughput and request rate. Further improvements in both throughput and perceived RTT can be made with an implementation in a faster CCN router, such as that in the CICN project [6]. We leave this for future work.

This paper is organized as follows. Section II provides an overview of CCN. Next, Section II-B overviews related work that motivates our design. Section III then describes CCVPN and its security is analyzed in Section IV. Performance analysis and experimental results are presented in Sections V and VI, respectively. Finally, directions for future work are outlined in Section VII.

II. Preliminaries

This section overviews the CCN architecture[1] and other related work. Given familiarity with these topics, this section can be skipped without loss of continuity.

A. CCN Overview

In contrast to IP networks, which focus on end-host names and addresses, CCN [7], [8] 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 matching *content object* message is returned to the consumer upon satisfaction of the interest. 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.

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., 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 field, containing 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. Importantly, if an interest carries a payload, then its name *also* carries the hash digest of the payload so as to make the interest unique and prevent cache hits. The reader is encouraged to review [8] for a complete description of all packet fields and their semantics.

Packets are moved in the network by routers. A router 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 router may also maintain an optional *Content Store* (CS) used for content caching. From here on, we use the terms *CS* and *cache* interchangeably.

Routers 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.

B. Related Work

Related work falls into two categories: (1) anonymity networks, such as ANDaNA [3] and AC3N [4], which were discussed in Sec. I, and (2) encryption-based access control techniques.

Content encryption addresses data confidentiality rather than privacy or anonymity. Encrypted content disseminated throughout the network cannot be decrypted without appropriate decryption key(s). Many techniques have been proposed based on: general group-based encryption [9], broadcast encryption [10], [11], and proxy re-encryption [12]. [13] generalized these specialized approaches in a framework called CCN-AC, an encryption-based access control framework that shows how to use manifests to specify and enforce 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 name-based access control scheme (NDN NBAC) [14] is similar to [13] 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. In the interest-based access control

[1]Named-Data Networking (NDN) [7] is an ICN architecture which is very similar to CCN with a few subtle differences. We do not focus on NDN in this work. Nevertheless, CCVPN can be made to work for NDN as well.

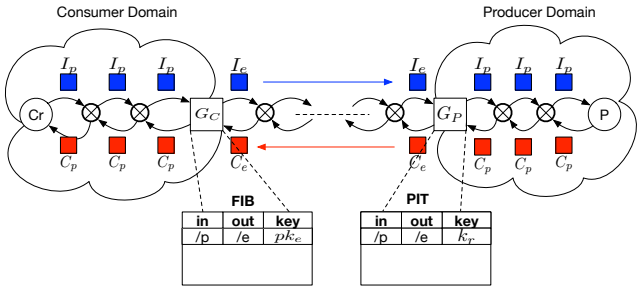


Fig. 1. CCVPN connectivity architecture

(IBAC) scheme [15] content is protected by making its name derivable only by authorized consumers. NDN-ACE [16] is a recent access control framework for IoT environments which includes a key exchange protocol for distributing keys to sensors. All these techniques, with few exceptions, use public-key cryptography to protect only the *payload* of content packets. They do not encapsulate complete packets between consumers and routers or producers that serve content. The only exception is CCNxKE [17] – a key exchange protocol that bootstraps secure sessions. As discussed later, CCNxKE can be optionally used by CCVPN to establish pair-wise shared keys between tunnel end-points.

III. CCVPN Design

A Virtual Private Network (VPN) supports secure communication across the Internet. It allows users to send and receive data across insecure public networks as if their computing devices were directly connected to the same private network [18]. The goal of CCVPN it to provide users with analogous functionality within the CCN architecture.

CCVPN involves four types of entities:

- *Consumer:* issues an interest for content it wants to retrieve.
- *Producer:* entity that creates and publishes content.
- *Consumer-Side Gateway (G_c):* end-point of the secure tunnel on the consumer side.
- *Producer-Side Gateway (G_p):* end-point of the secure tunnel on the producer side.

A secure CCVPN tunnel is uni-directional, i.e., for a given tunnel, the roles of Consumer and Producer are fixed. (We use the term *"uni-directional"* to mean that all interests flow in one direction, while all content flows in the opposite direction.) As discussed later, G_c and G_p can run on the same platform to enable bi-directional communication composed of two uni-directional tunnels. However, for clarity's sake, we present them as separate logical entities.

Fig. 1 shows the flow of a single interest and content exchange over CCVPN. Entities inside *Consumer* domain form a physically interconnected private network. Same holds for entities in *Producer* domain. The goal is to create an overlay network that joins these two private networks such that interest and content packets are only visible to entities within the respective domains.

G_c is an entity in *Consumer* domain that encrypts outgoing interests. Similarly, G_p decrypts and forwards incoming interests. When a content packet is forwarded in response to an interest, G_p decrypts incoming interests from G_c and forwards them in the *Producer* domain. G_p effectively serves as a proxy for the original consumer. When the intended content is returned to G_p, the latter encrypts (or encapsulates) the data before it exits the *Producer* domain and is forwarded to G_c. Finally, G_c decrypts the content packet and forwards it towards *Consumer*. In the rest of this section, we describe these steps in more detail.

NOTE: We use the term "encryption" to denote *"authenticated encryption"*, i.e., encryption and decryption algorithms that generate and verify integrity of data, respectively.

Upon arrival of an interest I_p, G_c performs a FIB lookup to check whether the interest's name prefix is among those used for VPN communication. (Here we assume that G_c's FIB is pre-configured with the list of prefixes that require VPN tunneling, i.e, prefixes associated with the producers in *Producer* domain in Fig. 1.) If the FIB lookup succeeds, G_c obtains G_p's name (actually, a prefix) and public key (pk_e). G_c then runs Algorithm 1 to generate a new interest I_e which encapsulates the original interest I_p.

First, Algorithm 1 generates a fresh and random symmetric key (k_r), used later for *Content* encryption and decryption. Next, it retrieves G_p's name and public key from the FIB. It uses the public key to encrypt[2] the symmetric key k_r and the original interest I_p.

It then creates the new interest I_e referring to G_p's name prefix and with encrypted I_p as the payload.[3] I_e is routed towards G_p. Since the payload is encrypted with G_p's public key, only G_p can decrypt it to obtain I_p and k_r.

Algorithm 1: Interest encapsulation (runs on G_c)

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;

Upon receipt of I_e, G_p verifies whether the interest name prefix matches one of its own. If so, it runs Algorithm 2, using its private key (sk_e) to decrypt I_e, which yields I_p and k_r. G_p then stores I_e and k_r in its PIT, as part of the entry for the pending interest I_p. In doing so, G_p acts as a proxy consumer.

I_p is forwarded inside the *Producer* domain until it reaches *Producer* or a cached copy of the target content at some router

[2]In fact, for the sake of efficiency, hybrid encryption is used. For details on the hybrid encryption implementation refer to [19].

[3]Recall that, since I_e carries an encrypted form of I_p in its payload, the name of I_e has a unique identifier appended to it so as to prevent cache hits between G_c and G_p.

Algorithm 2: Interest decapsulation (runs on G_p)

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;

within *Producer* domain. In either case, C_p is forwarded to G_p. Upon receiving C_p, G_p does a PIT look-up using C_p's name (i.e, name referenced in I_p) to retrieve k_r and I_e. Next, G_p encrypts C_p with k_r, yielding encrypted content C_e with the name I_e, as shown in Algorithm 3.

Algorithm 3: Content encryption (runs on G_p)

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;

Next, C_e is forwarded back to G_c using router state previously established by I_e. Upon receiving C_e, G_c invokes Algorithm 4: G_c performs a PIT look-up which matches C_e's name to the pending interest for I_e, and retrieves k_r. G_c then decrypts C_e with k_r and obtains C_p. Finally, C_p is forwarded to *Consumer* using router state set up by I_p.

Algorithm 4: Content decryption (runs on G_c)

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

As mentioned earlier, in the context of bi-directional communication, the same device can run both G_p and G_c. The same holds for actual end-points, i.e., the same "box" can act as Consumer in one uni-directional VPN tunnel, and as a Producer in the opposite-direction uni-directional VPN tunnel. This applies to interactive-type communication, such as conferencing or remote login where both end-points of the tunnel act as a producer and a consumer.

We also note that CCVPN does not inhibit in-network content caching within domains. Outside of the domains, i.e., between G_c and G_p, interest names are sufficiently random and prevent cache hits. This is a side effect of the authenticated encryption mechanism used to encapsulate I_p and form I_e. Finally, the roles of G_c and Consumer (or G_p and Producer) can be collocated, as is the case with IP-based VPNs. This enables private one-to-one, many-to-one, and one-to-many communication.

A trivial variation of CCVPN is the case when G_c and G_p have a shared pre-installed symmetric key. The only difference is that Algorithms 1 and 2 use symmetric key encryption instead of PKE. We refer to this variation as *symmetric-key CCVPN*. This variant can be configured if gateway keys are manually installed. Alternatively, a key exchange protocol such as CCNxKE [17] can be used to establish the gateway shared secret.

IV. CCVPN Security

This section discusses security considerations and properties of CCVPN.

A. System & Security Model

We consider the worst-case scenario, where Consumer issues an interest that has not been cached in any router in either Consumer or Producer domains. Therefore, the interest travels all the way to Producer.

Adversary Goals and Capabilities. The goal of the adversary $\mathcal{A}$ is either: 1) to learn some information about the original interest I_p or original content C_p, or 2) to learn the identities of Consumer or Producer.

$\mathcal{A}$ is also considered successful if it impersonates Producer by faking a content packet. We assume that $\mathcal{A}$'s presence and activities are confined to the public networks, i.e., the Internet. In other words, $\mathcal{A}$ has no presence in either *Consumer* or *Producer* domain. Also, since they are part of *Producer* and *Consumer* domain, respectively, we assume that G_c and G_p are not compromised.

We allow $\mathcal{A}$ to perform the following actions:

- **Eavesdrop on traffic:** $\mathcal{A}$ can eavesdrop on any link (outside Producer and Consumer domains), thus learning packet contents and other characteristics, such as timing and sizes.
- **Compromise existing, or introduce new compromised, routers:** this means that $\mathcal{A}$ can inject, delay, and discard interest or content packets at will. If $\mathcal{A}$ compromises an existing router, it learns all of private information, including all private keys and cached content.

B. Security Analysis

In this section we analyze security of CCVPN. We aim to prevent $\mathcal{A}$ from achieving goals outlined in Section IV-A. Formally, this translates into semantic security of all traffic, except for whatever can be inferred by traffic analysis. Consequently,

an off-path adversary can not forge content packets with non-negligible probability. Our analysis relies on arguments in the standard security model. It consists of assessing the security of the interest and content encapsulation algorithms. Moreover, we assume that all interest and content packets are *padded* to the standard MTU size between the gateways. (This is done in order to make packets indistinguishable.)

***Definition** 4.1: An interest encapsulation algorithm $Encapsulate(I_p)$ is indistinguishable iff, given any two interests I_p^1 and I_p^2, chosen by $\mathcal{A}$, and a randomly selected bit b, $\mathcal{A}$ has $1/2 + \epsilon$ probability of guessing the value of the bit b when given $I_e^b = Encapsulate(I_p^b)$, where ϵ is a negligible factor in terms of security parameter k.*

***Claim** 4.1: Let $Encapsulate_{pk}(I_p)$ denote the interest encapsulation routine described in Algorithm 1. If Enc_{pk} is a CPA-secure public-key encryption scheme, then $Encapsulate_{pk}(I_p)$ is indistinguishable.*

Proof– Suppose that Claim 4.1 is false. Then there is a polynomial adversary Adv capable of guessing b in Definition 4.1 with non-negligible advantage, when given $I_e^b = Encapsulate(I_p^b)$ with $b \leftarrow \{0, 1\}$ chosen at random. We show that if Adv exists, it can be used to construct another polynomial adversary $AdvCPA$ which breaks CPA-security of Enc_{pk}. $AdvCPA$ plays the CPA-security game with a challenger sending it two messages: m^0 and m^1. Following the CPA-security game, the challenger randomly chooses a value for the bit $b' \leftarrow \{0, 1\}$ and gives back $C = Enc_{pk}(m^{b'})$ to $AdvCPA$. To break CPA-security $AdvCPA$ must guess the value of the bit b' with non-negligible advantage. For that purpose $AdvCPA$ queries the challenger for encryptions of m^0 and m^1 ($c^0 = Enc_{pk}(m^0)$ and $c^1 = Enc_{pk}(m^1)$) and construct two interests $I_e^0 = createNewInterest(Gp_{name}, c^0)$ and $I_e^1 = createNewInterest(Gp_{name}, c^1)$, using the same $createNewInterest$ function used by algorithm 1, which is public (note that Gp_{name} is also public). Finally, $AdvCPA$ gives I_e^0 and I_e^1 as input to Adv and outputs whatever Adv outputs. Since under our assumption Adv guesses the bit b with non-negligible advantage, then $AdvCPA$ breaks the CPA-security of Enc_{pk}. Since this violates the hypothesis of Claim 4.1, Adv can not exist.

***Definition** 4.2: A content encapsulation algorithm $Encapsulate(C_p)$ is indistinguishable iff, given any two content packets: C_p^1 and C_p^2, chosen by $\mathcal{A}$, and random bit b, $\mathcal{A}$ has $1/2 + \epsilon$ probability of correctly guessing b when given $C_e^b = Encapsulate(C_p^b)$, where ϵ is negligible factor in terms of security parameter k.*

***Claim** 4.2: Let $ContentEnc_{k_r}(C_p)$ denote the content encapsulation routine described in Algorithm 3. If $EncryptThenMAC_{k_r}$ is an authenticated encryption (i.e., CCA-secure) symmetric-key scheme used to construct $ContentEnc_{sk}$, then:*

1) *$ContentEnc_{k_r}(C_p)$ is an indistinguishable content encapsulation algorithm;*
2) *$\mathcal{A}$ negligible probability of generating a valid fake encapsulated content I_c'*

Proof (Sketch)– Follows directly from the definition of CCA-security for authenticated encryption and from the same argument as in the previous proof.

We therefore claim that nothing is leaked in encapsulated interest or content packets as they are forwarded between gateways. Since the only information in these packets is G_p and a random nonce derived from the encrypted payload of I_e, Adv learns nothing about I_p or C_p, or the identities of Consumer or Producer.

C. Additional Considerations

Unlinkability between Consumer domain and encapsulated packets: An argument similar to those in Sec. IV-B can be used to show that, in a setting where multiple consumer domains establish tunnels to a given producer domain gateway Gp, an outside observer can not correlate (with non-negligible advantage) a given encapsulated packet (interest or content) to the domain where the original interest was issued.

In fact, this is an advantage of CCVPN when compared to VPNs over IP. In IPSec, for the sake of forwarding, each encapsulated packet carries source and destination addresses of correspondent tunnel end-points. Thus, an outside observer can easily determine the two domains that communicate encapsulated data. In CCVPN, encapsulated content packets are forwarded to the consumer domain according to routers' PITs, while interests carry no source addresses. Thus, observing encapsulated packets gives no more information about the correspondent consumer domain than observing their encrypted payload.

Gateway-to-Gateway Authentication: In CCVPN, any host that has G_p's public key can initiate a tunnel with G_p. In other words, our design does not include any authentication between tunnel end-points. We claim that $G_p - G_c$ authentication is not required since it not all application scenarios need it. For example, a producer offers its content to any consumer while requiring that the latter request and receive such content *privately*. In this case, there is no need for G_c to authenticate itself to G_p (as is the case today with majority of SSL/TLS web servers).

Another CCVPN use-case is where two physically separated private networks (e.g., offices of the same company in different countries) need to behave as a unified network. In that case, it is necessary to prevent extraneous G_c-s, from connecting to G_p. Standard host-to-host CCN security mechanisms can be used to authenticate G_p and G_c to each other, prior to VPN communication. We leave the evaluation and specification of gateway-to-gateway authentication protocols for future work (see Sec. VII). However, we note that CCNxKE [17] supports mutual authentication and could be used for this purpose.

Denial of Service: Since CCVPN gateways face the public network they are clearly exposed to DoS attacks. A DoS attack on G_p might involve flooding it with fake encapsulated interests, while a DoS attack on G_c would consist of flooding it with an enormous amount of encrypted content packets. The former is more dangerous, since interest decapsulation involves a public-key decryption operation. If symmetric-key algorithms were

used to encrypt interest and content packets, efficacy of DoS attacks would be reduced, though not negligible. We defer DoS counter-measures to future work.

V. CCVPN Performance

We now discuss performance aspects of CCVPN.

A. State Consumption

CCVPN has an immediate impact on a a gateway's FIB and PIT sizes. (Cache size remains unaffected since only decapsulated content objects are ever cached.) Let F_S be the total size of a standard CCN router FIB in bytes, and N_F – number of FIB entries. For simplicity, we assume that each name prefix in the FIB has a constant size of 64B. (We expect this to be a reasonable upper bound in practice). Thus, $F_S = N_F \times \ s$, where s is the size of each FIB entry; s includes a name prefix (64B) and a bit-vector that identifies matching links for the interface. We assume that a gateway has 128 links which is a safe upper bound. Therefore, $s = 80$B (= 16+64B). Now consider FIB size F_G for a CCVPN gateway. Some entries of a FIB will point to "private" prefixes, i.e., other domains, and therefore would be of larger size to account for the corresponding prefix and key material. For both public- and symmetric-key encryption, key size is the same: 32B [20]. Therefore, taking into account: FIB entry prefix key, target domain prefix (e.g., G_p), encryption key, and corresponding bit-vector, the total size of one "private" FIB entry is 176B, meaning that $F_G = 176N_F$B. Thus, in the worst case, CCVPN FIB is at most $F_G/F_S = 176/80 = 2.2$ times larger than the standard FIB. In practice, however, we expect growth factor to be much smaller, since the fraction of public-to-private FIB entries would be non-zero.

The same analysis is applied to PIT size. A standard PIT entry includes a complete name and ingress bit-vector.[4] A gateway PIT entry would contain the same elements as a standard PIT entry, plus a symmetric key (32B), a nonce (12B), and an encapsulation name (64B + 32B) – the name of an encapsulated interest that 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 CCVPN gateway PITs, respectively, and let N_P be the number of PIT entries in each individual table. Based on the above discussion, and assuming that a name is $\leq$ 64B, a standard PIT entry is 80B, while a gateway PIT entry is 204B. Therefore, in the worst case, CCVPN PIT would be $\leq \ P_G/P_S = 204/80 = 2.55$B larger than the standard PIT. Assuming a steady state size of approximately $1e^5$ entries [21], the PIT would be 20.4MB – well within the capacity of modern routers.

B. Processing Overhead

A CCVPN gateway adds some new steps to the data path of a packet. The main added computational burdens are packet encapsulation and decapsulation. In the public-key variant of CCVPN, interests are processed using public-key encryption, while content – using symmetric-key encryption. Let $T_E^P(n)$ and $T_D^P(n)$ be respective times to encrypt and decrypt nB of data using a public-key encryption scheme. Similarly, let $T_E^S(n)$ and $T_D^S(n)$ be respective times to encrypt and decrypt the same amount of data with a symmetric-key encryption scheme. Then, the latency of 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 original interest and content sizes, respectively. As a rough estimate, [22] lists the cost of AES-GCM to be $2.946\mu s$ for setup followed by 102MiB/second, on an Intel Core 2 1.83 GHz processor running Windows Vista in 32-bit mode (with AES ISA support). For packets that are $\leq \ 1,500$B, total processing time is $\approx \ 17\mu s$. Moreover, public-key encryption and decryption operations are always at least as expensive; thus, total latency is increased by at least $T \ = \ 4 \times 17\mu s \ = \ 68\mu s$. In comparison to network latency for a single packet, this might be unnoticeable, though for a steady arrival state of $\approx \ \ 1e^5$, it would lead to an unstable 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 system configuration and network conditions.

Another performance issue stems from gateways not being able to process packets without allocating memory. Specifically, each packet requires either an encryption or decryption. However, since this cannot be done entirely in-place, the gateway must allocate some memory for every packet, e.g., to store the MAC tag, to account for ciphertext expansion, or to apply padding. This overhead can outweigh that of cryptographic computations if packet arrival rate is high enough. Therefore, when implementing CCVPN, special care must be taken to ensure that memory allocation is minimized or avoided.

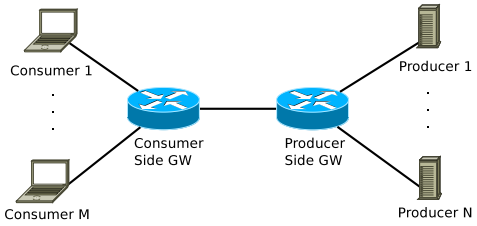


Fig. 2. Testbed network topology. M consumers and N producers

VI. Implementation and Performance Assessment

We implemented CCVPN as a network-layer service running on the gateways of private networks that compose the VPN (see Fig.1). The implementation uses the CCNx software stack [23] and libsodium cryptographic library [20]. Both are publicly available and written in C. For the public key version of CCVPN, we use the libsodium public-key authenticated encryption API in the interest encapsulation and decapsulation routines (Algorithm 1, and Algorithm 2 of Sec. III). Internally,

[4] They may also include optional `KeyId` and `ContentId`. However, since they are included in the gateway PIT as well, we omit them from this analysis.

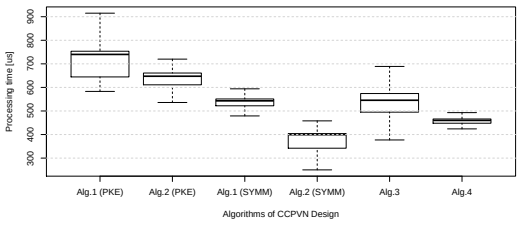
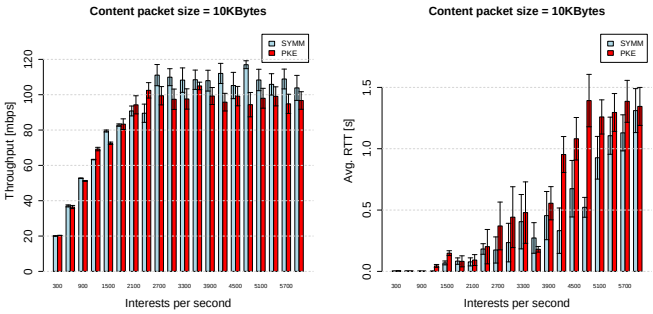


Fig. 3. Execution time of the algorithms in CCVPN design



(a) Throughput (b) Avg. RTT

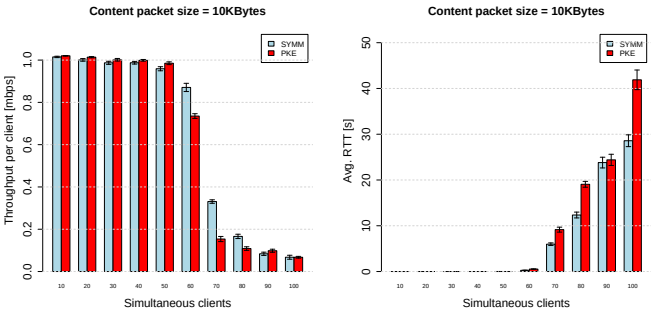
Fig. 4. CCVPN performance with one consumer and one producer for increasing interest issuance rates.

these perform a x25519 [24] key exchange to derive a symmetric key that is then used to encrypt and authenticate the interests in transit. AES256-GCM [25] is used to encrypt and authenticate content packets (Alg. 3, and Alg. 4 of Sec. III). Recall that symmetric keys used to encrypt and authenticate content packets are generated and sent together with the encapsulated interest in Alg. 1. In the symmetric key version of CCVPN, both interests and contents, are encapsulated with AES256-GCM under the assumption that the two gateways already share a symmetric key.

Experiments presented in this section were performed on an Intel Core i7-3770 octa-core CPU @3.40GHz, with 16GB of RAM, running Linux (Ubuntu 14.04LTS). Content payload size was set to 10 kilobytes. In every experiment, each of the two gateway processes (i.e., G_c and G_p processes) were assigned high priority and each ran in a single core. Fig. 3 presents box-plots of execution times for four algorithms involved in CCVPN data transmission, including both public and symmetric key versions for interest encapsulation and decapsulation.

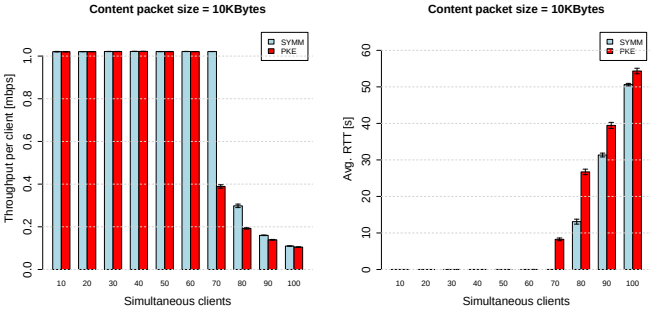
To evaluate the impact of CCVPN's cryptographic overhead on overall network performance, we measured network throughput and request-response round-trip time (RTT) under various topology settings. In our testbed, G_p and G_c are directly connected. N producers are connected to the former, and M consumers – to the latter, as illustrated in Fig. 2. We consider three variations for values of $[M, N]$:

- **One consumer and one producer** $[1, 1]$: We slowly increase interest issuance rate until we can to determine maximum network throughput and impact on RTT, as congestion increases.



(a) Throughput (b) Avg. RTT

Fig. 5. CCVPN performance with multiple consumers and one producer. Each consumer requests with 1 mbps rate.



(a) Throughput (b) Avg. RTT

Fig. 6. CCVPN performance with multiple consumers and multiple producers. Each consumer requests with 1 mbps rate.

- **Multiple consumers and one producer** $[M, 1]$: We fix interest issuance rate so that each consumer requests $\approx$ 1 mbps of content, and gradually increase the number of consumers, until throughput per consumer starts to decrease, i.e., until congestion starts to occur. All interests coming from all consumers are served by a single producer.
- **Multiple consumers and multiple producers** $[M, N]$: We gradually increase the number of consumers. However, we also increase the number of producers by the same amount in each round, i.e., $M = N$. The number consumers and producers is increased until congestion is detected.

In all of the these settings, every interest is a unique request for a unique content. Therefore, experimental results reflect throughput and RTT in the worst-case scenario, i.e., no content caching at the gateway. The results are presented with 95% confidence intervals.

Fig. 4 shows network performance when $[M, N] = [1, 1]$ as consumer's request rate increases. The network achieves maximum throughput of 100 mbps in the public key version and slightly higher throughput of 110 mbps in the symmetric key version. Average RTT per message starts to increase as interest issuance rate approaches maximum network throughput – a sign of congestion.

Results for multiple consumers requesting content from a

single producer ($[M, 1]$) are shown in Fig. 5. Each consumer receives close to requested throughput (1 mbps) when < 50 clients request content at the same time. With > 60 clients average RTT starts to increase due to congestion, and average throughput for each client gradually goes down.

Since a CCN producer must sign every content[5], congestion observed in Fig. 5 might be influenced by the overhead of having a single producer signing a large number of interests, in addition to gateways' cryptographic overhead. To evaluate this effect in Fig. 6, we show average throughput and RTT in the $[M, N]$ scenario, where $M = N$ and each consumer requests from a fixed producer. Here, results are slightly better. The network offers requested throughput (1 mbps per client) with $\leq$ 60 nodes, in the public key version, and $\leq$ 70 nodes, in the symmetric key version.

A. Discussion

CCVPN exhibits moderately good results with respect to network load capacity, considering overhead incurred by deploying secure tunnels over the CCN architecture. With gateway processes running each on a single core of a single processor, the VPN can provide reasonable throughput to $\leq$ 70 consumers. These performance results represent a lower bound that can be improved in several ways, such as:

1) **Implementation optimization:** CCNx software stack is an on-going research project and it prioritizes functionality over performance. We believe that CCVPN performance can be significantly improved by optimizations that do not rely exclusively on CCVPN design, but also in CCNx software.
2) **Distributed and parallel processing:** In a real deployment scenario, a large organization that wants to use CCVPN would have dedicated network devices running the gateway service, possibly in multiple cores. Also, multiple VPN gateways can share the load in a large organization.
3) **Caching:** Content caching is a major advantage of ICNs, as compared to IP. Our experiments evaluated worst-case scenarios, i.e., consumers always request distinct content packets and caching does not occur. In a real-world deployment, popular content (inside the VPN) would be cached, thus increasing throughput and reducing RTT.

VII. Conclusions and Future Work

We presented the design of CCVPN, discussed its implementation, and experimentally assessed its performance. CCVPN allows two CCN namespaces to be bridged by a secure tunnel across a public network. Unlike point-to-point tunnels, such as those enabled by secure session protocols, CCVPN allows many consumers to share the tunnel to access a private namespace. CCVPN is designed with efficiency in mind: public-key cryptographic operations are kept to a minimum during normal operation, while symmetric-key packet encapsulation algorithms are used to marshall packets between gateways. Experiments show that CCVPN results in a modest performance cost in the presence of a variable number of consumers and producers. Our results suggest that CCVPN can be used for private namespace tunneling in real-world CCN deployments.

For future work, we plan to use the CCNxKE key exchange protocol [17] to bootstrap symmetric-key tunnels between gateways. In addition, we plan to integrate CCVPN into CCN testbeds to assess its performance with real-world applications. These include video streaming and large-scale content dissemination applications. We also plan to investigate countermeasures to DoS attacks, such as the use of puzzles for tunnel establishment. Lastly, it would be interesting to compare CCVPN against IP-based VPNs and evaluate the benefits of a CCN-based tunnel.

[5]Unless the content can be requested by hash.

References

[1] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," DTIC Document, Tech. Rep., 2004.
[2] A. Luotonen and K. Altis, "World-wide web proxies," *Computer Networks and ISDN systems*, vol. 27, no. 2, pp. 147–154, 1994.
[3] S. DiBenedetto, P. Gasti, G. Tsudik, and E. Uzun, "Andana: Anonymous named data networking application," *arXiv preprint arXiv:1112.2205*, 2011.
[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. Kent, "Ip encapsulating security payload (esp)," 2005.
[6] CISCO, "Cicn project," https://wiki.fd.io/view/Cicn, 2017.
[7] V. Jacobson *et al.*, "Networking named content," in *CoNext*, 2009.
[8] M. Mosko, I. Solis, and C. Wood, "CCNx semantics," *IRTF Draft, Palo Alto Research Center, Inc*, 2016.
[9] D. K. Smetters, P. Golle, and J. D. Thornton, "CCNx access control specifications," PARC, Tech. Rep., Jul. 2010.
[10] S. Misra, R. Tourani, and N. E. Majd, "Secure content delivery in information-centric networks: Design, implementation, and analyses," in *ICN*, 2013.
[11] M. Ion, J. Zhang, and E. M. Schooler, "Toward content-centric privacy in ICN: Attribute-based encryption and routing," in *ICN*, 2013.
[12] C. A. Wood and E. Uzun, "Flexible end-to-end content security in CCN," in *CCNC*, 2014.
[13] J. Kurihara, C. Wood, and E. Uzuin, "An encryption-based access control framework for content-centric networking," *IFIP*, 2015.
[14] Y. Yu, A. Afanasyev, and L. Zhang, "Name-based access control," *Named Data Networking Project, Technical Report NDN-0034*, 2015.
[15] 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.
[16] W. Shang *et al.*, "Ndn-ace: Access control for constrained environments over named data networking."
[17] 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
[18] S. Khanvilkar and A. Khokhar, "Virtual private networks: an overview with performance evaluation," *IEEE Communications Magazine*, vol. 42, no. 10, pp. 146–154, 2004.
[19] Sodium, "Public-key authenticated encryption," https://download.libsodium.org/doc/public-key_cryptography/authenticated_encryption.html, 2017.
[20] ——, "The sodium crypto library (libsodium)," https://github.com/jedisct1/libsodium, 2017.
[21] 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.
[22] "Crypto++ 5.6.0 Benchmarks," https://www.cryptopp.com/benchmarks.html, accessed: 2016-11-21.
[23] PARC, "Ccnx distillery," https://github.com/parc/CCNx_Distillery, 2016.
[24] D. J. Bernstein, "Curve25519: new diffie-hellman speed records," in *International Workshop on Public Key Cryptography*. Springer, 2006, pp. 207–228.
[25] M. Dworkin, *Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC*. US Department of Commerce, National Institute of Standards and Technology, 2007.