# A secure encryption proxy

Marc Mosko*, Chris Wood

**Abstract**
With the growing use of end-to-end encryption on the Internet, mobile carriers find it more and more difficult to optimize the air interface because TCP proxies can no longer peek in to data streams and adjust them for the radio. We describe an *encryption performance enhancing proxy* (ePEP) using CCNx technology that may recover not only the lost ground from today's traditional PEPs, but even go beyond their performance. The CCNx ePEP may exceed today's PEP performance because we are able to run two control loops: one radio aware transport between the UE and the proxy, and one Internet transport between the proxy and data replica. The CCNx ePEP is a semi-trusted middlebox. It has visibility to some signaling between the consumer and replica, but it has no visibility to the data. The consumer maintains complete confidentiality, integrity, and authenticity.

**Keywords**
Content Centric Networks, Performance Enhancing Proxy, Encryption

*Palo Alto Research Center*
***Corresponding author**: marc.mosko@parc.com

## Contents



## Introduction

It has long been known that TCP has poor performance over wireless links [**?**, **?**, **?**]. This is caused by several factors, which come down to the fact that the radio link has a much different channel response for a transport protocol than a wireline link. TCP cannot distinguish between the radio link and the wireline link, so it under estimates the end-to-end capacity. Much work has thus gone in to proxy (or split TCP) topologies, where a middle box terminates the radio transport loop and establishes a second transport loop to the wireline system. This allows the transport protocols to optimize performance on each side of the proxy and keep the air interface full with a minimum of duplicate data due to TCP retransmission requests. These proxies are often called a Performance Enhancing Proxy (PEP).

The fate of the PEP is in decline [**?**]. Today's internet uses more and more encryption, which prevents middle boxes, such as PEPs, from intercepting an encrypted session. HTTPS and TLS prevent PEPs from doing much more than packet shaping to try and adjust data rates. This is bad news for mobile operators, as PEPs yield significant benefits, especially for large flows like videos.

Recent research on PEPs in actual carrier networks has shown what the penalty is for losing PEPs in large data flows. Because of the large variance in bandwidth of LTE links, TCP cannot fully utilize the wireless link. This results in over 71% of large flows under utilizing bandwidth by up to 50% [**?**]. In general, HTTPS is not proxied, which means those flows are losing out on up to the 45% improvement that split connections can offer [**?**].

This problem will continue to get worse. In 2014, the Internet Architecture Board released a statement on confidentiality that promote the ubiquitous use of encryption through the protocol stack and applications [**?**].

The CCNx ePEP has some similarity to multi-context TLS (mcTLS) [**?**]. In mcTLS, each TLS record may have different keys associated with it. Each record type *i* has a set of keys associated with it: end-to-end ($K_{endpoints}$), writers ($K_{writers}$), and readers ($K_{readers}$). Only the endpoints of have the endpoint keys, and certain designated middle boxes may have read or write keys. One issue with mcTLS is that within a record context shared with a middle box, *there is no confidentiality*. This means that for mcTLS to allow a mobile operator to shape traffic, there needs to be a higher-level protocol that sets up its own signaling channel over one TLS record context and a data channel over a second TLS record context and

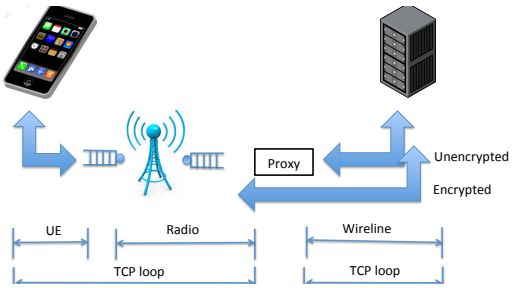


**Figure 1.** Traditional PEP with Encryption

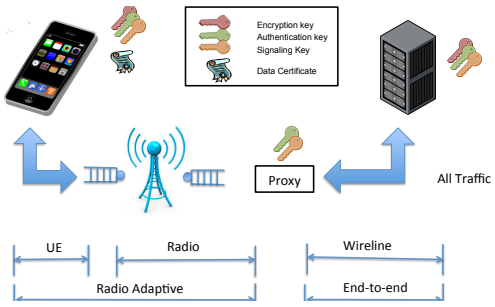


**Figure 2.** CCNx ePEP

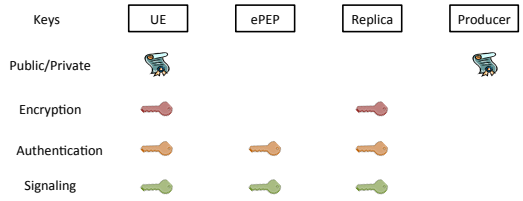


**Figure 3.** CCNx ePEP Keys

gives the middle box enough data in the signaling channel to optimize the data channel. However, because HTTP/2 uses its own frame channels within its session (and thus within the TLS data context record), the signaling channel does not have any visibility in to that. Only if one restructures how HTTP/2 organizes it frames in to different TLS records and then splits that signaling in to the signaling context, can one really achieve a useful PEP middle box.

The CCNx *Encryption Performance Enhancing Proxy* (CCNx ePEP) solves this problem by introducing a semi-trusted middle box. The ePEP participates in some end-to-end signaling while it runs two different transport loops. This maintains end-to-end confidentiality, integrity, and authenticity while benefiting from all the performance improvements a PEP can offer a mobile handset.

CCNx ePEP is described for use with a single ePEP. This is not an inherent limitation. One could distribute keys to multiple proxies, similar to the mechanisms of mcTLS, though the keys are used for different purposes.

CCNx ePEP is described in a simplex channel from consumer to replica. The consumer does not require, at any stage, its own routable name. There may be some situations where a duplex channel is required, in which case a consumer needs a routable name. In this situation, the system described below is executed twice, resulting in two simplex channels, one in each direction.

## 1. Traditional PEP

A traditional PEP only handles traffic it understands at the protocol level, such as HTTP or FTP. By using deep packet inspection, it can understand the data flow and adjust either the requests or TCP behavior, such as window size, to optimize the air interface. This allows the PEP to mask the high variance of the radio link from the wireline transport and to tune the radio side of the transport loop for high bandwidth-delay and loss.

Figure 1 shows an example setup, where the PEP is used for unencrypted traffic and encrypted traffic bypasses the PEP.

## 2. CCNx ePEP

The CCNx ePEP shares a set of keys between the UE and replica and a subset of those keys with the ePEP. This allows the ePEP to participate in some control signaling without violating the confidentiality, integrity, or authenticity of the data. This is done using a combination of session keys and consumer - producer signatures.

Figure 2 shows an example operator's network with a CCNx ePEP in place of the traditional proxy. This configuration allows a radio-aware transport protocol to execute on the radio side, an Internet end-to-end transport protocol to operate on the wireline side, and all traffic – both encrypted and unencrypted – to operate through the proxy.

Figure 3 shows how ePEP distributes keys. The original producer (which may be the same as the replica) has a public identity, so the consumer may verify the data using standard CCNx mechanisms regardless of the replica endpoint. This end-to-end authenticity could be achieved through other means than signatures, such as a group encryption or DRM or other shared state between the consumer and producer. The important feature is this state is independent of the replica. The consumer and replica share a set of encryption ($K_e$), authentication ($K_a$), and signaling ($K_s$) keys. These keys are used to secure and authenticate a logical data channel and logical signaling (control) channel. These are described below in Sec 2.1.

### 2.1 Logical Channels

CCNx ePEP uses two logical channels per encryption context (set of keys). Each encryption context is identified by a Session ID (SID). Within a SID, CCNx ePEP multiplexes flows within logical transactions. The transactions are similar to HTTP/2 frames. Each transaction has a signaling channel and a data channel.

CCNx ePEP uses two levels of naming. The outer names represent the encryption context. Within a data channel of an encryption context, the consumer and replica use inner names

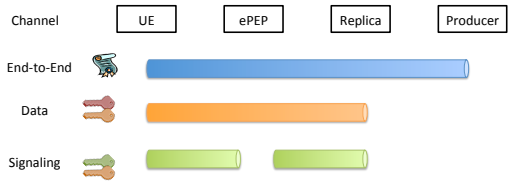


**Figure 4.** CCNx ePEP Logical Channels

to identify actual data. These inner names may be plaintext or may be further hidden by group encryption or anonymized transport manifests or other means.

At the outer name level, there are two more logical channels. The first is the SID-level control channel. Within the naming context of Name 1, a consumer - proxy - replica can exchange signaling. This is primarily used to shutdown an encryption context. The second is the XID-level control channel. This is used to exchange signaling and data relevant to a particular consumer transaction.

`/replica-prefix/SID=sid/Chunk=m` (1)

`/replica-prefix/SID=sid/XID=xid/Chunk=n` (2)

Within an XID context, consumers and replicas exchange transactions. The notation $E_k\{x\}$ means to encrypt (and authenticate) the element $x$ under key $k$. The notation $S_k\{x\}$ means to only authenticate $x$ under key $k$. For example, in an AEAD scheme like AES-GCM, the authenticate step computes a GHASH over $x$ using key $k$ and produces an authentication token.

$$S_{Ka}\{E_{Ks}\{\text{Signaling}\} \,||\, E_{Ke}\{\text{Data}\}\} \quad (3)$$

An equivalent method that drops the authentication key but co-signs a hash is:

$$E_{Ks}\{\text{Signaling, EDH}\} \,||\, E_{Ke}\{\text{Data}\} \quad (4)$$

$$EDH = \text{Hash}\{E_{Ke}\{\text{Data}\}\} \quad (5)$$

## 3. Protocol

We assume the consumer knows the identity of the proxy. This may be done via on-line discovery, such as in mcTLS, or via configuration, such as when a mobile node associates with a base station. Note that packets from the consumer will be addressed to the replica, not the proxy, so the operator network needs a method to put the proxy on-path.

We assume the consumer knows the identity of the replica. This could be done via the application, a name resolution service, or user actions. For example, a Foo application could ask `ccnx:/foo` for a set of replicas and be given the replica name. In another possibility, a mobile node could be configured with the local cache replica of the mobile operator.

We assume that the consumer, proxy, and replica have exchanged keys and established a Session ID (SID), as in Sec 4.

The CCNx Interest / Content Object protocol is executed within a transaction, though with some modification given that there is an inner and outer context. We also are not specifying the radio transport or wireline transport protocols here, though we give an example.

The consumer initiates a transaction by sending a first Interest, which initializes state for a new XID 1. The inner name `/data/manifest` requests the manifest All In One [**?**] stream for the prefix `/data`. The All In One stream will begin sending all of the manifest objects and all of the data objects for that manifest inside XID 1. This first interest roughly corresponds to an HTTP GET request.

$$\begin{aligned} &Interest\{ \\ &\quad Name = \texttt{/replica/SID=sid/XID=1/Chunk=0} \\ &\quad Signal = \\ &\quad\quad Data = Interest\{\texttt{/data/manifest}\}\} \end{aligned}$$

The proxy forwards this interest to the Replica, which responds with a corresponding ContentObject. The response carries the signaling of what the EndChunkNumber for XID 1 will be. This follows the CCNx Chunking Protocol specification, and it may be modified later if the replica does not know the actual chunk number (so long as it does not shrink before what is already read).

$$\begin{aligned} &Content\{ \\ &\quad Name = \texttt{/replica/SID=sid/XID=1/Chunk=0} \\ &\quad Signal = \text{EndChunkNumber} = z \\ &\quad\quad Data = Content\{\texttt{/data/manifest}\} \end{aligned}$$

The proxy may begin pipelining Outer interests to the replica after Chunk 0. This establishes the wireline control loop. We do not specify the transport protocol here, except to say that the proxy has control of issuing the right number of interest messages to the replica for the wireline channel.

There is a one-to-one correspondence between outer CCNx messages and inner CCNx messages. That is, one outer Interest carries at most one Inner interest and one outer Content Object carries at most one inner Content Object. Because the first Interest requests the All In One stream of the manifest, no additional inner interests are required on the wireline side. The outer interests generated by the Proxy will fetch the whole manifest tree and data.

The one-to-one correspondence *does not* mean that there is a correspondence between inner names. For example, outer name N1 could request inner name M1 and outer name N2 requests M2. In there downlink, outer name N1 could contain M2 and N2 contain M1.

### 3.1 Alert Messages

Similar to TLS, CCNx ePEP supports Alert messages. At preset, we only define four messages: REKEY, CLOSE, ERROR, KEEPALIVE. An ERROR implies a CLOSE. For example, if messages fail authentication, that is an error and will result in a close. A REKEY message will cause all parties to negotiate new session keys, such as via CCNxKE, and result in a new Session ID (SID). Existing open XIDs will carry over to the new SID. Once the new SID is in operation, the consumer and producer will terminate the old SID with close messages, but keep the XIDs open. Carrying over an XID means that the Chunk numbering of the XID also remains in tact.

When a consumer is done reading from a transaction, it must send a Close message. The proxy will forward the close to the replica and the replica will respond with its own Close message in the downstream signaling channel. if the consumer does not receive the downstream Close, it may timeout and resend it, or it could shutdown the XID.

If the consumer receives a CLOSE response on a SID for a non-existent XID, it will send an ERROR in the SID control channel to the replica. This will cause the replica to destroy the XID.

A replica may shutdown an XID at any time. It may put a CLOSE or ERROR message in the response to any outer pending Interest from the consumer. In this case, it does not put anything in the Data channel. This will cause an imbalance between pending interests and returning content objects in the data channel. However, because the replica indicated a CLOSE or ERROR, it will have flushed any remaining data so it does not matter.

A consumer or producer may request a REKEY at any time. The REKEY message is carried in the first outer namespace `/replica-prefix/SID=sid/Chunk=m`. Because the channel is simplex, the consumer must give the replica a chance to send it control messages on this channel. This is done by sending KEEPALIVE messages at some negotiated rate (e.g. every 30 seconds) from the consumer to the producer. The producer could respond with its own KEEPALIVE or a REKEY or a CLOSE. This maintains the interest/content object flow balance in this control channel.

## 4. Key Exchange

### 4.1 Provisioned proxy

In provisioned mode, the consumer establishes a key control channel with the proxy (by CCNxKE) and session keys with the replica (via CCNxKE). The consumer then shares the authentication ($K_a$) and signaling ($K_s$) keys with the proxy over the key control channel.

### 4.2 Dynamic proxy discovery

In transport offload, a client (e.g. 5G handset) uses a wireline proxy to execute the transport protocol (flow control and congestion control) to optimize the air interface.

To support private data but still allow the transport protocol to operate correctly, we need the proxy to authenticate data but not decrypt data. This section describes how to derive an authentication key (Ka) and a separate encryption key (Ke) among the three parties. It uses a method borrowed from mcTLS for deriving the keys only, not the encryption methods in the paper. The following items describe some relevant assumptions and notation.

- Three parties: Client, Proxy, Replica.
- Assumptions: The client trusts the server (e.g. through certificate validation).
- Topology: The client communicates with the server through the proxy.
- Modified mcTLS: For $n > 1$ proxies, there are n different security contexts derived using mcTLS Each context is set to the name of the entity or proxy and its KeyID with the appropriate string, e.g., “ka”, as shown below.

$$context = \text{“}proxyA\text{”}||\text{“}0x1abc2901..\text{”}||\text{“}ka\text{”} \quad (6)$$

  There are three supported key strings: “ka”, “ke”, and “ks”.

To begin, each party establishes a pair-wise shared key via CCNx-KE. For example, pair-wise keys are created for the client and proxy, proxy and replica, and the client and replica. After this step, the context keys – $K_e$, $K_a$, and $K_s$ – must be created. To do this, we use the per-context derivation technique described in the mcTLS text. Specifically, the client and replica generate per-context keys using the following technique:

$$K_i^C = PRF_{S_{c-r}}(< context > ||rand_C) \quad (7)$$

where $K_i^C$ is a context key derived at the client, $S_{c-r}$ is the traffic secret derived between the client and replica via CCNx-KE, “¡context¿” is the context string defined above (for the appropriate key), and $rand_C$ is fresh randomness generated by the client and given to the server in the CCNx-KE exchange. After these keys are created, the client and replica share them with the middlebox(es) as needed by encrypting them with the appropriate pair-wise keys. For example, since the proxy needs to obtain both $K_s$ and $K_a$, the keys $K_s^C, K_s^R, K_a^C, K_a^R$ would be shared with the proxy. The final coputation to derive $K_s$ and $K_a$ from these values is as follows:

$$Ka < -PRF_{K_a^C|K_a^R}('ka'||rand_C||rand_R) \quad (8)$$

The entire exchange requires two ”protocol executions”: One round of CCNx-KE to establish shared keys between each party, and another to establish and share per-context (per-proxy transport keys) with each party and each allocated middlebox.

## 5. Variations

### 5.1 Separate manifest from data

In one variation, the consumer requests the manifest (all at once) separately from the data. In this case, the consumer can request manifest branches as all in one streams, so there is still a substantial reduction in upstream traffic.

### 5.2 Transport manifests

In one variation, the consumer requests a transport manifest in the signaling channel and the proxy then begins fetching the data based on that transport manifest. In this case, the data is still in the data channel. The transport manifest needs to be padded and encrypted with $K_e$ and authenticated with $K_a$ to avoid leaking information. Since $K_a$ is used to authenticate the encrypted manifest elements, and since the proxy is given $K_a$, it is also to verify the contents of the manifest without seeing the plaintext data encrypted by $K_e$.

*Interest*{
*Name* =`/replica/SID=sid/XID=1/Chunk=0`
*Signal* =*Interest*{`/data/manifest/Chunk=0`}
*Data* =}

*Content*{
*Name* =`/replica/SID=sid/XID=1/Chunk=0`
*Signal* =*Content*{`/data/manifest/Chunk=0`}
*Data* =}

*Interest*{
*Name* =`/replica/SID=sid/XID=1/Chunk=1`
*Signal* =*Request*{`/manifestleaf`}
*Data* =}

*Content*{
*Name* =`/replica/SID=sid/XID=1/Chunk=1`
*Signal* =*Response*{`/manifestleaf`}
*Data* =*Content*{`/manifestleaf`}}

### 5.3 Consumer-replica Interests

In one variation, the consumer drives all Interest traffic with consumer-to-replica Interests in the Data channel, but the proxy traffic shapes the Interests within the XID to achieve the proper wireline downlink capacity to keep the radio link saturated. In this case, the only messages in the signaling channel are close messages.

## 6. mcTLS Support

The ePEP desgin can be emulated with mcTLS on top of IP. This can be done as follows: Within a single mcTLS session with at least one proxy, define two contexts for a single transaction: one for signalling information and one for data information. The client and server should not share the reader or writer keys for the data context with the proxy. The proxy should only receive the signalling context keys. Afterwards, the client and server should use the data context to transport application data and the signalling context to share transport-related information with all parties (including the proxy). The use of these channels follows exactly as described in this document.

If more than one transaction is needed in a session, then more than than one pair of context channels (signalling and data) should be created and shared appropriately. There is no restriction on the amount of channels that can be created.

## 7. Conclusion

## References


[1] R. Yavatkar and N. Bhagawat. Improving end-to-end performance of tcp over mobile internetworks. In *Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications*, WMCSA '94, pages 146–152, Washington, DC, USA, 1994. IEEE Computer Society.

[2] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, and Randy H. Katz. Mobility. chapter A Comparison of Mechanisms for Improving TCP Performance over Wireless Links, pages 401–417. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999.

[3] A. Bakre and B. R. Badrinath. I-tcp: indirect tcp for mobile hosts. In *Distributed Computing Systems, 1995., Proceedings of the 15th International Conference on*, pages 136–143, May 1995.

[4] Citrix statement on bytemobile failure. `http://broabandtrafficmanagement.blogspot.com/2015/11/citrix-de-invests-bytemobile-blaming.html`. Accessed: 2016-04-14.

[5] Junxian Huang, Feng Qian, Yihua Guo, Yuanyuan Zhou, Qiang Xu, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. An in-depth study of lte: Effect of network protocol and application behavior on performance. *SIGCOMM Comput. Commun. Rev.*, 43(4):363–374, August 2013.

[6] Xing Xu, Yurong Jiang, Tobias Flach, Ethan Katz-Bassett, David Choffnes, and Ramesh Govindan. *Passive and Active Measurement: 16th International Conference, PAM 2015, New York, NY, USA, March 19-20, 2015, Proceedings*, chapter Investigating Transparent Web Proxies in Cellular Networks, pages 262–276. Springer International Publishing, Cham, 2015.

[7] Iab statement on internet confidentiality. `https://www.iab.org/2014/11/14/iab-statement-on-internet-confidentiality`. Accessed: 2016-04-14.

[8] David Naylor, Kyle Schomp, Matteo Varvello, Ilias Leontiadis, Jeremy Blackburn, Diego R. López, Konstantina Papagiannaki, Pablo Rodriguez Rodriguez, and Peter Steenkiste. Multi-context tls (mctls): Enabling secure in-network functionality in tls. *SIGCOMM Comput. Commun. Rev.*, 45(4):199–212, August 2015.

[9] Marc Mosko and Ignacio Solis. All-in-one streams for content centric networks. *ICNS 2015*, page 122, 2015.