Enabling Off-the-grid Communication for Existing Applications: A Case Study of Email Access

Teng Liang
The University of Arizona
philoliang@cs.arizona.edu

Beichuan Zhang
The University of Arizona
bzhang@cs.arizona.edu

***Abstract*—Most of today's applications require an Internet connection to reach cloud servers essential for their functioning. However, in many scenarios, such connectivity is unavailable, such as when a user is in a remote location with no cellular coverage or during a natural disaster. In these situations, nearby devices can still connect to each other, and the applications running on each device can have data useful to each other; however, the applications are unable to exchange data directly between themselves.**

In this paper, we describe mechanisms to enable off-the-grid communication for existing applications, allowing them to directly exchange data between themselves. To achieve this goal, an application should be capable of discovering, fetching, and verifying data without the help of a server. We believe that Named Data Networking (NDN) can provide an application with these abilities. We utilize email access as a study case and propose *mailSync*, a framework to the goal of app-to-app communication for this application via NDN. Meanwhile, we identify the various differences between TCP/IP and NDN applications, and we generalize several steps to "NDNize" an existing application, including protocol translation, application layer framing, naming, data discovery and security management. We implemented a prototype of mailSync in Java on laptop and Android as a proof of concept.

I. Introduction

Most of today's applications require an Internet connection so they can communicate with cloud servers essential to their functioning. For example, an email client retrieves emails from a predefined server on the Internet, and a collaborative text editor uses a centralized server to maintain the shared state and to mediate all interactions among users.

However, Internet connectivity is unavailable in many locations and scenarios, such as when a user is in a remote location with no cellular coverage, when traveling on a cruise ship, or during a natural disaster. In these situations, many applications do not function properly. Some applications provide an offline mode with cached data and local operations, but in a multi-device scenario, these applications are unable to share cached data between devices (e.g., Google Maps). A few ad-hoc services can exchange data between two devices, but require pre-planned, application-specific, and point-to-point mechanisms (e.g., Apple Airdrop provides a file transfer service between two Apple products over Bluetooth).

Today's devices possess the physical capabilities necessary to store and exchange data between nearby devices, as they have a decent amount of storage (e.g., 16 GB or more) and can connect directly to each other through various technologies (e.g., WiFi-Direct, Bluetooth, Zigbee, etc.).

The key problem preventing local data exchange between applications is the dependency on cloud servers. This problem stems from the mismatch between applications and the TCP/IP network. In contrast to applications that focus on data, the TCP/IP network focuses on communication between end hosts. Thus, establishing communication to a pre-defined, centralized, and trusted server has become a simple and reliable model for application design. However, this client-server model makes direct data exchange among applications challenging.

To solve the problem, we should enrich an application with several abilities. First, an application should be able to discover if desired data exists in connected networks (e.g., point-to-point and ad-hoc networks), then it should be able to fetch the data. Additionally, it must be able to verify fetched data, since the data is not transmitted from a predefined, trusted server.

In this scenario, we believe that the Named Data Networking (NDN) architecture provides a better platform applications desiring these abilities than traditional TCP/IP networks. NDN networks provide built-in data discovery and fetching as a core functionality of the network: NDN names data, provides a pull-based communication model, and achieves stateful forwarding based on data names. In other words, an application can simply send a query for a data name to an NDN network, and the network will discover the queried data and return it to the application. In addition, NDN provides data-centric security, since each data packet in NDN is bound to a signature. Thus, data in NDN can be verified independently of the location, no matter from which it was fetched.

Although NDN is a good fit for distributed systems, we do not wish to design a pure NDN application in this paper. Instead, we study how to enable an existing application to share data directly with other peers in an NDN network. In addition, we intend to achieve the goal with little modification to the existing application, allowing them to seamlessly access data from either a cloud server or a peer. Specifically, we conduct a case study of email access, since it is a widely-used application with standardized protocols. In addition, we propose **mailSync**, a framework that enriches a legacy email client, allowing it to access emails directly from a peer using

NDN (shown in Figure 1).

In our target scenario, a user has downloaded all of their emails onto their phone in an airport. After boarding their plane, they wish to work on their emails from a laptop. Without Internet connectivity, an existing email client on the laptop is unable to access emails either from a cloud server or directly from the phone. However, with the help of mailSync, the email client on laptop is able to access emails downloaded on the phone (assuming that two devices can connect to each other).

In addition, the design of mailSync reveals several key differences between TCP/IP and NDN applications. Based on the differences, we have generalized several steps to "NDNize" an existing application, including protocol translation, application layer framing, data discovery, naming and security management. Moreover, we implement a prototype of mailSync in Java on laptop and Android as a proof of concept.

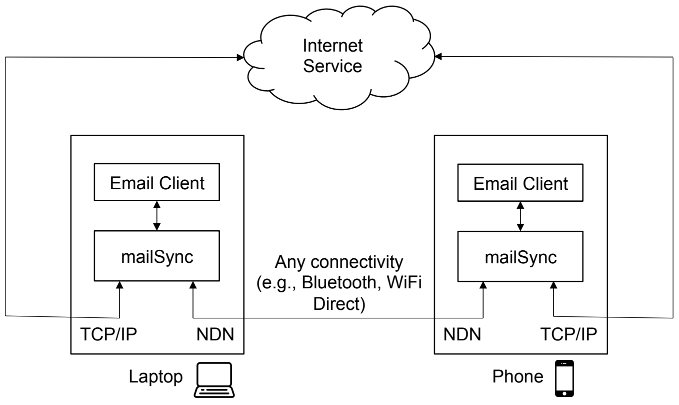

Fig. 1: Overview

II. Related Work

In TCP/IP networks, the proposed Legion framework [1] allows client web applications to replicate data from servers, and synchronize these replicas directly among themselves. However, Legion still requires an Internet connection (using WebRTC) to establish a peer-to-peer connection between two web clients. In addition, Legion achieves security with the help of a centralized server, meaning that, each client has to be authenticated by a centralized server before fetching a symmetric key for data encryption and decryption. Legion utilizes infrastructure services to build a peer-to-peer overlay network and synchronize data directly among web clients, in order to solve the scalability problem present in the client-server model; meanwhile, our work intends to remove the dependence on infrastructure from existing applications and enable them to fetch data from anywhere using any connectivity mechanism in NDN networks.

Within NDN community, a large amount of research has been conducted on NDN networks. In addition, many NDN applications have been designed and implemented from scratch, NDN-RTC [2], nTorrent [3], NDNFit [4], ChronoShare [5], and many more. As NDN differ from TCP/IP networks in many areas, including communication, security, and the data transfer model, an NDN application is totally different from a TCP/IP application. However, the differences between these two applications have rarely been explored. We intend to use mailSync to identify and explain these differences. Moreover, instead of building a new NDN application from scratch, we enable an existing TCP/IP application to access data via NDN and utilize its benefits.

In addition, translation between TCP/IP and Information-centric Networking (ICN) traffic have been explored at different layers. For instance, Moiseenko [6] proposes a TCP/ICN proxy capable of carrying TCP traffic over ICN networks; Refaei [7] proposes an IP-to-NDN gateway to convert TCP and UDP traffic into NDN traffic; Li proposes an HTTP-CCN gateway [8] to convert HTTP traffic into Content-centric Networking (CCN) traffic, in order to introduce real traffic onto a CCN testbed. These works intend to carry TCP/IP network traffic over an ICN network through two gateways, so that two TCP/IP based endpoints can still communicate. However, we intend to utilize NDN's benefits to provide existing applications with ability that is challenging to achieve in TCP/IP networks (namely, accessing data from anywhere). Moreover, we do more than a protocol translation, as we "NDNize" an existing application to access data in an NDN network, which includes protocol translation, data organization, data access, naming, and security management.

III. System Design

A. Overview

The design goal of mailSync is to enable an existing email client to access emails from either a cloud server via TCP/IP or a peer via NDN. In addition, mailSync is intended to have little modification to the client. Thus, we design mailSync as a framework to bridge the email client and different networks (shown in Figure 2). From email clients' perspective, mailSync is transparent, and it still uses standardized Internet Message Access Protocol (IMAP) [9] to access emails, except that the IMAP traffic will go through mailSync instead of directly to a cloud server.

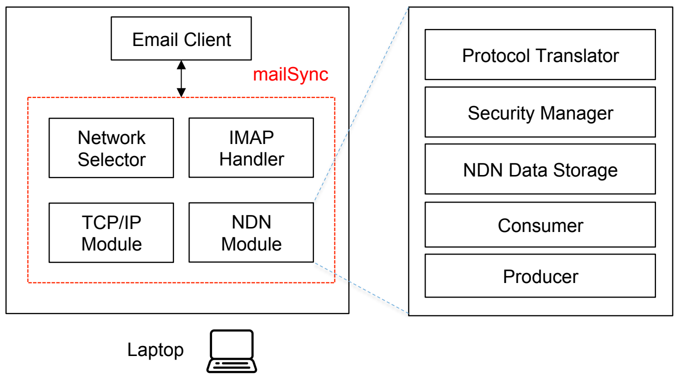

Fig. 2: The mailSync Architecture

On receiving IMAP traffic from email client, mailSync first decides whether to use cloud servers via TCP/IP or fetch data from NDN networks. As IMAP is a TCP/IP protocol, it has to be translated into NDN communication for NDN networks. To accomplish the protocol translation, mailSync must understand the semantics of IMAP and its corresponding NDN communication. Thus, as shown in Figure 2, mailSync contains four modules to handle these tasks:

- **Network Selector:** Network selector decides to use either TCP/IP or NDN networks.
- **TCP/IP Module:** TCP/IP module serves as a proxy between an email client and a cloud server, allowing an email client to seamlessly use Internet services.
- **IMAP Handler:** IMAP handler is an adapter between an email client and the NDN module. It is responsible to parse IMAP requests and return IMAP responses.
- **NDN Module:** NDN module provides core functionalities for email access in NDN networks and protocol translation between IMAP and NDN communication.

The NDN module is the key to “NDNize” an email application, so that an email client can access emails from anywhere. In the following subsections, we summarize the general steps to “NDNize” an existing application, and we describe the design details of mailSync accordingly. Meanwhile, we identify the differences between TCP/IP and NDN applications.

B. Protocol Translation

IMAP not only defines how emails are accessed, but also determines how emails are organized in TCP/IP networks. To translate IMAP into NDN communication, we first define the organization and access methods of emails in NDN networks. Then, we design the mapping between IMAP and the NDN based email access.

In order to simplify this translation, we basically follow IMAP’s email organization structure and access methods in NDN, except that we define smaller, fixed data access units in NDN to improve data consumption utilization (Section III-C). Accordingly, we design the related data naming in NDN (Section III-D). In addition, we describe how data discovery works in mailSync with the protocol translation(Section III-E).

As both IMAP and NDN use pull-based communication model, i.e., an IMAP client sends commands to a server for responses, and an NDN consumer issues Interest to fetch Data, the mapping between IMAP and NDN communication is straightforward. We simply translate IMAP’s command-response exchange to NDN’s Interest-Data exchange. More specifically, as shown in Figure 3, we translate one IMAP command to one or multiple Interest(s), and translate one or multiple Data to one IMAP response accordingly.

IMAP is a connection-oriented and stateful protocol, meaning that an IMAP server maintains sessions and states to provide email access services. Most IMAP commands are only valid in certain states. For example, message fetching command is valid in the *Selected* state, which can only be entered from *Authenticated* state. In contrast, an NDN protocol can achieve connectionless and stateless for both data consumer and producer, because an NDN packet name can explicitly describes the requested data, and both Interest

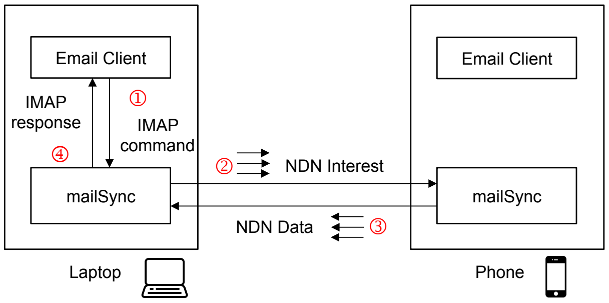

Fig. 3: The mapping between IMAP command-response exchange and NDN Interest-Data exchange

and Data carries signature to ensure integrity. Thus, unlike an IMAP server that has the complexity of query parse for arbitrary items, session maintenances and states verification, an NDN producer (e.g., mailSync) can serve data in a more lightweight and simpler way.

C. Application Layer Framing

As mentioned in Section III-B, we basically follow IMAP’s data organization structure and access methods as part of protocol translation in NDN, except that we apply an important principle, Application Layer Framing (ALF) [10], i.e., an application should break the data into Application Data Units (ADU), and the network should treat ADU as the unit of manipulation, so the network can avoid different expensive representations (e.g., TCP, IP headers), and the network and application processing pipeline becomes more efficient.

IMAP does not apply ALF, because the TCP/IP network does not handle ADU as the unit of manipulation. In IMAP, emails are organized into a hierarchical structure, i.e., each user has multiple mailboxes, each mailbox collects email messages, and each email message has attributes (e.g., flag and text structure) and text (including header and body). As IMAP does not apply ALF, it allows a client to make a request for arbitrary items, such as a subset of objects for a subset of emails. For instance, in Figure 4a, the IMAP request asks for message attributes FLAGS, RFC822.SIZE and ENVELOPE of emails with sequence number 1 to 3. In Figure 4b, the IMAP request asks for message header components of emails with sequence number 1 to 3.

NDN provides the opportunity to apply ALF, since the network layer is able to take ADU as the unit of manipulation (i.e., Data packets). However, if we translate IMAP directly to NDN communication without ALF, i.e., an NDN Interest name will contain arbitrary information on a range of message sequence numbers and a subset of requested items. Accordingly, NDN Data packets with these names are generated after receiving the Interest, and they will be cached anywhere. It results in two drawbacks: 1) an NDN producer is unable to generate all possible Data packets in advance, as an Interest can query an arbitrary combination of objects, 2) the utilization of cached Data packets is low, even if an Interest is requiring for a subset of one cached Data, the cached Data is not matched. In contrast, if we apply ALF to an NDN application, i.e., ADUs should be predefined and generated as NDN Data packets, and one Interest is issued for one (part of) ADU, the data consumption utilization will be improved.

IMAP Request:
A003 Fetch 1:3 (UID FLAGS RFC822.SIZE ENVELOPE)

⇕

NDN Interest Names:
…/[Message-1-Name]/status
…/[Message-1-Name]/meta-data
…/[Message-2-Name]/status
…/[Message-2-Name]/meta-data
…/[Message-3-Name]/status
…/[Message-3-Name]/meta-data

(a) fetch email message attributes

IMAP Request:
A004 fetch 1:3 (BODY.PEEK[HEADER.FIELDS (From To Cc Bcc Subject Date Content-Type Reply-To)])

⇕

NDN Interest Names:
…/[Message-1-Name]/Header
…/[Message-2-Name]/Header
…/[Message-3-Name]/Header

(b) fetch email text header

Fig. 4: Examples of translating an IMAP fetch command to multiple Interests fetching ADUs in NDN

Then how to define ADUs in NDN? Our definition is based on the semantics, granularity and producer of data. Specifically for email access in NDN, each email is constructed of several ADUs. Within an email, we preserve the structure and boundaries of existing email text structure (i.e., Internet Message Format [11]), meaning that email header is an ADU, (one part of [1]) email body is another ADU. Apart from email text, IMAP defines email message attributes. Depending on the producer of email message attributes, we separate them into two ADUs, one describes the status of an email in a mailbox (e.g., Flags, Message Sequence Number etc.) and one describes an email message itself (e.g., size, body structure etc.). We name the former *status* and the latter *meta-data*. Figure 4 shows two examples of translating an IMAP fetch command to multiple Interests fetching ADUs in NDN.

Note that we could use a smaller granularity to define ADU in email access, e.g., we could name each message attribute and each header field individually as ADU. However, it does not improve data consumption utilization but brings more overhead for each ADU (i.e., name and signature). Thus we define ADU in this proper granularity, including status, meta-data, header, and body for each email message.

D. Naming

Naming is essential to an NDN application, because it represents how data are organized and accessed. We design the naming of mailSync in Figure 5. It is based on how IMAP organizes and accesses emails, except that we define ADUs within each email. Specifically, we use */mailSync* as the prefix for forwarding purpose, then followed by the *username* name component that represents a user's email address. The *object* name component describes an object to be access, such as a mailbox (e.g., INBOX), or a message which is uniquely identified by the a tuple <*mailbox, UIDVALIDITY* [2], *UID*> defined in IMAP. The *IMAP_COMMAND* name component preserves the method that IMAP accesses an object. As discussed in Section III-C, data are accessed as ADUs in NDN, and each ADU is described as *ADU* name component.

Both *timestamp* and *segment* components are optional depending on context. Those data that can be updated has timestamp to indicate its version, such as the status of a mailbox, accessed by the SELECT command in IMAP, is often updated (e.g., a new email is added, the FLAG of an email is set to read, etc.). Those data that will never be modified after being produced does not need a timestamp, such a message text. When an ADU cannot fit into one NDN Data packet, it is segmented into multiple Data packets with segment numbers, e.g., an email body with a long text. In contrast, the meta-data of a message can always fit into one Data packet.

We design mailSync naming in this way to adapt to IMAP translation. An alternative pure data-centric naming can focus on message text itself instead of the data organization structure IMAP defines (i.e., the hierarchy of username, mailbox and message). The pure data-centric naming can achieve higher efficiency on data consumption, because the same message sent to a mailing list avoids different names due to different mailboxes in the current naming design. However, the alternative naming requires a different approach than IMAP to access data, making protocol translation challenging. This is a major difference between designing an pure data-centric NDN naming and an adaptive NDN naming for legacy applications.

E. Data Discovery

An application should be able to discover if desired data exists in a network. This ability is achieved differently in TCP/IP and NDN networks. In TCP/IP networks, data discovery is conducted in two steps: an application first discovers the end point that hosts data (e.g., an IMAP server), then it uses an application protocol (e.g., IMAP) to discover and fetch data on the end point. The first step is normally achieved through DNS lookups, and it requires an application to be aware of data location name in advance.

However, in NDN networks, data discovery and fetching are built-in functionalities, because NDN names data, uses pull communication model, and provides loop-free and name-based stateful forwarding. In other words, assume NDN nodes have physical connectivities to form a local area network, an NDN consumer can simply broadcast an Interest with the name of

[1]Email body can be defined as Multipurpose Internet Mail Extensions (MIME) type, which can have multiple parts, and we use the boundaries between parts to define ADUs

[2]UIDVADILITY is similar to the version number of a mailbox. Each email has a different UID in a different version of mailbox. The combination of mailbox name, UIDVALIDITY, and UID refers to a single immutable email message on a server forever.

Data Naming in mailSync:

/mailSync/<username>/<object>/<IMAP_COMMAND>/<ADU>/[#timestamp]/[#segment]

E.g., (1) /mailSync/philoliang@cs.arizona.edu/<mailbox>/SELECT/20180101000000
(2) /mailSync/philoliang@cs.arizona.edu/<mailbox, *>/FETCH/UID/20180101000000/#seg=1
(3) /mailSync/philoliang@cs.arizona.edu/<mailbox,UIDVADILITY,UID>/FETCH/status/20180101000000
(4) /mailSync/philoliang@cs.arizona.edu/<mailbox,UIDVADILITY,UID>/FETCH/metadata
(5) /mailSync/philoliang@cs.arizona.edu/<mailbox,UIDVADILITY,UID>/FETCH/body/#seg=1

Fig. 5: Data Naming in mailSync

desired data to the network; then the matched Data packet can be found and returned back (if exists). Thus, the key of data discovery for an NDN application is to figure out data names instead of data host location.

An NDN application can achieve name discovery through mechanisms such as using manifest and namespace synchronization. As mailSync preserves IMAP methods to access data, name discovery is achieved by a manifest alike mechanism along with the protocol translation. In IMAP, after an email client logs in to a server, it first sends a SELECT command to select and get the status of a mailbox, including the UIDVADILITY, the number of emails, etc. Then the client can use wild character asterisk (*) to fetch objects of all emails. mailSync uses the same mechanisms to retrieve the names of all emails.

Whenever the status of a mailbox is changed, e.g., a new email message is downloaded, a new NDN Data packet will be accordingly generated under the SELECT command namespace with new timestamp (Example (1) in Figure 5). As an NDN consumer is unaware of the timestamp name component, it constructs an Interest with the same name except the timestamp name component, and sets the selector *MustBeFresh* to true. With the name and selector, this Interest is able to fetch the latest status of a mailbox. Similarly, mailSync uses wild character asterisk (*) and the selector MustBeFresh to fetch UIDs of all latest emails (Example (2) in Figure 5). After that, mailSync is able to construct the names of all emails, and to fetch ADUs (Example (3-5) in Figure 5).

F. Security Management

IMAP ensures security by adopting typical security mechanisms for the client-server application model. Specifically, an IMAP client can only access data from a server after it is authenticated by the server. In addition, IMAP uses SSL/TLS to provide data confidentiality and integrity for communication sessions between client and server. However, in an NDN network, data can be stored at and fetched from anywhere, so these security mechanisms are not suitable for NDN applications. In fact, we take data-centric security mechanisms for mailSync, and here are several considerations:

Integrity: In NDN, data integrity can be verified by itself, since both Interest and Data packets can be signed with signature. We use RSA signing in mailSync, and sign all Data packets using the application's private key, so that a consumer is able to verify the Data using the application's public key. Then, *how to fetch the public key?* The signature of signed Data contains the name of a certificate, that can help a consumer to verify the Data. A certificate is a Data packet, that contains a public key and is signed by another private key. Next, *how to trust the fetched certificate?* It depends on an application's trust model.

Trust Model: A trust model defines trust relations in a system. Regarding mailSync's trust model, we assume that mailSync are running on trusted devices that a user has access to, thus mailSync can share the same trust anchor. More specifically, users issue certificates for all their devices' public key, a device issues certificate for mailSync's public key and data belongs to mailSync can only be signed by mailSync's private key. Following the trust model, one mailSync can verify the data of another mailSync through the trust chain. In addition, we use trust schema [12] to ensure that the signing relationships follow the trust model, so that data (including keys) can only be signed by legitimate keys.

Content Confidentiality and Access Control: In NDN, data confidentiality and access control are achieved at the same time by using encryption. Unlike the case in client-server model that data can only be accessed after the session is authenticated, data in an NDN network can be cached anywhere and be fetched by anyone. Thus, access control in NDN is achieved by encrypting data content and distributing the decryption key to those entities who can access data content. Our access control policy is simple, i.e., legitimate users can access all of their data in mailSync. Thus, mailSync maintains a symmetric key (for both encryption and decryption) for each user. How to maintain the same symmetric key is a general challenge in a distributed system. As our target scenario is simple, i.e., any one of user devices that first downloads emails from a server, and other user devices can sync with it. We could apply two approaches to achieve the goal: 1) each mailSync instance uses the same seed and mechanisms to generate the same symmetric key, e.g., using email account password with the same cryptographic hash. 2) we maintain a symmetric key for each user at the cloud server, and when mailSync access emails from the server, it synchronizes the symmetric key with the server; as a result, one device that has the latest emails from the server, also has the latest version of symmetric key, and other devices can sync the symmetric key with it before email access. For the second approach, we describe the symmetric key synchronization in the Bootstrap procedure.

Name Obfuscation: As the name of Interest and Data may carry sensitive information, various name obfuscation mechanisms are applied to protect privacy, such as hashing and encrypting name components. In our design, mailSync share a symmetric key for content encryption, which can also be used to encrypt the name components after the first one (/mailSync) for obfuscation purpose.

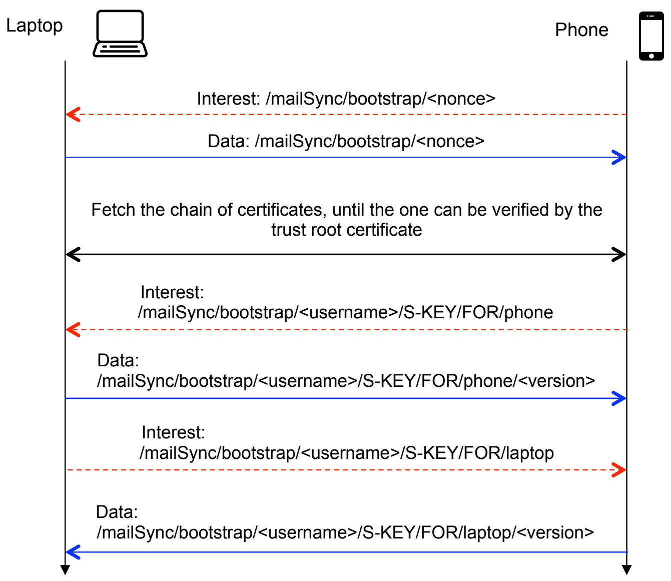
Laptop
Phone
Interest: /mailSync/bootstrap/<nonce>
Data: /mailSync/bootstrap/<nonce>
Fetch the chain of certificates, until the one can be verified by the trust root certificate
Interest: /mailSync/bootstrap/<username>/S-KEY/FOR/phone
Data: /mailSync/bootstrap/<username>/S-KEY/FOR/phone/<version>
Interest: /mailSync/bootstrap/<username>/S-KEY/FOR/laptop
Data: /mailSync/bootstrap/<username>/S-KEY/FOR/laptop/<version>

Fig. 6: Bootstrap procedure: exchanging certificates and synchronizing the symmetric encryption key

Bootstrap procedure: Based on these considerations, two mailSync instances have to exchange certificates and synchronize the symmetric encryption key before email access. As shown in Figure 6, we design the bootstrap procedure to handle these tasks. One mailSync instance first issues a signed Interest with the name */mailSync/bootstrap/<nonce>*, and another instance responds to it with a signed Data packet. This first Interest-Data exchange is a starting point for both sides to fetch the chain of certificates, including the root certificate. Again, both Interest and Data are signed by mailSync's private key and the signature contains the name of the verifying certificate. After both sides exchange certificates, the next step is to synchronize the symmetric encryption key with each other. For example in Figure 6, mailSync on the phone first issues an Interest with the name */mailSync/bootstrap/<username>/S-KEY/FOR/phone*. The name components *<username>* and *S-KEY* indicates the said user's symmetric key, while the name components */FOR/phone* tells the producer to encrypt the symmetric key using phone's public key, so that only the phone can decrypt the symmetric key using its private key. Similarly, mailSync on laptop fetches the encrypted symmetric key from the phone. With the version number in the Data name, both sides are able to choose the latest version of symmetric key.

IV. Conclusions

This paper studies how to enable an existing TCP/IP application to access data from anywhere using NDN. We intend to achieve the goal with little modification to existing applications. To find a general solution, we conduct a case study of email access, which is a wildly used application with standardized protocols. Moreover, we propose mailSync, a framework that allows an existing email client to access emails either from a cloud server via TCP/IP, or directly from a peer via NDN. We use the design of mailSync to generalize the steps of how to "NDNize" an existing application, including protocol translation, application layer framing, naming, data discovery and security management. In addition, we identify the differences between TCP/IP and NDN applications. We implement a prototype of mailSync in Java and Android as a proof of concept. In future, we plan to study applications that use different protocols.

V. Acknowledgments

This work is partially supported by the National Science Foundation under awards CNS-1345142, CNS-1513505, and CNS-1629009.

References

[1] A. van der Linde, P. Fouto, J. Leitão, N. Preguiça, S. Castiñeira, and A. Bieniusa, "Legion: Enriching internet services with peer-to-peer interactions," in *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2017, pp. 283–292.

[2] P. Gusev and J. Burke, "Ndn-rtc: Real-time videoconferencing over named data networking," in *Proceedings of the 2nd International Conference on Information-Centric Networking*. ACM, 2015, pp. 117–126.

[3] S. Mastorakis, A. Afanasyev, Y. Yu, and L. Zhang, "ntorrent: Peer-to-peer file sharing in named data networking," in *Computer Communication and Networks (ICCCN), 2017 26th International Conference on*. IEEE, 2017, pp. 1–10.

[4] H. Zhang, Z. Wang, C. Scherb, C. Marxer, J. Burke, L. Zhang, and C. Tschudin, "Sharing mhealth data via named data networking," in *Proceedings of the 3rd ACM Conference on Information-Centric Networking*. ACM, 2016, pp. 142–147.

[5] A. Afanasyev, Z. Zhu, Y. Yu, L. Wang, and L. Zhang, "The story of chronoshare, or how ndn brought distributed secure file sharing back," in *Mobile Ad Hoc and Sensor Systems (MASS), 2015 IEEE 12th International Conference on*. IEEE, 2015, pp. 525–530.

[6] I. Moiseenko and D. Oran, "Tcp/icn: Carrying tcp over content centric and named data networks," in *Proceedings of the 2016 conference on 3rd ACM Conference on Information-Centric Networking*. ACM, 2016, pp. 112–121.

[7] T. Refaei, J. Ma, S. Ha, and S. Liu, "Integrating ip and ndn through an extensible ip-ndn gateway," in *Proceedings of the 4th ACM Conference on Information-Centric Networking*. ACM, 2017, pp. 224–225.

[8] Z. Li, J. Bi, and S. Wang, "Http-ccn gateway: Adapting http protocol to content centric network," in *Network Protocols (ICNP), 2013 21st IEEE International Conference on*. IEEE, 2013, pp. 1–2.

[9] M. Crispin, "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1," RFC 3501 (Proposed Standard), Internet Engineering Task Force, Mar. 2003, updated by RFCs 4466, 4469, 4551, 5032, 5182, 5738, 6186. [Online]. Available: http://www.ietf.org/rfc/rfc3501.txt

[10] D. D. Clark and D. L. Tennenhouse, "Architectural considerations for a new generation of protocols," in *ACM SIGCOMM Computer Communication Review*, vol. 20, no. 4. ACM, 1990, pp. 200–208.

[11] P. Resnick, "Internet Message Format," RFC 2822 (Proposed Standard), Internet Engineering Task Force, Apr. 2001, obsoleted by RFC 5322, updated by RFCs 5335, 5336. [Online]. Available: http://www.ietf.org/rfc/rfc2822.txt

[12] Y. Yu, A. Afanasyev, D. Clark, V. Jacobson, L. Zhang *et al.*, “Schematizing trust in named data networking,” in *Proceedings of the 2nd International Conference on Information-Centric Networking*. ACM, 2015, pp. 177–186.