NDNizing Existing Applications: Research Issues and Experiences

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

Ju Pan
The University of Arizona
pjokk722@email.arizona.edu

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

ABSTRACT

A major challenge to potential ICN/NDN deployment is the requirement of application support, namely, applications need to be rewritten or modified in order to run on NDN networks *and* receive the full benefits. Using a proxy to translate between an application-level protocol and NDN offers a viable solution that balances between development cost and architectural benefits. In this paper, we study on the questions of how to facilitate and incentivize the development and deployment of such protocol translation proxies. We propose to enable existing applications to communicate "off the grid", i.e., using only local network connectivity without the global Internet, by translating between conventional client-server protocols and NDN. This provides deployment incentives by enabling a useful feature with no or minimal changes to existing applications. By giving the experience of a few protocols, we hope to abstract out some common design patterns that can be reused in developing other application-level proxies. This paper reports our work on IMAP/NDN translation for local email access and XMPP/NDN translation for local group chat. Based on this work, we identify and discuss a number of common design issues including application-level framing, namespace design, application protocol semantics, multiparty synchronization, security and privacy, and real-world deployment challenges.

CCS CONCEPTS

• **Networks** → **Network architectures**; **Application layer protocols**;

KEYWORDS

ICN; NDN; Proxy; Off the grid; Application Translation; ICN Deployment

ACM Reference Format:
Teng Liang, Ju Pan, and Beichuan Zhang. 2018. NDNizing Existing Applications: Research Issues and Experiences. In *5th ACM Conference on Information-Centric Networking (ICN '18), September 21–23, 2018, Boston, MA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3267955.3267969

1 INTRODUCTION

The core innovation in Information Centric Networking (ICN), including Named Data Networking (NDN) [35][1], is the use of application-level names as the identifier for network packets. This allows the network to identify data independent of a particular connection, and to retrieve desired data from anywhere instead of a single given host. While this design decision underpins the architectural benefits of NDN, it also comes with some challenges. The one challenge that stands out as a major roadblock to NDN deployment is the need of application support. In order to run on NDN networks *and* get the full benefits of the architecture, applications need to inform the network about the names of the data they are requesting or providing. Since existing applications are not designed to use this paradigm, they cannot run on NDN networks.

To the best of our knowledge, there are three approaches to solve this problem: (1) develop native NDN applications from scratch, which will enjoy the full benefits of NDN but incur the most development cost; (2) run a proxy that translates between TCP/UDP/IP and NDN, enabling existing applications to run on NDN networks without any change, but receive only limited benefits from NDN; and (3) run an application-level proxy that translates between the application's native application-level protocol and NDN, which incurs less development cost than the first approach and receives more benefits than the second approach.

Our research studies the third path. By investigating the approach of application-level protocol translation, we notice two major research questions. The first question is that, given there are so many application-level protocols, how do we minimize the effort of developing each translation proxy? Our hypothesis is that there are many issues, such as namespace design, name discovery, and security, common to most or all applications. If the solutions can be abstracted into a few key design principles or design patterns, we can significantly reduce the effort needed for developing each translation proxy.

The second research question we consider is how to create incentives for developers to upgrade their applications, even before any NDN deployment occurs in the network. To jumpstart upgraded application deployment, we need to find use cases where updated applications will enjoy significant benefits even before any existing network element has been upgraded.

In this paper, we propose to use the capability of "off-the-grid" communication to incentivize initial upgraded application deployment and hope to come up with a number of reusable design patterns to aid the development of application-level translation. Today's applications predominantly rely on cloud servers and the Internet connectivity to reach those servers. However, there are

[1]The work presented in this paper is done under NDN, but many discussions apply to other ICN designs as well.

scenarios where global Internet connectivity is poor or even not available, but local network connectivity is adequate - for example, after a natural disaster or onboard an airplane. In these scenarios, most existing applications will stop working, but an NDN application would be able to work to the fullest extent allowed by the network connectivity, discovering and retrieving data between two locally connected devices without going through the cloud. Translating existing application-level protocol and NDN through transparent proxies will allow existing applications to communicate seamlessly off the grid. This will not require any change to the application, nor any change to the network. Therefore it is a reasonable starting point for deploying NDN at applications and end hosts.

As a concrete basis for our work, we design IMAP/NDN translation for local email retrieval between two end devices, and XMPP/NDN translation for local group chat. Through these efforts, we identify key common design issues, including application-level framing, namespace design, application logic translation, security, privacy, and real-world deployment challenges. For each issue, we outline our design for the specific protocol, and discuss the rationale behind our design choices, and draw generalizable conclusions. Though we do not have complete solutions for all the issues yet, we hope the discussion will shed light on how to develop application-level translation in general, and lead to common design patterns that can be reused in translating other application protocols. Even for applications that use proprietary protocols, this general understanding and design patterns will still help developers convert such applications.

The rest of the paper is organized as follows. Section 2 further elaborates the rationale behind this work. Section 3 gives a system overview for application-level translation, and Sections 4, 5 and 6 provide more details and discussions on various design issues. Section 7 introduces the evaluation details, Section 8 is the related work, and Section 9 concludes the paper.

2 DESIGN RATIONALE

NDN was proposed as a *future* Internet architecture, in the sense that it is a fundamental change to the basic network service semantics, and it impacts almost all aspects of network systems. Research so far has been mostly focused on why this fundamental change is desirable, and how to do it efficiently. Past research has demonstrated and quantified many benefits enabled by the new architecture, which includes in-network caching, native anycast and multicast, resilient and efficient data retrieval [32], DDoS resistance [11], schematized trust [33], and data-centric security [37]. There are also many advances in performance optimization of various NDN mechanisms, such as fast and scalable name lookups [26], efficient caching policies [38], secure and scalable routing protocols [14], and so on. With a simulator ndnSIM [1], a software prototype NFD [27], and various libraries [28], NDN has been studied in many different network environments like Internet of Things [22], vehicular networks [12], building management and automation [23], in addition to the traditional Internet.

With all this progress, it is time to think about NDN's deployment: its potential paths, roadblocks, and possible solutions. This is *not* saying that NDN is mature and ready for large-scale deployment now, but rather, it is a start of research on deployment-related issues, which are as important to NDN's future as the aforementioned performance-related issues. In the short term, this research will help increase NDN experiments and trials in the real world to advance NDN research; in the long run, it will prepare us for the actual deployment, or at least deeper understanding of what it takes to make NDN a reality.

2.1 Challenges in Deployment

Full deployment of NDN requires upgrading the network and the applications. The need for network support is clear: at the minimum, network devices need to change from address-based packet processing to name-based packet processing; on top of that, they may incorporate storage for caching, different forwarding strategies for different contents or network environments, and some security measures. The core research question is how to make these changes *incrementally*. More specifically, how to make NDN devices co-exist with IP devices in the same network without interfering with each other in packet forwarding, how to ensure there are benefits for the first mover or initial deployment, and how to ensure incremental deployment will lead to increasing benefits. These are typical issues in deploying new network protocols and there is already work in this area for ICN/NDN. For example, Cisco's hICN [20] encodes names as IPv6 addresses to allow hICN packets to be processed by both upgraded and legacy routers, and the benefit of initial deployment is improved performance for content dissemination and streaming. For another example, Zhang et. al. [31] proposed a dual-stack scheme for NDN switches and IP switches to co-exist in local area networks, improving path diversity and redundancy.

The need to incentivize application support, however, has not been elaborated to the best of our knowledge. The core innovation of NDN is to use application-level names to name network packets. This allows the network to identify data independent of any address, port number, or connection. Thus the network can retrieve desired data from anywhere via any path, which is the foundation for all the benefits of the NDN architecture. However, sharing the same namespace between application and network also means that the interface between application and network has to change. Existing APIs such as the socket API only allow communication via addresses and port numbers, not via application-level names. This suggests existing applications have to be upgraded in order to run on NDN networks and to acquire full benefits.

Requiring support from both networks and applications is the direct result of NDN's change to the basic network service semantics: from "delivering packets to a destination" to "retrieving data of given names." It makes NDN deployment rather different from most network protocol deployment we have seen. The closest one is perhaps IPv6, which requires upgrading both network devices as well as applications to use 128-bit addresses. However, the application changes required by IPv6 are well-defined and relatively straightforward compared with the changes required by NDN. To upgrade an application for NDN, developers have to consider numerous issues such as namespace design, name discovery, data-centric security, and many others. Because the semantics of name-based API and address-based API are remarkably different, applications need more fundamental changes in order to use name-based API. It is the first time that in-depth application changes are needed in deploying a network architecture. This is the research problem that we will focus on in the rest of this paper.

2.2 Application Support of NDN

There are three approaches to solve the application support problem. They differ in the tradeoff between development cost and architectural benefits.

The most straightforward and complete solution to the application support problem is to rewrite an application from scratch. Developers would give up the traditional thinking of which server addresses are available, which server address to use, how to detect and recover from an unresponsive server address, etc. They would need to rethink the design and implementation from the data-centric point of view, e.g, how to frame the data, name the data, discover names, secure the data, and so on. NDN researchers have done lots of work developing native NDN applications, from simple network diagnosis tools [15], to file sharing [2] and video conferencing [13], to the Internet of Things [13] and augmented reality [3] applications. The advantage of this approach is the full support of NDN and full benefits from the architecture. We believe this will be the solution in the long run, especially when there is already significant deployment of NDN in the network. In the near term, however, the downside of this approach is a large amount of work required to develop such applications. Especially in large-scale complex software, most part of the software is non-network-related, such as user interface and internal data processing logic. Without the high-quality implementation of these parts, users might have no mind to use the application; but rewriting and maintaining these parts will take up most development efforts without significant benefits from using NDN.

Another approach is to run a proxy that translates between TCP/UDP/IP and NDN, such that existing applications can run over NDN networks without any change. A number of such proxy designs were presented in [17, 21, 25]. While this approach offers the best backward compatibility in that no application change is needed at all, the translation has to sacrifice some of the architectural features. For example, to translate a TCP packet into an NDN packet, the proxy needs to come up with a name for the resulting NDN packet. Based on information from TCP/IP headers, the proxy can make up a name uniquely identifying the packet, but the name will be associated to the specific TCP connection. This provides benefits such as caching within the same TCP connection (e.g., efficient retransmission of lost packets), but cannot support caching across different TCP connections (e.g., multicast to different consumers). Furthermore, security policies and trust models have to be defined based on addresses and port numbers, not application-level data names.

The third approach, a midpoint between the first two, translating between application-level protocols and NDN. For example, translation between HTTP and NDN can enable all HTTP-based applications to run on NDN without changes. The translation does not change existing applications' implementation, and incurs much less development cost than rewriting the application. The translation will also leverage the information in the application-level protocol header to generate meaningful data names across different network connections, which allows this approach to receive much more architectural benefits than running a translating proxy. Therefore, in the early stage of NDN deployment, we think this approach makes the best cost-benefit tradeoff.

Looking further into this approach, we notice two major research questions. The first question is that, given there are various application-level protocols, how do we minimize the effort of developing each translation proxy? Our hypothesis is that there are many issues, such as namespace design, name discovery, and security, that are common to most or all applications. Thus, after examining a few protocols, we might be able to summarize some design principles or design patterns, which will significantly reduce the effort needed for developing each translation proxy. For this purpose, we have chosen IMAP/NDN translation for local email access and XMPP/NDN translation for local group chat. The reason for choosing these two protocols is they are both public standard, have mature open-source implementations, and support numerous different applications. We use these protocols to demonstrate that, if implemented properly, this level of translation can enable NDN benefits the existing applications with only trivial configuration changes.

The second research question is that even if it is feasible and manageable, what are the incentives for developers to upgrade their applications for NDN before any NDN support being deployed to the network? This is important to break the 'chicken-and-egg' problem of which one to deploy first, network or application? To answer this question, we need to search for use scenarios in which upgraded applications can enjoy certain benefits even without infrastructural support. If we can achieve this, it will be possible for the deployment to start from the edge, i.e., the applications and end hosts, and as the demand increases, it will drive the deployment gradually to the network. The use scenario we identified is the so-called "off-the-grid" communication, to be elaborated in the next subsection.

2.3 Off-the-grid Communication

Most of today's applications are built on top of TCP/IP's point-to-point communication, a good match with the client-server model in which the server is responsible for providing most functionality, including serving the data, securing the data, controlling access to the data and so on. With the rising trend of cloud computing, most of today's applications have become dependent on cloud servers, hence the Internet connectivity to reach those servers. Without such Internet connectivity, applications simply do not work.

In real life, however, there is a number of scenarios where the Internet connectivity is inadequate, limited, or not even available, but local connectivity exists and allows device-to-device communication without going through the Internet. For example, after a disaster crippling the communication infrastructure, or when the user is onboard a cruise ship or an airplane, there is no Internet connectivity available, but it is trivial to turn a laptop or smartphone into a WiFi access point and have other devices communicate through this local WiFi network. Such local connectivity permits a single user to share emails, notes, and pictures between his/her phone and laptop, or a group of users to have a group chat. But existing applications cannot take advantage of the local connectivity since it does not allow them to connect to the cloud server which is essential to most of the modern applications. An example of limited Internet connectivity is when a user tethers a laptop to a smartphone to gain Internet access through the phone's LTE, in which case the user may want to minimize the traffic going through LTE, e.g., downloading emails only once but storing them at both the phone and the laptop. The existing apps download the same emails twice, one copy for the phone and the other copy for the laptop.

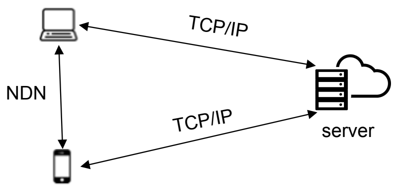

Figure 1: Off-the-grid Communication in NDN

It is clear that the traditional point-to-point client-server model does not meet the demands of increasing user mobility and local connectivity (e.g., WiFi, Bluetooth, Zigbee). No doubt that TCP/IP applications can be modified to use local connectivity. For instance, Apple's Airdrop can share files between Apple devices using Bluetooth. The amount of work needed to make such a change, however, is not trivial since a number of issues need to be resolved. Basically, in order to work in an unknown local network, an application needs to be able to (1) discover available data sources or services that it is interested in, (2) exchange data with potentially multiple data sources and sinks, and (3) ensure data integrity, security, privacy and access control without the cloud server. These issues pose significant challenges to application developers to devise solutions from scratch and contribute to the fact that majority of today's applications do not work with local connectivity only.

Named Data Networking (NDN) is a natural fit for off-the-grid communication (Figure 1). It allows applications to retrieve data from anywhere without having to specify a server address or hostname. Using application names in network packets, NDN facilitates service discovery, in-network caching, multicast, data integrity, and security. If every application was built on top of NDN, it would not care whether there is an Internet connectivity; it would simply work to the fullest extent allowed by the available connectivity. For example, on an airplane, a laptop would not be able to retrieve emails from the server but would have no problem retrieving emails from the user's phone. Therefore NDN presents the architectural advantage in supporting off-the-grid communication, and that is what we chose to enable by conducting application-level translations.

3 SYSTEM OVERVIEW

This work focuses on two use scenarios. The first scenario is to share emails using only the local network connectivity between two devices, e.g., a smartphone and a laptop, owned by the same user. For example, users may download emails through the Internet to the phone's local storage before boarding. During the flight, they want to work on these emails on the laptop, and the emails can only be fetched from the phone. Another example is that when the laptop is tethered to the phone for Internet access, the user wants to download emails to both devices but only costs one share of LTE data usage. In the second use scenario, multiple users are connected locally but not to the Internet, and they want to be able to chat through XMPP. In both scenarios, we assume the end devices are the only places that have deployed NDN.

As Figure 2 shows, all the functionality is achieved by running an *NDN Proxy* on each end device. Applications only need to be configured to connect to the local Proxy instead of a remote server, i.e., the IMAP server or the XMPP server. While the main job of the Proxy is to translate between application protocols, IMAP and XMPP in our scenarios, and NDN, it also needs to support communication to the cloud server when the Internet is accessible. The major components in NDN proxy include:

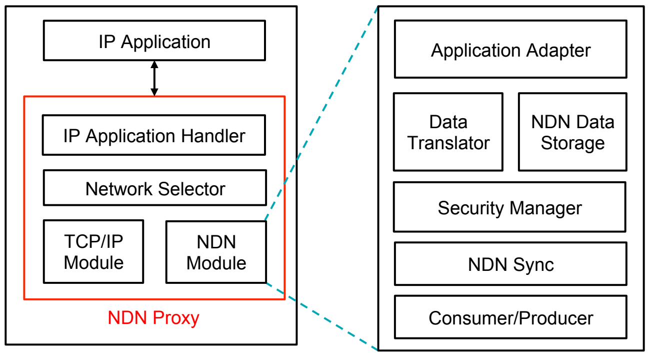

Figure 2: The NDN Proxy Architecture

- **IP Application Handler:** this module is responsible for communicating with the application through the application protocol.
- **Network Selector:** this module decides whether to use TCP/IP to reach cloud server or use NDN over the local network. The default setting is to access cloud server first, and if not reachable, turn to NDN over the local network. This behavior can be configured independently.
- **TCP/IP Module:** this module is responsible for the TCP/IP side of communication, i.e., accessing the cloud server, for backward compatibility purpose.
- **NDN Module:** this module provides core functionalities of protocol translation.

Within the NDN module, the proxy needs to translate application contents from/to NDN data packets (*Data Translator*) and store them in (*Data Storage*) for later sharing. This step requires application-layer framing of the data and the proper namespace design. The *Application Adapter* translates application requests to NDN Interest and translates NDN Data to application responses. The actual NDN traffic is done either through direct Interest-Data exchange (the *Consumer/Producer module*), or over the *NDN Sync* module, depending on the pattern and semantics of the communication. Finally, the *Security Manager* handles security, trust, and privacy issues.

4 DATA TRANSLATION

Before any network communication, we need to define what builds a Data packet and the name of each Data packet.

4.1 Application Layer Framing

Proposed by Clark et. al. in 1990 [5], Application Layer Framing (ALF) is a design principle that says applications should break the data into Application Data Units (ADUs), and the network should treat ADU as the unit of manipulation. With ALF, packet processing through the entire system is more efficient because packet repackaging at different layers is eliminated. However, since TCP doesn’t preserve ADU boundaries, existing applications usually do not implement the ALF principle.

In NDN, each Data packet has its unique name and a signature that binds the name, the payload, and the signing key. This structure is more rigid and suitable for ALF. The reasons are: first, since the name is the application-level name, it is necessary to have the payload also to be a unit semantically meaningful at the application layer, an ADU. Second, since composing a packet involves signing overhead, it would be better to prepare a packet only once and reuse it as much as possible. Third, reusing ADUs avoids signing the same data multiple times, and increases the chance of the cache hit.

Without the implementation of ALF, in order to retrieve data, existing client-server based applications allow clients to send requests to a server, and the server will push all requested data back. To translate this procedure into NDN, whether or not to apply ALF in NDN results in different translation designs. We give two concrete examples to translate IMAP request-response for data retrieval:

- **Without applying ALF in NDN:** as shown in Figure 3, without applying ALF, the NDN proxy can encapsulate any application request in an NDN Interest packet and any application reply in one or more NDN Data packet(s). This makes protocol translation trivial to achieve, but it incurs more overhead due to the process of signing packets on the fly, few cache hits, and one more round trip time if the response is divided into more than one Data packet.
- **With applying ALF in NDN:** as shown in Figure 4, this design reorganizes application data into ADUs and reuses them to satisfy Interests. Compared to the first design, this design makes NDN proxy more efficient, because ADUs can be generated as NDN Data packets in advance. Moreover, well-organized ADU is the foundation of running NDN Sync (Section 5.1). One downside of this design is that data fetching becomes less flexible, as one ADU may contain additional information than an application asks for. In addition, it requires an NDN proxy to understand application semantics to accomplish data retrieval translation. Overall we think ALF is a great fit for NDN, especially in the off-the-grid scenarios where no centralized server is available.

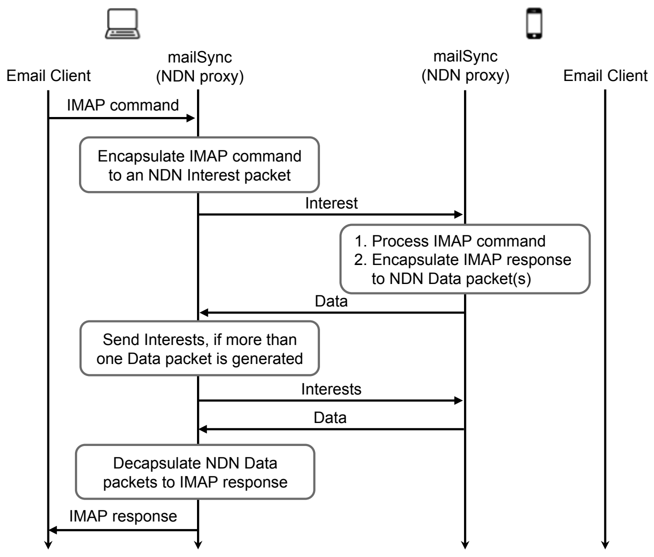
Email Client
mailSync (NDN proxy)
mailSync (NDN proxy)
Email Client
IMAP command
Encapsulate IMAP command to an NDN Interest packet
Interest
1. Process IMAP command
2. Encapsulate IMAP response to NDN Data packet(s)
Data
Send Interests, if more than one Data packet is generated
Interests
Data
Decapsulate NDN Data packets to IMAP response
IMAP response

Figure 3: To encapsulate/decapsulate IMAP to/from NDN without defining Application Data Units (ADU)

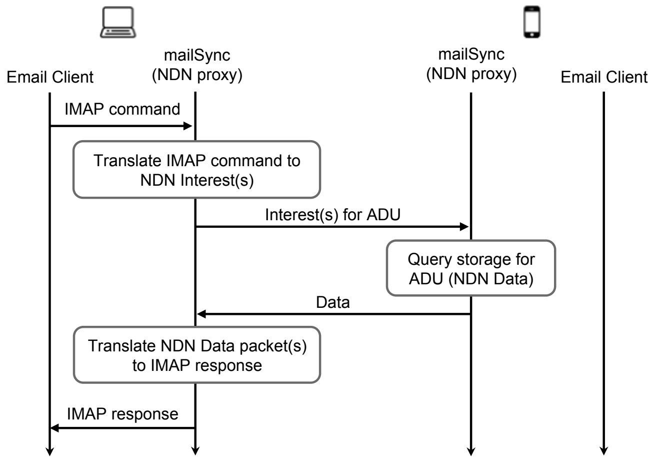
Email Client
mailSync (NDN proxy)
mailSync (NDN proxy)
Email Client
IMAP command
Translate IMAP command to NDN Interest(s)
Interest(s) for ADU
Query storage for ADU (NDN Data)
Data
Translate NDN Data packet(s) to IMAP response
IMAP response

Figure 4: To translate IMAP to/from NDN with defining Application Data Units (ADU)

4.2 Defining Application Data Units (ADUs)

The main design issue in adopting the ALF principle is to define ADUs. Here we use IMAP/NDN translation to illustrate that the ADU formation (Figure 5) should consider **application semantics**, **data producers** and **data granularity**.

First of all, NDN ADUs should be semantically meaningful application data blocks. For example, the left part of Figure 5 shows the basic IMAP data structure, which is the fundamental of forming various ADUs and naming them. However, the problem still exists within this structure: it does not differentiate information created by different producers, because email clients always fetch emails from one centralized server. For example, the attributes of an email message containing information specific to a mailbox, such as whether messages have been read, which for the same email can be different on different devices. The attributes also contain information related to the email message itself, such as message size and body structure. The former information should be created and signed by a mailbox, and the latter one should be created and signed by the composer of the email. Mixing them in the same ADU would make security and trust decisions extremely difficult in NDN. Therefore we split IMAP message attributes into two different ADUs, *status* and *metadata*, which correspond to the two kinds of attribute information mentioned at the beginning of the paragraph.

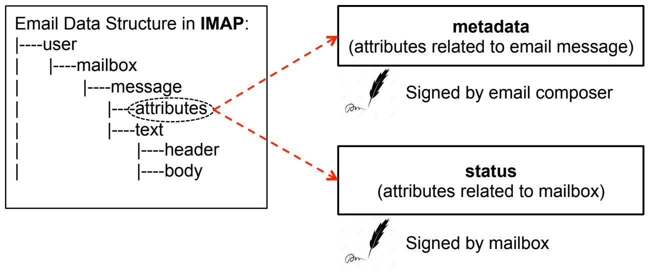

Figure 5: Basic Email data structure in IMAP and the two different data producers of message attributes

The third factor to consider is the data granularity. An ADU should be large enough to make the overhead of the packet header and signing operation reasonable, and small enough to reduce the unnecessary information when it's reused to answer future Interests. Take Figure 6 as an example, making every individual attribute of an IMAP message an ADU would be too fine-grained, incurring the significant overhead of packet header and signing operation. Put all these together, Figure 7 shows how we define NDN ADUs for IMAP data.

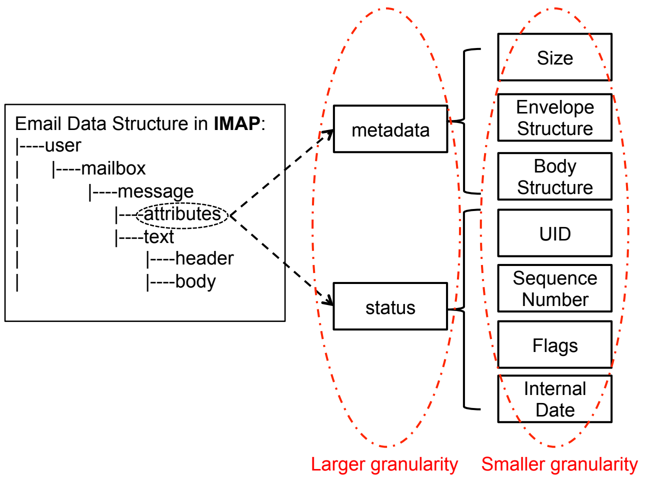

Figure 6: Defining each individual attribute as an ADU incurs significant overhead

4.3 Namespace Design

Once ADUs are defined, the following critical issue is how to name the data. Namespace design is a complex issue: the same set of data can be organized and named in different ways depending on how applications are designed to access the data.

In IMAP/NDN and XMPP/NDN translations, they adopt different data access logics to determine their namespace designs. Figure 7 shows the namespace design for IMAP/NDN translation. In this translation, we follow IMAP's client-server data access logic, because the logic that clients retrieving emails from one server is suitable in our scenario where one device retrieve emails only from the other one. XMPP has more complicated data access logics, such as publish-subscribe pattern and multiparty communication. In XMPP/NDN translation, we utilize NDN Sync (Section 5.1) as a basic mechanism to achieve these data access logics, and accordingly design NDN-XMPP namespace in Figure 9.

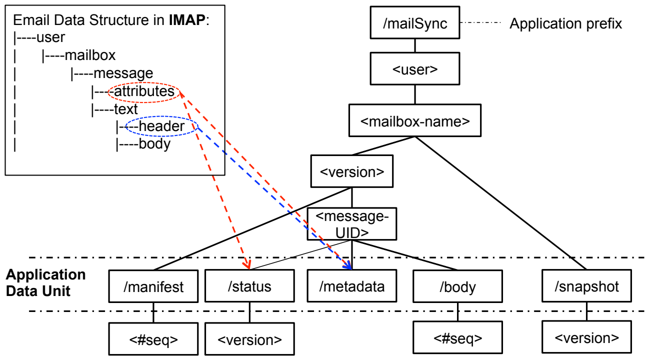

Figure 7: Redefining data structure in IMAP to NDN Application Data Units (ADUs)

Based on XMPP/NDN translation, we abstract one naming schema for Data, and another naming schema for signed Interests, which are often used as commands or negotiation requests (Figure 8). For data names, the first part - *application prefix* - is to distinguish data for different applications, and it should be routable in an NDN local network. The second part is the *data collection prefix*, which reflects how applications organize and access various datasets. For instance, in XMPP/NDN namespace design (Figure 9), we have three such data collection prefixes corresponding to three major application datasets, i.e., the dataset about users, the dataset about one-to-one chats, and the dataset about multi-user chats. Data collection prefixes provide well-structured units for applications to choose to synchronize by applying NDN Sync. The third part of the data name is the *producer name*, which indicates who generates the data. It is necessary when the data collection is contributed by more than one producer. Having this information in the name helps consumers apply security mechanisms or trust schema (Section 6). The last part of the name is *Application Data Unit* name, which is determined by the specific ADUs. Within this high-level naming schema, the actual name components need to include details such as version number, segment number etc.

While the above schema works well for static data in IMAP and XMPP, sometimes there's 'dynamic' data, which is produced by running a command or in response to a negotiation request. For example, an XMPP user sends a friend request to another user. Translating such process to NDN, a user sends a signed Interest to another user with the *Request* part of the name being the XMPP friend request verb (Figure 8). We use this naming schema to support this type of interaction as opposed to data retrieval or synchronization.

4.4 Data Translation Procedure

After defining ADUs and namespace, we are now able to translate application data into NDN packets, so they can be consumed in an NDN network. While the non-modified applications still *send* and *receive* IP packets using existing application protocols, these

1. Data Collection: /<Application Prefix>/<Data Collection Prefix>/<Producer Name>/<ADU Name>/

2. Signed Interest: /<Application Prefix>/<User Name>/<Request>/<timestamp>

Figure 8: Two Generalized Naming Schemas

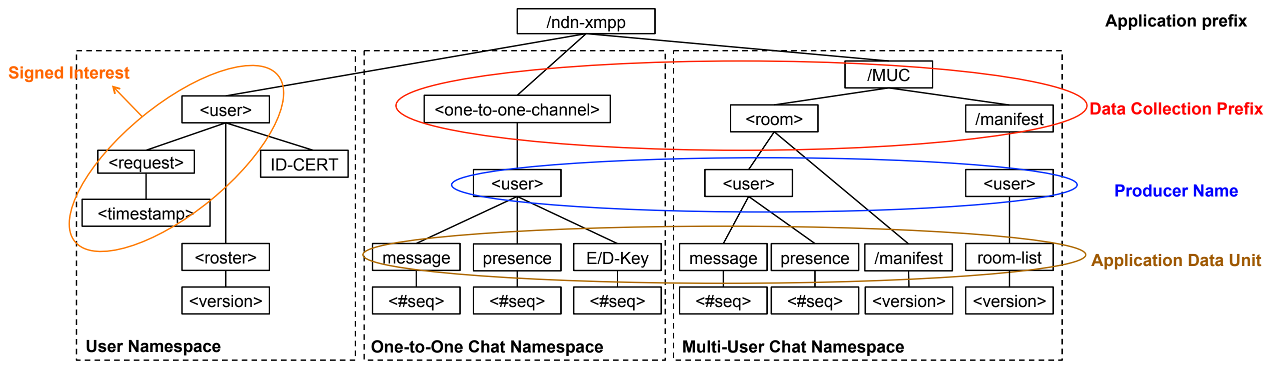

Figure 9: NDN-XMPP Namespace Design

packets are examined at application-level by the NDN Proxy and converted to NDN packets if needed.

When the Internet is available and the client is communicating with the cloud server, the NDN Proxy will relay the traffic to keep the communication going. At the same time, it examines each packet's application protocol header to determine whether the payload constitutes any useful application data. If so, it will construct corresponding NDN packets and save them into the NDN storage. When the application communicates over local connectivity via NDN, the NDN Proxy will already have some Data packets in storage that might be consumed to satisfy Interests. For example, in the IMAP/NDN translation, when the phone downloads emails from a server, the NDN proxy translates the downloaded emails into NDN Data packets to satisfy Interests sent from the laptop; in the XMPP/NDN translation, when a client sends a message to chat room, the message is translated into an NDN Data packet, no matter the Internet is available or not.

5 TRANSLATION OF COMMUNICATION PROTOCOL

After translating application data into NDN packets, this section explains the related issues and solutions in translating communication protocols.

5.1 NDN Communication Patterns

Before going into the details of protocol translation, we first introduce the basic NDN communication patterns that the translations utilize, *Interest-Data Exchange* and *NDN Sync*.

- **Interest-Data Exchange:** One-to-one Interest-Data exchange is the fundamental communication pattern that the NDN network layer follows. Therefore, an NDN application can directly retrieve a Data packet by issuing an Interest to an NDN networking, using the consumer/producer programming model [18]. In addition, Interest-Data exchange builds the foundation for NDN Sync.
- **NDN Sync:** NDN Sync provides the functionalities for NDN applications to reliably synchronize data within the shared namespace among multiple parties, serving as a transport layer. Despite that NDN Sync involves complex mechanisms built on top of Interest-Data exchange and has various designs (Section 8), this work treats NDN Sync as a basic communication pattern in NDN.

5.2 Translating Communication Patterns

Existing application protocols apply various application-level communication patterns, fitting in either **pull** or **push** communication model, such as *request-response* pattern and *publish-subscribe* pattern. To translate these patterns into NDN, we use both Interest-Data exchange and NDN Sync as basic building blocks.

Pull Model. : Request-response pattern is commonly used by clients to send requests, which may include *data retrieval request* and *command request*, such as fetching emails and inviting a friend. To translate such communications, the general approach is to convert client requests to Interests and the server replies with Data. As explained in Section 4.1, the simplest way is to encapsulate each request in one Interest and to encapsulate each reply in one Data. But to translate data retrieval requests, this approach does not fully utilize characteristics of NDN and often incurs significant overhead. We focus more on achieving application objectives instead of converting packets mechanically. Therefore, one data retrieval request/reply may be translated to one or more Interest(s)/Data(s), and vice versa, depending on the application's specific features. In addition, command requests can be converted to signed Interests, and data retrieval requests can also be translated to NDN using NDN Sync.

In our local email sharing scenario, there are only two devices, a smartphone, and a laptop. Thus it is straightforward for the Proxy to directly send Interest to retrieve Data. When an IMAP command specifies multiple messages to retrieve, it will trigger the Proxy to send multiple Interests. If an email is large in size or contains the attachment, it is also common to split the message into multiple Data packets and to send multiple Interests to retrieve them all. Nevertheless, basic Interest-Data exchange is needed for the 'dynamic' content, such as inviting a friend.

Push Model. : Push communication model is also widely used, for example, an XMPP client can push its *presence* information to all subscribers, and an XMPP server can send messages to users in the same chat room. To translate such communications, we utilize NDN Sync as the basic mechanism. More specifically, multiple parties agree to use the same *data collection prefix* and apply NDN Sync to it. Within this synchronization namespace, each party generates data under its own subnamespace (i.e., the producer name), and others can receive the data with the help of NDN Sync. This procedure translates the push based communication patterns to NDN.

In XMPP multi-user chat, the chat messages need to be shared among multiple users. NDN Sync fits in such scenario translation, because NDN Sync provides multi-party data synchronization. If the participants were interested in the same dataset in its entirety, NDN Sync would provide a great service abstraction to applications and an efficient implementation to handle various issues in group communication. We have found that using NDN Sync in XMPP can considerably simplify the translation design. We suspect other application protocol translation will also acquire similar benefits using NDN Sync.

5.3 Name Construction and Discovery

If applications always retrieve contents from a fixed server, there is not much data/service discovery needed because the server name/address is already configured. However, if the architecture allows contents to be retrieved from anywhere, name discovery becomes a necessary step. For example, in the off-the-grid scenario, a device needs to discover what contents are available, and from which peer it can retrieve the content of interest in the local network. We have found four mechanisms useful in our work.

Naming Convention. : One simple way is to define static and public keywords, algorithms, and rules, so NDN proxy is able to correctly and independently construct names, then to communicate with other parties. For example, as shown in Figure 9, *ndn-xmpp*, *MUC*, and *manifest* are static keywords. Putting them together as */ndn-xmpp/MUC/manifest* is the manifest collection prefix. For the variable name component <*one-to-one-channel*>, two friends can individually generate the value, applying the same algorithm with their identifiers as input. Such a mechanism helps two XMPP friends seamlessly switch between infrastructure mode and off-the-grid mode without negotiating the <*one-to-one-channel*> name component in advance when switching to the off-the-grid mode.

Prefix Match between Interest and Data. : This is a basic name discovery mechanism and it forms the foundation for some other mechanisms. NDN allows a Data to satisfy an Interest whereas the Interest name is a prefix of the Data name. For example, a consumer sends an Interest */A/B* and receives a Data */A/B/1* back. The Data might not be exactly what the consumer wanted, but the consumer is able to use the Data information to form a longer Interest name for a more accurate query in the future.

With proper settings, this mechanism can assist a consumer to fetch the latest version of data without knowing the version number in advance, assuming the producer is accessible. In the IMAP/NDN translation, we use such a mechanism to fetch the latest version of a mailbox's snapshot, responding to the IMAP SELECT command [6], which asks for a brief description about one mailbox, such as the number of email messages. For example,

(1) To translate an IMAP SELECT command, the NDN proxy creates an Interest */mailSync/teng@gmail.com/inbox/snapshot*, and sets *MustBeFresh* flag in the Interest to be true, meaning that an in-network cached Data can be matched only if it has not passed its *FreshnessPeriod* flag.
(2) This Interest will reach the producer (i.e., the NDN proxy on phone), and the producer will send the latest version Data back, e.g., */mailSync/teng@gmail.com/inbox/snapshot/**v2*** with *FreshnessPeriod* setting to 0.

Manifest. : It is a content object or a small collection of content objects that contains information about one namespace, such as existing name prefixes or names. Once the Manifest is retrieved, the application knows the more accurate name prefixes or names and avoids running discovery on a broader namespace. Thus this mechanism reduces the scope of name discovery to discovering and retrieving the Manifest. The Manifest can also be extended to contain more information to achieve synchronization control (Section 5.4).

Take the XMPP multi-user chat namespace in Figure 9 as an example:

(1) Alice creates a new multi-user chat room named *Arizona-Tea*, and puts the room name into the content of a Data packet, which is named as */ndn-xmpp/MUC/manifest/Alice/room-list/v1.* Alice at the same time, runs NDN Sync on the manifest collection prefix */ndn-xmpp/MUC/manifest.*
(2) Bob who also synchronizes this manifest prefix will get the newly created Data that contains the room's name, then Bob runs NDN Sync on */ndn-xmpp/MUC/Arizona-Tea* to retrieve messages in the room. Bob sends messages to the chat room with name prefix */ndn-xmpp/MUC/Arizona-Tea/Bob.*

NDN Sync. : NDN Sync synchronizes data with a namespace. Therefore, it can be used to discover the latest version of a content object, and to discover subnamespaces as well. We give another two examples:

(1) Following the previous example, Alice then creates another room named *Pizza*, and updates the room-list Data packet with a new version number */ndn-xmpp/MUC/manifest/Alice/room-list/v2.* As Bob runs NDN Sync on top of */ndn-xmpp/MUC/manifest*, he gets the latest version of Data that contains the latest room list.
(2) Another user Carol who just joined the network, runs NDN Sync on the manifest collection and serves her room-list Data packet */ndn-xmpp/MUC/manifest/Carol/room-list/v8.* Again, Bob is able to discover Carol's room-list Data and her user

identifier. This property is useful in scenarios such as when users want to discover people nearby.

5.4 Control and Management

Existing applications may have various control and management mechanisms. To translate them into NDN, we need to find proper mechanisms, extend existing mechanisms, or come up with new mechanisms in NDN. One example is **group membership management** in XMPP multi-user chat. To translate it, we perform the management with both the data synchronization eligibility and the data confidentiality in NDN.

For example, in XMPP, a user is able to create a multi-chat user room and explicitly specifies who is eligible and who is ineligible to join. In NDN, since data can be cached anywhere, preventing ineligible users from fetching data is a different case. Therefore, data is encrypted and the decryption keys are distributed to eligible users only, in this way, the access control is achieved. However, ensuring data confidentiality is not sufficient, because an attacker can poison the synchronization dataset if not controlling synchronization members. We prevent such attack by not synchronizing data from ineligible users and distributing the security configurations in the Manifest. More specifically, in NDN translation, when a user creates a multi-chat room, it creates a multi-chat room namespace and the metadata in the room manifest namespace *ndn-xmpp/MUC/<room>/manifest* (Figure 8). The metadata contains the **whitelist** for the eligible users and the **blacklist** for the ineligible users. After retrieving the metadata, only the eligible users are permitted to sync the multi-chat room namespace, meaning that they will not synchronize with ineligible users during the NDN sync protocols. This requires NDN Sync to provide such mechanism.

6 SECURITY ISSUES

Today's Internet applications mostly adopt a session-based security model: the client is preconfigured the hostname or IP address of the server, and establishes a secure session to the server, encrypting transmitted data via Secure Sockets Layer or Transport Layer Security (SSL/TLS). In addition, the client verifies the server's identity by checking its certificate issued by a trusted Certificate Authority (CA), and the server authenticates the client by verifying its username and password. This approach, however, does not apply to off-the-grid scenarios, because we can neither require a device to know which other devices it will retrieve data from, or require it to trust a secure session with any device that is willing to provide data. Therefore, we should adopt NDN's data-centric model to ensure data security and privacy in the application translation for the off-the-grid scenarios.

6.1 Data-Centric Security

We first briefly introduce the data-centric security mechanisms in NDN, including integrity, confidentiality, access control, trust schema, and group membership management in NDN Sync.

Integrity. : In NDN, data-centric integrity is ensured by binding signature to every NDN Data packet. Therefore, consumers are able to verify Data packets independently regardless of where they are retrieved. Note that Interests can also be signed and verified as Signed Interests. The signature carries the name of the certificate which can be used to verify the signature. Since certificate in NDN is a Data packet signed by another certificate, verifiers can easily fetch the chain of certificates from the network, by issuing the Interests for the certificates.s In our work, we use RSA signing to sign all Interest and Data packets with the NDN proxy's private key.

Confidentiality and Access Control. : Data confidentiality is achieved by encrypting the content part of Data packets. In addition, because NDN Data packets can be cached anywhere, access control is also achieved by encrypting Data packets and distributing the decryption key to legitimate consumers. For encrypted Data packet, the current NDN packet format does not contain a field for the encryption key name. One approach is to include the decryption key name as part of the Data name, such as Name-based Access Control [34].

In our work, we use a shared secret symmetric key for encryption and decryption of bulk data and use public cryptography to share the secret key among trusted peers. In IMAP/NDN translation, two NDN proxy instances synchronize the secret key after they set up trust. In XMPP/NDN translation, sharing the secret between two users in one-to-one chat is similar to the process in IMAP/NDN translation. The challenge is to synchronize the secret key among a group in the multi-user chat.

Trust Schema. : An application defines the trust model to describe trust relationships among different parties. In NDN, name-based trust schema [33] reflects trust relationships to signing relationships by specifying name relationships for a Data packet (including both raw data and key) and the signing Key, therefore it achieves data-centric trust management.

Group Membership Management in NDN Sync. : As mentioned in Section 5.4, NDN Sync should provide group membership management, so that eligible members can avoid synchronizing data with ineligible members. For example, in XMPP/NDN translation, a multi-chat room, users on the whitelist only synchronize data with others in the whitelist. If only blacklist is applied, eligible users will not synchronize with users on the blacklist.

6.2 Trust Model and Configuration

In application translation, two types of configurations involved, one is for backward compatibility, and the other one is for NDN communication. Take local email sharing as an example, and assume the user has a Gmail account for the ease of composition. First, since the application on the laptop is configured to access Gmail account via NDN Proxy (on localhost 127.0.0.1), it has to accept the Proxy's certificate as the Gmail server. Plus since the smartphone is accessing Gmail server, it has to know the user's username and password. Both need one-time input from the user and are necessary to be compatible with the existing system. Second, between the smartphone and the laptop, NDN communication requires the signature for every Interest/Data packet. We need to generate key pairs for both NDN proxy instances on the smartphone and the laptop and configure them to trust each other's key.

In order to implement data-centric security in NDN, the first thing is to define the application trust model, and then to generate certificates accordingly. While previous research has covered how to use public-key cryptography to achieve various security and privacy properties, the most prominent challenge in terms of deployment in our work is how to configure and retrieve these keys/certificates without putting burdens on the end user.

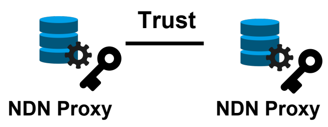

Figure 10: Two parties trust each other

In the off-the-grid scenarios, we find two potential trust models that can be applied in our work. Figure 10 shows the basic trust model that two parties trust each other. Because this model has no dependency on a third party, it is simple to achieve. The only requirement is that users have to manually set up the trust relationship because the two parties share no trust anchor. For example, in the local email sharing scenario, the NDN proxy instances on both laptop and phone trust each other's certificate with user's agreement; in XMPP/NDN translation, when a user wants to chat with another user in the off-the-grid scenario, such trust relationship can be set up through friend invitation and acceptance operations.

Note that the ideal case is that the data producer signs the data, and the consumer verifies the signature regardless of where the data was retrieved. In the current stage of deployment, however, we have to make applications work compatibly with existing systems, where application data is not signed as NDN packets. Thus in our implementation, the consumer verifies that whether the signature is attached by a trusted NDN proxy, not necessarily the original data producer.

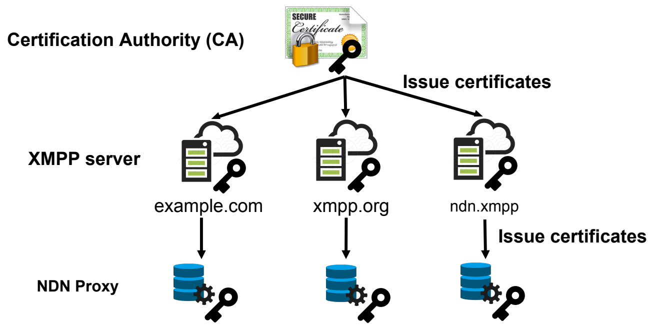

Figure 11: Trust support from the infrastructure

The first trust model is not suitable for communications involving multiple parties, because it requires any pair of participants to manually set up trust, putting many burdens on end users. For example, XMPP provides a public multi-user chat room that any user is able to join. To translate it into the off-the-grid scenario, it is possible that users joining the room do not trust each other. Therefore, we argue that a more convenient way is to apply the trust model in Figure 11, which requires the support from infrastructure. In this trust model, when XMPP uses infrastructure services through NDN proxy, the verified servers issue certificates for the NDN proxies. As a result, such trust relationships are still maintained in the off-the-grid scenarios, and the NDN proxies from different parties share the same trust anchor.

6.3 Name Obfuscation for Privacy

Although encrypting content prevents others from accessing the data, names, which are in the packet header, it may still leak some sensitive information. Name obfuscation can be achieved by encrypting certain sensitive name components. One of such designs is ANDaNA [8]. In our applications, we did not include such name obfuscation mechanisms. We rely on layer-2 protection since it is reasonably adequate in our use cases: the devices are within one hop of WiFi connectivity. But if the devices are cross multiple hops or using open layer-2 links, then name obfuscation may become necessary.

7 EVALUATION

We have implemented NDN Proxy for IMAP for the purpose of evaluation. The program is named as mailSync, and it is written in Java. It has a Mac version to run on the laptop, and an Android version to run on Android smartphones. We used the IMAP module of GreenMail [2], an open-source mail server implementation, as the IMPA handler to parse and compose IMAP messages. We also employed the JavaMail library to communicate with remote IMAP server, i.e., the Gmail cloud server. On the NDN side, each device runs NFD for the basic NDN functionalities, and the Proxy is implemented using the JNDN library. In the process of testing, the Android smartphone runs Gmail, NDN Proxy (mailSync), and NFD-Android; the laptop runs Thunderbird, NDN Proxy (mailSync), and NFD. The Android smartphone first retrieves emails from the Gmail server, then the laptop retrieves emails from the phone via NDN over local WiFi connectivity (without Internet access). After further debugging and tuning, we plan to release the code publicly.

During field tests of the programs, we found that WiFi networks in many places are not amenable to device-to-device or off-the-grid communication. Many public WiFi networks, e.g., the WiFi at airports, university campus, shops, and cafeterias, do not allow multicast traffic due to the security concerns. However, multicast is the basis for local discovery in both IP and NDN. Disable multicast means all local discovery mechanisms, IP or NDN, will not work. Even worse, many public WiFi networks also block unicast communication between two WiFi clients. The intention is to protect individual users from being attacked by another user on the same WiFi. The reason that most existing applications are not affected is they all rely on cloud servers reached through the Internet. However, this popular setup precludes the possibility of utilizing local connectivity when Internet is not reachable. Better security/privacy solutions are definitely needed here. For these reasons, we did not use public WiFi in our tests. Instead, we turned the smartphone into a hotspot mode and tethered the laptop to it, which is also a common use by many users.

8 RELATED WORK

Off-the-grid TCP/IP Works: Using local broadcast or multicast to discover services has been done in many protocols and applications, for instances, IPv6 Address Autoconfiguration [29] and DNS-based Service Discovery [4]. While service discovery is a necessary piece of the puzzle, it is not adequate for applications to

[2]Open Source suite of lightweight and sandboxed email servers supporting SMTP, POP3, and IMAP.

be fully functional without Internet access. Applications also need functionalities such as multi-party synchronization and data security management. NDN has mechanisms to handle all these tasks and it provides a common layer to be used by all applications.

Peer-to-Peer Applications: Although many peer-to-peer applications are able to directly communicate between peers, they are built as an overlay network on top of the infrastructure. Therefore, they still depend on infrastructure service to be functional. For example, Legion is a proposed framework [30] to allow client web applications to work without servers. Clients replicate data from servers and synchronize data directly among themselves. However, Legion still needs infrastructure services in order to build a peer-to-peer overlay network. In contrast, our work intends to enable an existing IP application to serve and consume data with direct app-to-app communication without reliance on any cloud server. For off-the-grid peer-to-peer applications, they are facing the same issues as other off-the-grid TCP/IP works.

IP-to-NDN Translation: There is a number of work that translates TCP/IP traffic from/to NDN traffic so that IP and NDN networks can co-exist and interoperate, e.g., TCP-to-NDN [17], IP-to-NDN [21], and IPoC [25]. However, translating TCP/IP traffic at network or transport layer does not change the networking primitive from address-centric to data-centric, because the translated NDN traffic is named either by an end-host identifier or a TCP 5-tuple. This means some of NDN's major benefits, including caching, security, and adaptive forwarding, will not be enjoyed completely. In order to take full advantage of NDN architecture, we choose to translate between application protocols (e.g., IMAP, XMPP) and NDN. Not only does this give each packet a semantically meaningful name, but also translates the protocol behavior more accurately and more efficiently.

Li proposes an HTTP-CCN gateway [16] to convert HTTP traffic into Content-centric Networking (CCN) traffic, in order to introduce real traffic onto a CCN testbed. Similar to TCP-to-NDN and IP-to-NDN, the goal of HTTP-CCN is to carry HTTP/TCP/IP traffic over an ICN network between two gateways so that two TCP/IP based endpoints can still communicate. However, our purpose is to use NDN to provide the off-the-grid capability to existing applications. It requires more than 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.

NDN Synchronization Protocols. NDN Synchronization protocols, short for NDN Sync, provide the functionalities for NDN applications to reliably synchronize data within the shared namespace among multiple parties, serving as a transport layer. We find NDN Sync a suitable and basic mechanism to translate client-server based applications protocols to NDN in the off-the-grid scenario, where distributed dataset synchronization is a common usage.

Shang surveys [24] the existing NDN Sync protocols, including CCNx Sync [19], iSync [10], ChronoSync [39], RoundSync [7] and PSync [36]. While PSync is originally proposed for the consumers to synchronize a subset of a large data collection with a single producer, and to reduce maintaining consumer states at the producer, other NDN Sync protocols more focus on synchronizing the full dataset of a data collection among multiple parties. Various NDN Syncs are proposed to improve both efficiency and effectiveness of data synchronization. The key is to efficiently figure out the differences among multiple states of data collection. In contrast to CCNx 1.0 Sync that uses the hash of the list of data names as the local state, iSync utilize a more efficient encoding mechanism - Invertible Bloom Filter (IBF) [9] to prepresent state. ChronoSync, RoundSync, and PSync simplifying state representation by adopting the sequential data naming conventions.

In this paper, we analyze the requirements of NDN sync from the application's perspective, especially in the practical translation of existing mature applications, IMAP and XMPP. We treat NDN Sync as a basic mechanism that can provide full data synchronization within a given namespace.

9 CONCLUSIONS

This paper is intended to identify research issues, and to generalize design patterns of translating existing application-level protocols to NDN. Such protocol translation enables existing applications with off-the-grid communication without application changes. As a concrete basis for our work, we design IMAP/NDN translation for local email retrieval between two end devices, and XMPP/NDN translation for local group chat. Based on these efforts, we identify key common design issues, including application-layer framing, namespace design, application communication translation, security, privacy, and real-world deployment challenges. In addition, we outline out design for the specific protocols, discuss the rationale behind our design choices, and draw generalizable conclusions. We have built the NDN proxy that conducts IMAP translation in Java and Android, tested them between a laptop and an Android phone, and collected deployment challenges.

ACKNOWLEDGMENTS

We are extremely grateful for the valuable suggestions made by our Shepherd, John Wroclawski, and detailed comments from anonymous reviewers.

This material is based upon work supported by the National Science Foundation under Grant No. CNS-1629009, CNS-1513505, and a Huawei grant. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

[1] Alexander Afanasyev, Ilya Moiseenko, Lixia Zhang, et al. 2012. ndnSIM: NDN simulator for NS-3. *University of California, Los Angeles, Tech. Rep* 4 (2012).
[2] Alexander Afanasyev, Zhenkai Zhu, Yingdi Yu, Lijing Wang, and Lixia Zhang. 2015. 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, 525–530.
[3] Jeff Burke. 2017. Browsing an Augmented Reality with Named Data Networking. In *International Conference on Computer Communication and Networks (ICCCN), International Conference on Computer Communication and Networks (ICCCN).*
[4] Stuart Cheshire and Marc Krochmal. 2013. DNS-Based Service Discovery. RFC 6763. https://doi.org/10.17487/RFC6763
[5] David D Clark and David L Tennenhouse. 1990. Architectural considerations for a new generation of protocols. In *ACM SIGCOMM Computer Communication Review*, Vol. 20. ACM, 200–208.
[6] M. Crispin. 2003. INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1. RFC 3501 (Proposed Standard). http://www.ietf.org/rfc/rfc3501.txt Updated by RFCs 4466, 4469, 4551, 5032, 5182, 5738, 6186.

[7] Pedro de-las Heras-Quirós, Eva M Castro, Wentao Shang, Yingdi Yu, Spyridon Mastorakis, Alexander Afanasyev, and Lixia Zhang. 2017. *The design of RoundSync protocol.* Technical Report. Technical Report NDN-0048, NDN.

[8] Steven DiBenedetto, Paolo Gasti, Gene Tsudik, and Ersin Uzun. 2011. ANDaNA: Anonymous named data networking application. *arXiv preprint arXiv:1112.2205* (2011).

[9] David Eppstein, Michael T Goodrich, Frank Uyeda, and George Varghese. 2011. What's the difference?: efficient set reconciliation without prior context. In *ACM SIGCOMM Computer Communication Review*, Vol. 41. ACM, 218–229.

[10] Wenliang Fu, Hila Ben Abraham, and Patrick Crowley. 2015. Synchronizing namespaces with invertible bloom filters. In *Architectures for Networking and Communications Systems (ANCS), 2015 ACM/IEEE Symposium on.* IEEE, 123–134.

[11] Paolo Gasti, Gene Tsudik, Ersin Uzun, and Lixia Zhang. 2013. DoS and DDoS in named data networking. In *Computer Communications and Networks (ICCCN), 2013 22nd International Conference on.* IEEE, 1–7.

[12] Giulio Grassi, Davide Pesavento, Giovanni Pau, Rama Vuyyuru, Ryuji Wakikawa, and Lixia Zhang. 2014. VANET via named data networking. In *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on.* IEEE, 410–415.

[13] Peter Gusev and Jeff Burke. 2015. Ndn-rtc: Real-time videoconferencing over named data networking. In *Proceedings of the 2nd ACM Conference on Information-Centric Networking.* ACM, 117–126.

[14] AKM Hoque, Syed Obaid Amin, Adam Alyyan, Beichuan Zhang, Lixia Zhang, and Lan Wang. 2013. NLSR: named-data link state routing protocol. In *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking.* ACM, 15–20.

[15] Siham Khoussi, Davide Pesavento, Lotfi Benmohamed, and Abdella Battou. 2017. NDN-trace: a path tracing utility for named data networking. In *Proceedings of the 4th ACM Conference on Information-Centric Networking.* ACM, 116–122.

[16] Zhaogeng Li, Jun Bi, and Sen Wang. 2013. HTTP-CCN gateway: Adapting HTTP protocol to Content Centric Network. In *Network Protocols (ICNP), 2013 21st IEEE International Conference on.* IEEE, 1–2.

[17] Ilya Moiseenko and Dave Oran. 2016. TCP/ICN: carrying TCP over content centric and named data networks. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking.* ACM, 112–121.

[18] Ilya Moiseenko and Lixia Zhang. 2014. Consumer-producer api for named data networking. In *Proceedings of the 1st ACM Conference on Information-Centric Networking.* ACM, 177–178.

[19] Marc Mosko. 2014. CCNx 1.0 Collection Synchronization. In *Technical Report.* Palo Alto Research Center, Inc.

[20] Luca Muscariello, Giovanna Carofiglio, Jordan Auge, and Michele Papalini. 2018. *Hybrid Information-Centric Networking.* Internet-Draft draft-muscariello-intarea-hicn-00. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-muscariello-intarea-hicn-00 Work in Progress.

[21] Tamer Refaei, Jamie Ma, Sean Ha, and Sarah Liu. 2017. Integrating IP and NDN through an extensible IP-NDN gateway. In *Proceedings of the 4th ACM Conference on Information-Centric Networking.* ACM, 224–225.

[22] Wentao Shang, Adeola Bannis, Teng Liang, Zhehao Wang, Yingdi Yu, Alexander Afanasyev, Jeff Thompson, Jeff Burke, Beichuan Zhang, and Lixia Zhang. 2016. Named data networking of things. In *Internet-of-Things Design and Implementation (IoTDI), 2016 IEEE First International Conference on.* IEEE, 117–128.

[23] Wentao Shang, Qiuhan Ding, Alessandro Marianantoni, Jeff Burke, and Lixia Zhang. 2014. Securing building management systems using named data networking. *IEEE Network* 28, 3 (2014), 50–56.

[24] Wentao Shang, Yingdi Yu, Lijing Wang, Alexander Afanasyev, and Lixia Zhang. 2017. *A Survey of Distributed Dataset Synchronization in Named Data Networking.* Technical Report. Technical Report NDN-0053, NDN.

[25] Susmit Shannigrahi, Chengyu Fan, and Greg White. 2018. Bridging the ICN Deployment Gap with IPoC: An IP-over-ICN protocol for 5G Networks. In *Proceedings of the 2018 Workshop on Networking for Emerging Applications and Technologies.* ACM, 1–7.

[26] Tian Song, Haowei Yuan, Patrick Crowley, and Beichuan Zhang. 2015. Scalable name-based packet forwarding: From millions to billions. In *Proceedings of the 2nd ACM conference on information-centric networking.* ACM, 19–28.

[27] NFD Team. [n. d.]. NFD DeveloperâĂŹs Guide. ([n. d.]).

[28] Jeff Thompson and Jeff Burke. 2014. NDN common client libraries. *Technical Report NDN-0024, Revision 1. NDN Project* (2014).

[29] S. Thomson, T. Narten, and T. Jinmei. 2007. IPv6 Stateless Address Autoconfiguration. RFC 4862 (Draft Standard). http://www.ietf.org/rfc/rfc4862.txt

[30] Albert van der Linde, Pedro Fouto, João Leitão, Nuno Preguiça, Santiago Castiñeira, and Annette Bieniusa. 2017. 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, 283–292.

[31] Hao Wu, Junxiao Shi, Yaxuan Wang, Yilun Wang, Gong Zhang, Yi Wang, Bin Liu, and Beichuan Zhang. 2017. On Incremental Deployment of Named Data Networking in Local Area Networks. In *Proceedings of the Symposium on Architectures for Networking and Communications Systems.* IEEE Press, 82–94.

[32] Cheng Yi, Alexander Afanasyev, Lan Wang, Beichuan Zhang, and Lixia Zhang. 2012. Adaptive forwarding in named data networking. *ACM SIGCOMM computer communication review* 42, 3 (2012), 62–67.

[33] Yingdi Yu, Alexander Afanasyev, David Clark, Van Jacobson, Lixia Zhang, et al. 2015. Schematizing trust in named data networking. In *Proceedings of the 2nd International Conference on Information-Centric Networking.* ACM, 177–186.

[34] Yingdi Yu, Alexander Afanasyev, and Lixia Zhang. 2015. Name-based access control. *Named Data Networking Project, Technical Report NDN-0034* (2015).

[35] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, Patrick Crowley, Christos Papadopoulos, Lan Wang, Beichuan Zhang, et al. 2014. Named data networking. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 66–73.

[36] Minsheng Zhang, Vince Lehman, and Lan Wang. 2017. Scalable name-based data synchronization for named data networking. In *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE.* IEEE, 1–9.

[37] Zhiyi Zhang, Yingdi Yu, Haitao Zhang, Eric Newberry, Spyridon Mastorakis, Yanbiao Li, Alexander Afanasyev, and Lixia Zhang. 2018. *An Overview of Security Support in Named Data Networking.* Technical Report. Technical Report NDN-0057, NDN.

[38] Weicheng Zhao, Yajuan Qin, Deyun Gao, Chuan Heng Foh, and Han-Chieh Chao. 2017. An efficient cache strategy in information centric networking vehicle-to-vehicle scenario. *IEEE Access* 5 (2017), 12657–12667.

[39] Zhenkai Zhu and Alexander Afanasyev. 2013. Let's ChronoSync: Decentralized dataset state synchronization in Named Data Networking. In *2013 21st IEEE International Conference on Network Protocols (ICNP).* IEEE, 1–10.