NDN, Technical Report NDN-0070, 2021. http://named-data.net/techreports.html
Revision 1: January 16, 2021

NDN Forwarder Manager: Improving the Usability of NDN Forwarders

Xinyu Ma
UCLA
xinyu.ma@cs.ucla.edu

Eric Newberry
UCLA
enewberry@cs.ucla.edu

Lixia Zhang
UCLA
lixia@cs.ucla.edu

ABSTRACT

Up to now, configuring NDN for use on end hosts has generally been difficult due to the absence of graphical configuration interfaces. To improve the usability of NDN, we have developed the NDN Forwarder Manager (NDN-FM). NDN-FM provides a graphical interface for users to manage a local instance of the NDN Forwarding Daemon (NFD), allowing them to monitor the status of the forwarder; create, update, and delete faces and routes; manage certificates; and run basic NDN debugging tools. NDN-FM can also be used to manage other NDN packet forwarders that support the NFD Management Protocol. In this report, we describe the implementation of NDN-FM, discuss the rationale for the design choices that were made during its development, and demonstrate the use cases of NDN-FM from the perspective of system administrators and end users to illustrate NDN-FM's improvements to NDN usability.

1 INTRODUCTION

Named Data Networking (NDN) is a new network architecture which lets applications communicate with each other by fetching named, secured chunks of data. This differs from traditional TCP/IP network architectures, which deliver packets to destination host addresses. The development of a fundamentally new network architecture necessitates the development of new management systems for packet forwarders.

However, up to this point, configuring NDN for use on end hosts has been difficult. In contrast to today's user-friendly graphical tools for configuring traditional networking equipment, such as wireless routers, one needs familiarity with command line tools to configure an NDN node. To illustrate this point, we will use the NDN Forwarding Daemon (NFD) [17], an NDN forwarder intended for use in general-purpose computing environments, as an example. At the present time, NFD is generally configured using a combination of a configuration file and command line tools. Upon startup, NFD automatically loads a configuration file, which provides "startup configuration". Meanwhile, the remaining portion of the forwarder configuration (the "runtime configuration") is provided through the `nfdc` [19] command line utility. Additionally, the `ndnsec` [16] command line utility is used to manage the NDN security configuration of the local host.

An earlier effort was made to create a graphical management interface for NFD [10]; however, the result was platform-specific and did not keep up with the pace of the NDN platform's development. However we do note the existence of a standardized management protocol [18], which allows NFD, and any other forwarder supporting this protocol, to be configured over native NDN protocols.

In this report we present the NDN Forwarder Manager (NDN-FM)[1] which offers a graphical, browser-based management interface for NDN forwarders. Utilizing a browser-based interface provides significant benefits over a standalone program, including simplified deployment and cross-platform support. NDN-FM allows end users to perform a number of operations critical to configuring and managing a local instance of a forwarder, including: monitoring the status of the forwarder; creating, updating, and deleting faces and routes; managing certificates; and running basic debugging tools. In this report, we describe the implementation of NDN-FM, discuss the rationale behind the design choices made during its development, and demonstrate the use cases of NDN-FM. We explain how NDN-FM provides increased usability for the deployment of NDN forwarders for both skilled NDN system administrators and end users with less knowledge about the inner workings of NDN forwarders and networks. We hope that this tool will ease the deployment and management of NDN instances on end devices on various types of networks and by users of various skill levels.

This report is organized as follows: Section 2 introduces NFD and its configuration and monitoring mechanisms and protocols. Section 3 introduces the design of NDN-FM. Section 4 discusses the implementation details of NDN-FM. Section 5 evaluates the usability of NDN-FM as compared to existing command-line management tools. Section 6 compares NDN-FM and the original NDN Control Center, and discusses the different design choices taken during the development of each. Finally, Section 7 provides a plan of future improvements and extensions to NDN-FM.

2 BACKGROUND

This section introduces the core NDN software as well as the options that can be configured. We assume that the reader has basic knowledge of how NDN works, including topics

[1]The code is available on GitHub: https://github.com/zjkmxy/ndn-cc

like Interest-Data exchange and the essentials of the NDN forwarding pipeline. For readers unfamiliar with these topics, we recommend [1].

2.1 NDN Software

A typical NDN-capable host runs several core software components. We now discuss each component, as well as the specific implementation we use for each:

- A library implementing the core NDN primitives, including Interest-Data exchange, security primitives, and the management protocol. In this paper, we will use *ndn-cxx*, which is written in C++ [11].
- A tool to configure the NDN security infrastructure on the local device. In this paper, we will use *ndnsec*, which is included as part of the ndn-cxx package [16].
- An NDN forwarder. In this paper, we will use *NFD*, which is built on top of ndn-cxx [17].
- A forwarder configuration tool. In this paper, we will use *nfdc*, a command line configuration tool [19] for *NFD*.
- A collection of basic tools for NDN, providing utilities for, among other purposes, "pinging" NDN prefixes and inspecting NDN packets. In this paper, we will use *ndn-tools*, which is implemented on top of ndn-cxx [12].

A host on an NDN network needs to install at the very minimum ndn-cxx and NFD (or equivalents). Most hosts will also have nfdc and ndn-tools installed for configuration and debugging.

While other forwarders have been developed for NDN (such as ndn-lite [13] and NDN-DPDK [22]), NFD is the only forwarder currently deployed on the official NDN testbed. As such, it is a core component of the NDN platform. It consists of six modules: the "Core", "Faces", "Tables", "Forwarding", "Management", and the "Routing Information Base (RIB) Manager" [2]. The *Core* module provides common functionality for the other modules. The *Faces* module implements NDN network interfaces on top of different protocols, including UDP, TCP, and Ethernet for remote faces, as well as Unix sockets, WebSockets, and TCP for local faces [2] (See § 2.2.2). The *Tables* module implements the Content Store (CS), the Pending Interest Table (PIT), the Forwarding Information Base (FIB), strategy choices, and other tables. The *Forwarding* module implements the central packet processing pipelines and interacts with the Faces and Tables modules (See § 2.2.3). The *Management* module implements the NFD management protocol, which allows applications and users to monitor the state of NFD, as well as configure Faces, FIB entries, strategy choices, and so on (except the RIB, whose management is handled in a separate module). It also parses the configuration file and handles startup configuration[2]. Finally, the *RIB Management* module manages routing information and, while using the same management protocol as the *Management* module, is implemented in a separate thread from the rest of management for scalability reasons, given the high cost of RIB manipulation [2].

NDN-FM provides a graphical interface for users to observe information obtained from the NFD management protocol and ndnsec, as well as to allow them to perform various common management operations. Among other operations, users can add/delete faces, FIB entries, and strategy choices, as well as manage local identities via NDN-FM.

2.2 NFD Configuration

The following four systems must be configured in order to properly set up an NDN host running NFD: (1) security, (2) faces, (3) forwarding (including routes), and (4) strategies. These configuration steps are traditionally performed using the command line tools bundled with NFD and ndn-cxx. However, these tools are not user-friendly to home users.

2.2.1 Security. NDN integrates security primitives into the network layer to verify the provenance of data both in transit and at rest [27]. Additionally, NFD's management systems rely heavily upon these security primitives to authenticate management commands. Therefore, security bootstrapping must be performed before any other management or configuration steps can be completed. To accomplish this, the system must obtain a name and any associated keypairs and certificates [26]. NFD's default trust model utilizes self-signed certificates that permit users on the local host to perform any management tasks. However, this trust model can be modified to restrict or permit access to different components of management by a user possessing a given key.

To generate an initial usable NFD security configuration, traditionally one would use `ndnsec-keygen` to generate an identity and self-signed certificate and then install these with `ndnsec-install-cert`. This allows NFD management to be accessed locally (relying on a local trust anchor). However, to allow local applications to validate content retrieved from an existing NDN network and produce content that can be validated beyond the current host, it is necessary to learn the network's trust anchor(s) and then obtain a certificate from this anchor. This can be accomplished using other tools, such as `ndncert` [15].

2.2.2 Faces. To establish connections with other nodes, users need to create *faces* to these hosts. A *face* is a generalization

[2]Since startup configuration is not the focus of this paper, when we refer to "configuration" in the following sections, we always mean "runtime configuration".

that can represent either a *network interface* or an *application interface*. A *network face* is a connection to the forwarder on a remote host. Unlike the TCP/IP protocol stack, NDN does not make a strict distinction between the network and link layers. This means that NDN can run directly on top of standard link layer protocols, such as Ethernet, WiFi, and Bluetooth, but can also operate as an overlay network on top of higher layer protocols such as UDP, TCP, or WebSockets (which in turn run on top of IP). Network faces can be either unicast (to a single remote forwarder) or multicast (to one or more remote forwarders). Meanwhile, an *application face* is a connection to an NDN application. Generally, this takes the form of a Unix socket or TCP connection. Application faces are created and deleted implicitly when an application connects to and disconnects from the forwarder and their creation and deletion is therefore generally outside of the scope of forwarder configuration operations.

Traditionally, faces between forwarders have been created and modified using the `nfdc face create` command, and deleted using the `nfdc face delete` command [19]. The first command supports a number of options that can modify the behavior of the face and are documented in the *nfdc* manpages [19]. To create a face, one needs to know the protocol and the address of the remote forwarder (including the port number, if necessary for the underlying transport protocol in use). For example, `ucp6://192.168.1.1:6363` specifies a network face running over UDP.

2.2.3 Forwarding. NFD contains a Forwarding Information Base (FIB), which stores pairs of the format (name prefix, outgoing faces), to find the appropriate nexthop(s) when forwarding Interests [25]. While forwarding rules can be added directly to the FIB, it is recommended to add them via NFD's Routing Information Base (RIB) instead. The RIB contains two types of rules: (1) Those added by routing protocols (if any are in use), as well as prefix registrations by directly connected applications and remote forwarders using *automatic prefix propagation* [24]. (2) Those added manually by administrators using `nfdc route add`. The former are outside of the scope of the NFD management tools used by end users, so we will only consider the latter in this paper.

2.2.4 Strategies. Forwarding strategies determine whether, when, and where to forward Interests. By default, the *best-route* strategy is used, which forwards a single Interest to the nexthop with the lowest cost in the longest matching prefix FIB entry [2]. However, forwarding strategies have great flexibility in their behavior. For example, the *multicast* strategy forwards Interests to all nexthops listed in the longest matching prefix FIB entry and the *asf* strategy forwards to the upstream host with the lowest measured RTT [2]. The `nfdc strategy set` command is used to set a forwarding strategy for a name prefix, with strategies applying hierarchically [19].

During our experience developing the NDN platform, we found that it can be difficult to remember how to perform management tasks entirely via the command line. We imagine this would especially be the case for end users with less domain-specific knowledge than ourselves. Therefore, we have developed NDN-FM to provide a more familiar mechanism for users to manage local forwarders and hopefully ease the deployment of NDN in future networks.

3 DESIGN

We have developed the NDN Forwarder Manager (NDN-FM) to simplify the management of NDN forwarders that implement the NFD Management Protocol [18]. At the time of writing, only the NFD forwarder itself implements this protocol, but this general purpose NDN management protocol could be implemented by another forwarder at some point in the future. Additionally, we support the various security operations needed to authenticate management operations and support local applications running on the same host. In the remainder of this section, we discuss the design goals of NDN-FM (§ 3.1), provide an overview of the design of NDN-FM (§ 3.2), and discuss the specific features supported by NDN-FM at the time of writing (§ 3.3).

3.1 Design Goals

The central goal of NDN-FM is to provide a usable, general, and flexible management solution for NDN forwarders. This is because, while at the moment NDN deployments are limited to research and development environments, they will eventually roll out into end-user environments. End users will likely need to configure and manage their local forwarders to at least some extent, as is commonly necessary for wireless routers in home and small business environments. NDN-FM is intended to be used directly by end users with varying levels of technical knowledge, so it must satisfy the three principles stated above. More specifically, it satisfies each principle like so:

- *Usability.* NDN-FM integrates frequently used forwarder management functions into a browser-based GUI. This relieves users of the burden of having to remember the `nfdc` commands needed to perform various NFD forwarder management operations. Instead, the user interface is organized into reasonable categories that allow users to quickly find the operation they wish to perform. Moreover, NDN-FM is easy to install and run, being based upon standard, cross-platform Python libraries.

- *Generality.* NDN-FM supports all operating systems [3] that can run NFD and Python 3. Since it controls NFD via the OS-independent NFD Management protocol [18], no operating system-specific differences are presented to the user.
- *Flexibility.* The NFD Management protocol provides a standardized mechanism through which a wide variety of management operations can be performed. Therefore, NDN-FM implements a number of management operations of varying levels of complexity and frequency of use. This includes common operations like route and face configuration, as well as more advanced operations like security management.

In future widely-deployed NDN networks, forwarders on each device will serve as a gateway for application connectivity to the wider network and perform a very similar role to that of today's home wireless routers. This not only matches the level of configuration required in today's IP networks, but goes beyond it, as packet forwarders must be configured on every end devices in addition to on intermediate network hardware. Therefore, similar tools to those available for wireless routers today must be developed for NDN forwarders that satisfy the above-listed requirements.

3.2 System Overview

NDN-FM provides a browser-based graphical user interface to interact with users and to process user queries and commands. The user's browser communicates with the NDN-FM backend using HTTP. Meanwhile, the backend communicates with the forwarder (e.g., NFD) using the NDN-based NFD Management protocol [18] to query the status of the forwarder and perform various management operations. This design is shown in Figure 1.

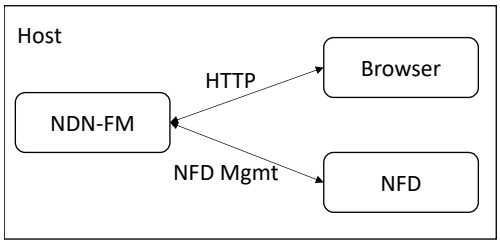


Figure 1: Design Overview

NDN-FM provides an icon on the system tray, which is used as a shortcut to the user interface and to allow the user to quit the application.

[3]While NFD officially supports a number of recent versions of Ubuntu and macOS, it has also been unofficially reported to run successfully on other platforms, such as recent versions of Debian, Gentoo, and Raspbian [7].

3.3 Supported Features

With NDN-FM, users can view, create, and delete faces and routes, as well as set strategies for given name prefixes. Additionally, users can modify the local security keychain, including adding and deleting identities, keys, and certificates. Moreover, users can view the status of the forwarder (including many counters), perform basic reachability tests, and even automatically connect to the nearest NDN Testbed [14] node.

4 IMPLEMENTATION

NDN-FM is implemented in Python 3, using python-ndn [5] and aiohttp [3] to realize the NDN-based management and user-facing HTTP interfaces, respectively. NDN-FM consists of two modules: a backend module and a system tray module. The backend module processes user input and obtains status information from the forwarder. Meanwhile, the system tray module provides an icon in the system tray (Figure 2).

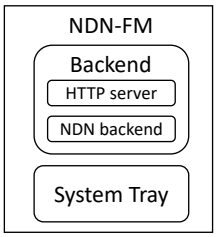


Figure 2: Components of NDN-FM

4.1 Backend Module

The backend module consists of two components, an aiohttp server and a python-ndn application. The aiohttp server responds to HTTP requests and translates between HTTP's request-response paradigm and NDN's Interest-Data paradigm. Meanwhile, the python-ndn component expresses these Interests and waits for corresponding Data. It also subscribes to face status change notifications from the NDN Management Protocol [18] and records such events.

The NFD Management Protocol is made up of "status datasets", "notification streams", and "control commands". Operations to query status datasets and issue control commands require a single Interest-Data exchange. Therefore, these two types of operations are blocking within the backend module. Meanwhile, notification streams operate via long-lasting polling Interests. Both of these types of operations run in the same thread.

4.2 System Tray Module

The system tray module uses pystray [23], a lightweight, cross-platform system tray module for Python, to display

a tray icon and the associated menu. By using psytray, we avoid including the entire PyQt package and therefore reduce the footprint of NDN-FM.

The system tray menu utilizes a shell command to perform various operations on the host system. In particular, to open the user interface, it uses `xdg-open` on Linux and `open` on macOS. Additionally, to start NFD, it runs `pkexec nfd-start` on Linux. This method assumes the paths to NFD and its command line tools are included in the *PATH* environment variable.

4.3 Usage

Since NDN-FM is written in Python, it can run without the need for prior compilation and can be run using the command `python3 main.py`. Additionally, we have created a macOS app that also bundles a recent version of Python 3, which can be installed by copying the app to the "Applications" folder. After starting NDN-FM, a system tray icon will appear – users can open the NDN-FM page from this icon.

4.4 System Requirements

NDN-FM officially supports Linux (Ubuntu and Raspbian) and macOS. However, it likely runs on other Linux distributions, but these environments have not been evaluated. Moreover, since the UI of NDN-FM is written in standard HTML, CSS, and JavaScript, it is assumed to be compatible with any modern web browser. We have tested it with Chrome (Chromium on Raspbian), Firefox, and Safari.

In our evaluations, we found the memory usage of the macOS app to be less than 50 MB in terms of resident set size, which is less than reported by the NDN Control Center [10]. Meanwhile, the required disk space for this macOS package, including the packaged Python3 interpreter, is 126 MB.

5 USABILITY EVALUATION

In this section, we evaluate three management operations that can be performed by NDN-FM by comparing them against identical operations performed using the `nfdc` command line tool [19].

5.1 Face Management

Before configuring routes, a user must first configure the forwarder's face table. Generally, when a user wishes to connect a specific set of NDN nodes manually instead of connecting to the nearest testbed hub, they need to create faces directly using the IP addresses (or other network identifiers) of those nodes.

To allow for this, NDN-FM provides a page for face management (Figure 3). On this page, users are provided with a list of faces sorted by face ID (in ascending order). They can remove faces using a "Remove" button to the right of every face. By default, all information about the face except the face ID and remote URI is hidden to provide a cleaner interface to the user. However, users can access more detailed information about a face, including its properties and statistics (e.g., counters), by clicking the triangle icon near the remote face URI. A form at the bottom of the page can be used to create a new face by specifying the remote URI of the new face. At the moment, NDN-FM only supports creating UDP and TCP faces (hence the label of "IP Addr" for the remote URI), but support for creating Ethernet faces is planned for future work [4].

The CLI tool *nfdc* supports all of the above-mentioned functions, but is not as easy to use as NDN-FM. The command `nfdc face list` shows a list of all faces. However, all properties are printed on one line, requiring users to manually parse out the information. Additionally, the command `nfdc face destroy` removes a face. The face to remove can be specified using either face ID or remote URI. Finally, the command `nfdc face create` creates a face. At a high level, these commands works similarly to NDN-FM, although they provide more control to the user as far as setting face properties. Currently, users cannot change face properties using NDN-FM and can only perform the coarse-grained actions of face creation and removal.

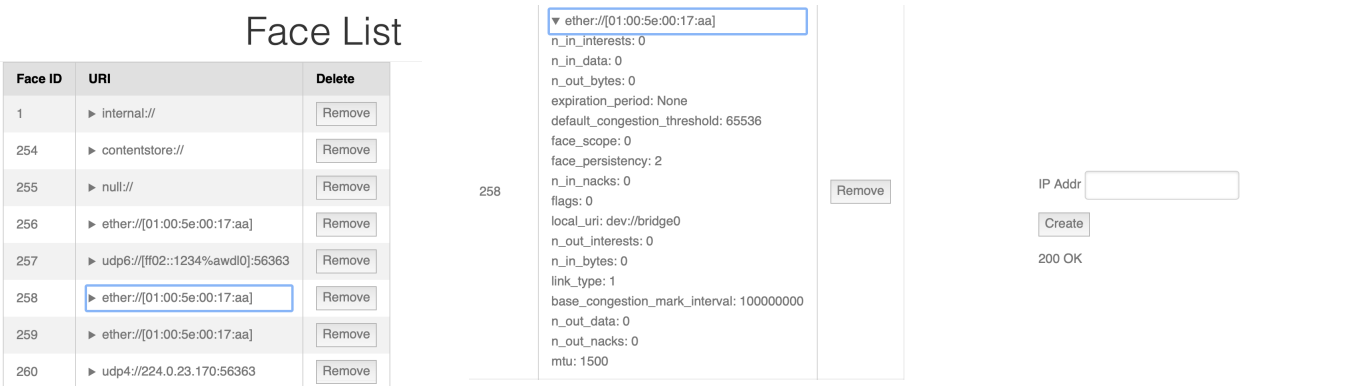
5.2 Route Management

If a routing protocol or other configuration mechanism (such as `ndn-autoconfig` [9]) is not used, users will need to manually configure all routes. One current use case for this NDN-FM feature is an NDN application developer testing their application on their own devices.

NDN-FM provides a page for route management (Figure 4). This page shows a list of routes grouped by name prefix. Users can click the "Remove" button to remove a specific route and can add new routes by specifying a name prefix and nexthop face ID (adding via remote face URI is planned for future work). During face creation, we do not provide options to set the "cost" or "origin" of the route. It is not necessary to allow these parameters to be set as they are not utilized when no routing protocol is deployed.

The CLI tool *nfdc* supports the addition and removal of routes. `nfdc route list` prints a list of all routes. This list is sorted by name prefix, although routes are not aggregated by name prefix as they are in NDN-FM. `nfdc route show` prints the routes for a specific name prefix. Users can use `nfdc route add` and `nfdc route remove` to add and remove routes.

[4] At the time of writing, NDN-FM face creation has only been tested with IPv4 addresses.



(a) A list of all faces. **(b) The details of a face.** **(c) The form used to create a new face.**

Figure 3: The face management page of NDN-FM.

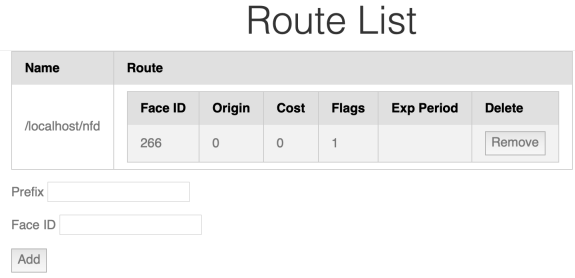


Figure 4: The route management page of NDN-FM.

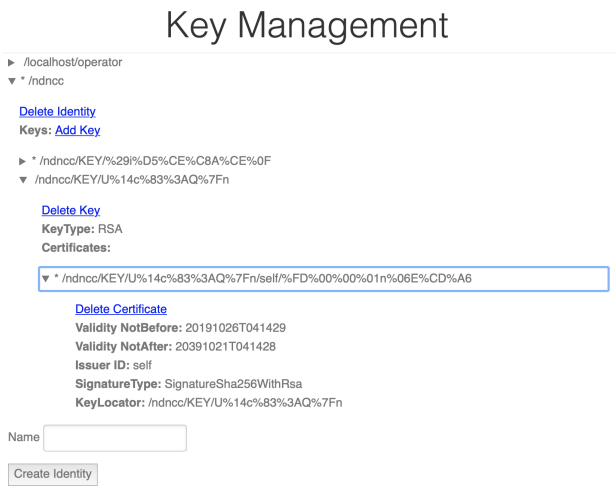


Figure 5: The keychain management page of NDN-FM.

5.3 Keychain Management

To publish content on a NDN network, users need to ensure that each node possesses the appropriate keys to sign data it creates. To this end, NDN-FM provides a page to manage the ndn-cxx security keychain (Figure 5). This page renders Identity–Key–Certificate relationships as a tree. Each node is folded by default and nodes can be expanded by clicking on the triangle located to the left of each entry. Additionally, users can add a key with a self-signed certificate to an identity by clicking on the corresponding "Add Key" link. They can also delete a specific identity, key, or certificate by clicking on the corresponding "Delete" link. Moreover, at the bottom of the page, there is a form that allows users to create new identities.

The CLI tool `ndnsec` supports all of these functions. `ndnsec list` prints the tree of certificates. However, its verbosity can only be specified globally, meaning that it most likely presents a large amount of extraneous data to the user. Therefore, when users wish to obtain the details of a specific certificate, `ndnsec`'s output will likely either be too brief or too verbose. Next, `ndnsec delete` is used to delete a keychain object, but users need to know the full name of the object to delete before invoking this command. Additionally, the type of object can be specified: "-k" for keys, "-c" for certificates, or no option for identities. Finally, `ndnsec key-gen` is used to create a new identity or key.

6 DISCUSSION

There are two main differences between NDN-FM and the NDN Control Center [10]: (i) NDN-FM uses a "browser user interface" (or "BUI"), while the NDN Control Center uses a traditional, desktop-based GUI. (ii) NDN-FM does not bundle a copy of NFD, while the NDN Control Center does. We elaborate on these differences in this section.

6.1 Browser User Interface

NDN-FM uses a browser user interface (or "BUI") to simplify our implementation and provide easier cross-platform support. As more and more functionalities are added to front-end frameworks, almost all the needs of a GUI can be implemented in web pages. As a result, many modern pieces of software have been written using BUIs, such as Slack and Discord [4]. Using a BUI makes it possible for us to exploit existing front-end frameworks, allowing for easier development. On the other hand, the structure of Python packages requires that the full Qt package be installed, which increases the size of the NDN-FM package.

6.2 Decoupling Management from NFD

Unlike the NDN Control Center, NDN-FM does not bundle NFD. The reason for this decoupling is threefold:

(1) NDN-FM is designed to support different versions of NFD that may have been released before or after the relevant version of NDN-FM. However, this cross-version support assumes that the NFD Management API has not changed.
(2) NFD is still under development and iterates rapidly. Bundling NDN-FM with NFD would force a new release of NDN-FM for every NFD release, even the NDN-FM itself does not change.
(3) Binary formats, library versions, and the installation process differ between platform. Therefore, bundling NFD would limit the number of platforms that could be supported without requiring users to recompile the NDN-FM package.

7 FUTURE WORK

In the current implementation of NDN-FM, we have added basic security configuration options, such as the administration of local identities. We hope to later expand this to include certificate management through the ndncert package [15]. We have added the basic structure of these pages to NDN-FM. However, at the time of writing, the tables in these pages remain unpopulated and users cannot perform any management operations using them. In the future, we plan to complete the implementation of user-friendly management of this critical security aspect of NDN forwarders.

Moreover, there are some features of the NFD Management Protocol that we do not currently support. These include Content Store management, which allows users to control the capacity of the Content Store, view its contents and statistics, erase specific contents from it, and even control whether content can be admitted to/served from it [6]. While adding this feature would add to the completeness of our system, we leave it to future work given its lower relative importance to tasks like face, route, and security management.

In addition to features controlled through management, many settings of NFD are controlled via a configuration file. These include such important options as enabling and disabling faces for specific protocols, which are controlled via the forwarder's "startup configuration". To edit the forwarder's startup configuration, NDN-FM would need to directly edit the configuration file. This would require that NDN-FM run with administrative privileges to edit the file, which it currently does not by default. Additionally, we would need a way to edit the configuration file programmatically. Given that the config file is in the INFO format, we could directly use `infoedit` for this purpose [8].

In addition to supporting additional features of the NFD Management Protocol, it would be beneficial to enable the management of remote NDN forwarder instances with NDN-FM. This would be helpful in a scenario where a user wished to manage an intermediate NDN node, such as their home router. However, in its current state, NDN-FM features no authentication features, such as a login requirement to view and change settings. These features have become standard on wireless router management interfaces due to the high potential for abuse by unauthorized parties. Such a feature would be straightforward to add to NDN-FM, but is likely not necessary in the current design where the administrative interface is only accessible locally, which already requires users to be authorized to use the system.

In addition to NFD, another core component of many NDN hosts, particularly in the core of the network, is a routing protocol, such as NLSR [20]. Like NFD, NLSR features a command-line management tool, known as `nlsrc`, that allows for the management of the routing protocol [21]. In particular, `nlsrc` allows for one to view the status of NLSR, as well as retrieve the current link-state database and routing table and advertise/withdraw specific prefixes. Adding management interfaces for NLSR would be an excellent addition to NDN-FM and would expand its scope beyond simply managing the forwarder to managing a complete NDN host.

8 CONCLUSION

We have developed the NDN Forwarder Manager (NDN-FM) to ease the management of NFD, as well as theoretically any other NDN forwarder implementing the NFD Management Protocol [18]. Our manager is platform independent, supporting any system that can run an NDN forwarder such as NFD. This is because it is written in Python 3 and utilizes a browser-based user interface (BUI).

Currently, NDN-FM can simplify many tasks that are frequently performed when managing an NDN node. In particular, it can automatically connect to the NDN Testbed [14];

manage faces, routes, and the local keychain; and view status information about the forwarder. This is provided to users using a clean, uncluttered interface to enhance their comprehension of this information.

We evaluated NDN-FM by comparing its functionality and usage to equivalent command line tools, namely `nfdc` [19] and `ndnsec` [16]. In addition, we enumerated future functionality that we hope to add to NDN-FM to further expand its feature set.

ACKNOWLEDGMENTS

This work was partially supported by the National Science Foundation under awards CNS-1629922 and CNS-1719403. The authors would like to thank Philipp Moll for his help in proofreading this report.

REFERENCES

[1] A. Afanasyev, J. Burke, T. Refaei, L. Wang, B. Zhang, and L. Zhang. 2018. A Brief Introduction to Named Data Networking. In *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*. 1–6. https://doi.org/10.1109/MILCOM.2018.8599682

[2] Alexander Afanasyev, Junxiao Shi, Beichuan Zhang, Lixia Zhang, Ilya Moiseenko, Yingdi Yu, Wentao Shang, Yanbiao Li, Spyridon Mastorakis, Yi Huang, Jerald Paul Abraham, Eric Newberry, Steve DiBenedetto, Chengyu Fan, Christos Papadopoulos, Davide Pesavento, Giulio Grassi, Giovanni Pau, Hang Zhang, Tian Song, Haowei Yuan, Hila Ben Abraham, Patrick Crowley, Syed Obaid Amin, Vince Lehman, Muktadir Chowdhury, and Lan Wang. 2018. *NFD Developer's Guide.* Technical Report NDN-0021, Revision 10. Named Data Networking. https://named-data.net/wp-content/uploads/2018/07/ndn-0021-10-nfd-developer-guide.pdf

[3] Aiohttp Contributors. [n. d.]. aiohttp. https://docs.aiohttp.org/en/stable/

[4] OpenJS Foundation. [n. d.]. Electron Apps. https://www.electronjs.org/apps

[5] Xinyu Ma, Zhaoning Kong, and Eric Newberry. [n. d.]. python-ndn. https://github.com/zjkmxy/python-ndn

[6] Named Data Networking. [n. d.]. Content Store Management. https://redmine.named-data.net/projects/nfd/wiki/CsMgmt

[7] Named Data Networking. [n. d.]. Getting Started with NFD. https://github.com/named-data/NFD/blob/master/docs/INSTALL.rst

[8] Named Data Networking. [n. d.]. Infoedit: Boost INFO file editor. https://github.com/NDN-Routing/infoedit

[9] Named Data Networking. [n. d.]. ndn-autoconfig. https://named-data.net/doc/NFD/current/manpages/ndn-autoconfig.html

[10] Named Data Networking. [n. d.]. NDN Control Center. https://github.com/named-data/NDN-Control-Center

[11] Named Data Networking. [n. d.]. ndn-cxx. https://named-data.net/doc/ndn-cxx/current/

[12] Named Data Networking. [n. d.]. NDN Essential Tools. https://github.com/named-data/ndn-tools

[13] Named Data Networking. [n. d.]. ndn-lite. https://ndn-lite.named-data.net/

[14] Named Data Networking. [n. d.]. NDN Testbed. https://named-data.net/ndn-testbed/

[15] Named Data Networking. [n. d.]. ndncert. https://github.com/named-data/ndncert

[16] Named Data Networking. [n. d.]. ndnsec. https://named-data.net/doc/ndn-cxx/current/manpages/ndnsec.html

[17] Named Data Networking. [n. d.]. NFD - Named Data Networking Forwarding Daemon. https://named-data.net/doc/NFD/current/

[18] Named Data Networking. [n. d.]. NFD Management Protocol. https://redmine.named-data.net/projects/nfd/wiki/Management

[19] Named Data Networking. [n. d.]. nfdc. https://named-data.net/doc/NFD/current/manpages/nfdc.html

[20] Named Data Networking. [n. d.]. NLSR. https://named-data.net/doc/NLSR/current/

[21] Named Data Networking. [n. d.]. nlsrc. https://named-data.net/doc/NLSR/current/manpages/nlsrc.html

[22] National Institute of Standards and Technology. [n. d.]. NDN-DPDK. https://github.com/usnistgov/ndn-dpdk

[23] Moses Palmér. [n. d.]. pystray. https://github.com/moses-palmer/pystray

[24] Li Yanbiao, Alexander Afanasyev, Junxiao Shi, Haitao Zhang, Zhiyi Zhang, Tianxiang Li, Edward Lu, Beichuan Zhang, Lan Wang, and Lixia Zhang. 2018. *NDN Automatic Prefix Propagation.* Technical Report NDN-0045, Revision 1. Named Data Networking. https://named-data.net/wp-content/uploads/2018/03/ndn-0045-1-auto-prefix-propagation.pdf

[25] Cheng Yi, Alexander Afanasyev, Ilya Moiseenko, Lan Wang, Beichuan Zhang, and Lixia Zhang. 2013. A case for stateful forwarding plane. *Computer Communications* 36, 7 (2013), 779 – 791. https://doi.org/10.1016/j.comcom.2013.01.005

[26] Haitao Zhang, Yanbiao Li, Zhiyi Zhang, Alexander Afanasyev, and Lixia Zhang. 2018. NDN Host Model. *SIGCOMM Comput. Commun. Rev.* 48, 3 (September 2018), 35–41. https://doi.org/10.1145/3276799.3276804

[27] 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. *IEEE Communications Magazine* 56, 11 (2018), 62–68.