NDN Host Model

Haitao Zhang
University of California, Los Angeles
haitao@cs.ucla.edu

Yanbiao Li
University of California, Los Angeles
lybmath@cs.ucla.edu

Zhiyi Zhang
University of California, Los Angeles
zhiyi@cs.ucla.edu

Alexander Afanasyev
Florida International University
aa@cs.fiu.edu

Lixia Zhang
University of California, Los Angeles
lixia@cs.ucla.edu

This article is an editorial note submitted to CCR. It has NOT been peer reviewed.
The authors take full responsibility for this article's technical content. Comments can be posted through CCR Online.

ABSTRACT

As a proposed Internet architecture, Named Data Networking (NDN) changes the network communication model from delivering packets to destinations identified by IP addresses to fetching data packets by names. This architectural change leads to changes of host functions and initial configurations. In this paper we present an overview of the basic functions of a host in an NDN network, together with necessary operations to configure an NDN host. We also compare and contrast the functionality and configuration between an NDN host and an IP host, to show the differences resulted from the different architecture designs.

CCS CONCEPTS

• **Networks → End nodes**; **Network manageability**;

KEYWORDS

NDN, host function, host configuration

1 INTRODUCTION

The concept and properties of a network *host* are well-understood with respect to today's Internet architecture [1]. It is a physical or virtual device that runs applications. Hosts are provisioned, either manually or via automated protocols, with IP addresses, forwarding configurations, DNS server settings, and other parameters. A host encapsulates and sends application data to destinations identified by IP addresses. Generally speaking, the network setting of an IP host (either an IPv4 host or an IPv6 host) is transparent to applications, but with a few exceptions. For example, whenever a host moves to another network, it needs to be re-provisioned with the parameters associated with the new location, which may require dynamic updates to DNS-IP address mapping, or potentially impact the applications running on the host (e.g., breaking the existing TCP connections).

In Named Data Networking (NDN) [2–4], a proposed data-centric network architecture, the concept of a network host still applies: it is a node running applications. However, due to the change to the communication model, an NDN host differs from an IP host in some fundamental ways regarding the parameters to be configured and their impact on applications. In particular, NDN application and network layers share the same namespace. The hierarchically structured names assigned to data by applications are used by NDN forwarders to direct requests toward data and by the hosts to demultiplex the requests to the right application process. Moreover, NDN requires all *Data* packets to be properly signed; therefore all the hosts and applications running on them need to be configured with cryptographic keys and certificates associated with their names, together with appropriate trust anchor(s), and trust policies.

One notable point is that NDN host and application names are not necessarily associated with the name of the network the host is attached. Thus, when a mobile host attaches itself to a new network, it may not need to re-provision its host and application names; however, it may need to re-advertise those names, so that consumer requests can be forward to the host.

To help the reader establish a mental context for this paper, we start with a brief review of IP host's basic functions and configurations (Section 2), followed by a quick introduction of the basic concepts in NDN (Section 3). We then describe the three contributions this paper aims to make: (i) a comprehensive overview of an NDN host's functions (Section 4) and configurations (Section 5); (ii) a comparative analysis of NDN vs IP host functions and configurations (Section 6); and (iii) identification of the remaining tasks needed to automate NDN host configurations (Section 7).

2 IP HOST FUNCTIONS AND CONFIGURATIONS

In an IP host, different namespaces are used at different protocol layers. An application running on a host produces data following application-defined schemas, and passes the data to the transport layer for delivery. The application process looks up DNS to convert the names used by the application to destination IP addresses; it also uses a well-known port number (based on the application type), or may use a system-assigned port number (the application can influence which port is assigned). The transport layer encapsulates (decapsulates) application data into (from) segments, adds (removes) the transport layer header, and performs multiplexing (demultiplexing) based on port numbers. The network layer encapsulates (decapsulates) transport segments into (from) network datagrams, adds (removes) the network layer headers, and forwards those datagrams whose destinations, i.e., IP addresses passed by the transport layer, are not on the local subnet to the gateway router. The data link layer encapsulates (decapsulates) network datagrams into (from) frames, adds (removes) the data link layer headers, and

delivers outgoing frames to next hop (determined by other mechanisms). Notice that, each of the transport, network, and data link layer *adds a new header* to the packet, to carry information needed to perform its own functions.

The IP network layer has its own IP address space. When connecting to a network, an IP host needs to be configured with IP addresses, network prefixes, gateway routers, and DNS servers, either manually, or by some auto-configuration protocols, such as Dynamic Host Configuration Protocol (DHCP) [5, 6], or stateless address autoconfiguration mechanisms [7]. Notice that, as far as IP connectivity is concerned, an IP host does not see application semantics, so IP host configurations for network connectivity do not involve application configurations; applications perform their configurations independently in their own ways.

3 NAMED DATA NETWORKING

In the NDN architecture, a host performs work at four layers: application (transport functions are implemented in system libraries used by applications), network, data link, and physical. Compared with the IP architecture (see Figure 1), NDN changes the protocol layers above network layer, and runs NDN network layer over link layer (the link layer can be virtual links made of TCP/UDP/IP tunnels).

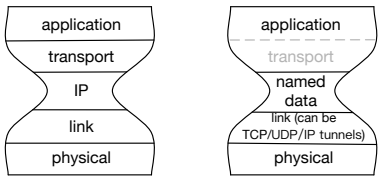


Figure 1: TCP/IP Architecture vs NDN Architecture

In NDN, every entity [8] (e.g., a host, an application, or a user) has a semantically meaningful name, which is associated with a public/private key pair, and the public key should be certified by a self-signed or certificate authority (CA) issued *certificate*; a certificate is simply a piece of named data and is embedded in a *Data* packet (in the rest of this paper, we use "*Data*" to represent both "*Data* packet" and "*Data* packets").

In an NDN host, a data-producing application signs every *Data* it produces using its private keys directly, or indirectly (e.g., signing a manifest made of the digest of a set of *Data* [9]). It may encrypt the *Data* if the content access control is needed. Data-consuming applications verify *Data*'s signatures by fetching the producer's public key certificates, then decide whether to trust the *Data* according to their own trust anchors and trust rules [10]. If the *Data* is access-controlled and hence encrypted, authorized consumers will be granted the corresponding decryption keys. *Interest* packets (in the rest of this paper, we use "*Interest*" to represent "*Interest* packet") are not signed by default; however, when an *Interest* is used as a command or a producer needs to authorize the consumer for specific purpose, an *Interest* can also be signed [11].

NDN directly uses application names for network layer packet delivery. NDN network layer forwards application-created *Interests* and *Data* directly, with no addition of new header or modification. Specifically, it forwards *Interests* based on their names, and forwards *Data* by reversing the path taken by the requesting *Interest*. One *Interest* retrieves one *Data* with the same name or a matching prefix.

NDN generalizes "network interfaces" and "application interfaces" to "faces" (Figure 2). Faces unifies (i) lower-level (layers under NDN network layer) transmission mechanisms/protocols that deliver packets to other hosts, and (ii) communication channels towards local applications into the same packet processing logic which is implemented in the NDN forwarding module. The forwarding module maintains three basic data structures [4, 12]: a Pending Interest Table (PIT), a Content Store (CS), and a Forwarding Information Base (FIB). The PIT stores all received *Interests* that have not been matched with *Data* and have not expired. The CS caches, according to cache policies, previously retrieved *Data* to satisfy future *Interests* for the same data. FIB entries, each is keyed by a name prefix, record one or multiple faces via which potential source(s) of matching Data can be reached. In addition, the forwarding module also maintains *Forwarding Strategies*, which determine whether, when and where to forward each *Interest*, according to the longest-prefix-matched FIB entry, the observed performance from the 2-way *Interest*/*Data* exchange, and the local policies. Upon receiving an *Interest*, the NDN forwarding module first checks CS to see whether any cached *Data* has a matching name, and replies with that *Data* if a match is found; otherwise, the forwarder checks PIT to see whether any PIT entry has the same name, if so it aggregates the new Interest with the existing PIT entry. If both checks fail, then the Interest is handed over to the forwarding strategy, which looks up the FIB and makes a forwarding decision. Details of NDN packet processing logic can be found in [4], [12] and [13].

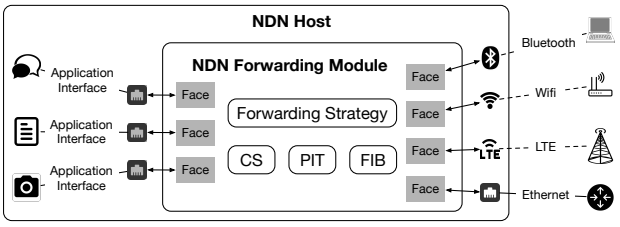
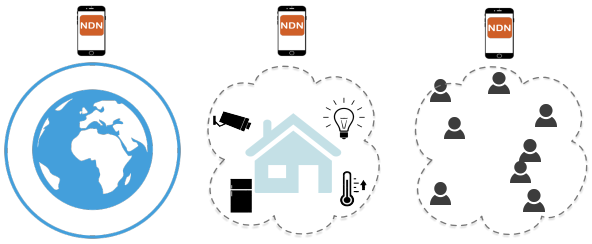


Figure 2: NDN forwarding module and faces in a host

An NDN Host Example

To facilitate explanations of NDN host functions and configurations, we use a mobile phone as an NDN host example in this paper. A user *Alice* has an NDN-enabled mobile phone, and she uses this phone to browse the global NDN network (Figure 3(a)) which relies on the infrastructure support, to control her own smart home network (Figure 3(b)), or to communicate with other NDN hosts within her local vicinity (Figure 3(c)). NDN enables the latter two cases to work through wireless broadcast channels and ad hoc connectivity, without reliance on infrastructure support .



(a) global network (b) smart home (c) ad hoc network

Figure 3: An NDN Host in Different Networks

4 NDN HOST FUNCTIONS

NDN host may performs four basic functions, as shown in Table-1. From an application's perspective, an NDN host provides it necessary running environment, and allocates to it a sub-namespace and issues certificates;[1] From the network's perspective, an NDN host runs the NDN forwarding module to dispatch packets among all the faces, and sometimes behaves like a packet mule when the host is on the move. This section uses the mobile phone example described earlier to illustrate these functions.

Table 1: NDN Host Functions and Configurations

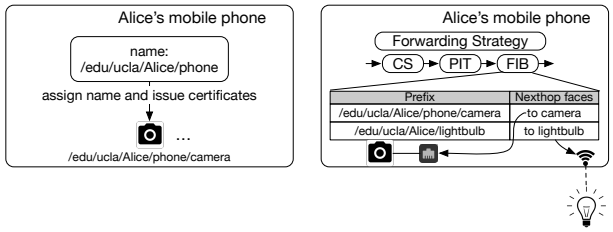
from whose perspective	functions	required configurations
application	application running environment provider	N/A
	namespace manager	name, public/private key pair and certificate; trust rules and trust anchor
network	packet dispatcher	faces, FIB and forwarding strategies
	packet mule	cache policy

4.1 Application Running Environment Provider

Alice's mobile phone may host multiple applications and provides necessary resources for them to work properly. That includes the storage to install applications and store data, system resources—such as CPU and memory—to run applications, and system libraries supporting various commonly needed functions, e.g. reliable data fetch, or NDN's transport functions, such as distributed dataset synchronization protocols [14].

4.2 Namespace Manager

After obtaining a name, generating a public/private key pair for the name and receiving a certificate for the public key (see Section 5.1 for how to perform these functions), Alice's mobile phone can delegate sub-namespaces and issue certificates to other NDN entities under its namespace. As an example, Figure 4(a) shows that Alice's mobile phone has a name "`/edu/ucla/Alice/phone`" and its keys and certificate. An application running on Alice's mobile phone, e.g. the camera application, can obtain a sub-namespace "`/edu/ucla/Alice/phone/camera`", and request certificate from the phone. All the applications, whether running on Alice phone or elsewhere, can authenticate each other's data if they share the same trust anchor; applications with different trust anchors can also build trust relationships among them through proper trust rules [15].



(a) Namespace Manager (b) Packet Dispatcher

Figure 4: Alice's Phone Works as an Namespace Manager and Packet Dispatcher

4.3 Packet Dispatcher

The NDN forwarding module's name-based forwarding logic and face abstraction make Alice's mobile phone a dispatcher for local applications (Figure 4(b)). More specifically, the forwarding module on the mobile phone creates a face to reach each local application and adds a FIB entry with that face as the next hop to reach the app's namespace; it also creates a face for each network interface and adds a FIB entry to reach external namespaces in the similar way. This setup enables the forwarding module to directly "dispatch" received *Interests*, which may come from either local or external faces, that match local applications namespaces to the right local faces, and "dispatch" received *Data* according to PIT entries.

4.4 Packet Mule

Equipped with PIT and CS, Alice's mobile phone can buffer both *Interests* and *Data* and carry them along as a packet mule when it moves. This feature enables resilient forwarding when network connectivity is unstable or intermittent, such as in an ad hoc scenario in Figure 3(c). We use the example shown in Figure 5, a more elaborated pictire of Figure 3(c), to illustrate packet muling.

In this ad hoc network, Bob's and Cathy's mobile phones are too far apart to communicate with each other directly; however, both Bob's and Cathy's mobile phones have unstable and intermittent connections with Alice's mobile phone situated in the middle. Bob's mobile phone wants to fetch a *Data* "`/edu/ucla/Cathy/phone/pic-1`" produced by Cathy's mobile phone, so it keeps sending *Interests* to request the *Data*.

Interest Muling: As Alice's mobile phone can occasionally communicate with Bob's phone, it receives the *Interest* for "`/edu/ucla/Cathy/phone/pic-1`" and inserts it into the PIT. Alice's mobile

[1]Note that a host may also host applications whose namespace is not under that of the host.

phone then becomes an *Interest* mule: this *Interest* is kept in its PIT, even when the connectivity to Bob's mobile phone is lost.

Data Muling: In the same way, Alice's mobile phone forwards this *Interest* to Cathy's phone, retrieves the requested *Data* back, and stores it into its CS. Alice's mobile phone then becomes a *Data* mule by caching this *Data* in the CS, even if its connectivity with Cathy's mobile phone is lost. Assuming that Bob's mobile phone keeps re-expressing the unsuccessful *Interest*, Alice's mobile phone will receive the same *Interest* again and reply with the cached *Data*.

The same logic should work even when Bob and Cathy's mobile phones can reach each other intermittently via multiple hops. However we must note that the PIT and CS in each NDN host provide only the buffering function for Interest and *Data* to enable packet muling. Additional mechanisms, such as the simple persistent trying in this example, are needed to achieve effective communications.

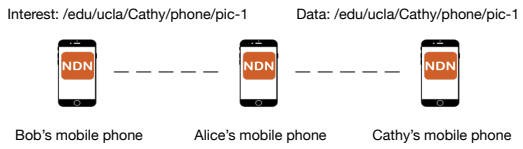


Figure 5: Alice's Phone Works as a Packet Mule

5 NDN HOST CONFIGURATIONS AND NAMESPACE REACHABILITY

To perform NDN host functions summarized in Section 4, an NDN host, Alice's mobile phone in our example, needs to be properly configured. A few necessary configurations enables the mobile phone to discover namespaces, learn reachability to external namespaces, and propagate its own namespaces out by following given policies, so that others can send *Interests* to the host—Alice's phone.

Table 1 summarizes the configurations needed for each host function. The current implementation of NDN forwarding module pre-defines a few simple cache policies and forwarding strategies, which we do discuss here. Except these two, a host needs three other configurations. First, an NDN host needs to learn at least one trust anchor together with default trust rules to enable it to verify received Data. Second, it must obtain a name under which it can publish *Data* and allocate sub-namespaces and associated security credentials. Third, its local forwarding module needs to set up initial faces and FIB entries for NDN communication.

In this section, we group host configurations into two categories: 1) security bootstrapping (Section 5.1) and 2) initial forwarding configurations (Section 5.2). After the initial configuration, the forwarding module in a host continues to adapt to applications' requirements as well as connectivity changes through namespace discovery and propagation (section 5.3).

5.1 Security Bootstrapping

When an NDN host first appears online, it must get a name, then go through a security bootstrapping process, through which the host obtains its trust anchor, its certificate, and learns trust policies. Alice's mobile phone, at this step, needs to securely obtain its name and go through security bootstrapping as described below. The description is brief due to the space limitations; readers interested in learning more about NDN security bootstrapping may find further details from [15].

Obtain a Name. The first step of host configuration is to obtain a name if the host does not have one yet. Generally speaking, some out-of-band knowledge is required to decide which entity should assign the host a name and how the name is structured. For example, Alice's mobile phone is named ("`/edu/ucla/Alice/phone`"), either via manual configuration or from the owner of its parent name space (e.g., "`/edu/ucla/Alice`").

Learn Trust Anchor and Trust Rules. With a name, Alice's mobile phone must obtain a trust anchor, which will allow the phone to obtain a certificate, fetch the trust rules in order to establish trust relationship with other NDN entities and learn about who can do what. In general, the trust anchor can be pre-installed, through out-of-band operations, into the mobile phone. The phone can fetch an initial set of trust rules by following a given naming convention, and establishing the trust anchor enables the phone to verify the received trust rules.

Generate Public/Private Key Pair and Request Certificate. To prove its ownership of its name obtained in the first step, an entity needs to get a CA-issued certificate; notice that the CA should be trustworthy as well according to its trust anchor(s) and trust rule(s). The mobile phone generates a public/private key pair, with the private key safely stored locally and the public key named under its name; then it requests a pre-defined (or discovered) CA—such as UCLA CA in the global network—to sign the public key to generate a certificate; finally, it receives the certificate and saves it.

Security bootstrapping aims to establish initial trust, and the process itself, which involves several *Interest/Data* exchanges between the mobile phone and the CA, must be secured. To this end, the authenticity of each side and integrity of every packet should be ensured (e.g., via HMAC, with a symmetric key pre-shared between both sides through a secure out-of-band operation), and dynamic challenges are required to prevent potential replay attacks (e.g., via including randomized tokens in communication messages).

5.2 Initial Forwarding Configurations

To initialize NDN communication, the mobile phone's forwarding module loads a configuration file to create initial faces and FIB entries accordingly. More specifically, the forwarding module creates some *local faces* for NDN forwarding module's internal use, such as by the forwarding module management; it also enumerates all physical network interfaces and creates one or multiple *non-local* faces on each of them to reach external namespaces, independent from the specific underlying communication technologies those interfaces use.

At the same time, some well-known and locally scoped name prefixes are registered on those faces mentioned in the last paragraph, creating initial FIB entries. Prefixes starting with "`/localhost`" are registered on the local faces , so that management commands (such as *Signed Interests*) can be dispatched to corresponding management modules; prefixes starting with "`/localhop`" are registered on non-local faces, enabling name discovery within one physical

hop. With those initial FIB entries, the mobile phone can start to receive and forward *Interests*. Readers can find further details about the NDN node forwarding configuration from [12].

5.3 Namespace Reachability

Interest forwarding is determined by forwarding strategies (which is not discussed in this paper) with input from the FIB, which indicates which namespaces can be reached through which faces. An NDN host, such as Alice's phone, does not run a routing protocol in general, so its FIB is not manipulated by routing protocols. Instead, the forwarding module fills the FIB using a combination of default routing, self-learning, and local application registrations, together with automatic adaptation to connectivity changes.

New FIB entries would be created in the local forwarding module when new namespaces become reachable. First, after being authenticated, every local application can ask the local forwarding module to create faces (between the forwarding module and local applications), and register name prefixes on those faces to create FIB entries. For example, the camera application, with a name and certificate obtained from the local host, can ask the forwarding module on Alice's mobile phone to create a FIB entry for "`/edu/ucla/Alice/phone/camera`". Second, the local NDN forwarding module can create FIB entries based on information learned from others. More specifically, if the "self-learning" strategy [16] is triggered by an *Interest*, the forwarding module will broadcast the *Interest* to all available faces; it can then learn a prefix announcement provided by the target producer from the first response, and create a FIB entry, as well as a face if necessary, for forwarding future *Interests* under the prefix. Besides, an NDN host can propagate [17] its own prefixes, aggregated from prefixes in the local FIB, to other nodes, for them to create faces and FIB entries to forward *Interests* matching those prefixes to the host. For instance, Alice's phone can propagate a prefix "`/edu/ucla/Alice/phone`" to a global network router.

FIB adaptations can also be triggered by connectivity changes. One example is, in dynamic environments, like the ad hoc network in Figure 3(c) where connectivity is intermittent, or the mobile phone moves from one WLAN to another, its network interfaces may go up and down repeatedly or have link-layer parameters changes (such as SSID or radio channels). When a network interface goes down, the corresponding face(s) will be closed; the dead face(s) will also be removed from the associated FIB entries to avoid *Interests* being forwarded to them. When a network interface goes up, one or more faces will be created immediately according to forwarding module's configurations, or later on along with the FIB entry via "self-learning". Moreover, link-layer parameter changes will also trigger face closures and creations with similar forwarding adaptions.

6 NDN HOST & IP HOST: A COMPARISON

NDN's building block—named, signed *Data*, and shared namespace between application and network layers—fundamentally changes the functions and configurations of hosts. To help the reader see these changes clearly, this section compares NDN hosts with IP hosts from four aspects, to illustrate the differences resulted from the network architecture changes.

6.1 Addresses vs. Topology-Independent Names

IP networks (IPv4 or IPv6) use their own address space for network layer packet forwarding, which is totally independent from the namespaces used by applications. When an IP host moves to a new network, it must be re-provisioned with new IP address(es) and related parameters, to fit itself into the new network location.

In contrast, application and network layers on an NDN host share the same namespace. The NDN network layer directly forwards application-created *Interests* and *Data* based on names. Once an NDN host and applications running on it get configured with their names and associated security credentials, the host may propagate the namespace prefixes (maybe not all) reachability to attached NDN routers and other hosts. When an NDN host moves to another network, no reconfiguration is needed. Depending on the network management policies, these application prefixes may not propagate far, especially for mobiles. Thus, NDN's use of topology-independent names requires additional means, such as *Forwarding Hints* [18], to make all *Data* reachable. Forwarding Hints introduces additional complexity and overhead, as a necessary cost for the gains of using application names for network layer delivery.

6.2 Security: Channel-Based vs. Data-Centric

Security solutions were patched into IP protocol stack after its initial design. IP hosts secure communication channels by using IPsec [19] protocol at the network layer and Transport Layer Security (TLS) [20] at the transport layer; in cases stronger security is needed, applications adopt other protocols at the application layer.

NDN was designed with security support: it uses named and signed packets to design and implement security solutions. Packet signatures enforce packet-level verification, and trust rules define who can do what, enabling fine-grained authenticity; NDN also provides other mechanisms, such as name-based access control (NAC) [21], to enforce confidentiality when privacy is a major concern.

NDN's data-centric security framework ensures that *Data* are secured at their generation time; then they stay secured, independent of whether they are in transit or not, and where they are stored. As application-created packets are directly used at the network layer in NDN, once applications properly secure their packets, the network communications are naturally secured. Meanwhile, compared with current channel-based security practice used on IP hosts, NDN's new security mechanism does not depend on powerful servers or middle boxes to handle secured channels with NDN hosts; each secured NDN packet may be repeatedly consumed by multiple consumers, which is more efficient than channel-based security model, where each piece of *Data* needs to be secured separately for each consumer.

6.3 Forwarding

Hiding Transmission Details by Adopting Unified Interface. In an IP host, looking up from the network layer, the transport layer multiplexes outgoing application data and demultiplexes incoming network datagrams based on port numbers. When forwarding outgoing datagrams, which is based on destination IP addresses and forwarding rules, and looking down to the data link layer, an

IP host needs to run ARP or NDP (or other protocols serving the same purpose) to determine the link layer destination address, then creates and delivers frames to the link layer destination.

By treating not only physical network lower-layer communication channels between hosts but also inter-process communication channels between NDN forwarding module and local applications as faces, NDN hides transmission details and provide unified interfaces to the network layer. Further, by configuring which name prefixes should go to which faces in FIB, NDN hosts unify incoming and outgoing packets processing logic; thus, they can directly forward the packets to other hosts or local applications based on Forwarding Strategies and FIB.

Adapting to Namespace Reachability and Network Connectivity Changes. As IP network uses its own address space, once an IP host is provisioned, all applications running on it should make use of those network configurations, rather than the network configurations adapt to applications' requirements and changes. In some cases, such as when using TCP (but not MPTCP [22]) as the transport protocol, applications should also adapt to host network configuration changes, such as tear down previous connections and use new IP address(es) to set up new connections with network servers.

Utilizing application-layer namespaces, which usually do not change along with host movement or connectivity changes, directly at the network layer, NDN forwarding does not need to change along with host movement or connectivity changes, but rather, it needs to adapt to namespace reachability and network connectivity changes (see Section 5.3 for details of how to perform adaptation). As a result, applications running on NDN hosts do not need to take account network configurations changes, instead, they simply focus on their own logic.

Supporting Host Multihoming. An IP host is multihomed when it gets configured with multiple IP addresses, each represents a network location, and are connected to multiple networks. Usually, when application or transport layer protocols use IP addresses to identify connectivity/connections, i.e., when higher-layer identifiers are coupled with network locations, they cannot take advantages of host multihoming. To support host multihoming in IP, network-location-independent higher-layer identifiers are required. For example, MPTCP [22] uses multiple IP addresses, each is used in a subflow; QUIC [23] identifies its connections using connection IDs, which are independent from IP addresses.

Compared with IP, NDN utilizes application names, naturally independent from network location and connectivity/connections. As a result, NDN supports host multihoming at network layer and allows one to send *Interests* and fetch *Data* via all/any available faces.

Utilizing Forwarding Strategy. In IP hosts, whether, when, and where to forward network datagrams are determined by destination IP address (of datagrams being forwarded), its type, and forwarding rules. Datagrams destined to broadcast and multicast IP addresses are broadcasted and multicasted respectively; datagrams destined to unicast IP addresses could only be forwarded via one physical interface according to the configured local subnet and default router.

In contrast, since NDN inherently supports host multihoming, the NDN forwarding module needs a decision module—that are forwarding strategies—to decide whether, when, and where to forward *Interests*. Importantly, NDN's *Interest-Data* packet exchange, where *Data* are forwarded back to where requesting *Interests* came, provides rich feedback of current network conditions, thus enabling forwarding strategies to make more intelligent decisions. For example, *Best Route Strategy* ranks nexthop faces from lowest cost to highest cost, sends *Interests* to them one by one if a previous does not work; *ASF strategy* prioritizes nexthop faces based on their performance in *Data* retrieval delay [12]. Apart from those existing ones, new forwarding strategies can be designed and added when the existing ones cannot achieve the desired goals.

6.4 Data Cache

In IP architecture, as the network layer does not understand application layer data, generic in-network caching of IP packets is not beneficial. Therefore, a host discards packets after forwarding them or finding that it does not know how to forward them. NDN's data-centric communication model, along PIT and CS inside NDN forwarding module, enables in-network cache, making hosts to be natural *Interest* and *Data* mules (application level cache may still be needed if NDN network level packet mule function is not enough).

As a result, an IP host can deliver data to another host only when they are stably connected; while in NDN, consumers can fetch *Data* not only from data producers, but also from other hosts that have requested or forwarded the *Data* before, even there is only unstable and intermittent connectivity (see Figure 5 for an example). This amplifies host forwarding function and improves data availability.

7 CONCLUSION AND FUTURE WORK

As a new Internet architecture design, NDN changes the host model from TCP/IP architecture in a fundamental way. To help people interested in NDN, especially new NDN developers, get a basic understanding of the NDN host model, this paper introduces the basic functions of NDN hosts, and the bootstrapping process for an NDN host to obtain necessary parameters to enable an NDN host to function.

Different from an IP host which obtains one or multiple IP addresses from the network(s) it is attached to, an NDN host obtains semantically meaningful name(s) that may or may not be location specific. In addition to names, an NDN host must also possess cryptographic key(s) and obtain certificates. The certified names, along with trust rules and trust anchors, enable an NDN host to establish trust relationship with other NDN entities, which further enable NDN forwarding to adapt to namespace reachability and network connectivity changes. We compare the NDN hosts with IP hosts, demonstrating a number of differences in configuration, security, forwarding and data cache.

The NDN development is still on-going, thus our understanding of the NDN host model is likely to evolve over time as well, together with our implementations in supporting NDN host functions and configurations. Although we have implemented a number of mechanisms, protocols and tools for configuring NDN hosts to perform its basic functions, a big gap remains between how the configurations should be done ideally and what has been implemented so far. For instance, the cache policy and forwarding strategies are currently preconfigured in the NDN Forwarding Daemon (NFD) [12]; in the

future, we plan to support application or user defined cache policies and forwarding strategies. As another example, NDN Certificate Management Protocol (NDNCERT) [24] aims to automate security configurations for new NDN hosts with UIs to obtain names and request certificates from NDN testbed CAs; however we are also porting NDN to run on small devices which have no or limited user interfaces, this argues for eliminating dependency on UIs from NDN configuration designs in general.

ACKNOWLEDGMENTS

This work is partially supported by the National Science Foundation under awards CNS-1629922 and CNS-1719403. The writing also greatly benefitted from the comments of the CCR editor.

[1] Kevin R Fall and W Richard Stevens. *TCP/IP illustrated, volume 1: The protocols*. addison-Wesley, 2011.

[2] Van Jacobson, Diana K Smetters, James D Thornton, Michael F Plass, Nicholas H Briggs, and Rebecca L Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 1–12. ACM, 2009.

[3] Lixia Zhang et al. Named Data Networking (NDN) project. Technical Report NDN-0001, NDN, 2010.

[4] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, et al. Named Data Networking. *ACM SIGCOMM Computer Communication Review*, 2014.

[5] Ralph Droms. Dynamic host configuration protocol. RFC 2131, 1997.

[6] Ralph Droms, Jim Bound, Bernie Volz, et al. Dynamic host configuration protocol for IPv6 (DHCPv6). RFC 3315, 2003.

[7] Susan Thomson, Thomas Narten, and Tatuya Jinmei. IPv6 stateless address autoconfiguration. RFC 4862, 2007.

[8] Zhiyi Zhang, Haitao Zhang, Eric Newberry, et al. Security in named data networking. Technical Report NDN-0057, NDN, 2018.

[9] Ilya Moiseenko. Fetching content in named data networking with embedded manifests. Technical Report NDN-0025, NDN, 2014.

[10] Yingdi Yu, Alexander Afanasyev, David Clark, et al. Schematizing trust in Named Data Networking. In *Proceedings of the 2nd ACM Conference on ICN*, pages 177–186. ACM, 2015.

[11] NDN Project Team. Signed Interest. https://named-data.net/doc/ndn-cxx/current/specs/signed-interest.html, 2018.

[12] 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. Nfd developer's guide. Technical Report NDN-0021, Revision 10, NDN, July 2018.

[13] Cheng Yi, Alexander Afanasyev, Ilya Moiseenko, Lan Wang, Beichuan Zhang, and Lixia Zhang. A case for stateful forwarding plane. *Computer Communications*, 36(7):779–791, 2013.

[14] Wentao Shang, Yingdi Yu, Lijing Wang, Alexander Afanasyev, and Lixia Zhang. A survey of distributed dataset synchronization in named data networking. Technical Report NDN-0053, NDN, May 2017.

[15] Zhiyi Zhang, Yingdi Yu, Haitao Zhang, Eric Newberry, Spyridon Mastorakis, Yanbiao Li, Alexander Afanasyev, and Lixia Zhang. An overview of security support in Named Data Networking. Technical Report NDN-0057, Revision 3, NDN, June 2018.

[16] Junxiao Shi, Eric Newberry, and Beichuan Zhang. On broadcast-based self-learning in named data networking. In *Proceedings of IFIP Networking*, 2017.

[17] Yanbiao Li, Alexander Afanasyev, Junxiao Shi, et al. NDN automatic prefix propagation. Technical Report NDN-0045, NDN.

[18] Alexander Afanasyev, Cheng Yi, Lan Wang, Beichuan Zhang, and Lixia Zhang. SNAMP: Secure namespace mapping to scale NDN forwarding. In *Proceedings of 18th IEEE Global Internet Symposium (GI 2015)*, April 2015.

[19] Rodney Thayer, Naganand Doraswamy, and Rob Glenn. IP security document roadmap. RFC 2411, 1998.

[20] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, RFC Editor, August 2008. http://www.rfc-editor.org/rfc/rfc5246.txt.

[21] Yingdi Yu, Alexander Afanasyev, and Lixia Zhang. Name-based access control. Technical Report NDN-0034, NDN, 2015.

[22] Alan Ford, Costin Raiciu, Mark Handley, Sebastien Barre, and Janardhan Iyengar. Architectural guidelines for multipath TCP development. RFC 6182, 2011.

[23] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. The QUIC transport protocol: Design and Internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 183–196. ACM, 2017.

[24] Zhiyi Zhang, Yingdi Yu, Alexander Afanasyev, and Lixia Zhang. NDN certificate management protocol (NDNCERT). Technical Report NDN-0050, NDN, 2017.