Lessons Learned Building a Secure Network Measurement Framework using Basic NDN

Kathleen Nichols
Pollere, Inc.
Montara, CA, USA

ABSTRACT

The Named-Data Networking Project has moved from a multi-university NSF-funded Future Internet Architecture project to an open source codebase with world wide contributors and a growing body of applications. Researchers have applied NDN to applications like lighting control, vehicular communications, and augmented reality but more work is needed to make the data-centric and security features of NDN accessible. Users are currently required to become experts on the internals of the codebase, a difficult task further complicated by the lack of well-documented examples and the project adding new features. While implementing a secure, distributed network measurement framework for NDN, we encountered two major difficulties: the lack of a library of application-usable communications models (built on top of the NDN layer) and the difficulty of integrating trust rules with the NDN codebase.

This paper describes our NDN network measurement framework and the co-developed tools that implement its secure, publish/subscribe communications model. Our goals are both to present the network measurement framework and to motivate developers to evolve NDN by creating frameworks, libraries, and includible headers rather than bloating NDN's waist.

CCS CONCEPTS

• **Networks** → **Layering**; *Transport protocols*; *Security protocols*; *Network measurement*.

KEYWORDS

NDN, transport, trust schema, ICN, network measurement

ACM Reference Format:
Kathleen Nichols. 2019. Lessons Learned Building a Secure Network Measurement Framework using Basic NDN. In *6th ACM Conference on Information-Centric Networking (ICN '19), September 24–26, 2019, Macao, China.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3357150.3357397

1 INTRODUCTION

Modern distributed systems (cloud computing map/reduce, virtualization, IoT, network monitoring) are information-centric and employ one-to-many, many-to-one, and any-to-any communication patterns rather than the Internet's source-destination model. Although IP multicast and anycast expand the definition of "destination", the majority of Internet traffic follows a source-to-destination model. As a result, modern applications use complex adaptation code in order to translate between information-centric communications models and host-centric communication mechanisms. The model mismatch makes the resulting systems complex and brittle, e.g. an inherently distributed process ends up with a centralized bottleneck in its implementation or a broadcast communications model on a broadcast media ends up sending information in separate connections. Information-Centric Networking (ICN), in particular, Named-Data Networking (NDN)[1], provides an excellent foundation for modern applications due to its information-based security and ability to exploit broadcast media. Originally developed as a Future Internet Architecture, current efforts are focused on NDN for edge networks like Internet of Things (IoT) [31, 33], building management [32], critical military communications [11], solar energy reporting[18, 23], augmented reality[7], and vehicular communications[20].

The range of NDN applications is encouraging but the lack of application-enabling infrastructure is not. NDN's core, or generic primitives, can permit implementation of information-centric communications models more efficiently than IP networking (e.g. [17]) but does not in itself implement a communications model. Internet applications use models like sockets, streaming, transactions, and publish/subscribe rather than calls to, or even awareness of, the network or transport layers. NDN needs reusable packages that can be used to implement a range of communications models and handle the details of Faces and Interest and Data packet construction and parsing, managing mechanisms that move and secure information while integrating application-specific trust rules and application layer data framing. Rather than expecting applications to function without transport functionality, the original NDN architecture envisioned moving transport functions out of a kernel-based layer and into application-accessible libraries and a data synchronization *Sync* building block where they could be customized into bespoke transports that meet the needs of important classes of applications. Although this approach is laid out in the "Transport" section of [3], it has had insufficient use and attention, leaving application writers to work directly with network layer primitives. As the development of the Internet's protocol stack shows, core mechanisms are best kept simple while specialization happens on top of them [12] and new physical media can be added below. The Internet's use of this "hourglass model" allowed it to dominate competing protocols while continuing to enable innovation. Jacobson [19] builds on this history and makes the case for more attention to following the model in NDN.

Linking application communications closely to network layer primitives is creating two major issues. First is that working directly

ICN '19, September 24–26, 2019, Macao, China

ACM ISBN 978-1-4503-6970-1/19/09…$15.00
https://doi.org/10.1145/3357150.3357397

with network primitives causes implementers to add features to the generic core rather than creating code modules to implement features on top of the generic core. This can be seen in [8, 39] which both have the goal of publish/subscribe communications over an Interest/Data network layer. iHEMS [39] creates an architecture and proof-of-concept for a secure ICN-based publish/subscribe infrastructure for a Home Energy Management System with innovative group key management. Its publish/subscribe API is directly mapped to network layer primitives and creates "pesistent" subscriptions by altering the architecture of the network layer using a single Interest to enable the forwarding of multiple Data packets which changes the model to one where PIT state is *not* consumed by Data packets. This alters the critical flow balance attribute of the architecture and alters the forwarder implementation. COPPSS [8] has the goal of an efficient pub/sub content delivery system for ICN. The authors note that equating a subscription query with an NDN Interest (again trying to make direct use of the network layer) leads to obvious problems since a single Data annihilates an Interest. Rather than making use of the built-in multicast primitives of NDN (where Interests serve the role of "joins" in PIM-SM), they propose COPSS and COPSS-aware routers which have an additional element, a Subscription Table, requiring an altered forwarder implementation. A recent paper develops a framework for ICN remote method invocation [16]. The authors have the stated goal of making minimal changes to protocols and forwarder behavior, but their approach does make idiosyncratic use of the FIB in order to create a particular connection-oriented client-server path to reflect the client-server nature of the relationship created by the required four-way handshake. Interoperable deployment of NDN will be difficult or impossible with multiple flavors of its network protocol and variations in Forwarder requirements will complicate development of measurement and management approaches.

The second issue is the lost opportunity to move toward usable, shareable solutions that provide packages, templates, libraries, and includible headers to implement communications models, evolving a best practice and toolkits for each. The lessons learned in implementing each NDN application are lost because they cannot be generalized, shared, and used to ease the writing of further future applications. Projects to develop reusable modules exist [1, 2], but it does not appear that NDN developers are taking the "dogfood" approach of using work developed in earlier projects as scaffolding for further work. Code lacks documentation to guide a user and the applications work at the level of Interests, Data, Faces, and network-specific elements of Names. Moisenko et. al. [22] propose an API for a consumer/producer communications model using data dissemination in NDN networks with a strict separation of "two programming abstractions: one for consumer applications, and another one for producer applications" and does not include a Sync block. The github code uses the deprecated selector Interest field, is not documented and appears to be dormant (no updates in two years). Although a few Sync protocols have been developed [35], PSync and ChronoSync appear to be the only ones in active use. Both of these synchronize on producer-specific streams of sequentially numbered Data, ChronoSync using cryptographic state digests and PSync using and Inverted Bloom Filter to capture the state of a producer's set of datastreams. Abraham et. al. [5] seek to improve ICN usability by adding a non-user space Information-Centric Transport (ICT) to the architecture. In addition to being added to end-points, processes running flavors of this ICT must be deployed by network operators on intermediate nodes. This is presented as a general model with generic examples as the authors note that the true needs and requirements of ICN applications are as yet unclear. ICTs are not currently easing the writing and securing of applications and the need for operator intervention for their use adds some complexity to deployment.

We became intimately acquainted with all of these issues while creating a NDN-based network measurement protocol, an important application championed by NIST.[1] Thus far in NDN development, measurement has attracted little attention but new approaches are needed because NDN's communications are not based on IP's "endpoint" and "flow" abstractions. Further, NDN's stateful forwarding results in NDN Forwarding Daemons (NFDs) containing a lot of operationally useful information. A network measurement approach that makes this and other NDN network health data securely available to network operators is essential to evolve NDN from "research vehicle" to "practical networking infrastructure".

The need for privacy and security in network measurement is widely acknowledged but has been mostly unaddressed in the large number of number of measurement platforms created for various, sometimes overlapping, purposes [6] as the Internet evolved. The IETF LMAP Working Group, chartered to develop an information model and protocols for the secure measurement and control of network access devices, outlined a measurement framework in [14] that enumerates important privacy issues and security threats but does not suggest architectural or protocol means to mitigate them. An initial NDN Network Measurement Framework proposed in [26], largely consistent with [14], also acknowledges, but does not address, the need for data confidentiality and security via an appropriate authentication structure. It is past time to move the need for security in network measurement into reality.

We designed and prototyped a Distributed Network Measurement Protocol (DNMP)[2] as a secure, role-based framework for requesting, carrying out, and collecting measurement of NFDs. DNMP applications both send and receive data that is organized by hierarchical topics, rather than by producers, requiring a new NDN transport. Our design process uses a *bespoke transport* model of includible header files and customizable templates to ease development as well as to create an extensible approach for users with more interest in gathering measurements than in learning the details of the NDN codebase. The bespoke transport model gives a framework for developing user space transports for NDN on top of the basic NDN primitives which, in our experience, provided a sufficient network layer on which to create a useful publish/subscribe transport; all issues were elsewhere in the stack. For example, NDN provides security primitives that ensure provenance of Data, simple solutions for key distribution problems [27, 41], and a trust schema model that allows for specifying trust based on application name structures[36]. Unfortunately for application writers, going from trust specification to secured application has been a complex task lacking in tools for validating or securing trust schemas. Our

[1]Initial work was funded by NIST under 70NANB18H186.

[2]Some (including one reviewer) have pointed out that DNMP could be used for management as well as measurement. If more management applications are developed, the name is easily changed while initials remain the same.

measurement protocol was co-developed with a Versatile Security (VerSec) toolkit that integrates with the transport framework to simplify integration of trust rules. We agree with Abraham et. al. that true needs and requirements of ICN applications will need to come from implementing them and we found our small-scale "dogfood" approach served DNMP, VerSec, and the bespoke transport model well.

Network measurement proved an excellent NDN application, matched well to its communications and security. Communications are inherently multi-way: e.g., a measurer querying network devices for status is one-to-many and those network devices reporting is many-to-one. DNMP was implemented as a fully functional proof-of-concept (POC) deployed on machines running Linux and MacOS connected by a single wifi network shared with other IP traffic. Section 2 describes DNMP. Section 3 provides more background on enabling modules used by DNMP that can be used in any bespoke transport. Section 4 summarizes our proof-of-concept system and gives our current approach to multiple segment network deployment. Section 5 covers future directions and Section 6 concludes. DNMP, its bespoke transport modules, and VerSec are GPL licensed open source.

2 THE DNMP FRAMEWORK

DNMP is a secure role-based distributed framework for NDN measurement. It is designed to collect and export locally obtained measurements (e.g., at a particular device) only in response to properly authorized local or remote requests where measurements may be secured for privacy. Authorization uses the *role* of the requestor and can be even more fine-grained including types and times of requests. Measurements can be simple status counts or statistical distributions from a particular device or combinations of exported data from several devices, reported on demand or at regular intervals. DNMP provides a communications framework and API for its application processes.

Soliciting and collecting network measurements requires one-to-many, many-to-one, and any-to-any communications about *information* rather than connections between devices, making the publish/subscribe communications model a natural fit. Pub/sub has over 30 years of history but has become more important in recent years as it is applicable to a range of modern problems from data center tasks to IoT. In particular, in the open source MQ Telemetry Transport (MQTT)[4], publishers and subscribers communicate through hierarchically structured topics and subtopics, using publications to implement message passing and event signaling. Publishers, which may also be subscribers, do not have specific knowledge of subscribers and publications are made available asynchronously. The number of publishers and subscribers to a topic can range from none to as many as the underlying system supports. This communication model of MQTT is attractive, although its implementation on IP networks is not information-centric.

Portions of the DNMP Framework, like a publish/subscribe Sync, are quite general. With the motivation of following a "dogfood" approach and of creating packages that can be utilized in other NDN applications, DNMP was used as a prototype application for two other Pollere projects, bespoke transport and versatile security (VerSec), described here as they apply to DNMP.

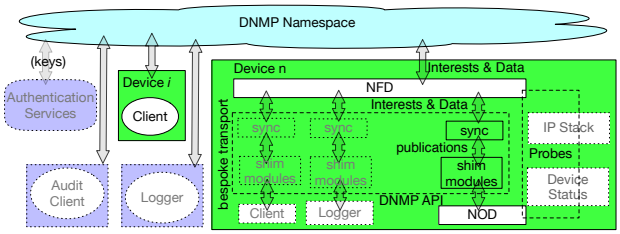

Figure 1: Block Diagram of DNMP

2.1 Basic Operation

DNMP has three required application types, based loosely on [26]: Probes, Network Observer Daemons, and Clients. *Probes* are functions that perform measurements at the direction of a *Network Observer Daemon* (NOD) while *Clients* request measurements. Probes run as subroutines or threads in a NOD. Clients can run anywhere in the network. The Device *n* block in Figure 1 has been expanded to show required and optional portions of DNMP. NODs are persistent processes, one-to-one with NFDs, with installed Probes that access measurements from the NFD and (potentially) from an IP stack or the device hardware. *Client* processes may be ephemeral, varying in complexity and amount of user interaction. Clients may be implemented as shell utilities that initialize, run, and terminate immediately or background tasks that continually monitor network measurements. Bespoke transport functionality is provided by a pub/sub Sync and by Shim modules. The shim implements a simple, intuitive DNMP API to the processes, integrates DNMP trust rules, and handles the interface to a Sync protocol. Shims use run-time libraries from VerSec to ensure properly validated packets are received from and constructed for publication by the Sync and provide subscription callbacks. DNMP security rules are specified in a trust schema that links identities and keys to components of publication names. The Authentication Services block represents the set of configurable but strict conventions that DNMP imposes on existing NDN security mechanisms like signing keys, signing chains, key enclaves (PIB and TPM), trust schemas, identities, verifiers, etc. (sections 2.4 and 3.3). Loggers and Audit Clients are additional DNMP applications under development (section 3.2).

The DNMP API in Figure 1 takes a topic, target and content from DNMP applications and creates publications and subscription requests that are passed to the Sync which converts these into Interests and Data to pass to the NFD (section 3.1). The Sync uses Interests from the NFD to update subscriptions and extracts publications in matching topics from Data to pass to the shim. DNMP uses two topics to solicit and collect measurements, *command* and *reply*. Clients solicit network measurements from targets via publication in command topics and receive the measurements by subscribing to associated reply topics. NODs subscribe to commands and publish replies whose name is derived from the corresponding command. Reply publication may be used to indicate successful command execution and/or command receipt and contains the result or name of the result.[3] Receipt of a new command causes a NOD to execute

[3]Commands have a *target* but replies do not. A reply is targeted to the command, not its source Client, accomplished by starting a reply name with a prefix that's a transformation of the related command.

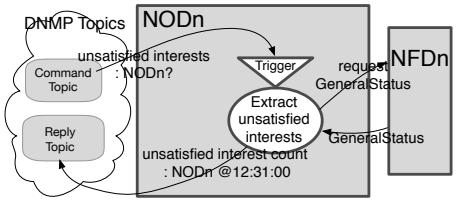

Figure 2: Unsatisfied Interests Probe

a particular Probe; the command publication name contains the information necessary for execution. As command and reply topics are related, they are handled by the same Shim, CRshim. The command API passes the information content of commands and the types of entities for which they are intended (the *target*). *Target* is used by the DNMP framework and becomes part of the publication name. Although publication names are constructed as NDN Names, they are not visible "on the wire" and do not need to have any relationship to network level abstractions like routing prefixes, node names or addresses. Thus DNMP commands identify target NODs using specifiers like "all", "local", "adjacent", configured identities or particular attributes. CRshim and its API includes *only* the elements required by its pair of topics; one Shim instance is responsible for handling the CRshim publications for a specific *target*.

Conceptually, Probes have a *trigger*, a *source*, an *action*, and a *destination*. Triggers can be immediate (i.e., "do this now"), timer-based, or condition-based. Trigger conditions can be anything that is appropriate for a probe, ranging from changes in particular tracked status indicators (e.g. content store size or link utilization) to the publication of new data to a topic (e.g. by another probe). A source can range from NFD counts and status values to timers to topic subscription updates. An action can be as simple as reading a counter but may include more sophisticated computation. Destination is the name the probe output is published under, either extracted from a command or configured as part of the Probe. In Figure 2, a Probe responds to the immediate *trigger* of a new command requesting the unsatisfied Interests count. The Probe uses the GeneralStatus data block of the NFD Management Protocol (the *source*) and takes the *action* of extracting the number of unsatisfied interests to publish in a timestamped reply in (sub)topic NFDData/UnsatisfiedInterests/NODn (the *destination*) where it can be accessed by *any* authorized subscriber. Publication of probe results can be used to trigger another probe that subscribes to that topic.

2.2 Identities and Publications

A Client's DNMP *role* determines what it can access and is set at runtime, deriving from the keys available in the login identity's TPM (Trusted Platform Module). Client roles of operator, user, and guest are used to determine capabilities, e.g. the right to issue certain commands to certain target NODs. Currently, capabilities link roles to accessible targets, with **user** limited to *local* and **operator** permitted to target anything. The trust schema supports extensive specialization that can allow, for example, designated users to issue designated commands to designated NODs at particular times of day. Both Client identities and machine or device IDs are required components of DNMP names, adding *who* and *where* forensic information to the intrinsic *what* and *when*. Access control is generally assymetric e.g., clients can't issue replies, and NODs can't issue commands, preventing compromised NODs from being used to attack DNMP.

NOD identity derives from its host's configuration and its DNMP keys, i.e. identity is derived from an authorized configuration entity for the particular network. Where measurement data privacy is required, access to decryption keys will be more limited than access to the data itself in order to enforce more fine-grained control. If privacy is requested for a reply, it is encrypted with the public key of the identity that published the command.[4] Additional results may be published under *any* name and the publisher can generate symmetric encryption keys, good for specific data and over specific intervals of time, to be used as symmetric AES encryption keys for the data and publish these keys with local Authentication Services, using asymmetric encryption, in a key enclave. Use of symmetric keys reduces the computational overhead on NODs and limits a subscriber's ability to decrypt measurement data to that collected when a specific key was in use. The exact approach depends on performance trade-offs (time to check identities vs time to send/receive data) and such considerations as whether security is compromised by knowing the value or the existence of a particular metric. Frequent updates of encryption keys allows the effective removal of clients.

The format and structure of full publication names (section 2.4) are used to enable fine-grained role-based security by expressing roles and permissions in exact match trust schema rules. This is exactly how the NDN trust model relates to its names [36, 41]: DNMP publication names are NDN names that make use of those naming conventions[37]. Publications must be signed with the proper identities so that the security rules, embodied in a trust schema, can be applied.

Publishing commands as Data rather than through signed Interests lets DNMP take advantage of the delivery properties of Data where all interested entities can access the published Data. In contrast, an Interest is consumed by a *single* Data response, making it unreliable for dissemination to a group of unknown size and enforcing one-to-one communications. In addition, command publications can be used to provide a log that can be audited if necessary since they contain all the forensic information of who did what to whom and when. As publications are Data, this audit trail can be created without perturbing the operation of DNMP. All publications contain a timestamp, covered by the signature, that serves triple duty: to bound the state needed to prevent replay attacks, to bound publication lifetime, and to determine one-way and round-trip response times.

Publications remain local until or unless there is an authorized remote subscriber. This is fundamentally different from several existing NDN sync protocols where Content has a long-lived utility (e.g. video, newspapers) and provision of transparent longer-term caching is automatic. CRshim communications are more akin to an ephemeral RPC request/response and generally there is no reason to preserve Replys, though individual Client applications might

[4]The NOD gets this public key to validate the command; encrypting with it means only that identity can decrypt the reply.

store a history of results. DNMP's bespoke transport uses different shims to implement the specific communications models required by specific tasks, allowing storage tasks for content that needs to be available longer than its network-layer lifetime to be separated from ephemeral communications tasks so that storage solutions can be made specific to the storage intent (section 3.2).

2.3 Client and NOD Shim Use[5]

Shim modules provide intuitive APIs and translate them to the topic-oriented view of a pub/sub Sync. Each shim has an associated trust schema used, via a run-time schema library, to handle all the publication construction, signing, validation and parsing issues (section 3.3) and implements a specific API. DNMP applications (e.g., Clients, NODs, Probes) use only this API to communicate. Authentication checks are performed by a validator module (section 3.3) called from the Sync so that any command passed "upward" to the callback by the Sync has already been checked for format.

The pseudocode templates of Figure 3 show how shims streamline NOD and Client code. Current Clients are executed from the command line, with arguments for probeType and probeArgs. NODs and Clients create one CRshim[6] per *target*; successful creation means execution can continue. NODs use three targets: those specifically addressed to them (my_ID()), **local** topics from this device, and to **all**. NODs invoke a *waitForCmd* method on each shim to start its subscription. A shim that receives a validated publication invokes the supplied callback with arguments from the publication. The callback results in a NOD probeDispatcher invoking the specified Probe, passing in probeArgs, and, placing the returned value in the Content of a reply packet derived from the command. Clients call a *doCommand* method (passing in the desired probeType, probeArgs, and callback) which subscribes to the expected reply topic, builds and publishes the command. A validated publication in the reply topic invokes the callback, passing the result from the content field as an argument. The callback processes the reply, here printing and exiting. A more complex Client could issue more commands, possibly based on reply content.

NOD Pseudocode

```
auto probeDispatch(Command c) {
   auto pType = c["probeType"];
   auto pArgs = c["probeArgs"];
   return (probeTable[pType])(pArgs);
}
string targets[] = {"nod/local","nod/all","nod/my_id"}
for (t: targets) {
   auto s = CRshim(t);
   s.waitForCmd(probeDispatch);
}
```

Shell Utility Client Pseudocode

```
void processReply(const Reply& r) {
   cout << r << endl;
   exit(0);
}
int main(int argc, char* argv[]) {
      ... parse arguments
          try {
              CRshim s(target);
              s.replyTimeout(replyWait, doFinish);
              s.doCommand(ptype, pargs, processReply);
          } catch (const std::exception& e) {
                    std::cerr << e.what() << std::endl;
          }
}
```

Figure 3: Pseudocode templates for NOD and Client

2.4 Publication Names and Signing Keys

DNMP publication names are constructed to reflect their functionality and the trust schema security model. Trust schema rules express organizational and operational policies as syntactic relationships between the keys of a publication's signing chain. These signing rules establish which identities have the authority to publish under which name prefixes. Since any name component can be matched against either a component in another key or a literal in a trust schema template, these rules allow fine-grained control over exactly what an identity can do. For verifiability, simplicity and robustness, trust schemas do only *equality* comparison between name components, those components must be at *fixed* locations in the name, and signing relationships must strictly follow the schema's signing chain specification. Command and Reply publications have a timestamp component that bounds their lifetime and all keys (including trust schemas which have the form of keys) have components which specify their validity period, allowing for fine-grained control not only of *who* can do *what* but also *when*.

Figure 4 shows the DNMP command and reply publication Name format where Name Components are grouped by function and written as <*group name*>for readability. (A group can contain multiple components or groups.) Conventionally, **bold** indicates literal content, forward slashes (/) denote component separation, and vertical lines (|) are used for separations inside a component. The reply name format uses the *domain* and *target* groups from the command that caused the probe to execute, changes the topic to **reply**, includes a *commandIdentifier (cmdID)* group that is an exact copy of the final three groups of the corresponding command**,** and appends its own three groups that give the number of publications in a reply[7], its unique identifier, and its timestamp. Specification of <*root*> is up to the local network administrator (by addition to the DNMP basic trust schema) and is used to construct the certificate for its instance of DNMP. Examples include root = /ndn/com/bigco/netops/sanjose or root = /myHouse. NOD *identity* is a function of the intrinsic host or device identifier, e.g. an id number (6) or tag (mac_laptop) or multi-part identifier (building10|floor2|gateway6) but is contained in one component. *Origin* gives the name of the machine where the command was published and is intended for forensic purposes. *All* of the name components, with exception of probeType and probeArgs for a command and the Content for a reply, can be supplied by CRshim using the run-time trust schema to fill in publication names. Publication names capture the trust

[5]This section describes the approach of the POC; the in-progress version is additionally streamlined by with c++ Promises and a more capable shim.
[6]Shim modules for future topics will use existing modules as a template.
[7]Where publication must fit in a Data packet.

<domain><target>**command**<srcID><directive><timestamp>
<domain><target>**reply**<cmdID><dCnt><rSrcID><timestamp>
domain = <root>/**dnmp**
root (or *networkID*) identifies the particular network
target = **nod**/<nodSpec> where directive is to be performed
nodSpec used to specify NOD(s) (e.g., **all**, **local**, <*identity*>)
command or **reply** exact value denoting Topic
srcID = <*roleType*>/<*ID*>/<*origin*> identifies publisher
roleType is operator, user, or guest
ID role-specific identifier
origin identifies the publication origin network-attached device
directive = <*commandType*>/<*probeType*>/<*probeArgs*>
commandType: only currently defined type is **probe**
probeType: descriptive name of the particular probe
probeArgs: single component, makes command more specific
timestamp = <*UTC microsecond timestamp*> (creation time)
cmdID = <*srcID*>/<*directive*>/<*timestamp*>
exact copy of **command**'s last three groups
dCnt = **<0>** or <kln>
exact value of **0** is used if only this Data packet in the reply
<kln> indicates the kth Data packet out of a total of n
rSrcID = **nod**/<*nodID*>, replying entity
nod*ID* identifier uniquely derived from host and/or NFD

Figure 4: command and reply names

and permissions used by CRshim communications and *only* need to be used in the shim (see 3.2 and 3.3).[8]

Figure 5 shows the signing rules for command and reply publications. Each publication type has Components that the trust schema maps directly to specific types of keys that must be signed in the order shown. Using "cpub" and "rpub" as shorthand for the publications, "<=" should be read as "is signed by." Recall (2.2) that commands use NOD specifiers, which can be identities or literals whose use is governed by *role*, while replies must have NOD identities, which are certified through the device configuration path. Each name is at a fixed location in the signing chain. Rules specified by DNMP's trust schema are applied to the publications it sends and receives to the Sync rather than to the packets on the wire.

2.5 Example: a command reaches many NODs

Pub/sub communications affords efficiencies in network measurement that can be illustrated by an example. Consider a blackhole utility that lets an operator query all the network's devices to find each NFD's *blackholes* (i.e., NFDs with no route to a prefix). The "straightforward" approach of enumerating and querying each of the NODs individually could be very inefficient. In DNMP, a blackhole query can be issued to an entire network at once by publishing a single item to a command topic (one-to-many) and requesting a response from NODs with matching blackholes, i.e. no route. Responses from these NODs are published to the associated reply topic and retrieved by subscription to that topic (many-to-one).

In figure 6 the central gray arrow shows the progress of the measurement and the namespace with time. Short arrows are used to indicate clients and NODs publishing data in the namespace or receiving subscription callbacks due to new data in the namespace.

[8]Names should use fewer components when prototyping is completed.

command pub definition and signing chain		
cpub	=	<domain>/**nod**/<nodSpec>/**command**/<roleType>/<ID>/<origin>/**probe**/<pType>/<pArgs>/<timestamp>
roleCert	=	<domain>/<roleType>/<ID>/<_key>
dnmpCert	=	<domain>/<_key>
domain	=	<root>/**dnmp**
cpub <= roleCert <= dnmpCert <= netCert		

reply pub definition and signing chain		
rpub	=	<cpub **command** => **reply**>/<dCnt>/<rSrcID>/<rtimestamp>
nodCert	=	<domain>/**nod**/<nodID>/<_key>
devCert	=	<root>/**device**/<devID>/<_key>
configCert	=	<root>/**config**/<configID>/<_key>
rpub <= nodCert <= deviceCert <= configCert <= netCert		

Figure 5: Signing chain for command/reply packets

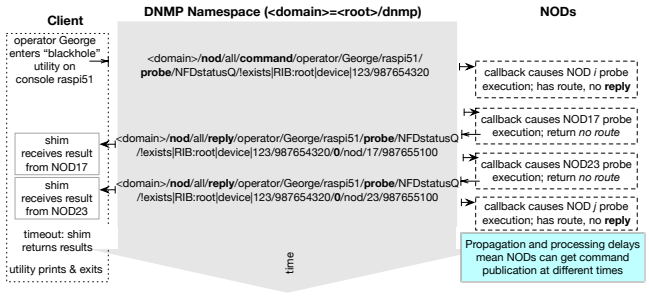

Figure 6: Multiple NODs respond to blackhole utility

"|" represents the namespace and the arrow direction relative to the "|" should be interpreted as the direction with respect to the namespace. The NODs have initialized by subscribing to nod/local and nod/all command topics and are awaiting new publications. *Operator* George on console raspi51 runs a blackhole client that publishes a command to all NODs asking them to issue a local NFD status query for routes to device 123 (RIB:route|device|123) and respond only if there are *none*. The shim publishes the command and subscribes to the corresponding reply topic where NODs will publish any results. The publication of this command causes a subscription callback to NODs subscribed to nod/all. NOD17 broadcasts its reply saying it checked at time 987655100 and didn't have a route. The client's pub/sub sync adds NOD17's reply to its subscription digest and solicits (via Interests containing its digest) any others prompting NOD23 to broadcast its reply that it didn't find a route when it checked at 987655200. The other NODs in the network have routes to that location and do not respond.[9] When the Client's topic digest for the reply topic contains both of these, further Interests will time out.

3 ENABLERS: SYNCPS, SHIMS, AND SCHEMA

This section covers the pub/sub Sync, describes the functions of the shim, and DNMP's use of VerSec. Though NDN literature and

[9]Variations could report only if a route is found or all NODs could report route status.

codebase provide much work to leverage regarding security and key distribution[27, 28, 41]), NDN lacks an approach to securing an application's trust schema or making it usable at run-time. DNMP benefits from being co-developed with VerSec and the bespoke transport model that makes use of it. The rest of this section describes features that enable DNMP's secure communications and simplified API: *syncps* in section 3.1, the role of the shim in section 3.2, and role of VerSec's schemer in section 3.3. The specifics of implementation are for the proof-of-concept implementation, though some in-progress work is discussed.

3.1 Pub/sub sync

Early in our work, it appeared that PSync [40] might work for DNMP but its communications model is one of multiple consumers and a producer where producers can replicate content in a *separate* communications context to provide redundancy. PSync proved a poor fit to a model where applications both publish and subscribe through hierarchically named topics and individual publications are distinguished by their names and creation timestamps. DNMP publications are, generally, a unit of ephemeral communication.

syncps is a lightweight publish-subscribe Sync, partially motivated by MQTT, that enables topic-based communications. The MQTT pub/sub protocol has over a decade of development and widespread use, including the open source HomeAssistant [9] and Samsung's IoT ARTIK Cloud [30]. Its communications model presents simple calls and message types with three delivery quality-of-service types: at most once, at least once, and exactly once (for more on general pub/sub delivery QoS see [10]). Additional assured delivery semantics, caching and long-term storage, and other custom functionality are easily layered on top of this model. Less appealingly, MQTT is implemented on TCP and uses TLS and userid/password for security. Compared to MQTT, *syncps* shows the value of using NDN as it removes the need for a centralized broker and enhances security. *sync*ps implements a simple delivery and any-to-any communications over NDN, providing API methods of *subscribeTo* (a Topic) that specifies a callback and *publish* that submits an item for publication in a Topic.

A *syncps* instance announces its set of currently known publications by sending an Interest containing a Difference Digest [15, 21]. Difference Digests solve the multi-party set-reconciliation problem without prior context and with communication proportional to the size of the subset difference. Receipt of these Interests does three jobs simultaneously: (1) announces new publications, (2) notifies of publications that peer(s) are missing and (3) acknowledges publication receipt. The first results in sending an Interest to get the new publications. The second results in satisfying the Interest with a Data containing all the missing publications that will fit. The third results in a progress notification sent up the protocol stack so anything waiting for delivery can proceed. *syncps* makes use of NDN's optimizations for broadcast media where an NFD uses Interests it hears to suppress its own and gets any Data for which it has an outstanding Interest. This means that one-to-many publications (like the command sent to **nod/all** in the blackhole example of sec.2.5) require one Interest and one Data to be sent, independent of the number of entities in the target set (the theoretical minimum possible for reliable delivery). Similarly, replies require *at most*[10] one Interest and Data sent per responder. The command-to-reply latency for the first response is twice the network propagation time (meeting the theoretical minimum bound) and each additional reply requires at most two additional propagation times.

Pub/Sub	publish(Publication&& pub); subscribeTo(Name&& topic, UpdateCb cb);
Upcall Config	setIsExpiredCb(IsExpiredCb cb); setFilterPubsCb(FilterPubsCb cb);
Security Config	setPubSecurity(PubSecurity&& ps); setPacketSecurity(PacketSecurity&& ps);
Pub/Sub methods access the communication functions of syncps. Upcall config lets the shim control pub lifetime (IsExpiredCb) and delivery priority (FilterPubsCb) policies of syncps mechanisms. Security config lets the shim supply the signing info and validator for pubs (PubSecurity) and NDN packets (PacketSecurity).	

Figure 7: syncps public methods (API)

The Interest's digest size can be controlled by publication lifetime, dynamically constructing the digest to maximize communication progress[24, 25] and, if necessary for a large network, dynamically adapting topic specificity, e.g., a client is only interested in replies to its most recent command(s). In our small-building-scale prototype implementation, lifetime bounding[11] keeps Interests small so the dynamic adaptations have yet to be implemented.

As described in sec.2.2, publications are NDN Data objects secured by a trust schema which are bundled into *another* Data packet when syncps satisfies an Interest who's digest requires them. This on-the-wire Data packet is secured using a standard NDN validator. Since the trust schema protects against application level threats, syncps defaults to a SHA validator to protect against data corruption but relies on the local network's security to protect against DoS attacks and information leakage. The syncps API allows any validator to be set, so shims can configure the NDN communication layer appropriately for site-specific threats.

Syncps implementation details are described in its GPL'd code and documentation available on github and are not covered here. What's relevant to DNMP is the API presented to the shim layer (Figure 7) which supports the policy/mechanism separation needed to create bespoke transports. Shims implement application-focused communication abstractions, described in the next section, using the mechanisms and policy hooks supplied by syncps and schemer. For example, CRshim uses the syncps *publish* and *subscribeTo* mechanisms to implement the basic command/reply exchange, the *isExpired* upcall from syncps to implement its publication lifetime policy and the *filterPubs* upcall to implement its communication progress policy.

[10] As mentioned earlier, multiple publications can be aggregated into one Data which typically results in a multiplicative reduction in both packet exchanges and average response latency. See [21, sec.5] for background.

[11] Command and reply publication lifetime is already kept short to minimize replay protection state burden.

3.2 Shims customize transport

Distinct DNMP topics require distinct (though similar) shims. CRshim is an ephemeral, RPC-like invocation of distributed measurement (by Probes) to retrieve a result. It has at-most-once delivery QoS and must be protected against replay. CRshim achieves this by assuming that NOD clocks are loosely synchronized,[12] and interpreting a publication's timestamp field as the start of its life. Publications are "active" (meaning *syncps* will give them to a peer that doesn't have them and that peer may deliver them to a subscriber that doesn't have them) for 5 sec[13] after their creation, then become inactive. Inactive publications are never given to a peer or subscriber but are kept in the syncps publication table for another 10 sec to prevent replays and churn due to clock skew. *syncps* has to manage active and inactive publications to know when and if to communicate them, but it knows nothing about the format or semantics of publications. Thus publication lifetime is managed by the shim indicating active/inactive via an upcall from and downcall to syncps. In consequence, almost all the code is in *syncps* but the application-visible interface and semantics are determined by the shim.

Other DNMP communications models are implemented with different shims, in particular, storage entities have their own shim and API.[14] Returning to Figure 1, a *Logger* provides storage for asynchronously generated measurements, i.e., those results not directly returned via reply, e.g., measurements generated by a time-based probe, results that are extremely large or inherently of archival value, or for content that may need to be available longer than the life of its publisher. Like NODs, Logger identity comes from their host, thus authorized Loggers can be constrained to devices with resources that can support their functionality. A Logger can be physically located anywhere, including on the same device as a NOD, but the device must have the required capabilities (e.g., "always on", stable storage, a certain processing power). A Logger can have sophisticated rules about when and what to archive, what to overwrite, etc. but these are local policy decisions implemented in its code and not part of their communications model. Loggers subscribe to **archive** topics, where NOD probes publish measurements, and subscribe to a different set of targets than NODs (e.g., "log/all" is not sensible but "log/any" could be useful). To offload work and resource use from NODs, **archive** publications are *only* subscribed by Loggers (i.e, *not* by Clients). If the archive publication has been appropriately signed and is otherwise well-formed, a Logger publishes in a corresponding **archived** topic to indicate the content of of the publication has been archived. NODs subscribe to the archived publications and can use subscription notifications to implement rules such as "at least one", "at least n", or "at least one non-local" successful Logger subscription. An **archive**/**archived** shim is similar to CRshim but its delivery QoS is at-least-N, replay isn't an issue, and it uses **query/response** topics.[15] Since its goal is get items safely archived despite transient communication or device outages, publications are considered active for 5 minutes after their creation time (they still die 10 sec after becoming inactive). Some or all of the Loggers' archived measurements can be stored in a database in the form of column stores or LSM (log-structured merge) trees, which could make the data Client-accessible via SQL-like queries fronting standard database packages like Apache Cassandra or InfluxDB. Basic data queries using temporal and value based range filters are stateless and have set-valued results (as opposed to results tied to the state embodied in some query agent). We are currently evaluating the trade-offs of different approaches.

An Audit Client can be used for operational audits. It subscribes to all (or a critical subset of) command (and possibly all reply) topics and thus can provide a secure, tamper-proof audit trail of *all* DNMP actions and actors. Housekeeping rules on what to keep and for how long are part of the audit client, not its communications model. Auditors use a **snoop** shim which has very different lifetime requirements. It never publishes anything itself; its purpose is to deliver network peer publications to a *local* Audit Client subscriber. Since it may not be configured with the trust schemas for those topics (which means it may not be able to validate incoming publications) and it doesn't know their lifetime rules, it should never send a publication it has received back out to peers. Thus publications are marked as inactive as soon as they're received (to keep them from being sent to peer(s)) but kept alive for 10 minutes (to filter out duplicates of long-lived publications like archive requests).

3.3 Versatile security simplifies shim

A simple API often results in a complex implementation. This is not the case with DNMP, in part due to a simple mapping of DNMP's pub/sub communication model onto the *syncps* API but mostly because of an unexpected synergy between DNMP's networking and security requirements that resulted in a remarkable simplification of the implementation.[16] This is best explained with examples from the CRshim. Consider the implementation of the Client-visible *doCommand()* which must:

(1) Locate a signing key consistent with the client-supplied parameters and acceptable to the command topic trust schema.
(2) Fill in the missing topic information (a valid command topic name has eight components of which the client supplies three).
(3) Generate the valid reply name for the command and *subscribeTo* the reply(s).
(4) Publish the command, collect replies as they arrive and return a result to the Client when there are a sufficient number or the command's lifetime expires.

Item (4) requires only simple calls to the lower level syncps framework but items (1)-(3) touch the core of DNMP's Authentication, Authorization and Access Control (AAA) model. The targets and/or Probes that a client is allowed to put in commands are controlled by the DNMP role signing key(s) held by the user that invoked the client. The NDN library's trust schema validator project [29] can be used at a receiver to verify that the entire signing key hierarchy is valid, but the sender needs this to work in reverse, going

[12] There are other, more complex, ways to accomplish this without assuming synchronized clocks if, for example, the replay protection needs to function across reboots. As synchronized clocks are generally useful for correlating network measurements, DNMP devices already NTP clock sync, so 1 sec is a reasonable skew bound.
[13] Our initial timing for a home/small office style wireless network. Experience may lead to changes.
[14] DNMP storage solution implementation is an in-progress design.
[15] The archive/archived shim adds NOD measurements to the database while query/response lets Clients access them.
[16] An apparently unique instance where security requirements made an implementation simpler and more robust rather than more complex and brittle.

from the keys held by the user to the command formats allowed by those keys. Unfortunately, the library validator *can't* be used that way. Since trust schemas work by defining equivalence relations between name components and key names in a signing chain and equivalence relations are *bidirectional* mappings, it's always possible to create a *single* validation structure that can be used to both build and validate commands.[17] This is the approach of VerSec used by a DNMP-specific *schemer* library to make problems (1)-(3) above trivial:

(1) Given the specified command components (target, probeType and probeArgs), call the validator routine that returns the key schemas for all keys that a) directly sign commands and b) are compatible with the user's roleType being set to the user's values.
(2) If there are no such key schemas, throw an error. Otherwise call the NDN library routines that search the user's TPM for matching keys. If none, throw an error; if multiples, choose the least privileged.
(3) Use this signing key to select the particular command schema it signs and fill in the user's values plus schema-specified literal, computed and command-to-key correspondence values. If any field of the prototype command isn't filled in, throw a missing parameters error. Otherwise the command is complete and ready to be signed and sent.

Using VerSec's method, trust specifications are written in a simple, declarative language matching the usual way NDN applications describe trust rules (e.g., see [33, 34, 36, 41]). VerSec's trust schema compiler converts this specification to a compact, binary form that can be signed and distributed as an NDN key. This signed schema is *the* authoritative form of publications, used at run-time to validate, parse, and build them, which makes publication security and syntax transparent to NODs and Clients. This removes a coding burden from DNMP implementors. More importantly, since all the site specific policy information is in trust schemas which are loaded and validated at runtime, clients and NOD *binaries* become portable and oblivious to most network and configuration changes.

Run-time use of the trust schema is illustrated in Figure 8. The gray arrows show signing relationships. For example, the Network Key is the trust anchor which (indirectly) signs all role and device keys as well as the trust schema. The black arrows indicate the verification relationships between name components of keys, publications, and schema literals (bold in Figures 4 and 5). The run-time schema records all signing and verification relationships in a compact graph structure that can be traversed in any direction from any point.[18]

4 IMPLEMENTATION STATUS

DNMP's c++ POC includes a few probes and communicates via *syncps* and a CRshim. The compiled trust schema is used by the shim to build, validate, and parse publications. POC Clients are

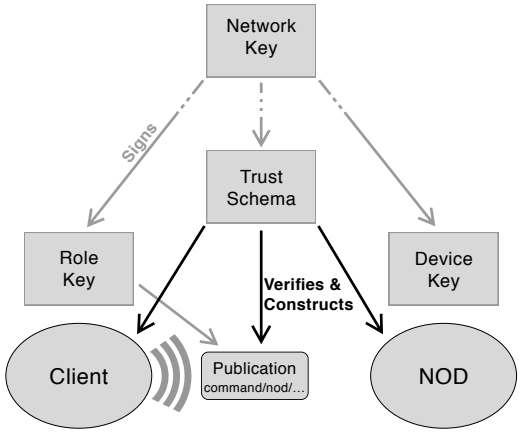

Figure 8: Trust Schema at run-time

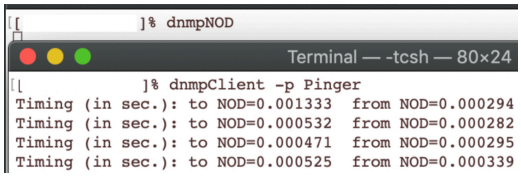

Figure 9: Continuous pinger

invoked via the command line with flags for the *probeType* and any *probeArgs*. Six Probes were implemented: four provide simple access to metrics available via the NFD Management Protocol, one is a continuous ping of a target, and one is a version of a black hole client. All print a final line giving outbound and inbound times. Probe functions are added to the NOD by registering them under their *probeType* name. Probes return a string (in future, an NDN TLV) that is placed in the Content field of the reply.

Initial performance numbers were on order of 600uS NOD-to-client and 5-500mS client-to-NOD. Commands and replies should have similar, sub-millisecond RTT (see sec.3.1).[19] On investigation we discovered that NFD was rejecting the NOD's syncps Interest with a NACK rather than putting it in the PIT so performance included the periodic Interest re-expression time (100s of mS), evident in both the NFD General Status showing thousands of NACKs and in the packet level debug log. This behavior is a clear violation of the architectural intent expressed in [3, 38]. Subsequently, we created an NFD code patch to allow proper broadcast behavior. The improvements can be seen in Figure 9 where the first reply publication includes the registration overhead (a device dependent 1-3ms), but subsequent pings show the same order of magnitude both NOD-to-Client and Client-to-NOD. Optimization of registration could reduce this.

Though not yet optimized, the POC is relatively efficient in lines of code. DNMP-specific code is in four files: `client.cpp` (98 lines of code), `nod.cpp` (68 lines), `probes.hpp` (probe functions of 3 to 36 lines), and CRshim (78 lines). `syncps` is 487 lines of code. The POC

[17] Think of the validator data structure like a set of building plans that are used by a) the city planning department to guarantee that what's built will conform to codes, b) the building contractor to construct the building, and c) the city building inspectors to verify that what was built matches the approved plans. A good validator framework will support all these use cases and more.
[18] As opposed to the trust chain linkage derived from objects' key locators which enumerates only one branch of the trust tree from leaf-to-root.
[19] Local performance is bounded by signing+verification time which should be 100-200uS for our test machines based on Supercop ECDSA/ED25519 benchmark results.

trust schema is expressed in 25 lines (excluding comments) over three files: a top level file that contains the local network root cert name, a site-independent specification of the POC's three role AAA, and the DNMP core definitions (e.g., name components). These are compiled by the schema compiler into the binary trust schema which is currently manually added to the shim. The shim, *syncps, schemer* and run-time validator are all modern (C++17) header-only includes, not libraries, which reduces the security perimeter and allows all the code to be inlined and optimized, potentially making it suitable for embedded environments. All code is GPL'd and on Pollere's github account[20].

5 STATUS AND FUTURE WORK

DNMP provides a framework where new features can be added without the need to work out low-level network protocol and security details. DNMP storage implementations (sec.3.2) are in-progress. The existing code is compact, relatively easy to understand and written with a view toward serving as templates for future development. To that end, it is GPL licensed and publicly available.

The NFD Management Protocol was useful for creating proof-of-concept probes but NFD needs better instrumentation. In-progress work includes a Probe that keeps a t-Digest[13] measurement of the *time to satisfy Interests* (TTS). This required NFD code changes as it was otherwise impossible to access the values. NFD's Measurement Table is currently only used by Strategies and its Management Protocol interface was never implemented. An NDN community effort to agree on forwarder instrumentation would be useful. DNMP can provide the fine-grained role-based secure access to this instrumentation.

NDN is rich in security features but they lack ease of use. The versatile security approach employed by DNMP is a significant step toward making NDN trust specifications more usable, straightforward and secure.

The current NFD implementation has been extensively tested over point-to-point links but is both untested and unfriendly to broadcast networks. We have encountered and fixed several significant issues (including one that made it almost impossible to broadcast Interests) and made patches available.

Our prototype goal is DNMP in a multi-device, multi-hop, single administrative domain network with a locally controlled and administered trust schema. Thus everything has a signing chain derived from the same root, including the signing chains that give every device and individual their identities, configuration and roles giving the system a built-in trust root with no key distribution issues and no external dependencies.[21] A configuration-free approach to making DNMP work across a multiple hop wireless networks is being explored. DNMP hides low-level addressing/naming details at the application level via its *target* abstraction. Our current multi-network strawman significantly extends the expressability of *target*. A Client should be able to issue commands to any device or collection of devices, such as "all the devices in this room", "all the devices in the bedrooms on the second floor", or "all the devices in the house", without knowing the network's topological organization. Since application-visible targets have no relationship to a NOD's network identity or topological location, a mapping table is already maintained by the *schemer*. Our present proof-of-concept statically maps "local" to NDN prefix "/localhost" and everything else to NDN prefix "/localnet" which is statically routed on every node to all UDP muticast interfaces. This makes any and all targets reachable at the cost of sitewide multicast of all DNMP packets.[22] For a multi-network prototype, all nodes connected to two or more networks create and disseminate per-network broadcast prefixes to the per-network-scoped topic (/localhop) which allows NODs to learn the name(s) of their attached networks and publish the bindings of their application level-name(s) to a 'directory' topic published in "/localnet", making it available to all NODs. The topic's trust schema will guarantee provenance of the bindings and appropriate setting of publication lifetimes will guarantee that the information stays current but generates minimal traffic. Each NOD will essentially provide a cache of this directory that local clients can access via (to be designed) query topics using a **nod/local** target.

6 CONCLUSION

In the course of implementing an NDN measurement protocol we learned a few lessons, some of which are old lessons. Lesson 1 was "go back to basics". By returning to the original simple network protocol of NDN and following its early guidance to develop application-specific transport models, we followed a modular design path, separating the application processes from bespoke transport modules on top of the NDN protocol. Lesson 2 was to "make security a first class citizen" which led to the extremely useful co-developed VerSec project. Lesson 3 was to use a "dogfood" approach and leverage the interaction of DNMP, the bespoke transport modules, and VerSec to make each piece better. Working this way gave lots of insights and was quite enjoyable.

ACKNOWLEDGMENTS

The author would like to thank Van Jacobson who contributed materially to DNMP discussions, versatile security, and editing suggestions; also Abdella Battou and Lotfi Benmohamed of NIST for suggesting and sponsoring DNMP.

REFERENCES

[1] [n.d.]. Named Data Networking. http://named-data.net/
[2] [n.d.]. Named Data Networking Code Base. https://github.com/named-data/
[3] [n.d.]. Named Data Networking: Motivation and Details. http://named-data.net/project/archoverview
[4] 2019. MQ Telemetry Transport. http://mqtt.org/
[5] Hila Ben Abraham, Jyoti Parwatikar, John DeHart, Adam Drescher, and Patrick Crowley. 2018. Decoupling Information and Connectivity via Information-Centric Transport. In *Proceedings of 2018 ACM Conference on Information-Centric Networking*. ACM.
[6] Vaibhav Bajpai and Jürgen Schönwälder. 2015. A Survey on Internet Performance Measurement Platforms and Related Standardization Efforts. *IEEE Communications Surveys and Tutorials* 17, 3 (2015), 1313–1341.
[7] J. Burke. 2017. Browsing an Augmented Reality with Named Data Networking. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. 1–9. https://doi.org/10.1109/ICCCN.2017.8038469

[20] At https://github.com/pollere/DNMP.
[21] Sharing of information between domains with different administrative owners is not considered at this time.
[22] This cost is zero for single network sites but may be significant for large networks with large amounts of DNMP traffic.

[8] Jiachen Chen, Mayutan Arumaithurai, Lei Jiao, Xiaoming Fu, and KK Ramakrishnan. 2011. Copss: An efficient content oriented publish/subscribe system. In *Proceedings of the 2011 ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems*. IEEE Computer Society, 99–110.

[9] Home Assistant Community. 2018. Smarter SmartThings with MQTT and Home Assistant. https://community.home-assistant.io/t/smarter-smartthings-with-mqtt-and-home-assistant/42493

[10] Angelo Corsaro, Leonardo Querzoni, S Scipioni, Sara Tucci-Piergiovanni, and Antonino Virgillito. 2006. *Quality of Service in Publish/Subscribe Middleware*. Vol. 8. https://www.researchgate.net/publication/237100885_Quality_of_Service_in_PublishSubscribe_Middleware

[11] DARPA. 2018. Secure Handhelds on Assured Resilient networks at the tactical Edge. https://www.darpa.mil/program/secure-handhelds-on-assured-resilient-networks-at-the-tactical-edge

[12] Steve Deering. 1998. Watching the waist of the protocol hour-glass. In *Keynote Address at 6th IEEE Int. Conf. on Network Protocols*.

[13] Ted Dunning and Otmar Ertl. 2019. Computing Extremely Accurate Quantiles Using t-Digests. *CoRR* abs/1902.04023 (2019). arXiv:1902.04023

[14] Philip Eardley, Al Morton, Marcelo Bagnulo, Trevor Burbridge, Paul Aitken, and Aamer Akhter. 2015. A Framework for Large-Scale Measurement of Broadband Performance (LMAP). *RFC* 7594 (2015), 1–55.

[15] David Eppstein, Michael T. Goodrich, Frank Uyeda, and George Varghese. 2011. What's the difference?: efficient set reconciliation without prior context. In *Proceedings of the ACM SIGCOMM 2011 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Toronto, ON, Canada, August 15-19, 2011*. 218–229.

[16] Michal Krol et al. 2018. RICE: Remote Method Invocation in ICN. In *Proceedings of 2018 ACM Conference on Information-Centric Networking*. ACM.

[17] C. Gundogan, P. Kietzmann, M. Lenders, H. Petersen, T. Schmidt, and M. Wahlisch. [n.d.]. NDN, CoAP, and MQTT: A Comparative Measurement Study in the IoT. *Proceedings of 2018 ACM Conference on Information-Centric Networking* ([n. d.]).

[18] Small Business Innovation and Research. 2018. Field Gateway Distributed Transaction Ledger for Utility-Scale Solar. https://www.sbir.gov/sbirsearch/detail/1523933

[19] Van Jacobson. 2019. Watching NDN's Waist: How Simplicity Creates Innovation and Opportunity. keynote talk at NSF/Intel ICN-WEN Anuual Workshop, Santa Clara, CA. http://pollere.net/talks.html

[20] H. Khelifi, S. Luo, B. Nour, H. Moungla, Y. Faheem, R. Hussain, and A. Ksentini. 2019. Named Data Networking in Vehicular Ad hoc Networks: State-of-the-Art and Challenges. *IEEE Communications Surveys Tutorials* (2019), 1–1. https://doi.org/10.1109/COMST.2019.2894816

[21] Michael Mitzenmacher and Rasmus Pagh. 2018. Simple multi-party set reconciliation. *Distributed Computing* 31, 6 (2018), 441–453.

[22] Ilya Moiseenko, Lijing Wang, and Lixia Zhang. 2015. Consumer/producer communication with application level framing in named data networking. In *Proceedings of the 2nd ACM Conference on Information-Centric Networking*. ACM, 99–108.

[23] Department of Energy. [n.d.]. Project Profile: Operant Solar (T2M3). https://www.energy.gov/eere/solar/project-profile-operant-solar-t2m3

[24] A. Pinar Ozisik, Gavin Andresen, George Bissias, Amir Houmansadr, and Brian Neil Levine. 2017. Graphene: A New Protocol for Block Propagation Using Set Reconciliation. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2017 International Workshops, DPM 2017 and CBT 2017, Oslo, Norway, September 14-15, 2017, Proceedings (Lecture Notes in Computer Science)*, Joaquín García-Alfaro, Guillermo Navarro-Arribas, Hannes Hartenstein, and Jordi Herrera-Joancomartí (Eds.), Vol. 10436. Springer, 420–428. https://doi.org/10.1007/978-3-319-67816-0_24

[25] A. Pinar Ozisik, Gavin Andresen, Brian Neil Levine, Darren Tapp, George Bissias, and Sunny Katkuri. 2019. Graphene: efficient interactive set reconciliation applied to blockchain propagation. In *Proceedings of the ACM Special Interest Group on Data Communication, SIGCOMM 2019, Beijing, China, August 19-23, 2019*, Jianping Wu and Wendy Hall (Eds.). ACM, 303–317. https://doi.org/10.1145/3341302.3342082

[26] Davide Pesavento, Omar Ilias El Mimouni, Eric Newberry, Lotfi Benmohamed, and Abdella Battou. 2017. A network measurement framework for named data networks. In *Proceedings of the 4th ACM Conference on Information-Centric Networking, ICN 2017, Berlin, Germany, September 26-28, 2017*. 200–201.

[27] Lei Pi and Lan Wang. 2018. Secure bootstrapping and access control in NDN-based smart home systems. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops, INFOCOM Workshops 2018, Honolulu, HI, USA, April 15-19, 2018*. 1–2.

[28] Lei Pi and Lan Wang. 2018. Secure bootstrapping and access control in NDN-based smart home systems. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops, INFOCOM Workshops 2018, Honolulu, HI, USA, April 15-19, 2018*. IEEE, 1–2.

[29] NDN project. [n.d.]. Validation Configuration File Format. https://named-data.net/doc/ndn-cxx/current/tutorials/security-validator-config.html

[30] Samsung. 2016. Samsung Announces Commercially Available IoT Cloud Platform to Deliver Interoperability Between Devices and Applications. https://news.samsung.com/us/samsung-announces-commercially-available-iot-cloud-platform\protect\discretionary{\char\hyphenchar\font}{}{}deliver-interoperability-devices-applications/

[31] 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 *2016 IEEE first international conference on internet-of-things design and implementation (IoTDI)*. IEEE, 117–128.

[32] 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.

[33] Wentao Shang, Zhehao Wang, Alexander Afanasyev, Jeff Burke, and Lixia Zhang. 2017. Breaking out of the Cloud: Local Trust Management and Rendezvous in Named Data Networking of Things. In *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation, IoTDI 2017, Pittsburgh, PA, USA, April 18-21, 2017*. 3–13.

[34] Lan Wang, Vince Lehman, A. K. M. Mahmudul Hoque, Beichuan Zhang, Yingdi Yu, and Lixia Zhang. 2018. A Secure Link State Routing Protocol for NDN. *IEEE Access* 6 (2018), 10470–10482.

[35] et. al. Wentao Shang. 2017. A Survey of Distributed Dataset Synchronization in Named Data Networking. *Named Data Networking Technical Reports* https://named-data.net/publications/techreports/ (May 2017).

[36] Yingdi Yu, Alexander Afanasyev, David D. Clark, kc claffy, Van Jacobson, and Lixia Zhang. 2015. Schematizing Trust in Named Data Networking. In *Proceedings of the 2nd International Conference on Information-Centric Networking, ICN '15, San Francisco, California, USA, September 30 - October 2, 2015*. 177–186.

[37] Yingdi Yu, A Afanasyev, Z Zhu, and L Zhang. 2014. Ndn technical memo: Naming conventions. *NDN, NDN Memo, Technical Report NDN-0023* (2014).

[38] Haitao Zhang, Yanbiao Li, Zhiyi Zhang, Alexander Afanasyev, and Lixia Zhang. 2018. NDN host model. *ACM SIGCOMM Computer Communication Review* 48, 3 (2018), 35–41.

[39] Jianqing Zhang, Qinghua Li, and Eve M Schooler. 2012. iHEMS: An information-centric approach to secure home energy management. In *2012 IEEE Third International Conference on Smart Grid Communications (SmartGridComm)*. IEEE, 217–222.

[40] Minsheng Zhang, Vince Lehman, and Lan Wang. 2017. Scalable name-based data synchronization for named data networking. In *2017 IEEE Conference on Computer Communications, INFOCOM 2017, Atlanta, GA, USA, May 1-4, 2017*. 1–9.

[41] 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.