Securing Instrumented Environments over Content-Centric Networking: the Case of Lighting Control and NDN

Jeff Burke
University of California, Los Angeles
jburke@remap.ucla.edu

Paolo Gasti
New York Institute of Technology
pgasti@nyit.edu

Naveen Nathan, Gene Tsudik
University of California, Irvine
{nnathan,gts}@uci.edu

***Abstract*—Instrumented environments, such as modern building automation systems (BAS), are becoming commonplace and are increasingly interconnected with (and sometimes by) enterprise networks and the Internet. Regardless of the underlying communication platform, secure control of devices in such environments is a challenging task. The current trend is to move from proprietary communication media and protocols to IP over Ethernet. While the move towards IP represents progress, new and different Internet architectures might be better-suited for instrumented environments.**

In this paper, we consider security of instrumented environments in the context of Content-Centric Networking (CCN). In particular, we focus on building automation over Named-Data Networking (NDN), a prominent instance of CCN. After identifying security requirements in a specific BAS sub-domain (lighting control), we construct a concrete NDN-based security architecture, analyze its properties and report on preliminary implementation and experimental results. We believe that this work represents a useful exercise in assessing the utility of NDN in securing a communication paradigm well outside of its claimed *forte* of content distribution. At the same time, we provide a viable (secure and efficient) communication platform for a class of instrumented environments exemplified by lighting control.

I. Introduction

The Internet has proven to be a tremendous global success. Billions of people worldwide use it to perform a wide range of everyday tasks. It hosts a large number of information-intensive services and interconnects millions of wired, wireless, fixed and mobile computing devices.

Core ideas of today's Internet were developed in the 1970-s, when telephony – i.e., a point-to-point conversation between two entities – was the only successful example of effective global-scale communication technology. The world has changed dramatically since then, and the Internet now has to accommodate new services and applications as well as different usage models. To keep pace with changes and move the Internet into the future, several research efforts to design new Internet architectures have been initiated in recent years.

Named-Data Networking (NDN) [18] represents one of these efforts. It exemplifies Content-Centric Networking (CCN) approach [11], [15], [17]. NDN explicitly names content instead of physical locations (i.e., hosts or network interfaces) and thus treats content as a first-class entity. NDN also stipulates that each piece of named content must be digitally signed by its producer. This allows decoupling of trust in content from trust in the entity that might store and/or disseminate that content.

NDN's long-term goal is to replace TCP/IP. In order to succeed, NDN must prove that it can be used to efficiently implement all kinds of communication commonly performed over IP today and envisaged for the near future. NDN's *forte* is clearly content distribution [15]. Recent and on-going research has shown that NDN fares well with respect to real-time [14] and anonymous communication [4]. NDN also offers some advantages over IP for less content-centric communication paradigms, such as cyber-physical systems [3] and group communication [22]. In particular, better security and explicit (as well as more meaningful) naming and discovery capabilities motivate deployment of such applications over NDN.

This paper is focused on securing lighting control systems running over NDN. Lighting control systems are a special case of Building Automation Systems (BAS). The latter provide a hardware and software platform for control, monitoring and management of: heating, ventilation and air conditioning (HVAC), lighting, water, physical access control and other building components. Prominent current trends in BAS include: (1) increasing use of IP and Ethernet for industrial control, (2) convergence of previously separate networks for IT and automation, enabled by this new common infrastructure, and (3) increasing popularity of cyber-physical systems that take advantage of internetworking of physical and digital elements to bring about new types of applications.

At the same time, growing emphasis on energy management and the "smart grid" suggest that physical network segregation is not a viable long-term approach for securing instrumented environments. Besides lower complexity and greater interoperability, deployment of BAS over public networks has certain advantages. First, there is no need to design, deploy and manage a separate network infrastructure, since BAS can benefit from high-speed, low-latency, fault-tolerant networks already deployed for general-purpose communication needs. Second, BAS can be physically distributed: devices might span buildings and sites with applications accessing them from multiple locations.

While some types of BAS might tolerate variable delays up to a few seconds for actuating or sensing, latency requirements for lighting control are stricter and represent an overlap with industrial and process control. To provide a sense of "real-time" interaction, architectural lighting might require execution of commands within a few hundred milliseconds from pressing a switch. By designing and implementing a secure lighting framework suitable for such low-latency systems coupled with a meaningful namespace, we target a hybrid design space that corresponds to the so-called "thin waist" for highly heterogenous BAS of the future.

Our goal is to use NDN to name and address all components of the system. Component names should relate to their identity or function. This is in contrast to an addressing *hodgepodge* that spans layers and systems (e.g., VLAN tag, IP gateway address, protocol port #, fixture address), as in current implementations. Furthermore, NDN naming can be used to reflect access restrictions, rather than require a separate policy language. The main motivation is that a namespace is consistently accessible within any NDN-compliant device or process. This obviates the need for application-specific access control protocols.

Organization. We proceed with NDN overview in Section II, followed by the description of a base-line lighting control protocol in Section III, which also introduces our framework. Implementation details and performance evaluation results are discussed in Section V. Next, Section VI summarizes related work. The paper concludes with future work in Section VII.

II. Overview of NDN

NDN [18] is a communication architecture based on named content. Rather than addressing content by its location, NDN refers to it by name. A content name is composed of one or more variable-length components, separated by "/". For example, the name of a CNN news content might be: `/ndn/cnn/news/2011aug20`. Large pieces of content can be split into fragments with predictable names: fragment 137 of a YouTube video could be named: `/ndn/youtube/video-749.avi/137`.

Since the main abstraction is content, there is no explicit notion of "hosts" in NDN. Communication adheres to the *pull* model: content is delivered to consumers only upon explicit request. A consumer requests content by sending an *interest* packet. If an entity (a router or a host) can "satisfy" a given interest, it returns the corresponding *content object*. Interest and content are the only types of packets in NDN. A content packet with name X in NDN is never forwarded or routed unless it is preceded by an interest for name X.[1]

When a router receives an interest for name X and there are no pending interests for the same name in its PIT (Pending Interests Table), it forwards the interest to the next hop, according to its routing table. For each forwarded interest, a router stores some state information, including the name in the interest and the interface on which it was received. However, if an interest for X arrives while there is already an entry for the same name in the PIT, the router collapses the present interest (and any subsequent ones for X) storing only the interface on which it was received. When content is returned, the router forwards it out on all interfaces from which an interest for X has been received and flushes the corresponding PIT entry. Note that, since no additional information is needed to deliver content, an interest does not carry a "source" address. Any NDN router can provide content caching. Consequently, content might be fetched from routers caches, rather than from its original producer. (Hence, no "destination" addresses are used in NDN).

NDN deals with content authenticity and integrity by making digital signatures mandatory for all content. A signature binds content with its name, and provides origin authentication no matter how, when or from where it is retrieved. Public keys are treated as regular content: since all content is signed, each public key content is effectively a "certificate". NDN does not mandate any particular certification infrastructure, relegating trust management to individual applications. Private or restricted content in NDN is protected via encryption by the content publisher.

[1]Strictly speaking, content named $X' \neq X$ can be delivered in response to an interest for X, but only if X is a prefix of X'.

III. Lighting Control over NDN

In this section we introduce our framework for secure lighting control over NDN. Our setup involves four parties: a configuration manager (CM), one or more fixtures (Fix), one or more applications (App) and an authorization manager (AM). CM is in charge of the initial fixture configuration. This includes, on a per-fixture basis: assigning a fixture its NDN namespace, installing a trusted public key (owned by AM) that identifies the local domain, and giving a fixture its identity represented by a unique public key. Note that, in NDN parlance, "namespace" refers to content published by some entity, whereas, "identity" refers to a public key associated with some entity that publishes content. AM determines which applications are allowed to access each fixture, signs applications' public keys and (optionally) issues signed access control lists. While CM and AM represent distinct functions, in practice, they are likely to be physically co-located.

A. Base-Line Protocol

We start by observing that NDN can be easily used to securely implement basic lighting control without requiring any new features or components. When application App needs to send a command to fixture Fix, the base-line protocol works as follows:

1) App creates (and signs) a new content object cmd containing the desired command.
2) App issues an interest int_A with a name in Fix's namespace that references the name of cmd.
3) Fix receives int_A, stores it in its PIT, and issues an interest int_F for the name of cmd.
4) App receives int_F and responds with cmd.
5) Fix receives cmd, (1) checks its access control list to determine if App is authorized to execute the command in cmd, (2) verifies the signature of cmd, (3) executes the command, and (4) replies with an acknowledgement (from here on abbreviated as "ack") in the form of a new signed content object.
6) App receives the ack and verifies its signature.

The main drawback of this protocol is its high latency and bandwidth overhead: a single command requires 4 rounds and 4 messages, instead of the ideal 2 rounds/messages. Thus, this approach is a poor match for delay-sensitive lighting control.

Alternatively, Fix could continuously issue interests that solicit App's commands. This way, whenever App issues a new command, it does so by simply satisfying the most recent interest. This approach, however, introduces new problems. First, Fix would have to always issue one interest per each App allowed to control it. In an installation with multiple applications (m) interacting with a large number of fixtures (n), the overhead of periodic $O(mn)$ interests would be significant.

Also, an application would be unable to generate a rapid burst of commands to the same fixture, since App can only issue a new command after it receives an interest from Fix.

B. Whither Authenticated Interests?

We now consider another approach that, at least in principle, violates the tenets of NDN. Recall that NDN stipulates that all content objects must be signed. Each entity implementing the NDN protocol stack must be able to verify content signatures. Interests, however, are not subject to the same requirement. One reason for this design choice is efficiency: public-key signature generation and verification is expensive. Moreover, signatures from different parties prevent straightforward interest aggregation. Another reason is privacy: traditional public-key signatures carry information about the signer. There are, however, applications that could benefit from authenticated interests and control of building systems like lighting seems to be one.

Authenticated interests can be implemented using both public key and symmetric authentication mechanisms, i.e., signatures and message authentication codes (MACs), respectively. For the sake of generality, we refer to the output of both as *authentication tags*. Due to the flexibility of NDN naming, where name components can be application-determined and are opaque to the network, an authentication tag can be placed into an NDN name as a *bona fide* component thereof. This way, an authentication tag becomes transparent to NDN routers and only the target of the command would interpret and verify it. Regardless of their type, computation of authentication tags must be randomized to ensure uniqueness, based on either nonces or timestamps.

In general, using authenticated interests is fairly straightforward. CM configures each fixture with a specific namespace and AM assigns a set of rights to each application, tied with the application public key or to a symmetric key shared with the fixture. The name reflected in an authenticated interest would contain three parts: (1) prefix part (used for routing) that corresponds to the fixture namespace, (2) actual command, and (3) randomizer (nonce or timestamp) along with the authentication tag computed over the rest of the name:

$$\boxed{\underbrace{\texttt{/fixture-namespace}}_{(1)}\underbrace{\texttt{/command}}_{(2)}\underbrace{\texttt{/randomizer||auth-tag}}_{(3)}}$$

The idea is that, when Fix receives such an interest, it verifies the authentication tag, executes the command and replies with a signed ack as content. The first task (verifying the authentication tag) is simple only if one application controls the fixture or if part 1 of the name somehow uniquely identifies the requesting application. Whereas, if multiple applications are allowed to issue the same class of commands to a given fixture and use the same type of authentication tags, the fixture would need to determine the exact application by repeatedly verifying the authentication tag. This could translate into costly delays. This issue can be easily remedied by overloading NDN names even further and including another explicit part that identifies the requesting application (or its key).

A drawback to both this and the base-line approach is that the fixture needs to sign, in real-time, the acknowledgement, which is represented as content. Since a typical fixture is a relatively anemic computing device, signature generation may involve non-negligible delays.

IV. Secure Lighting Control Framework

Based on the preceding discussion, we conclude that a more specialized approach to secure lighting control over NDN is necessary in order to obtain reasonable performance while adhering to NDN principles. To this end, we construct a security framework that includes:

- A trust model wherein public keys are associated with NDN namespaces. The framework relies on this functionality to determine the entity that "owns" a particular namespace. For example, this allows us to ensure that a content object issued by a fixture in response to a command has been generated by the correct party.
- A syntax for key attributes and access control policies that binds a public key with its attributes, as determined by the signer (certifier) of this key.
- A protocol that defines how fixtures are initialized and how applications and fixtures handle authenticated commands.

In the design of our framework, we consider an adversary that can control the communication channel between App and Fix. The goal of the adversary is to (1) produce a command of its choice that is executed by Fix; (2) undetectably delay, or replay legitimate commands from App; (3) provide an acknowledgment to App for a command that has not been executed.

A. Trust Model

Our trust model allows an entity (e.g., applications and fixtures) to publish content only in its namespace or any of its children. ($name_A$ is a child of $name_B$ if the latter is a prefix of the former).

Zero or more public keys are associated with each namespace. A content object published under namespace $name_A$ must be signed using the key associated to $name_A$ or any of its ancestors.

A trusted third party (TTP) – e.g., AM – generates the key-pair $K_{root} = (pk_{root}, sk_{root})$ and distributes pk_{root}. This public key is used as root of trust; a signature on a content object computed using sk_{root} is always accepted. In order to associate pk_P, belonging to producer P, with namespace $name_P$, TTP publishes, under $name_P$`/key`, a content object containing pk_P.

P can delegate a key $sk'_P \neq sk_P$ to sign content in namespace $name_P$`/sub-namespace` by publishing the corresponding public key pk'_P under $name_P$`/sub-namespace/key`. This mechanism allows TTP to delegate some of its certification capabilities to each producer.

P can prove to anyone its ownership of a key linked to $name_P$ through a simple challenge-response protocol. The challenger issues an interest for a content object with name $name_P$`/nonce` where `nonce` is a fresh random string selected by the challenger. P is able to respond with a valid content object only if it owns the signing key linked to $name_P$, one of its ancestors, or TTP's signing key.

B. Key Attributes & Access Control Policies

Attributes of a public key are expressed using the name under which such key is published. Each attribute is a name/value pair expressed as two consecutive namespaces: the first indicates a key attribute name, and the second – its value.

Recall that an NDN content object is bound to its name by a public-key signature. According to our trust model, such a signature must be issued either by the TTP or by the owner of the namespace that contains the public key. Applications can define their own set of attributes. For example, a public key pk_P published under `/uci/ics/domain/lighting-domain-1/appname/light-board-1/access/full-access/expires/20151231235959Z/key` specifies that pk_P belongs to application *light-board-1* in domain *lighting-domain-1*, that has "full access" to fixtures in such domain.

C. The Protocol

We now introduce the protocol for controlling NDN-connected light fixtures. The protocol is composed of three sub-protocols: bootstrapping, application authorization and control.

Bootstrap. New fixtures must be *paired* with CM and *bootstrapped* in order to be able to receive commands from applications. The pairing process consists of the distribution of a symmetric key from Fix to CM. For example, in our implementation CM scans a barcode on Fix's enclosure, that represents a symmetric key factory-installed on Fix.

Next, CM initializes Fix. Fixture initialization consists of selecting an NDN name for Fix, (loosely) synchronizing CM and Fix clocks and installing (on Fix) a trusted public key that belongs to AM. This public key identifies the domain under which Fix operates. CM then communicates a signing key-pair to Fix.[2] This key-pair is linked to Fix's namespace, and it represents the identity of Fix. Additionally, CM can specify the NDN name of one or more ACLs that Fix must use to determine applications' permissions. At this time, Fix also generates a long-term secret master key k_{Fix}. This key is optionally used later to derive application-specific symmetric keys (i.e., k_{App}) for authentication purposes. Once Fix is correctly initialized, it responds with the current time and a hash of all the information exchanged during bootstrap.

Application Authorization. AM grants control privileges to an application by signing the latter's public key. Given pk_{App} belonging to App and intended permissions $perm_{\mathsf{App}}$, AM first constructs a namespace $name_{\mathsf{App}}$ containing "`access/`$perm_{\mathsf{App}}$", as specified in Section IV-B. Then it signs pk_{App} and publishes it (as content) under $name_{\mathsf{App}}$. Any fixture under control of AM can verify that App owns permission $perm_{\mathsf{App}}$ by asking it to prove ownership of namespace $name_{\mathsf{App}}$, as in Section IV-A.

Control Protocol. The protocol is designed for resource-constrained fixtures interacting with a large number of applications. Thus, we aim to minimize computation and communication costs and amount of memory required to perform interest authentication. We avoid storing per-application long-term information (e.g. application keys) on each fixture. A fixtures stores a constant amount of state for each application currently interacting with it. We emphasize that, in order to issue and verify commands, applications and fixtures do not need to communicate with either CM or AM.

[2]Fix can also generate a signing key-pair and communicate the public key to CM.

Application App, that owns a key distributed under $name_{\mathsf{App}}$`/key`, issues an interest with command `cmd` for fixture Fix with NDN name $name_{\mathsf{Fix}}$ as follows:

$$\boxed{name_{\mathsf{Fix}}/name_{\mathsf{App}}/\texttt{cmd/auth-token}}$$

The string "`cmd`" represents a fixture-specific command. Since our framework does not specify any particular format for commands, this string is simply treated as an *opaque* binary field. For example, a simple command could be: "`on`" or "`off`", while a more complex one could be: "`intensity/+10/rgb-8bit-color/F0FF39`".

The field `auth-token` encodes the command authentication token, constructed as: `state` || `authenticator`. The first part represents state information required to prevent timing and replay attacks. It is, in turn, composed of: sequence number, timestamp and estimated round-trip time (RTT) between App and Fix. The `authenticator` part is a signature or a MAC. In either case, it is computed over: "$name_{\mathsf{Fix}}/name_{\mathsf{App}}$`/cmd/state`". App signs its commands using the private counterpart of $name_{\mathsf{App}}$`/key`. The key used to compute and verify commands authenticated with MAC is negotiated between App and Fix as detailed below.

When a fixture receives the interest above, it determines whether to execute `cmd`, as follows:

1) Examines attributes in $name_{\mathsf{App}}$ to determine whether App is allowed to issue `cmd`.
2) If available, uses a local or remote ACL specified by CM during the bootstrap phase.
3) Verifies the state of the command. First determines whether the interest is current. Then, if it has no record of previous commands from App, Fix extracts the sequence number from `auth-token` and stores it as: $(name_{\mathsf{App}}, sequence\_number)$. Otherwise, it checks that the stored sequence number is lower than the one in `auth-token`.
4) Verifies `authenticator` – signature or MAC on the interest. In the former case, Fix retrieves public key $name_{\mathsf{App}}$`/key` and stores it in its local cache.

If a pair $(name_{\mathsf{App}}, sequence\_number)$ stored by Fix is not updated for a pre-determined amount of time, it is considered stale and deleted. This way, at any given time, a fixture only retains state information related to active applications.

D. Symmetric Authentication.

By default, fixtures and applications authenticate commands and acks using public-key signatures. However, for performance reasons, they can switch to MAC-s at anytime, which requires establishing a shared secret key. Recall that, at bootstrap, Fix generates a long-term secret key k_{Fix}. When App asks Fix to switch to symmetric authentication, the latter uses k_{Fix} to compute an application-specific key k_{App}. After verifying that App owns the namespace $name_{\mathsf{App}}$ (see Section IV-A), Fix computes $k_{\mathsf{App}} = \mathrm{PRF}_{k_{\mathsf{Fix}}}(name_{\mathsf{App}})$, where PRF refers to a suitable cryptographic pseudo-random function. Then, Fix sends k_{App} to App encrypted under public key $name_{\mathsf{App}}$`/key`.

Note that Fix does not need to store these application-specific symmetric keys: it can compute k_{App} from k_{Fix} whenever needed. Therefore, the amount of symmetric-key-related state stored by Fix does not depend on the number of authorized applications.

E. Ack Authentication

As usual with instrumented systems, a device being controlled (light fixture in our setting) needs to provide feedback after processing a command. In the context of NDN, this naturally results in a closed-loop control system and allows NDN routers to flush PIT entries corresponding to processed commands (interests). For obvious security reasons, acks must be authenticated. In resource-constrained environment of light fixtures, the cost of computing per ack public-key signatures is quite high, especially considering that a fixture might receive numerous closely-spaced commands. For this reason, we propose an NDN extension allowing fixtures to efficiently produce authenticated acks.

A natural and efficient alternative to public-key signatures are (symmetric) MACs. An application and a fixture could share a key, and use to authenticate acks, i.e., replace a signature on the content object (that carries the ack) with a MAC. Unfortunately, this approach is unworkable if fixtures and applications communicate through a public network. Since MACs are not publicly verifiable, intervening NDN routers cannot authenticate MAC-d content and may simply drop it.

Next, we describe a technique that allows public verifiability of acks without requiring public-key operations by fixtures, applications or NDN routers.

Encryption-based Authentication. This technique assumes that App and Fix share a symmetric key k, itself derived from Fix-App shared key k_{App}, which is generated at bootstrap time. To begin, App generates a random s-bit value x and, using a block cipher E with block size s, computes $y = \mathrm{E}_k(x)$, $z = \mathrm{H}(x)$ where H is a collision-resistant hash function and E is used in the ECB mode. App includes the pair (z, y) as part of the command to Fix. Recall that this command is represented as an NDN interest and, on the path to Fix, it leaves state in all intervening NDN routers.

Upon receiving an interest, Fix extracts x' from y as $x' = \mathrm{E}_k^{-1}(y)$ and re-computes $z' = \mathrm{H}(x')$. If $z' \neq z$, then Fix aborts; otherwise, it issues an ack in the form of an empty content object with x as a signature.

Although x is clearly not an actual signature, this technique allows public verifiability. An NDN router that observes the (ack) content object carrying x must have a corresponding interest (and therefore z) in its PIT. It can efficiently determine the relationship between the interest and the content by checking whether $\mathrm{H}(x){=}z$.

Commands that are not acknowledged can be retransmitted until they time out. Once a command expires, it must be reissued using a new challenge. Despite public verifiability, App cannot prove to a third party that it successfully interacted with Fix. This is because App can unilaterally produce any number of challenge/response pairs without any interaction with Fix.

Operation	Intel Core2Duo	ARM Cortex A8
Create auth. command (RSA-1024, pub exp 3)	1.981 ms	21.553 ms
Verify command (RSA-1024)	0.096 ms	0.435 ms
Compute HMAC key from Fix's secret	0.005 ms	0.046 ms
Create auth. command (HMAC-SHA256)	0.006 ms	0.067 ms
Verify command (HMAC-SHA256)	0.013 ms	0.152 ms

TABLE I
PERFORMANCE OF RSA AND HMAC AUTHENTICATED COMMANDS.

V. PROTOTYPE EVALUATION

In order to evaluate the performance of the proposed architecture, we implemented a library – called `NameCrypt` – designed for lighting control in a theatrical environment. We also deployed it in an actual theatrical lighting installation. In this setting, applications and lighting fixtures interact over a LAN.

Our setup involves three applications: (1) a sequencer that outputs pre-generated patterns, (2) a controller that uses algorithmic patterns and (3) a fader. These applications control eleven lights, connected to five embedded devices. The target platform of the lighting fixture is an off-the-shelf embedded device based on the Gumstix Overo Air [12] computer-on-a-module. This device is running a 600 MHz Texas Instruments OMAP 3503 ARM Cortex-A8 CPU with 256MB RAM.

A. Performance Evaluation

Experiments were performed on a commodity laptop – MacBook with 2.53GHz Intel Core2Duo CPU – that runs the sequencer, controller and fader, and on a low-powered embedded system representative of a lighting fixture or a low-power fixture controller. Table I shows the results of micro-benchmarks in command authentication. Time required to generate an RSA signature on the Intel platform is comparable to typical LAN latency and does not significantly affect the performance of the whole protocol. Verification is below average network latency on both platforms. For this reason, we believe that features provided by digital signatures and their relative low cost justifies their use in an environment where commands are generated on non-constrained device. On the other hand, benchmarks show that low-power devices are not well-suited for generating real-time signatures on commands. In this case, we recommend the use of HMAC.

Symmetric authentication incurs negligible performance impact. Fixtures must generate a symmetric key for each application starting from their secret. This requires far less than a millisecond on our test devices. Similar to command authentication, digital signatures do not introduce any significant delay on the Intel platform, while signature generation is relatively expensive on the ARM. Whenever viable, our tests show HMAC provides adequate performance. However when public verifiability is required and standard signatures are too expensive, encrypted challenges are an appealing option, as shown in Table II.

VI. RELATED WORK

Most current protocols descend from legacy control architectures based on serial communication [16]. These legacy architectures rely on a separate communication infrastructure. In this transition from serial to IP, vendors have rarely implemented additional security measures [19]. As a consequence, such protocols must

Operation	Intel Core2Duo	ARM Cortex A8
Sign content object (RSA 1024, pub exp 3)	2.018 ms	26.418 ms
Verify content object (RSA 1024, pub exp 3)	0.046 ms	1.301 ms
Authenticate/verify (HMAC)	0.006 ms	0.070 ms
Encrypted challenge – create	0.003 ms	0.043 ms
Encrypted challenge – answer	0.001 ms	0.015 ms
Encrypted challenge – verify	0.001 ms	0.015 ms

TABLE II
PERFORMANCE ANALYSIS OF ACK AUTHENTICATION.

be often run over VLANs, VPNs, IPSec or physically segregated networks [13], [19]. Physical segregation is often difficult or even impossible when protocols are running over RF. In this case several of them have been show to be insecure [19].

Most modern lighting control protocols descend from or implement DMX512 [5] by encapsulating DMX payload over modern media such as wired/wireless Ethernet [2] or RF [6]. This has resulted in various competing technologies such as Art-Net [2], ACN [16], ETCNet/ETCNet2 [7] which bridge lighting systems and allow them to coexist with newer technology and integrate into BAS.

ACN [16] is a set of ANSI standards which define a protocol suite for controlling lighting, networked entertainment devices, and existing control systems. It has been designed to address several shortcomings of existing lighting control protocols, specifically (1) having both an open protocol and specification, (2) media agnostic control, and (3) generalizing to any device that can be controlled [16]. ACN defines several protocols on top of UDP, and therefore naturally extends to any medium that can carry IP communication. Security is not addressed in this standard, which assumes that ACN data is transported over a secure network.

Alongside aforementioned lighting control protocols, there are proprietary vendor-centric solutions, such as Philips Dynalite and Philips KiNET. To the best of our knowledge, neither Dynalite nor KiNET offer any form of authentication or encryption between devices.

In contrast with lighting, BAS protocols interconnect multitudes of sensors and actuators across a building, including HVAC, building controls, as well as home and office lighting.

BACNet [1] is an open standard specifying a BAS protocol at the backbone level. BACNet supports encryption and authentication, although it has been shown to be insecure (see [19], [13], [21], [8], [10]). DES is the only supported block cipher, and authentication is susceptible to man-in-the-middle attacks [19].

KNX began as an open standard converging several existing standards in home automation and intelligent buildings. As outlined in [9], [8], [10], KNX provides no data security. The control communication to the fixture is limited to a rudimentary access control scheme. An extension proposed to improve the security of KNX is EIBsec [9].

LonTalk is the communication protocol for the LonWorks BAS. It supports several media types, such as RF, Infrared, Coaxial cable, and Fiber Optics. As detailed in [19], [20], [9], [8], [10], LonTalk provides minimal security. Each entity is limited to a single authentication key of up to 48 bits. All entities must share the same key if they want to verify messages amongst each other. Significant overhead is incurred for authentication as the protocol requires a 4-round challenge-response protocol invoked for each message the sender transmits.

VII. SUMMARY AND FUTURE WORK

This paper focused on securing instrumented environments connected via Content-Centric Networking (CCN), motivated by the increasing integration of Building Automation Systems (BAS) with enterprise networks and the Internet. In particular, we explored lighting systems over Named-Data Networking (NDN), a prominent instance of CCN. We identified security requirements in lighting control and constructed a concrete NDN-based security architecture. We then reported on the prototype implementation and experimental results.

Clearly, this work represents only the initial step towards assessing suitability of NDN for communication settings far from its *forte* of content distribution. Much more work is needed to securely adapt NDN to other types of instrumented environments.

Within the lighting domain, we plan to extend our current design to support multicast communication – i.e., controlling multiple fixtures with a single message.

REFERENCES

[1] ANSI. Standard 135-1995, BACnet a data communication protocol for building automation and control networks, 1995.

[2] Specification for the Art-Net 3 Ethernet Communication Standard. http://www.artisticlicence.com. Retrieved Feb. 2012.

[3] J. Burke, A. Horn, and A. Marianantoni. Authenticated lighting control using named data networking. Technical report, UCLA, Oct. 2012.

[4] S. DiBenedetto, P. Gasti, G. Tsudik, and E. Uzun. ANDaNA: Anonymous named data networking application. In *NDSS*, 2012.

[5] DMX512-A. http://www.opendmx.net/index.php/DMX512-A. Retrieved Feb. 2012.

[6] Lighting systems made easy a guide to lighting installations. http://www.leprecon.com/catalogs/280075BLightingMadeEasy.pdf. Retrieved Feb. 2012.

[7] Electronic Theater Controls (ETCNet). http://www.etcconnect.com/. Retrieved Feb. 2012.

[8] W. Granzer and W. Kastner. Security analysis of open building automation systems. In *SAFECOMP*, pages 303–316, 2010.

[9] W. Granzer, W. Kastner, G. Neugschwandtner, and F. Praus. Security in networked building automation systems. In *IEEE WFCS*, pages 283–292, Jun. 2006.

[10] W. Granzer, F. Praus, and W. Kastner. Security in building automation systems. *IEEE IES*, 57(11):3622–3630, Nov. 2010.

[11] M. Gritter and D. Cheriton. An architecture for content routing support in the internet. In *USENIX USITS*, 2001.

[12] Gumstix Overo Air. http://www.gumstix.com/store/product_info.php?products_id=226. Retrieved Feb. 2012.

[13] D. Holmberg and D. Evans. Bacnet wide area network security threat assessment. http://fire.nist.gov/bfrlpubs/build03/art034.html, Jul. 2003.

[14] V. Jacobson, D. Smetters, N. Briggs, M. Plass, J. Thornton, and R. Braynard. VoCCN: Voice-over content centric networks. In *ReArch*, 2009.

[15] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, and R. Braynard. Networking named content. In *ACM CoNEXT*, 2009.

[16] W. Jiang, Y. Jiang, and H. Ren. Analysis and prospect of control system for stage lighting. In *IEEE CISP*, volume 8, pages 3923–3929, Oct. 2010.

[17] T. Koponen, M. Chawla, B. Chun, A. Ermolinskiy, K. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. In *ACM SIGCOMM*, volume 37, pages 181–192. ACM, 2007.

[18] Named data networking project (NDN). http://named-data.org. Retrieved Feb. 2012.

[19] N. Okabe, S. Sakane, K. Miyazawa, K. Kamada, A. Inoue, and M. Ishiyama. Security architecture for control networks using IPsec and KINK. In *IEEE SAINT*, pages 414–420, 2005.

[20] C. Schwaiger and A. Treytl. Smart card based security for fieldbus systems. In *IEEE ETFA*, volume 1, pages 398–406, Sept. 2003.

[21] J. Zachary, R. Brooks, and D. Thompson. Secure integration of building network into the global internet. `http://fire.nist.gov/bfrlpubs//build03/art027.html`, Oct. 2002.

[22] Z. Zhu, J. Burke, L. Zhang, P. Gasti, Y. Lu, and V. Jacobson. A new approach to securing audio conference tools. In *AINTEC*, 2011.