NDN, Technical Report NDN-0011, 2012. http://named-data.net/techreports.html
Revision 1: October 2012

Authenticated Lighting Control Using Named Data Networking

Jeff Burke, Alex Horn, Alessandro Marianantoni
UCLA REMAP
{jburke, nano, alex}@remap.ucla.edu

DRAFT
October 10, 2012

Introduction

This report discusses the design and implementation of authenticated lighting control, as an example of actuation in Building Automation Systems (BAS)[1], using Named Data Networking (NDN). It describes the application motivation and outlines preliminary designs for device bootstrapping, assigning permissions to applications, and authenticated control. Additionally, it reviews our testbed implementation, deployment and initial testing. The purpose of this document is to provide a brief overview of work-in-progress. The security approaches described here are treated in more detail in an upcoming paper created in collaboration with the NDN security group at UC Irvine (Burke et al., 2012).

Motivation

NDN is a newly-proposed networking architecture that shifts the "thin waist" of the Internet from IP's host-centric model to a data-centric model, together with two important consequences. First, data are named by applications, and the network directly uses these data names, rather than host addresses, for packet delivery. Second, each name is associated with a cryptographic key, which is used to secure data directly (Zhang et al., 2010).

In the work described here, we explore lighting control over NDN as an example component of a Building Automation System (BAS), which in turn is an example of an "instrumented environment" or "cyberphysical system." BAS provides an interesting and challenging application domain for NDN because information-centric networking is generally discussed in the context of large scale content dissemination, as opposed to control, actuation, or remote execution. Lighting represents a broad class of actuators while posing limited physical safety concerns in comparison with, for example, motor control. Additionally, there is increasing use of IP-based networking of lighting fixtures in new architectural and entertainment deployments.

Application Context

IP networking is now common as the backbone of modern BAS, both within individual systems, such as lighting, and to interconnect various systems, such as lighting, HVAC, and energy management. (Often, IP is used to interconnect controllers that use legacy serial protocols or contact closures to control individual devices). To secure a BAS, current practice typically relies on physical or VLAN-based segregation. Unfortunately such physical or logical segregation not only requires advanced networking expertise to set up and maintain, but also makes interoperability with IT systems difficult and configurations brittle to change. Segregating the networks physically or through VLANs limits the very interoperability—between, say, energy management and lighting control—promised by IP based control. Moreover, this segregation is often temporarily or permanently violated in order to provide web access to some devices—e.g.,

[1] For readers not familiar with BAS, see, for example, the recent special section on it in IEEE Trans. on Industrial Electronics (Dietrich & Zucker, 2010), or one of the many textbooks on the topic.

allowing a dedicated piece of equipment Internet access so that it can download firmware updates. In addition, IP addressing (and other configuration information) for devices is often hard-coded in at least some applications running on the network, making the infrastructure changes painful to implement. The lack of available IPv4 addresses and deployment of NAT further complicate the access and interoperability of these networks.

Figure 1 illustrates a common context for lighting control (itself shown in the upper part of the diagram) as part of a larger building automation system. It is often deployed simultaneously with heating, ventilation, and air conditioning (HVAC) control, energy monitoring, and other building systems. While it may share physical network infrastructure in some cases, control traffic for each system may be isolated to particular VLANs, causing challenges for integrated monitoring and user-facing applications.

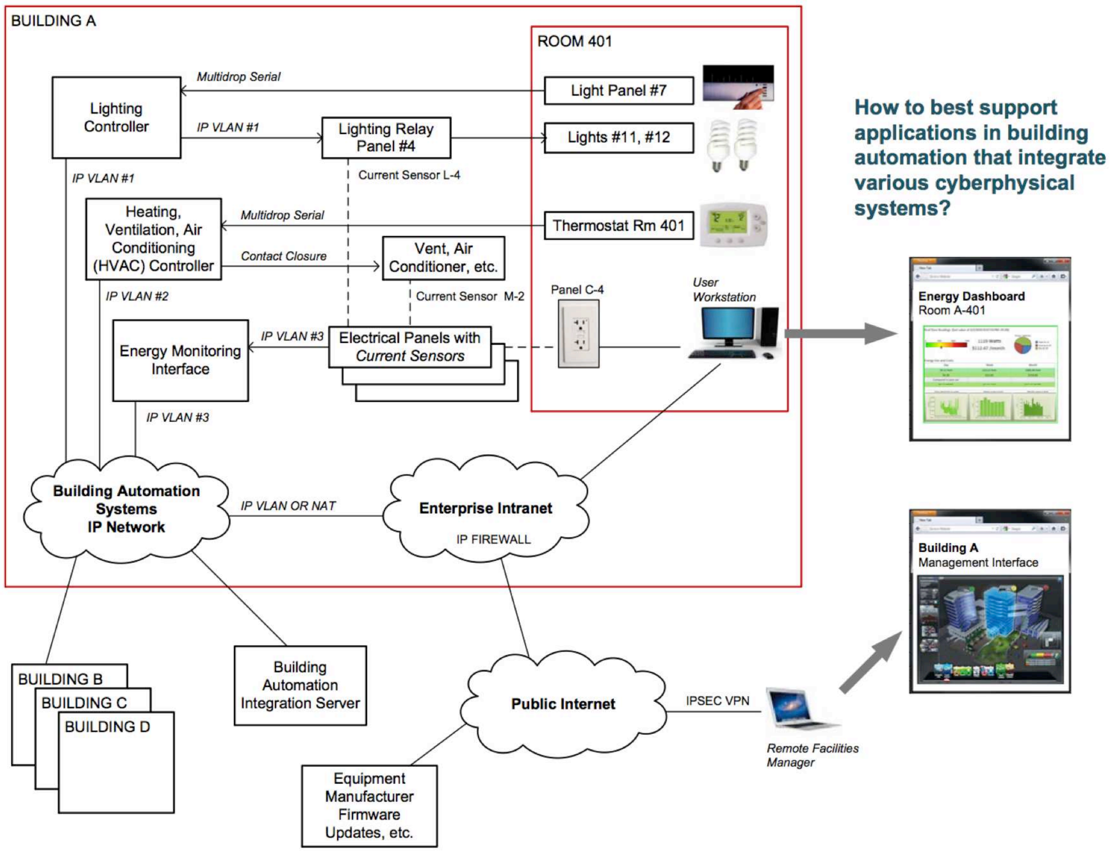

Figure 1. Building automation context.

Figure 2 shows our working abstraction of the building automation context for recent development, including the addition of critical control (e.g., fire / life-safety systems). It includes the following components:

- ***Fixtures and lighting interfaces*** – the lights themselves and their NDN gateways, embedded controllers connected to the lights via some legacy protocol (including IP). Eventually this functionality may be contained entirely in lights themselves.
- ***Configuration manager*** – an administrative process that assigns names and authorizes keys.
- ***Controllers*** – one or more controllers that affect lighting attributes (e.g., intensity).

- ***Fire / life-safety*** – a special category of process that can override control of other processes control of the fixtures to, for example, turn lights on or off in case of an emergency.
- ***Environmental monitoring*** – processes that have read-only access to lighting attributes.

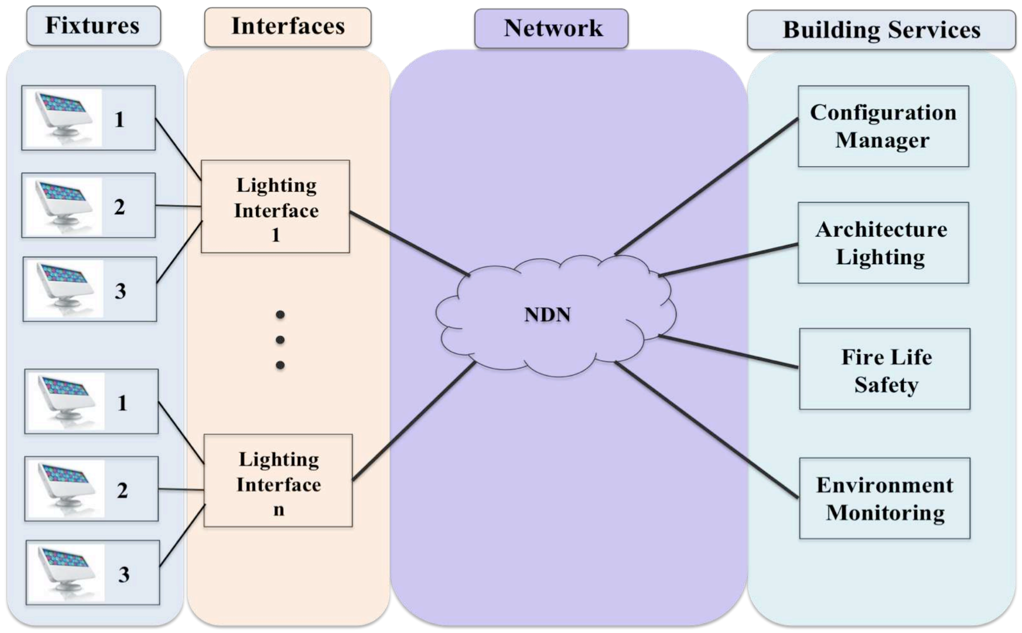

Figure 2. Relationship of services to lighting fixtures and the NDN network.

The "Building Services" in Figure 2 are considered to be processes with NDN connectivity, not hosts with IP connectivity.

Benefits of Name-based Networking

With this motivation and context in mind, we describe some of the potential benefits of using NDN to implement name-based networking in BAS that are consistent across applications and lower network layers, rather than relying on various application layers to translate network addresses / configuration to and from names.

In addition to using standard IP network naming approaches such as DNS, in many practical scenarios, BAS platform manufacturers provide middleware APIs (both public and internal to the manufacturer) that abstract away the addressing and protocol details. For example, an integration server may provide a programming interface that enables "light #11" to be addressed within applications and UIs as BUILDING_A.ROOM_401.LIGHT.11, and the middleware would provide the mapping from this application-meaningful name to the specific VLAN/IP address/Port-number/relay. Middleware may also expose common mechanisms through an industry standard protocol such as BACnet, enabling the control of this specific light by other systems using the same protocol. However, despite this simplification, important issues remain. First, the middleware itself can be complex and require significant development time. Second, the networking configuration task (VLAN, IP subnetting, port number etc.) still remains to be done and is handled outside of the middleware.

Significant knowledge is thus bound up in areas not accessible to application software developers or end users: VLANs embody boundaries between systems identified during design and deployment, but are typically invisible or inaccessible, as is IP subnetting and routing which reflect device organization and interconnectivity. Firewall configurations describe the rules for access between systems that can be difficult to change. Keys and certificates for SSL connections and VPNs may identify connections. VPN configuration and enterprise authentication hold network access permissions. None of these are typically visible or accessible to application software in traditional systems; they are "network configuration." In fact, they represent important system control logic that is often replicated ad-hoc in application configurations. A simple example is how an application must be configured to know that 192.168.2.1/24 is lighting, and 192.168.3.1/24 is HVAC.

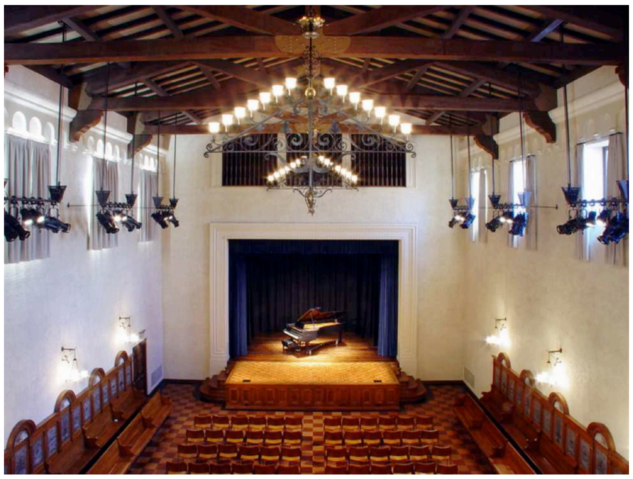
Figure 3. Example lighting control installation.

In NDN, names with meaning to the application can be used that are consistent across facilities and installations, e.g., `<enterprise_root>/lighting/<building>/<room>/<light>.`

Addressing is dealt with in one place in the architecture and used by both applications and network infrastructure.

Names may be used to address fixtures from different application perspectives. Consider, for example, the architectural lighting installation shown in figure 3. Here names used by applications could include:

- Manufacturer-assigned (used for bootstrapping): `/lighting/Philips/0041F31C493EF01A`
- Controller-assigned: `/<controller_root>/<port#>/<chan#>`
- Physical location: `/enterprise_root/<controller_id>/<fixture_id>`
- Region of Responsibility: `/room_root/region/wall_west/downlight`
- Designer-assigned: `/app_root/piano_spotlight`

NDN Lighting Control

To explore these potential benefits, we designed an approach for lighting control over NDN with the following design goals:

1. Satisfy low latency requirements for communication between software (or hardware) controllers and lighting dimmers or fixtures.[2]

[2] While not discussed in detail here, we target a maximum user-observed delay of 100 ms and update rates of (eventually) 100 Hz for all lights that need to be simultaneously controlled. This is in order to achieve perceptually real-time control from user interfaces.

2. Use NDN content naming to address all components of the system, with names related to their identity or function rather than a combination of addressing that spans layers and systems (e.g., VLAN tag, IP address of gateway, port of protocol, address of fixture) as in current implementations.
3. Given that NDN makes widespread use of content signatures, identify every entity in the system by a distinct public key.
4. Control access to fixtures via authorization policies, coupled with strong authentication. (Current BAS and lighting systems typically rely on physical or VLAN-based segregation for security, making interoperability with IT systems challenging, configurations brittle to change, and requiring advanced networking expertise to set up and maintain.)
5. Use NDN naming itself 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.
6. Develop security mechanisms suitable for low-power systems, initially targeting cell-phone class devices with a planned transition to microcontrollers typical of IP-connected lights today, such as the Phillips Color Kinetics ColorBlaze, that uses a 72-MHz ARM processor.

Figure 4. UCLA Television Studio #1.

Namespace

The initial namespace design uses names that made sense to applications authors with a background in architectural and theatrical lighting design. We also pursued general naming schema for naming control and data points in a BAS on NDN, which is not covered in this paper, and will be discussed in forthcoming reports on building automation[3].

In the system as designed and deployed, our application names for fixtures follow the schema:

```
<light> := <topological root>/<application>/<room name>/<device type>
```

For example, for lights on the set shown in figure 4, we use the following names:

```
/ndn/ucla.edu/apps/lighting/TV1/fixture/
            kitchen
            living-room-fill-right
            stairs
            bedroom
            living-room-front
            entrance-door
            window-left
```

The relationship between these desired application names and the names required to bootstrap and configure fixtures is described below, after a brief discussion of trust.

[3] See also related work such as (Culler & Ortiz, 2010), which describes a hierarchical naming approach for measurement instruments in a building energy management context.

Trust model / Name-based Permissions

The trust model developed for fixture control is SDSI-like (Rivest, 1996). Every entity in the system has a public and private key pair, and one or more trusted configuration managers assign keys to fixtures and controllers, expressing the controllers' permissions through the namespace in which its key is published (and signed by the controller). Those capabilities include *name*, *configure*, *control*, *read*, and *override*. The configuration managers themselves may be part of a set of processes with their own hierarchical trust relationship expressed in the namespace.

To authorize an application with a capability:

1. Permissions are granted to applications by publishing their keys under names that represents their authorized capabilities.
   `/<root>/lighting/<capability>/<app_name>/key`
   `/ndn/ucla.edu/apps/lighting/control/light_board/key`
2. Key data is signed by a key (for example, that of the Configuration Manager) that is already authorized.
3. To enable lookup by the fixture, these are published as:
   `<path-to-key>/authority/<fixture_name>/<capability>`[4]
4. Application issue authenticated interests in this namespace, with a signature including a locator for the authorized key.

For example, a controller's public key would be available as:

```
/<path_to_fixture>/lighting/<capability>/<app-name>/key
/ucla.edu/apps/lighting/control/light_board/key
```

Upon receiving a command to authenticate (described below), trust delegation is checked against the following lists:

1. Local cache of keys already authorized.
2. Trust anchor list generated through the bootstrapping step above.
3. If it is in neither list, the fixture issues an interest for
   `<path-to-key>/authority/<fixture_name>/<capability>`
   `and checks that it is signed by an authorized key.`

Discovery and Bootstrapping

To participate in the network, fixtures are assigned application-specific names. We expect that pervasive computing and instrumented environment applications will all use certain discovery mechanisms to find and configure new devices. We envision the following scenario to develop a discovery approach for lighting fixtures: When a new fixture is received from a manufacturer for installation in the field, it comes pre-configured with 1) an NDN name following well-known conventions for device discovery, as well as 2) a manufacturer-assigned public key[5] with the key fingerprint playing the role of today's serial numbers, and 3) a shared secret that will be used to authorize initial configuration. Note that the shared secret could be configured by the installer at the fixture, if the fixture had a front panel or exists on a barcode that could be scanned and

[4] Delegation could be handled by checking whether a key higher up in the name hierarchy signed both the requesting application and the fixture, but we are not sure yet if this will be a common hierarchy so this approach is more general.

[5] If the manufacturer's key is not trusted, additional steps could be taken to install a user-supplied public/private key pair.

communicated to the configuration process. Once the key fingerprint and secret are known to the configuration manager, discovery and bootstrapping proceed as follows.

1. Devices bootstrap by registering names of a known form "out of the box":
   `/ndn/lighting/devices/<manufacturer>/<type_components>/<serial>`
   `/ndn/lighting/devices/philips/ColorBlast/00-16-EB-05-FB-48`

   And publishing a manufacturer-assigned public key:
   `/ndn/lighting/devices/philips/ColorBlast/00-16-EB-05-FB-48/key`

2. The Configuration Manager periodically expresses discovery interests on a broadcast channel to which the lights are connected (e.g., a particular Ethernet segment) using a known convention:
   `/ndn/lighting`

3. Using an authenticated interest (described below) application-specific names are then assigned by the CM, which authenticates with the fixture via a shared secret passed out-of-band:
   `/<root>/<building>/<room>/<lights>/<fixture_path>`
   `/ndn/ucla.edu/melnitz/1471/lights/west_wall/wash_down`

In our application scenario, this type of discovery is used at fixture installation to locate fixtures with a manufacturer-supplied name. Then, interests in the application namespace(s) can be issued to discover fixtures available for control by a given application. Importantly, there is a limited amount of preconfigured information per fixture: an asymmetric key pair and shared secret.

Actuation via Authenticated Interests with Side Effects

A substantial contribution to building automation offered by NDN is the authentication of control messages and verification of data using basic features of the architecture. Data verification is not treated here as, at least for asymmetric signatures, it is intrinsic to the architecture and the primary challenge is developing the appropriate key hierarchy. Work on the lighting application focused on the authentication of control messages. For most applications, we believe that Interest/Data encryption of such commands will be unnecessary, but that authentication is a basic requirement poorly handled in many current IP-based control systems.

To keep command latency low and take advantage of Interest routing, our design uses Interests for control as shown in Figure 5a, to trigger side effects in the fixtures (e.g., changing the intensity or color of a light). The semantics are consistent with NDN: the controller is interested in the status data resulting from changing an actuator's property to a value expressed in the interest name. To authenticate the commands, we include a signature (or HMAC) in the name, calculated using the controller's key over the light name, capability, and desired value.[6] Our approach to authenticated control was developed in collaboration with the NDN security group at UCI, and is shown in Figure 5b.

[6] Signing Interests ties them to a particular creator, circumventing the normal anonymity of content requests in NDN; it also prevents them from being cached. We do not suggest this approach as a general-purpose extension, but rather as an engineering solution for this particular case.

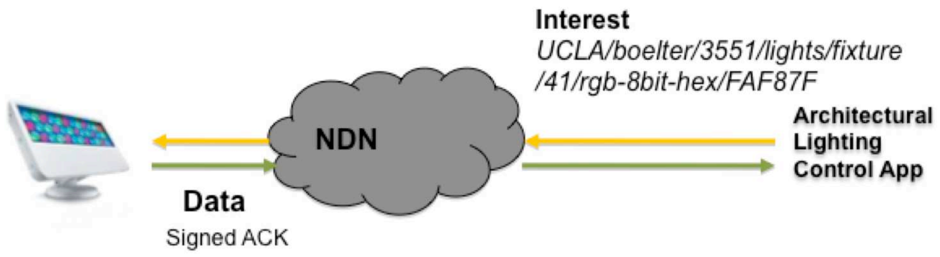

Figure 5a. Unauthenticated control via Interests.

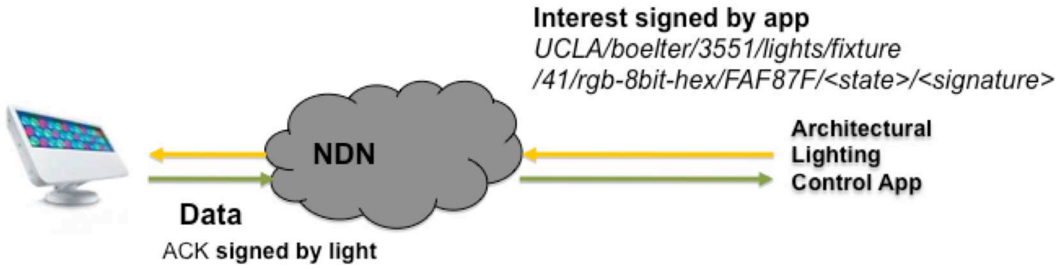

Figure 5b. Authenticated control via signed Interests.

For example, a lighting fixture in our testbed might be named using the following pattern:

```
<light> := /<path_to_fixture>/<building>/<room>/<lights>/<fixture-path>
 /ucla.edu/melnitz/TV1/lights/living-room-left
```

Verifiable control Interests follow the format:

```
      <light>/<param_pattern>/<parameters>/<state><signature>
```

e.g.,

```
      <light>/control/rgb-8bit-hex/F0FF39/[statebits][sigbits]
```

Verification algorithms

With UCI, we have explored both asymmetric and symmetric signatures as the verification mechanism. The details are discussed further in (Burke et al., 2012) and summarized briefly here. Notably, the signing time for asymmetric signatures has led us to explore HMACs using symmetric keys or hash chains for faster per-packet signatures.

Asymmetric Signatures

For RSA signatures, the name has a signed, empty ContentObject concatenated onto the name, just as in ccnd face registration. (In our implementation, the name is implicit in the ContentObject, and not carried over the wire.) In this case, the name also contains the KeyLocator for configuration manager public key, and the shared secret of fixture being addressed, encrypted with its public key.

HMAC Verification

In many lighting control applications, UDP is used to transmit control updates at high rates (50+ Hz). While NDN-specific optimizations or command compression may be possible (in particular, transmitting fade curves to fixtures rather than individual intensity values along the curve), we first aim to lower the latency to closer to what is found in current IP-based control. To lower computational costs, authorized application can propose a temporary symmetric key to fixture.

Then, controller uses a Hash-based Message Authentication Code (HMAC) to "sign" the Interest.

ACK Generation Time
Notably, the embedded controller must also generate a verifiable ACK, which requires a signature (or HMAC generation) at the fixture. Because this is done at the embedded device providing the fixture interface, it is slower has a more significant computational burden than at the controller, which is currently a workstation. ACK signing was noted as an RTT-limiting step, leading us to consider other techniques described in (Burke, et al. 2012), notably HMACs on replies. Initial tests of UCI code for HMAC on the embedded controller suggest at least a 200x speedup over asymmetric signing.

Implementation & Deployment

We have completed a working end-to-end lighting control system, consisting of a daemon running on an embedded Linux controller, along with a straightforward port of the CCNx routing daemon, and a remote control process running on a workstation. The lighting testbed is in a television studio on the UCLA campus, shown in Figure 4. We have eleven lights controlled by five TCP/IP controllers, which are in turn controlled by one embedded processor running *ccnd* that serves as an NDN gateway. We have also added real-time hardware (fader) control of the lights, in addition to software sequencing and an HTTP web controller. Each application can name the lights anything it chooses.

Testbed hardware

Light Fixtures
Fixtures are the individual lights controlled via NDN. In the future, these may contain NDN interfaces built-in. For now, they require connection to a power supply (e.g., a Philips Color Kinetics "Data Enabler") for both low-voltage power and control via IP over Ethernet or DMX[7].

Data Enabler
The Color Kinetics Data Enablers powering and controlling the fixtures have Ethernet interfaces and multiplex power and data to fixtures through one or more ports. The ethernet interfaces use a proprietary UDP/IP protocol called KiNET, which is used to both discover and control fixtures.

Lighting Interface
To connect the fixtures to the NDN network, a Linux-based embedded device (Gumstix Overo AIR, based on the ARM Cortex A8) with two Ethernet ports is connected to the local Data Enablers on one link, and to the rest of the NDN testbed on the other link.

Testbed software

The lighting control software was originally developed in C, but to ease development and aid integration with other projects, it was reimplemented in 2011 using Python. This provided an opportunity to test the new Python bindings for CCNx on an ARM-based embedded platform.

The component processes, representative of the abstractions in Figure 2, include:

Configuration Manager (CM)
Responsible for all network bindings between names, applications, and fixtures, the CM's public key is the root of authority in our lighting control system. It is currently only run when needed to discover and initialize new devices, and to authorize new applications.

[7] An industry standard multi-drop serial protocol based on RS-485.

Lighting Interface Controller

Running on the embedded device ("NDN Module" in Figure 6), this process listens for Interests in its connected lights' prefix, controls the lights, and publishes ACKs as data.

Runtime Applications

Two runtime control applications have been created: The *Sequencer* issues patterns of Interests to the lighting fixtures, and the *Fader* provides an interface to a physical lighting control panel.

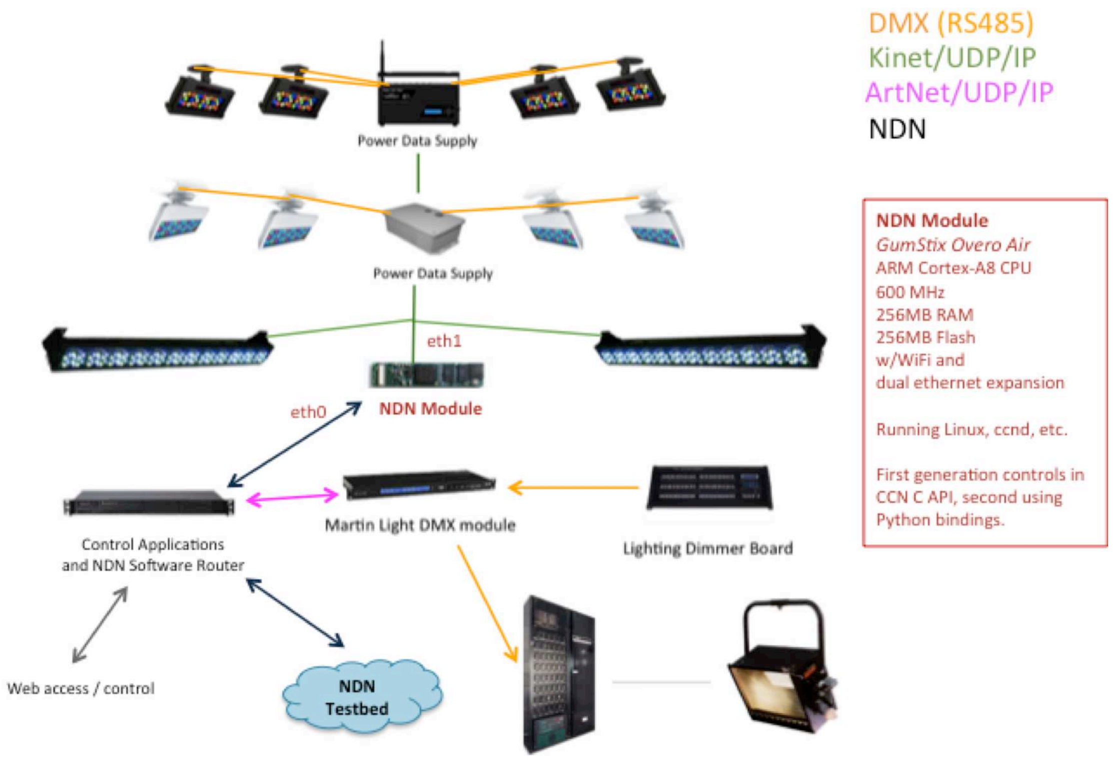

Figure 6. Lighting testbed components.

Keystore

Every entity has a 1024 bit RSA keypair, stored as a PEM file. On application instantiation, these public keys are published immediately as an NDN content object.

Implementation Limitations

There are known limitations in our initial implementation:

- *Configuration*: We plan to store configuration in device names, CCNx repos and on-device storage that can be accessed via NDN. Currently, each fixture and application has its own configuration file, containing its RSA keypair, names, NDN prefix, and capabilities. Configuration parameters are automatically generated during IP-based discovery.
- *Routing*: Fixtures in the NDN testbed connect to the upstream CCND daemon on the UCLA NDN hub via a static route. This allows each fixture to listen for its names within `<root>/<app_prefix>.` Next, we plan to use new autoconfiguration tools from the architecture and routing team.
- *Topology*: Currently, the NDN Module providing the interface to the fixtures looks like a 'gateway', with many fixtures connected through one NDN interface; the eventual goal is

one NDN interface per device, which may be approximated by running per-fixture processes on the embedded gateway.

Performance Tests

Forthcoming in the next draft.

Future Work

Going forward, lighting control as developed here will be used in deployed applications, where engineering will continue to decrease latency and take advantage of authentication to use an open network (the NDN testbed as an overlay on the public Internet) for control. More sophisticated building automation applications will be explored. These will include both sensing and actuation. We would like to simplify configuration further by using the new autoconfiguration and bootstrapping tools emerging from the other groups. Other open challenges include developing models for transmission of rare but critical events that seem best done with a "data push" approach, and testing group control of fixtures as described below.

Group control: Reversing Packet Flow

For synchronized control of many fixtures or to lower bandwidth, *fixtures* could issue (unsigned) interests, and receive commands in return. This trades off delay for group addressing. Content caching allows fixture to get state at low network cost.

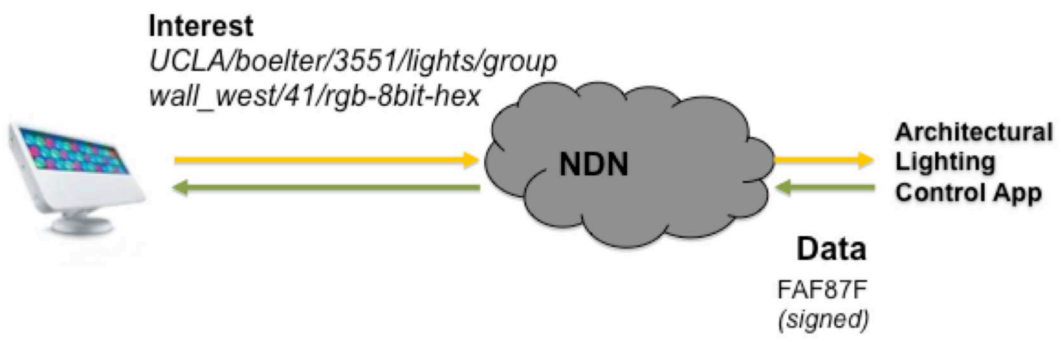
Interest
UCLA/boelter/3551/lights/group
wall_west/41/rgb-8bit-hex
NDN
Architectural Lighting Control App
Data
FAF87F
(signed)

Figure 9. Group-based control, reversing the packet flow.

Conclusion

Our current design and testing of lighting control over NDN, as a special case of actuation in building automation, incorporates the following features: 1) *identification* of entities on the network, including fixtures and applications, via an asymmetric key pair; 2) *capability authorization*, expressed entirely in the namespace; 3) *discovery and bootstrapping* via manufacturer-assigned names and a shared secret, in which fixtures are assigned *one or more application-specific names*; 4) *authenticated interests* for control, using either RSA signatures or HMACs that are verified and acknowledged through data packets authenticated by the fixtures.

The results of the approach are that 1) NDN's routing based on data names brings network semantics closer to application semantics, and 2) names/address are gathered into one place in the architecture. Because no configuration is required to express an Interest, 3) bootstrapping is also simplified. Finally, 4) authentication of commands provides intrinsic security for actuation.

Acknowledgements

Thanks to Gene Tsudik, Paolo Gasti and Naveen Nathan in the NDN security group at UC Irvine for their help in envisioning, designing, and implementing authenticated interests. Thanks also to Lixia Zhang at UCLA and Van Jacobson of PARC for many conversations and ideas about how to best use NDN's architecture for this application. Finally, thanks for Derek Kulinski for expanding and maintaining the Python bindings for CCNx used in later versions of this application.

Citations

Burke, J., P. Gasti, N. Nathan, and G. Tsudik. Securing Instrumented Environments over Content-Centric Networking: the Case of Lighting Control. (in submission) arXiv:1208.1336, August 2012. http://arxiv.org/abs/1208.1336

Culler, D. and J. Oritz, A System for Managing Physical Data in Buildings. Technical Report No. UCB/EECS-2010-128, EECS Department, University of California, Berkeley, September 28, 2010

Dietrich, D. and G. Zucker. Communication and Computation in Buildings: A Short Introduction and Overview. IEEE Trans. On Industrial Electronics, November 2010, pages 3577-3584

Rivest, R. L. and B. Lampson. SDSI - A Simple Distributed Security Infrastructure. Technical report, MIT, 1996.

Zhang, L., D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, et al. Named Data Networking (NDN) Project. PARC Technical Report NDN-001, October 2010.