**Application Perspective on Autoconfiguration / Publisher Mobility**

Revision 3 - updated October 14, 2015

Jeff Burke - jburke@ucla.edu

Major notes from Alex Afanasyev marked with [AA].

Purpose of this document

* Clarify and provide a single point of reference for the application team that describes the exact steps by which NDN connectivity is obtained “from scratch” by a mobile device that wishes to consume and publish data.
* Determine what functionality is part of the stack (as currently implemented by NFD and related tools), what should be part of the libraries (e.g., ndn-cxx or ndn-ccl), and what is the responsibility of the application, as well as how communication between these components occurs.
* Distinguish between mechanisms for “now” (next two months) and “what is planned within the NP timeline” (by May 2016) and “what is the eventual best way” (next few years).
* Establish open questions, etc.
* Figure out a practical way to support straightforward use of NdnCon and the NP environment sample applications.

Terminology

*local node* - a slight generalization of an NDN instance that wants to autoconfigure: an instance of one forwarder with a given default identity, libraries, multiple apps, etc. (e.g., a host, a browser tab (someday), a VM, a mini-NDN container, etc.)

*upstream node* – any node within one hop of the local node (e.g., localhop scope) that will forward interests to/from the local node

Application group use case

Provide continuous connectivity to testbed for mobile nodes and ensure that applications can work, despite (local) network connectivity changes.

For example,

* Current *NdnCon* user, who has a certificate assigned for the testbed in, say /ndn/edu/ucla, wishes to travel from campus to campus, communicating via NdnCon with no location-specific steps needed.

Identities

We need agreed terminology for the identities involved in the use case, and a clear definition of the trust relationship between them.

There seem to be at least three identities involved in a simple node operation, which might all be conflated into a single cert but play different roles. Can we settle on terminology for them, and clarify when they are used (in short- vs. medium- term):

1. Identity used by NFD in a default installation

/local/<what?>

2. Identity used for a given communication provider

/ndn/edu/uiuc/<what?>

3. Identity to publish in a given prefix

/ndn/edu/ucla/<what?>

When are these used? When are they the same? When are they different? How do they relate to the testbed identity?

Aspects of Autoconfiguration

**#1 – Bootstrapping Local Node Identity**

( [AA] There is no such thing. )

Current function: Install a default identity (self-signed certificate) associated with a given forwarder instance, and install it as the default to use for NFD?

General function: Obtain a local node-level identity that can be used by the local forwarder to issue commands to upstream forwarders necessary for further autoconfiguration steps.

Supported by: ndnsec [installed with nfd] for cert generation

Defined in: nfd.conf?

Described in: <http://named-data.net/doc/NFD/current/INSTALL.html>

 <http://named-data.net/doc/NFD/current/FAQ.html#how-to-configure-nfd-security>

Use case: Prior to installing NdnCon, the user / configurer of the local node installs nfd, and configures the identity for the forwarder. [Following what steps exactly?]

There is no assumed trust relationship between the cert of a local node’s nfd and any associated testbed user. [AA]

For our use case:

What the local node does: Self-signs a cert used for the node?

What the upstream node does: N/A.

What the library does:

What the app does: Nothing.

Questions:

* What is the nature of the cert used by NFD for data/interest signing?
* Why is repo-ng not needed (per AA) to serve this cert?
* In the future, how will one bootstrap local node identity with no connectivity?
	+ This is something related to local configuration (e.g., to allow run configuration commands). [AA]
	+ See some comments in next section that might apply?

**#2 – Bootstrapping connectivity-authentication identity (currently “testbed user”)**

Note: We put this as step 2 because it is required for the fallback (step 3) option of ndn-autoconfig, which connects to the hub hardcoded as associated with a given prefix. Further: it is essential part of automatic prefix propagation; it may or may not be directly used by applications. [AA]

Current function: Generate and authorize a certificate for publishing on the NDN testbed, install this for use as the default cert for applications within a given containerlocal node.

General function: For a given administrative domain (upstream node, at the simplest), authorize a given identity to publish data under a given prefix… perhaps one of many belonging to a “user”.

Supported by: <http://ndncert.named-data.net>

 ndnsec

 ndn-pib

 repo-ng (?)

Described in: <http://ndncert.named-data.net/help>

<http://named-data.net/doc/ndn-cxx/current/manpages/ndnsec.html>

<http://redmine.named-data.net/projects/ndn-cxx/wiki/PublicKey_Info_Base>

<http://redmine.named-data.net/projects/repo-ng/wiki>

[AA-]

Almost full instruction are on ndncert page.

However, there are “hidden” parts that depend on the way nfd is installed.

* When nfd installed manually from sources, nothing else to be done
* When nfd installed on Ubuntu from PPA, OSX from Macport, OSX from Homebrew, certificate (or derivative) needs to be added to NFD’s

A few highlights:

* Users and individual applications may have completely separate private/public/cert local nodes (in long term and on special platforms = always have separate). As of right now, the local node is solely controlled by $HOME variable (each user, minus special cases, has unique HOME, for apps HOME can be overwritten).
* On Ubuntu/PPA, nfd is running as root with custom HOME variable (/var/lib/ndn/nfd; path can be checked in /etc/init/nfd.conf);
* Macports and homebrew similar, but different paths: /opt/local/var/lib/ndn/nfd (+custom client.conf forcing file-based TPM); /usr/local/var/lib/ndn/nfd (+custom client.conf forcing file-based TPM)

The bootstrap would need (depending on how things installed) the following:

* creating derivative key for <identity> or <identity>/rib (this one is to ensure the cert can be used only for RIB tasks)
* publishing cert somewhere (with ndn-pib or repo-ng)

Use case: Prior to installing NdnCon, the user / configurer of the local node generates or retrieves a certificate associated with *an identity*, and this can be installed as the default identity for applications built against the standard NDN libraries. This cert can be used to sign data in the appropriate namespace. It can be default, but doesn’t need to be (this is longer term, but apps should use “schema” for appropriate key/cert selection).

For our use case:

What the local node does: Is it configured to use this as the identity for NFD too? (Are #1 and #2 conflated?)

What the upstream node does: Verifies commands are signed by an identity with the right trust relationship to the testbed? Verifies data are signed by this identity too?

What the library does: Configured to use this identity as default?

What the app does: Nothing.

Questions:

* Is the same testbed authentication cert used on different nodes operated by the same user? Wouldn’t it be preferably to identify these nodes differently, in the short term?
* In the long term, does it make sense to distinguish between “the cert a node uses to gain connectivity” (and register prefixes?), “the cert that authorizes a node to publish in some potentially non-localhop propagated prefix”, and a cert associated with a particular real human? Are these indeed conflated in the short term?
* Is there still a default application identity defined in client.conf or not?
* In the short term, how does an application know what identity to use as its root?

**#3 - Obtaining NDN Connectivity**

Current function: Establish a face to the best IP endpoint for a forwarder connected to the NDN testbed.

General function: On a change in connectivity, automatically (re)establish faces to one or more NDN forwarders that can provide the local local node with best NDN connectivity over its available communication media, given the identity established for the local node in #1.

Supported by: ndn-autoconfig

Described in: http://named-data.net/doc/NFD/current/manpages/ndn-autoconfig.html

 http://redmine.named-data.net/projects/nfd/wiki/HubDiscovery

 http://redmine.named-data.net/projects/nfd/wiki/NFD\_autoconfiguration [incomplete]

Use case: “ndn-autoconfig –d” (re)creates routes to the testbed. Changes are / could be passed up to NdnCon through a notification mechanism?

Currently, this must be run manually in from-source installations. With Ubuntu/ppa, homebrew, macports, this is as trivial as creating a upstart/system/launchd script. [JB: Does this script exist on these platforms? I think it does but am not sure, don’t have one handy to check.]

[AA:] Currently, ndn-autoconfig does not delete anything, rather adds or re(creates) connection. These faces are not persistent. (It is intentional for face to be persistent.) Non-reachable faces should be removed when detected.

Questions:

* How does autoconfig know what certificate on the local node to use to determine the fallback prefix (autoconfig step 3)?
* Currently, when NDN applications are running and autoconfigured prefix(es) change, applications are not informed of this. (Nor of adding/removing default route.) Will they be in the future? In the meantime, how should they detect changes that may require them to publish in a different namespace? Probe via #6 periodically?
* Why does autoconfig only return a single route? Might it be generalized in the future.
* How fast does autoconfig detect and re-establish connections after changes?
* What connectivity changes are detected by autoconfig?
* Should faces created by autoconf be persistent?

For our use case:

What the local node does: Runs autoconfig client; accesses the testbed cert installed on the local node.

What the upstream node does: NFD provides autoconfig responder.

What the infrastructure provides: Potentially DNS

What the library does: Informs application of connectivity changes? Provides LINK functionality?

What the app does: Reacts to connectivity changes? Notifies the user of prefix changes requiring new namespace selection for publishing?

Library action items:

* Is there / will there be notification of route changes (caused by autoconfig or nfdc) that should be handled by the libraries / available to applications?

**#4 – Autoregistration of Backroutes**

Current function: On the testbed, when a local node creates a route to a testbed node, some prefixes (such as (/ndn/multicast) are automatically registered, resulting in interests in those prefixes being forwarded to any local node connected to that testbed node.

General function: When a local node creates a face to another (upstream) node, the latter registers a predetermined set of prefixes with the local node, so the local node will receive interests in those “default” prefixes.

Supported by: ndn-autoreg server

Described in: <http://named-data.net/doc/NFD/current/manpages/nfd-autoreg.html>

Question:

* Is the local NFD informed by the upstream NFD when a backroute is created?

Use case: Whenever an NFD instance used by NdnCon sets up connectivity to the testbed and sends an Interest, the /ndn/multicast prefix is automatically registered as a backroute, enabling it to participate in chats / have access to a shared control channel, and discover other users (via ChronoSync mechanism). Note that an Interest must be transmitted for the backroute to be created, so the upstream forwarder knows about the local node in the case of connectionless transport.

For our use case:

What the local node does: Forwards interests normally.

What the upstream node does: Installs the backroute when receiving an interest from the face?

What the library does: Can detect the backroute installation through mechanism in #6?

What the app does: Issues an interest that unwittingly triggers backroute installation?

**#5 – Explicit Prefix Registration & Propagation**

Current function: 1) creates a “route” from the local forwarder to the app; and 2) creates a “route” from the upstream forwarders to the local node NFD (and in turn to the application).

General function: Per-application, per-face registration of a prefix for which the application wishes to receive Interests from local hops. 1) For the local node, creates a “route” from the local forwarder to the app; 2) creates a “route” from the upstream forwarder to the local local NFD.

[AA:] How far this propagates depends on the configuration (which keys NFD on the hub possesses). For now, assumption is that this propagation goes till the node that runs NLSR (the announced prefix supposed to be already configured in NLSR; if not configured, something will break).

Note, the upstream NFD’s nfd.conf’s rib…localhost\_security needs to be configured to the appropriate trust schema. [Where is this scheme defined?]

Supported by: Libraries and forwarder

For our use case:

What the local node does: Propagates routes

What the upstream node does: Installs and propagates routes?

What the library does: Provides API for registration? Can detect propagation using #6?

What the app does: Might react to events from library regarding registration/propagation?

Described in: Li, Y. et al, “Automatic Prefix Propagation.” NDN Tech Report NDN-00XX.

Question:

* Is any new library support needed for propagation? My understanding is this is behind-the-scenes?
* How does this relate to support for listening on an available communication media (e.g., wireless broadcast channel) without registering an explicit route with a remote forwarder? This feature comes up in IoT related discussions quite a bit.

**#6 – [Local] Prefix Discovery**

Current/General function: Determine the prefixes for which the upstream forwarder will forward interests to the local node; typically used for globally routed prefixes.

[AA:] This mechanism is for discovery of “globally” routable prefixes (prefix that app should use if it wants to communicate beyond 2 hops).

Questions:

* Why not have a similar function to tell what prefixes the upstream forwarder is willing to forward on?

Supported by: nfd-autoconfig-server

Described in: <http://named-data.net/doc/NFD/current/misc/local-prefix-discovery.html#local-prefix-discovery>

Use case: Until support is provided in underlying libraries (?), NdnCon should check forwardable prefixes returned by this Interest against the prefixes it wishes to publish data in, and take action to either republish in the new prefix and inform interested consumers of the change in an application-specific way, *or* follow step #7 to provide redirection.

For our use case.

What the local node does: Forwards interests/data as usual.

What the upstream node does: Responds to request in the namespace described above?

What the library does: Provides API calls for this? Should detect mismatches between locally registered prefixes and what is forwarded and notify app?

What the app does: Must detect mismatches between desired publishing prefix and forwarded prefixes, and employ either an application specific mechanism or a standard mechanism as described in #7 below.

Library action items:

* What library support is intended? Is it already available in ndn-cxx?

Aspects of Publisher Mobility Support

**#7 – Prefix redirection / Mobile publisher support**

Arguably not part of autoconfiguration. *But,* relates to typical steps that an application / local node must take to ensure that it can publish in the prefixes that it wishes to… And these steps may need to get reconfigured/changed upon changes in connectivity (routable prefix) – doesn’t that make it an autoconfiguration step?

Current / General function: For prefixes that a local node wishes to publish but are not forwarded by the upstream node, provide a mechanism to redirect interests to a prefix that it can publish under.

Supported by: Register LINK in NDNS, or application specific mechanism (like a message in /ndn/multicast). (Libraries for redirection?)

[AA:] SNAMP has 2 use cases: routing scalability (we don’t have this problem yet), mobile upload (not the case).

Described in:

* What is canonical ndns ref?
* Afanasyev, Alexander, et al. "SNAMP: Secure Namespace Mapping to Scale NDN Forwarding." Proceedings of 18th IEEE Global Internet Symposium (GI 2015). 2015.
* Link specified in: <http://redmine.named-data.net/issues/2587>
* Forwarding behavior specified in: <http://redmine.named-data.net/issues/3000>

Use case: When a local node that is configured for a user at UCLA (Step #2) obtains NDN connectivity via the UIUC hub, it registers a redirection link with its /ndn/edu/ucla NDNS to enable interests in that prefix to find it.

Somehow the trust chain for signed data in the local prefix has to be traceable back to both authorization for the local prefix AND for the permanent prefix?

What the local node does:

What the upstream node does:

What the library does:

What the app does: Nothing.

Questions:

* In NdnCon, when a user has connectivity from an upstream node that doesn’t forward it’s home prefix, what does it use to sign packets?
* Are LINKs used to redirect prefixes or on an interest by interest basis?
* How does the app / node get notified of a change in the link?
* In practice in the short to medium term, which parts of publisher mobility are supported in the forwarder, which in the libraries, and what does each application need to do?