POLLERE, INC.
network analysis & architecture, performance

Building a Bridge from Applications to NDN

Kathleen Nichols

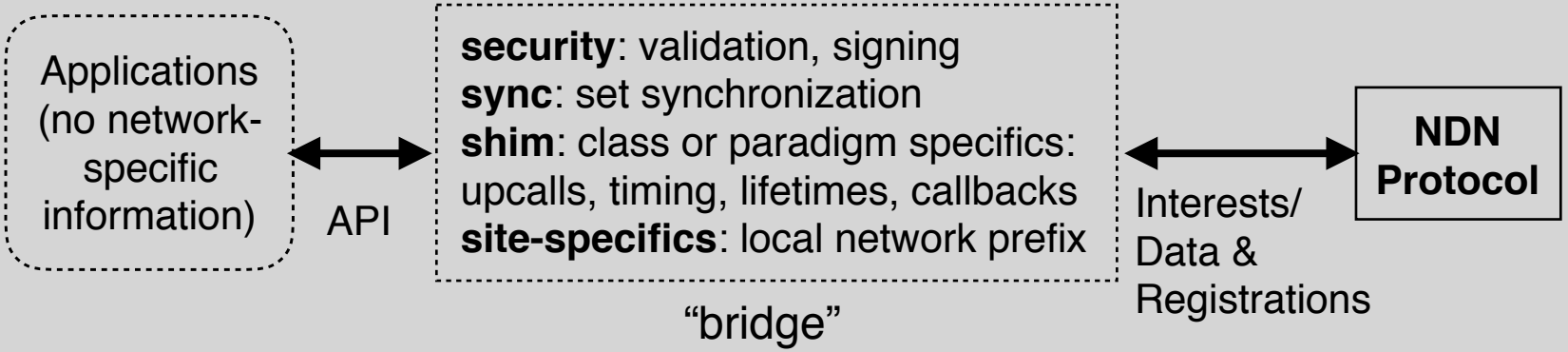
NDN Community Meeting

September 5, 2019

Why I like the NDN Architecture

- Multicast network protocol with security as first class citizen
- Interest/Data pairing guarantees flow balance for multi-source /multi-destination traffic (unlike IP multicast)
- Signed Data requirement basis for strong security
- Trust schemas and Name structure can provide rich security models
- Data transport based on set synchronization rather than conversation provides potential for efficient communications on today's broadcast channels
- User-space transport for Application Layer Framing

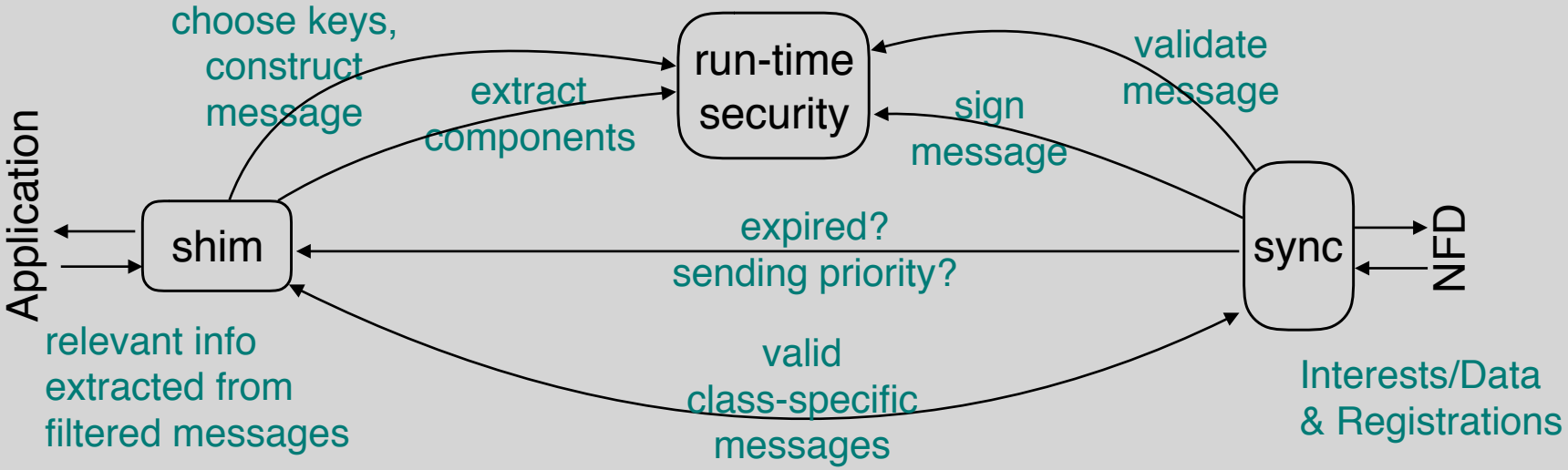
Toward portable, easy-to-write applications



- Applications need help, but not a straight-jacket
- Applications belong to a class or communications paradigm that provides the transport functionality and an API that only requires the application-relevant information
- A "bespoke transport" should provide class specificity using common functional modules and frameworks that provide validity checks for data, both security and expiry, to construct valid packets. Configured with site-specifics (networks, keys)

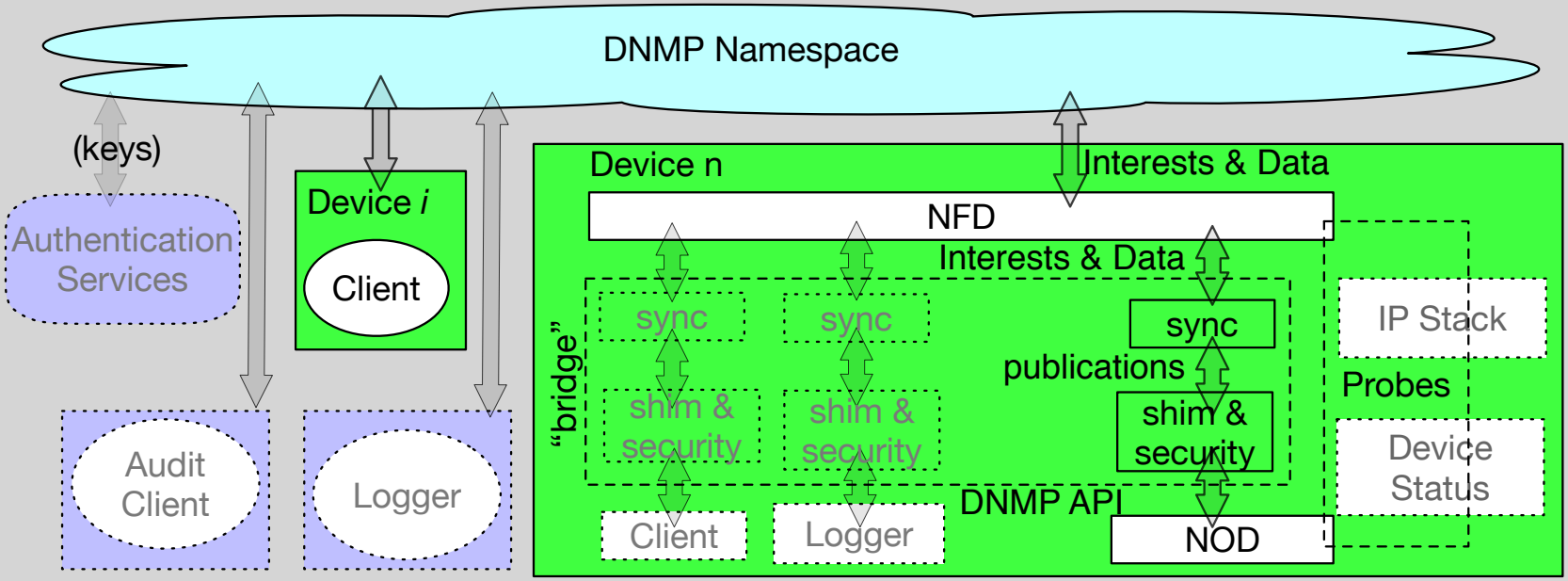
Constructing a bridge

Functional modules, not rigid layers; use upcalls to provide signing, validation, lifetime, and priority information



- API passes application-relevant information
- Shim and security modules apply paradigm-specific information and site-specific configurations
- These create and parse requests and messages (e.g. publications) that are specific to this paradigm that the sync translates to and from Interests and Data exchanged with NFD

Distributed Network Measurement Protocol



- Clients, NODs (Network Observer Daemons), Loggers, Audits are applications.
- DNMP API provides a topic-based communications paradigm, passing commands and their targets and a callback for results, receives commands, passes results
- DNMP "bridge" enforces trust schema and provides topic-specific logic, creating and parsing publications

Holes in the bridge: Sync

- Available Syncs use producer/consumer model
- Our goal is MQTT-like sync utilizing NDN to be brokerless and broadcast-efficient
- Easier to write new publish/subscribe sync, *syncps*
  - Interests sent that give Topic and IBLT that indicates what publications sender has
  - Receivers put all new publications in Topic in a Data packet
  - Publications have a limited lifetime and a timestamp that bounds state needed to prevent replay, bounds publication lifetime
  - Names constructed to reflect their functionality and the trust schema, e.g. command/reply akin to ephemeral RPC request/response
  - DNMP's trust schema applied to the publications sent and received from *syncps*, **not** the packets on the wire

Holes in the bridge: performance issues

- "This doesn't work the way you think it does"
- The NFD code doesn't match the architecture, particularly devastating impact on multicast
  - Interests are not held in PIT until timeout, but only put in PIT on forward
  - PIT not checked on new FIB entry, e.g. new registration
  - LP::Nacks cause premature Interest death
  - No Interest suppression reduces efficiency
  - RETX suppression causes premature Interest death
- Patches completed for these problems
  - Mostly involve *removing* code
  - Insufficient broadcast testing is being done on codebase additions

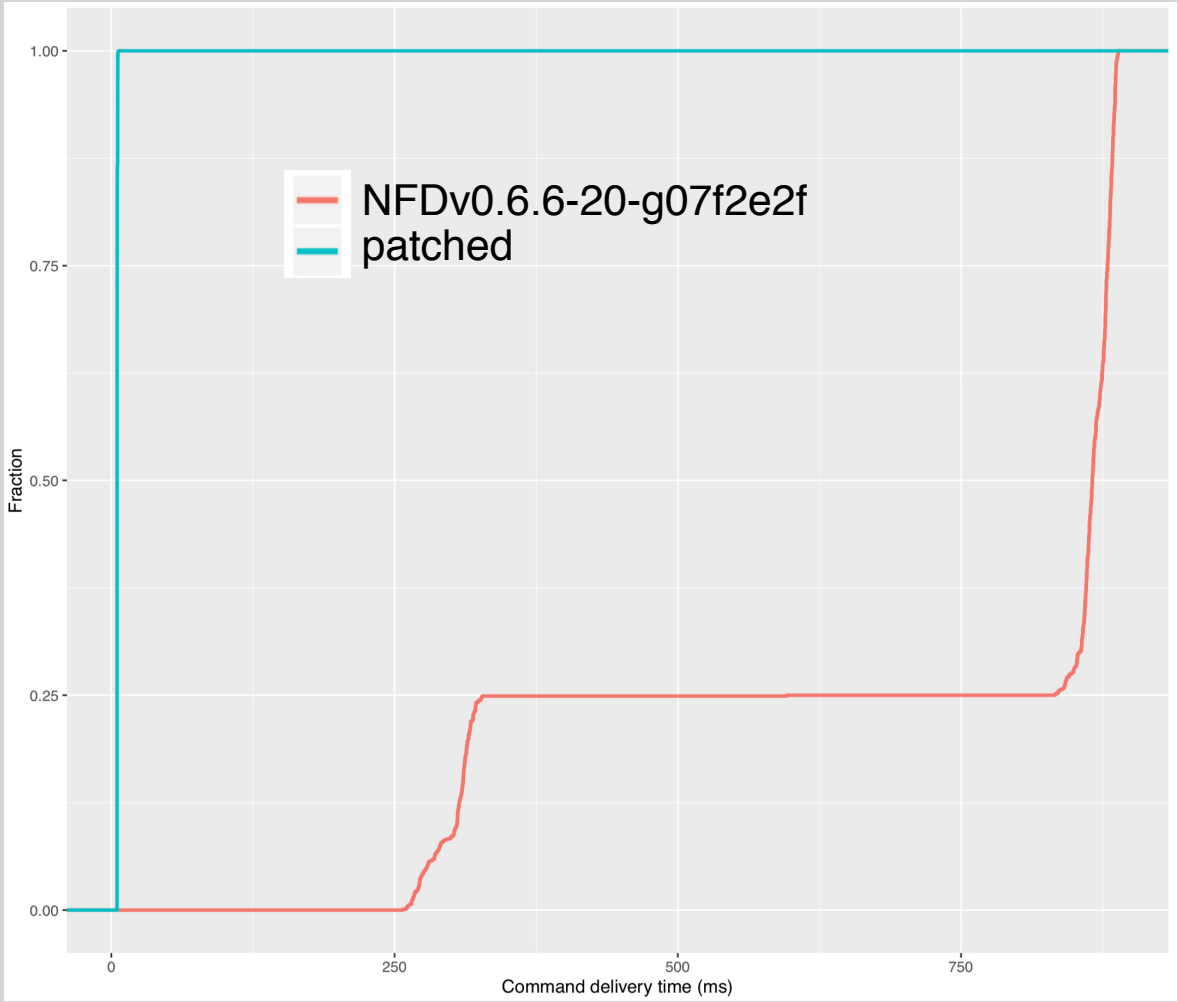
Patch fixes and more explanation at github.com/pollere/NDNpatches

Patches for LP::Nack and PIT discard issues

Test uses echo measurement (origination timestamps of both initial Command and its Reply) 20,000 exchanges:

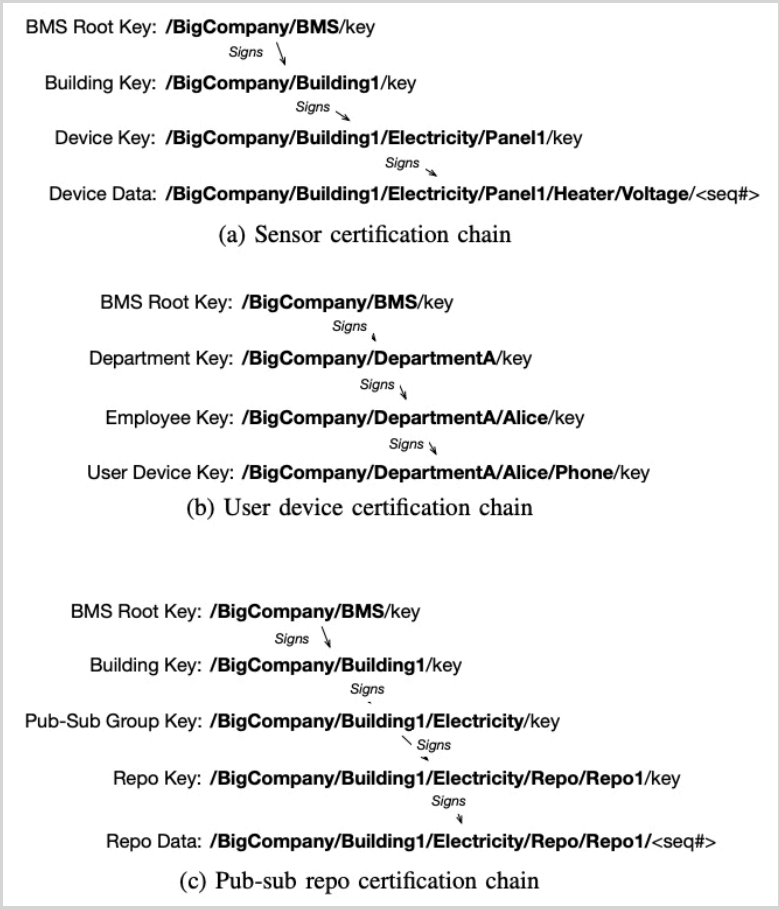
Before patch: mean=730ms, median=866ms

After patch: mean=median=5ms



Takeaway: rigorous application-driven testing and measurement must be performed so that applications get known quantity

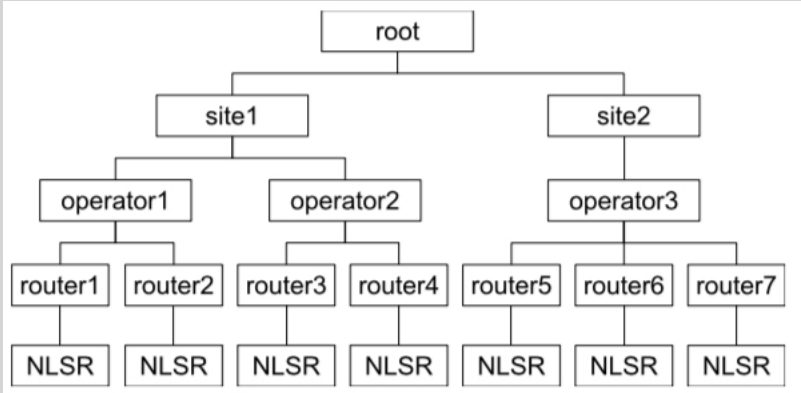
Specifying Trust Rules (some examples)



"Publish-Subscribe Communication in Building Management Systems over Named Data Networking"

Figure 7: Trust schema the blog website framework with "/a/KEY/1" as the trust anchor

"Schematizing Trust in Named Data Networking"



"Secure Link State Routing Protocol for NDN" (NLSR)

DNMP publication names and trust rules

<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 <k|n>
exact value of **0** is used if only this Data packet in the reply <k|n> 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

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		

- Names are "verbose" for debugging
- Redundant components can be removed for deployment

From "Lessons Learned Building a Secure Network Measurement Framework using Basic NDN" *to appear in* Proceedings of ACM ICN 2019.

Holes in the bridge: applying trust rules/schema

- The regular expression language for validator input doesn't mirror the human specification and can't cross-validate rules
- At best, existing validator only checks *some* components, *some* Names
- But trust rules *define* Names and signing relationships and should be usable to*:
  - check soundness of the trust schema
  - construct packets and automatically choose signing keys
  - validate *entire* signing chain, syntax and authorizations

*See "Lessons Learned" paper and github.com/pollere/versec (later this month)

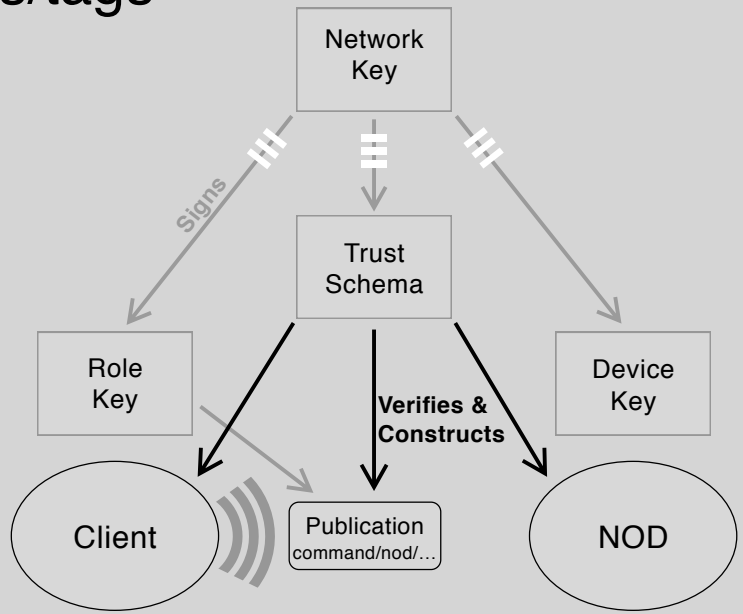
RegEx security section of nlsr.conf

```
security
{
  validator
  {
    rule
    {
      id "NLSR Hello Rule"
      for data
      filter
      {
        type name
        regex ^[^<nlsr><INFO>]*<nlsr><INFO><><><>$
      }
      checker
      {
        type customized
        sig-type rsa-sha256
        key-locator
        {
          type name
          hyper-relation
          {
            k-regex ^([^<KEY><nlsr>]*)<nlsr><KEY><>$
            k-expand \\1
            h-relation equal
            p-regex
^([^<nlsr><INFO>]*)<nlsr><INFO><><><>$
            p-expand \\1
          }
        }
      }
    }

    rule
    {
      id "NLSR LSA Rule"
      for data
      filter
      {
        type name
        regex ^[^<nlsr><LSA>]*<nlsr><LSA>
      }
      checker
      {
        type customized
        sig-type rsa-sha256
        key-locator
        {
          type name
          hyper-relation
          {
            k-regex ^([^<KEY><nlsr>]*)<nlsr><KEY><>$
            k-expand \\1
            h-relation equal
            ; the last four components in the prefix should
be <lsaType><seqNo><version><segmentNo>
            p-regex
^<localhop>([^<nlsr><LSA>]*)<nlsr><LSA>(<>*)<>
<><><><>$
            p-expand \\1\\2
          }
        }
      }
    }

    rule
    {
      id "NLSR Hierarchy Exception Rule"
      for data
      filter
      {
        type name
        regex
^[^<KEY><%C1.Router>]*<%C1.Router>[^<KEY><
nlsr>]*<KEY><><><>$
      }
      checker
      {
        type customized
        sig-type rsa-sha256
        key-locator
        {
          type name
          hyper-relation
          {
            k-regex
^([^<KEY><%C1.Operator>]*)<%C1.Operator>[^<K
EY>]*<KEY><>$
            k-expand \\1
            h-relation equal
            p-regex
^([^<KEY><%C1.Router>]*)<%C1.Router>[^<KEY>
]*<KEY><><><>$
            p-expand \\1
          }
        }
      }
    }

    rule
    {
      id "NLSR Hierarchical Rule"
      for data
      filter
      {
        type name
        regex ^[^<KEY>]*<KEY><><><>$
      }
      checker
      {
        type hierarchical
        sig-type rsa-sha256
      }
    }

    trust-anchor
    {
      type file
      file-name "root.cert"
    }
  }

  prefix-update-validator
  {
    rule
    {
      id "NLSR ControlCommand Rule"
      for interest
      filter
      {
        type name
        ; /<prefix>/<management-module>/<command-
verb>/<control-parameters>
        ; /<timestamp>/<random-value>/<signed-
interests-components>
        regex ^<localhost><nlsr><prefix-
update>[<advertise><withdraw>]<><><>$
      }
      checker
      {
        type customized
        sig-type rsa-sha256
        key-locator
        {
          type name
          regex
^([^<KEY><%C1.Operator>]*)<%C1.Operator>[^<K
EY>]*<KEY><>$
        }
      }
    }

    rule
    {
      id "NLSR Hierarchy Rule"
      for data
      filter
      {
        type name
        regex ^[^<KEY>]*<KEY><><><>$
      }
      checker
      {
        type hierarchical
        sig-type rsa-sha256
      }
    }

    trust-anchor
    {
      type file
      file-name "site.cert"
    }
  }

  ; cert-to-publish "root.cert"  ; optional, a file
containing the root certificate
                               ; Only the router that is designated
to publish the root cert
                               ; needs to specify this

  ; cert-to-publish "site.cert"  ; optional, a file containing
the site certificate
                               ; Only the router that is designated
to publish the site cert
                               ; needs to specify this

  ; cert-to-publish "operator.cert" ; optional, a file
containing the operator certificate
                                ; Only the router that is
designated to publish the operator
                               ; cert needs to specify this

  cert-to-publish "router.cert"  ; required, a file
containing the router certificate.
}
```

Not expecting you to read these 151 lines!

A New Approach: Versatile Security Toolkit (VerSec)

- A language for expressing the trust rules and a compiler to check the rules that outputs a binary form trust schema
- Run-time security methods, *schemer*, for validation and building packets, also allow applications to reference Name components by names/tags



See: http://pollere.net/Pdfdocs/ICN-WEN-190715.pdf, https://vimeo.com/354013644

Example: VerSec compiler input for NLSR

```
# NLSR schema (from github/named-data/NLSR/docs/SECURITY-
CONFIG.rst)

# site-specific config
net = ndn
site = edu/ucla

# site-independent config

# entities
operator = Operator/<opId>
rtr = Router/<rtrName>

# packet names

# (format from nlsr/src/hello_protocol.cpp)
# 3rd parameter is <net>/<site>/<rtr> prefix but stuck into one
# component so it can't be validated.
hello = <net>/<_nsite>/<_nrtr>/nlsr/INFO/<_rtr>/<_version>

# (format from nlsr/src/lsdb.hpp)
discovery = <_seqNo>
segment = <_seqNo>/<_version>/<_segmentNo>

lsa = localhop/<net>/nlsr/LSA/<site>/<rtr>/<_type>/(<discovery>|
&lt;segment&gt;)

packet = <hello> | <lsa>

# key names

# <_KEY> is a built-in definition of the 4 parameters that terminate
# an NDN key name: KEY/<_keyId>/<_issuerId>/<_version> (see
# http://named-data.net/doc/ndn-cxx/current/specs/certificate-
format.html)
# This info is validated by the key's signature, not the schema

netCert = <net>/<_KEY>
siteCert = <net>/<site>/<_KEY>
opCert = <net>/<site>/<operator>/<_KEY>
rtrCert = <net>/<site>/<rtr>/<_KEY>
nlsrCert = <net>/<site>/<rtr>/nlsr/<_KEY>

# signing chain

packet <= nlsrCert <= rtrCert <= opCert <= siteCert <= netCert
```

- Fifteen lines of code, fifteen lines of comments
- Not unlike the rule specifications

Use of shim and schemer simplified DNMP Client

omitted code to parse input line, set variables

```
try {
    // make a CRshim with this target
    CRshim s(target);
    // builds and publishes command and waits for reply
    s.doCommand(ptype, pargs, processReply);
} catch (const std::exception& e) {
    std::cerr << e.what() << std::endl;
}
```

note the use of component names

```
void processReply(const Reply& pub, CRshim& shim)
{
    const auto& c = pub.getContent();
    if (c.value_size() > 0) {
        std::cout << std::string((const char*)(c.value()), c.value_size()) << "\n";
    }

    // Using the reply timestamps to print cli-to-nod & nod-to-cli times
    std::cout << "Reply " << to_string(++nReply) << " timing (in sec.): "
              << "to NOD=" + to_string(pub.timeDelta("rTimestamp", "cTimestamp"))
              << "  from NOD=" + to_string(pub.timeDelta("rTimestamp")) << std::endl;

    if (--count > 0) {
        // wait then launch another command
        timer = shim.schedule(interval, [&shim]() {
            shim.issueCmd(ptype, pargs, processReply);
        });
        return;
    }
    if (target == "all") {
        // wait for more replies
        timer = shim.schedule(interval, []() { doFinish(); });
        return;
    }
    doFinish();
}
```

doFinish() for this client just exits

see:

github.com/pollere/DNMP

Status / Summary

- Started with NDN's roots: multicast and security
- These critical features need(ed) work
  - trust schema: usability and audits
  - set synchronization communication models
  - multicast strategy on mulitcast networks (*not* replicated unicast)
  - performance and behavior on wire (or over air)
- Co-development of DNMP and bespoke transport
  - NFD patches (more to come)
  - VerSec toolkit takes trust schema design to useful code
  - "bespoke transport" model of collection of functional modules that handle: Data transport, security validation, application class specificity
- Co-development and edge network starting point critical

Opinions

- Despite the prevalence of the hourglass in NDN papers, the “narrow neck” has not been respected
  - things added to the NDN protocol layer
  - no rich library of application-focused set synchronization transports
- NDN is unlikely to replace the Internet anytime soon, if ever, but offers a lot of promise for “edge” applications.
  - The edge is radio but not much work on testing or optimizing this
  - Data muling is a powerful feature, not in NFDv0.6.6-20-g07f2e2f
  - Walk first. **Performance test with applications.** Can’t rely on simulator
- NDN offers the opportunity to get security right. Its architecture allows fine-grained role-based security
  - tools to make use of this are lacking
  - lack methods of *securing* the trust rules - make the schema signable
  - use the schema at run-time (*schemer.hpp*) to access Name components so that changes in Names don’t require changes in application code

Example VerSec compiler output

cmd = { /myhouse/dnmp/nod/local/command/user/<uID>/
<_c_MachineID>/probe/<_pID>/<_pArgs>/<_c_TimeStamp> /myhouse/
dnmp/nod/local/command/operator/<opID>/<_c_MachineID>/probe/
<_pID>/<_pArgs>/

<_c_TimeStamp> /myhouse/dnmp/nod/all/command/operator/<opID>/
<_c_MachineID>/probe/<_pID>/<_pArgs>/<_c_TimeStamp> /myhouse/
dnmp/nod/<_nodID>/command/operator/<opID>/<_c_MachineID>/probe/
<_pID>/<_pArgs>/

<_c_TimeStamp> }
roleCert = {

/myhouse/dnmp/user/<uID>/KEY/_/_/_

/myhouse/dnmp/operator/<opID>/KEY/_/_/_ }

dnmpCert = { /myhouse/dnmp/KEY/_/_/_

}
netCert = {

/myhouse/KEY/_/_/_ }

reply = { /myhouse/dnmp/nod/local/reply/user/<uID>/<_c_MachineID>/
probe/<_pID>/<_pArgs>/<_c_TimeStamp>/nod/

<nodID>/<_r_TimeStamp> /myhouse/dnmp/nod/local/reply/operator/
<opID>/<_c_MachineID>/probe/<_pID>/<_pArgs>/<_c_TimeStamp>/

nod/<nodID>/<_r_TimeStamp> /myhouse/dnmp/nod/all/reply/operator/
<opID>/<_c_MachineID>/probe/<_pID>/<_pArgs>/<_c_TimeStamp>/nod/

<nodID>/<_r_TimeStamp> /myhouse/dnmp/nod/<_nodID>/reply/operator/
<opID>/<_c_MachineID>/probe/<_pID>/<_pArgs>/

<_c_TimeStamp>/nod/<nodID>/<_r_TimeStamp> }
nodCert = {

/myhouse/dnmp/nod/<nodID>/KEY/_/_/_ }

deviceCert = { /myhouse/device/<devID>/KEY/_/_/_

}
configCert = {

/myhouse/config/<confID>/KEY/_/_/_ }

30

netCert = { /myhouse/KEY/_/_/_

}

```
13 unique literals (66 bytes):
 KEY(8) all(2) command(4) config(1) device(1)
dnmp(12) local(4)
myhouse(16) nod(13) operator(7) probe(8) reply(4)
user(3)
5 unique refs (23 bytes):
 confID(1) devID(1) nodID(5) opID(7) uID(3)
6 unique params (46 bytes):
 c_MachineID(8) c_TimeStamp(8) nodID(2) pArgs(8)
pID(8) r_TimeStamp(4)
reference map:
  uID: cmd[6](1) roleCert[3](1)
  opID: cmd[6](2,3,4) roleCert[3](2)
  nodID: nodCert[3] reply[13](1,2,3,4)
  devID: deviceCert[2]
  confID: configCert[2]
validation chains:
  uID in cmd[6](1) validated by roleCert[3](1)
  opID in cmd[6](2,3,4) validated by roleCert[3](2)
  nodID in reply[13](1,2,3,4) validated by
nodCert[3]
```