LHCONE → NSI Adaptation
An Applications' Perspective

Jerry Sobieski
NORDUnet

CERN
Dec. 13/14 2012

Outline

- Purpose:
  - Explain [some of] the "user aspects" of adapting to NSI
- Begin with a brief overview of key NSI features
- Structure of applications and messages within the NSI context
- Some NSI specific details to get started
- A testing/evaluation strategy
- Some points for thought on global service regions…

NSI CS

- NSI is a *consensus* framework for defining a set of globally adopted services/protocols that will [ultimately] allow guaranteed performance services (Connection Services) to be *ubiquitously available, extensible, secure, and fully automated*.
- NSI Connection Service Protocol “NSI-CS” is the first protocol within the NSI framework.
  - A simple set of messages between a “Requesting Agent” and a “Provider Agent” that manage a Connection through its life cycle
- A “Connection”
  - A logical conduit between two endpoints over which user data is carried, unmodified, from ingress to egress.
  - (Note: says nothing about the technology used to realize the service instance.)

NORDUnet

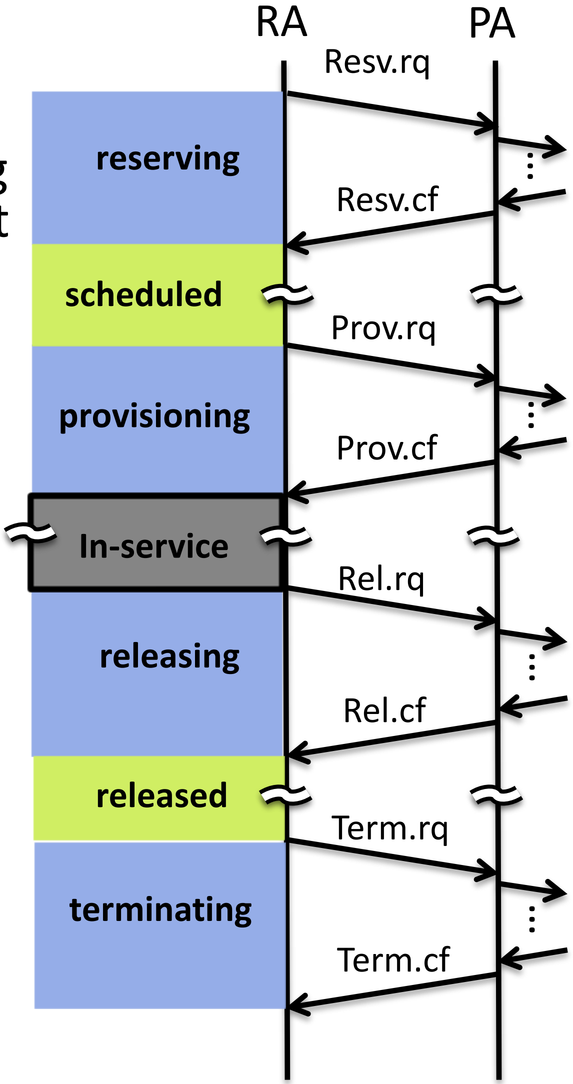
Nordic infrastructure for Research & Education

NSI Service Domains

- NSI works from the top down:
  - The world is made up of network service domains…not hardware infrastrucutre.
- This allows NSI to abstract the capabilities of the infrastructure to provide a well defined (constrained) service to users.
- This also simplifies the user perspective:
  - The user asks for the *service instance* rather than a technology resource.
  - Internally, each network domain decides how to map the service request to the available infrastructure.

NSI CS Protocol

- Each CS primitive is issued as a message from one agent to another
  - [Most] primitives originate with a Requesting Agent (RA) and are sent to the Provider Agent (PA)
  - The PA issues a corresponding response (confirm or fail) back to the RA.
- Each NSA manages a database containing all Connections it is providing or has requested. This database contains the child segmentation associations and lifecycle state(s).
- The CS protocol is designed to provide consistent life cycle state transitions for all NSI connections regardless of how they are segmented



NSI CS

- Other Messages:
  - Query() – provides information regarding a connection
    - Basic query provide basic summarized life cycle state
    - Detailed query “walks the tree” to collect and expose as much information as possible (e.g. path) regarding a connection.
  - Notify() – sends a message from the PA to the RA informing the RA of an unsolicited change in state of a segment (e.g. faults)
  - Modify() – changes performance parameters associated with a connection (optional).

NSI CS Security

- Since each NSI CS primitive crosses domain boundaries, each domain will need to insure that NSI services requests are "safe"
  - All information exchanged is subject to classification
  - AAI credentials in each primitive allow domains to maintain security and privacy in a multi-domain [global] service environment
  - NSI has this security built-in to each primitive
- NSI CS session between NSAs is authenticated and authorized
  - Allows each domain to qualitfy which remote agents it will serve…
  - And which neighbors it will trust
- Each NSI-CS primitive operation contains credentials of the entity that wishes to perform the operation.
  - These credentials are also authenticated and authorized before the operations is granted

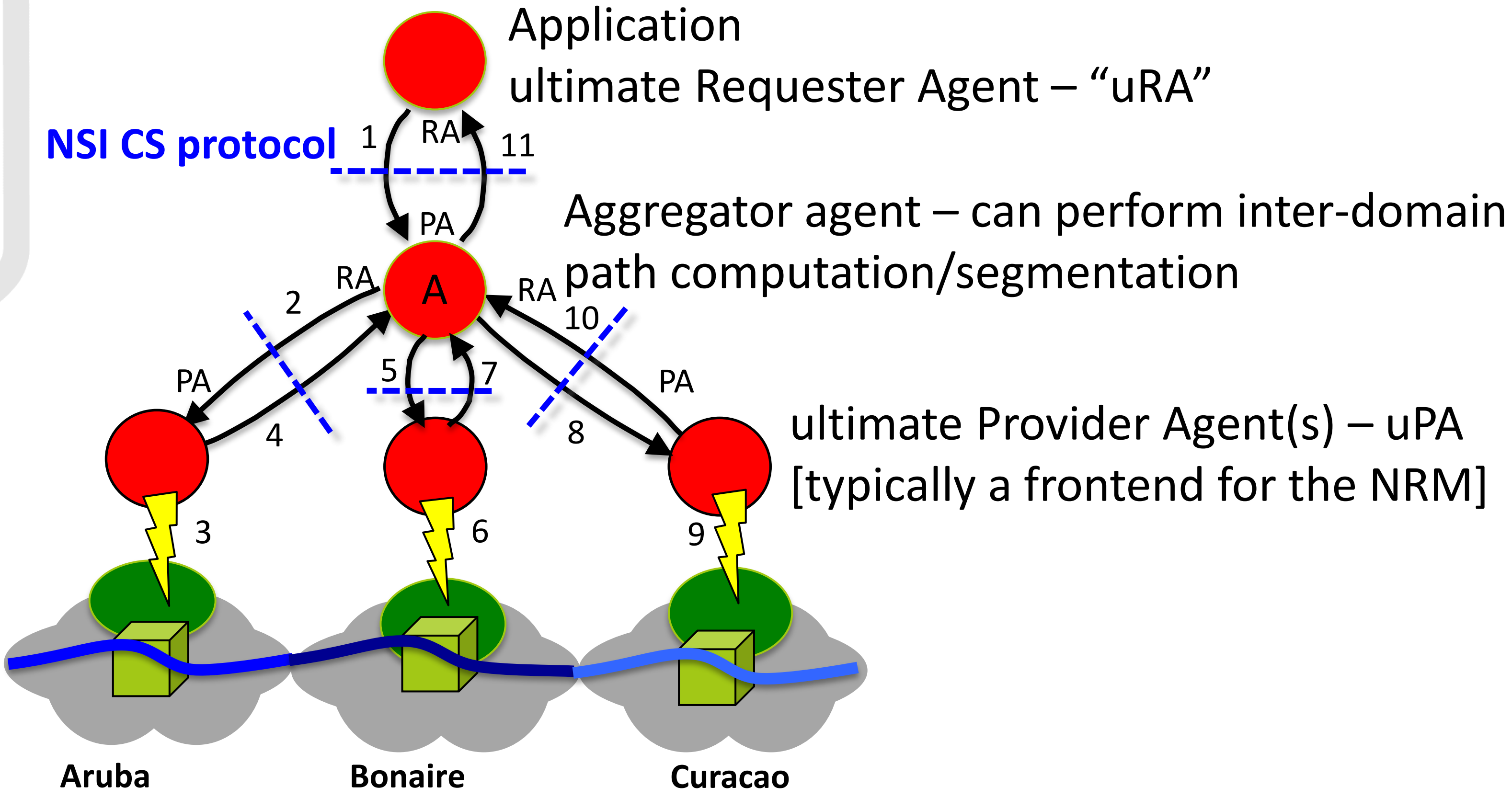
The NSI-CS Service Tree

- A Connection request is “segmented” by decomposing it into a sequence of pieces
  - The pieces (segments) are defined as part of the path finding process.
  - Each segments typically correspond to one or more transit domains along a selected path
  - These segments are concatenated to form the end-to-end Connection
- Each segment of a request is referred to as a *child* of that request
- As each request is segmented into children, and connection requests are recursively issued to other NSAs for those children, a “**service tree**” is constructed spanning all NSAs (network domains) involved in the end-to-end path.

NSI CS Service Tree

- “Tree” Processing – the Aggregator segments a request and issues child requests directly to each NSA long the desired path

  (The uPA could do the segmentation itself instead of asking an intermediate aggregator NSA to do so as in this example.)



Application
ultimate Requester Agent – “uRA”

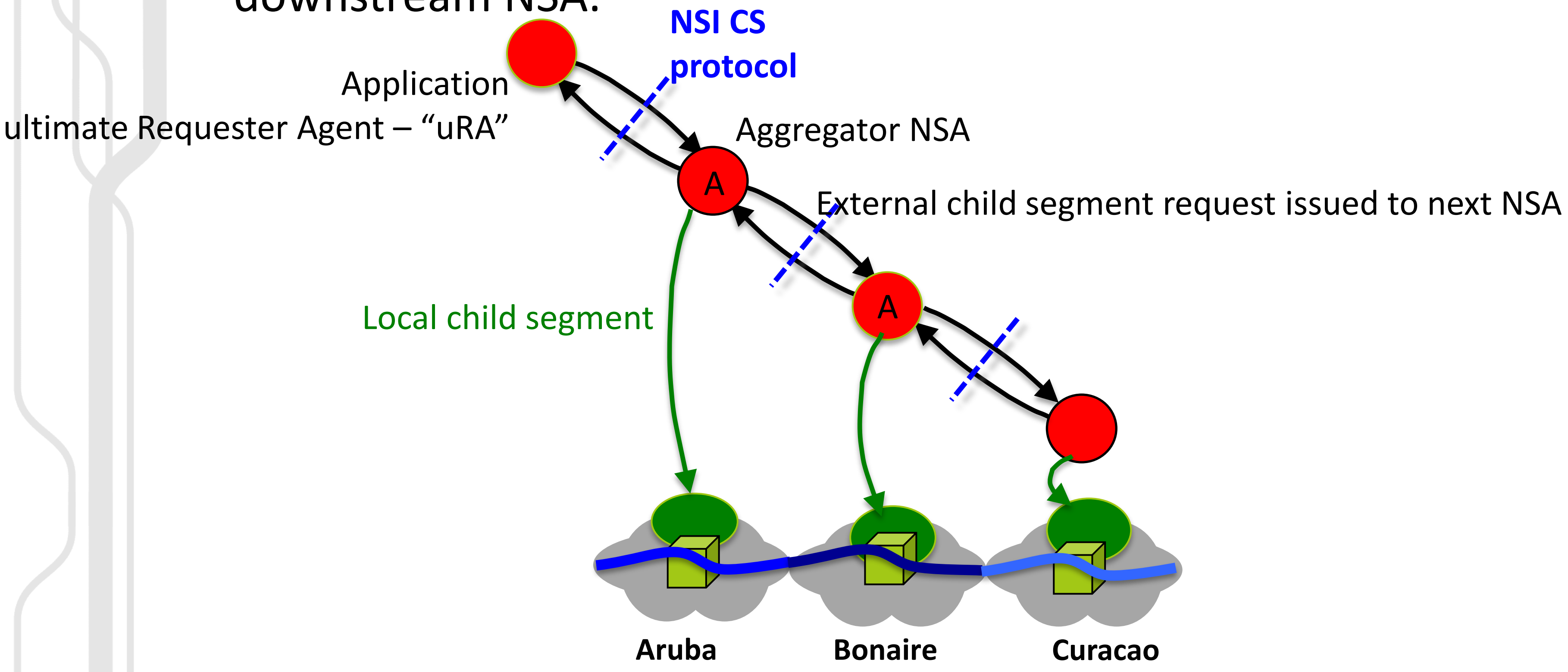
NSI CS protocol

Aggregator agent – can perform inter-domain path computation/segmentation

ultimate Provider Agent(s) – uPA
[typically a frontend for the NRM]

Options: Tree and Chain

- Conventional hop-by-hop resource reservation - "Chain" processing - is a degenerate case of Tree processing where the tree consists of a single branch
- Each PA segments the request into a local segment and an external segment.
- The PA passes the local segment to the local NRM, …
- …And passes the external segment to an appropriate adjacent downstream NSA.


NSI CS protocol
Application
ultimate Requester Agent – "uRA"
Aggregator NSA
A
External child segment request issued to next NSA
A
Local child segment
Aruba
Bonaire
Curacao

Path Selection Control

- A Requesting Agent can issue an end-to-end request to a single Aggregator NSA and expect a conformant Connection to be established…or
- The RA may assert path preferences itself via several methods:
  - The inter-domain path may be constructed by issuing segment requests to the desired NSAs/Networks directly ("tree" method)
  - Intra-domain paths can be specified via use of Explicit Route Objects within a segment request.
  - Use of type-value pairs within a request to define/constrain end point or transit STPs
- Each PA is able to assert their own path selection criteria when the Request does not otherwise explicitly specify transit criteria.
  - A PA may perform local internal path selection and pass the entire external path determination to a downstream NSA. ("chain" method)

NSI CS agents

- An “ultimate Requesting Agent” (uRA)
  - the agent that originated the Connection request
  - issues CS primitive request - but does not accept primitive requests
  - ***Applications are typically uRAs***
- A “ultimate Provider Agent” (uPA)
  - the leaf provider agent – the agent that interfaces to the NRM in a particular domain.
- An “aggregator agent”
  - An Aggregator agent is capable of inter-domain topology analysis and path selection (it can process and understands global [N3] topology)
  - Aggregators do path segmentation and “aggregate” the state of children Connection segments to present a single view back up to the RA.
- A “provider only” agent
  - Accepts service requests, but does not issue service requests
  - Does not (cannot) do inter-domain path finding - can only service requests that are completely in the local domain.
  - uPAs are typically local NRMs with a simple NSI-protocol-to-local-command translational interface.

User vs Network

- NSI does not distinguish between "users" and "networks"
  - NSI only recognizes "requester agents" and "provider agents" in the Connection Service protocol exchange
- User applications requester agents have access to as much network information and control as any other NSI agent
  - Access to this information and control is authorized and strictly enforced according to local policy…
  - …But policy in these early days and among the R&E community is intentionally very loose

NSI CS only builds “Pipes”

- NSI CS only constructs point to point connections.
  - Connections are conduits that carry user information from ingress STP to egress STP
  - Connections do not process the user datagrams - that is a user function at the end points.
  - NSI connections provide predictability and reliability - i.e. guaranteed performance, scheduled reservations – across the transport network(s) between the user specified locations (STPs)
- It is up to the “user” to construct higher layer functionality from these atomic “pipes”
  - E.g. NSI does not build “VLANs” even though the STP access framing may utilize 802.1Q to define user payloads.
  - E.g. NSI does not configure end systems, though end system processes can be developed that participate in the NSI CS provisioning
    - in order to be notified when such configuration might be necessary.

Application Development

- “Users” need to be aware of:
  - End Points (STPs)
  - Services Definition(s) and their parameters
  - Serving NSAs or entire topology
  - Authorization credentials
  - Topology – simple or sophisticated
- Many of these aspects can be handled transparently for the end user/application through the use of [web based] GUIs or workflow middleware

Application Development

- Training-
  - The NSI WG should offer applications development training to interested users and code developers
  - This is not a network engineering class, but a user class for how to access and manage NSI Connections
  - Perhaps we can initiate such training in Spring 2013?
    - 1 Or 2 day workshops where code jockeys are able taught the ins and outs of NSI primitives

Stage 1 Development

- Step 1: Down load the NSI specifications document from OGF
  - www….
- Step 2: Download the WSDL
  - Understand how the NSI CS messaging is built and encapsulated.
- Step 3: Set up a test environment
  - Option A: Build your own:
    - Choose one of the existing NSI provisioning systems and install it in a lab.
  - Option B: Use the GLIF Automated GOLE Facility
    - Test against *any* implementation in the AGF
    - Realistic environment, lots of experts,…

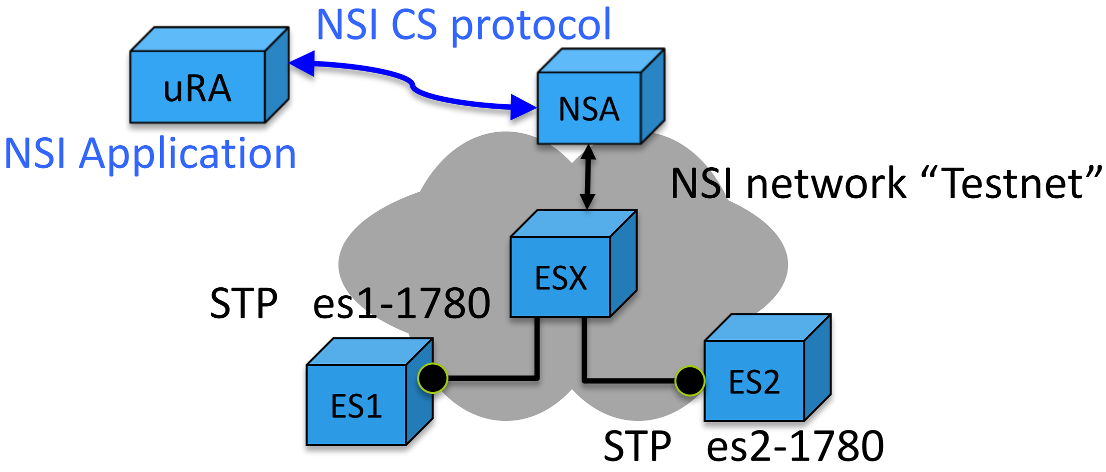
NSI Implementations

- Network Service Agent (NSA) packages:
  - **OSCARS**: http://code.google.com/p/oscars-idc
    - Developed by Esnet, deployed by Esnet, Internet2, other networks and on several GOLES,
  - **AutoBAHN**: ffi: Radek Krzywania <radek.krzywania@man.poznan.pl>
    - Developed by GEANT Consortium, deployed within/across GEANT
  - **OpenNSA**: http://git.nordu.net/?p=opennsa.git;a=summary
    - Developed by NORDUnet, deployed (experimentally) in NORDUnet, UvA, StarLight, GLORIAD, others
  - **OpenDRAC**: ffi: John MacAuley <john.macauley@surfnet.nl>
    - Devloped by NORTEL/Ciena and now SURFnet, deployed in NetherLight
  - **G-LAMBDA-A** ffi: Tomohiro Kudoh <t.kudoh@aist.go.jp>
    - Developed by AIST in Japan contact: Tomohiro Kudoh
  - **G-LAMBDA-K** ffi: Takahiro Miyamoto <tk-miyamoto@kddilabs.jp>
    - Developed by KDDI Labs, deployed experimentally in JGNX
  - **DynamicKL**: ffi: Jeonghoon Moon <otello90@gmail.com>
    - Developed by KISTI (Daejeon, KR) deployed in KreoLight GOLE (exp)

- All of these have at least one node in the Automated GOLE
  - So you can test against any/all of them

Setting Up a NSI Test Facility

- Five components:
  - A server to run your NSA
  - A [simple] switch that is recognized by your NSA
  - Two end systems attached to the switch
  - A system that can act as a platform for the uRA
- Some of these components can be staged on a single host
  - The end systems are only needed to verify the connection is functional
  - The uRA can be placed on an end system
  - The switch can initially be a DUD (fake switch) while you refine the NSI CS protocol interactions (which means end systems are unneccesary)



Psuedo Code

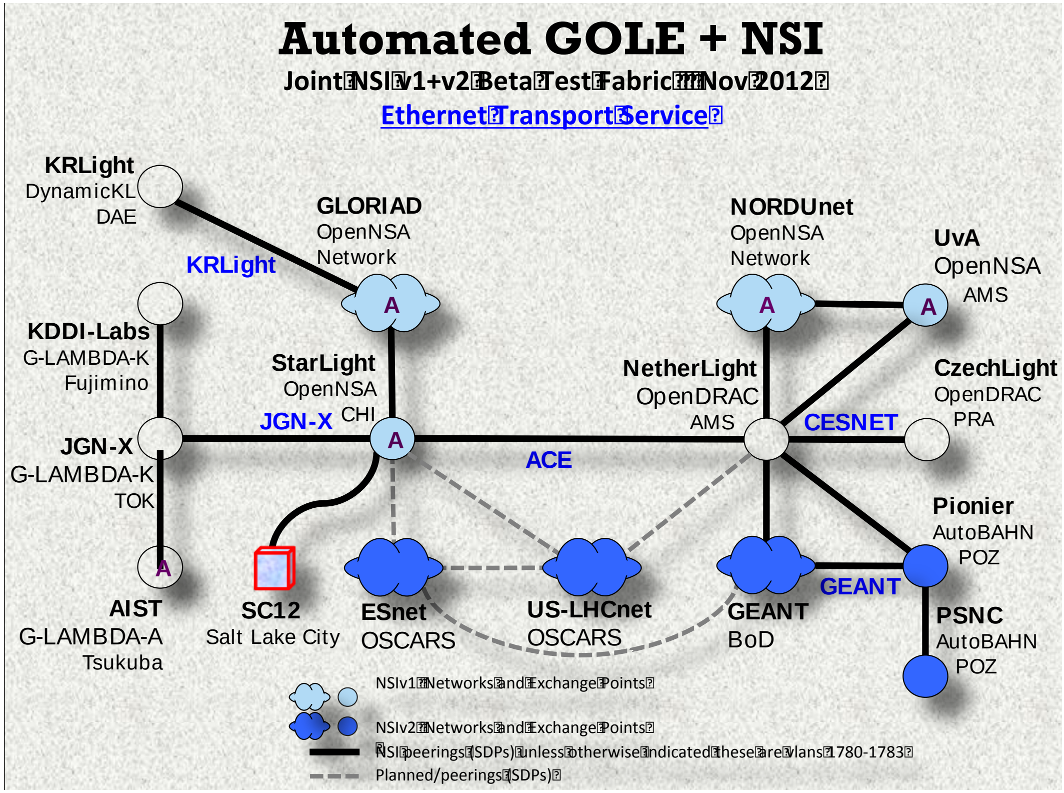
- Step 4: Write a small code to walk a connection through the life cycle

```
– Send: Reserve(destNSA = testnet,
   sourceSTP = testnet:es1-1780,
  destSTP = testnet:es2-1780,
   startTime = <now+2min>, endTime = <now+3min>,
   auth = jv@internet2.edu )
– Recv: Reserve.cf (connectID=foo)
– Send Provision(foo)
– Recv: Prov.cf(foo)
– From ES1:  ifconfig eth0.1780 ipaddr=10.1.1.1/24
– From ES2: ifconfig eth0.1780 ipaddr=10.1.1.2/24
– From ES1: ping 10.1.1.2
– Send: Query(foo, detailed)
– Recv: Query.resp(foo, info);  Print info;
– Send: Release(foo)
– Send: Term(foo)
```

- Note: most of these tags are actually URNs

Stage 2: Field Testing

- Now migrate your small test code to the Automated GOLE Facility:



- Use topology data from the AGF topo repository at: https://github.com/jeroenh/AutoGOLE-Topologies/tree/nsiv2

Refine your connection code:

- Step 5: More sophisticated multi-domain provisioning (using Automated GOLE Facility)
  - 5.1 same sequence as step 4, but allow an aggregator NSA to build service tree, and look at Query() details…

  -
    ```
    Send: Reserve(destNSA = Aist.ets,
       sourceSTP = NetherLight.ets:ps-1780,
       destSTP = NorthernLight.ets:ps-1780,
       startTime = <today@17:00:00>,
       endTime = <today@17:02:00>,
       auth = jv@internet2.edu )
    ```
  - 5.2 Do the segmentation in the application uRA:

  -
    ```
    Send: Reserve(destNSA = Netherlight.ets,
       sourceSTP = NetherLight.ets:ps-1780,
       destSTP = netherlight.ets:cph-1780,
       startTime = <today@17:00:00>,
       endTime = <today@17:02:00>,
       auth = jv@internet2.edu )
    Send: Reserve(destNSA = NorthernLight.ets,
       sourceSTP = NorthernLight.ets:ams-1780,
       destSTP = NorthernLight.ets:ps-1780,
       startTime = <today@17:00:00>,
       endTime = <today@17:02:00>,
       auth = jv@internet2.edu )
    ```

Command Line Interface

- In addition to the Web Services API, some NSI packages also offer command line tools that can be used for scripting
  - CL tools can be useful for establishing connections, and immediately configuring the local interface on a server
    - E.g. setting the VLANID sub-int, assigning IP addressing, setting MTU, etc.
  - Check the NSI package you elect to use (or might use) to see if there are CL tools.
    - These can often aid in debugging as well – e.g. Query() a connection you think your code reserved to see its state.

Stage 3: Service Deployment

- Having mastered the programmatic interface and having developed code that can leverage NSI…
- Now we need to coordinate with the network engineering team(s) to field LHCONE services…
  - define the end point STPs we want/need [for the app.]
  - to build and/or place the topology file(s) for the [LHCONE] environment
  - Identify the NSAs and their WS end points
    - need to know which are Aggregators and which are Provider-Only agents.
  - The [LHCONE] networks need to agree to a common service – and the respective networks need to implement it as a NSI service.

Deployment:

- **Step 1:** Applications developers and network engineers must identify the end points that are to be part of the NSI CS landscape
  - These end points will be named as STPs.
  - These STPs can be advertised via the local domain topology description
- **Step 2**: Network Engineering function should be responsible for constructing topology files.
  - These files will contain the STPs for that domain
  - And the service type and parameters that will be honored.
  - Adjacency information to neighboring NSI domains
  - And any other topology information the domain is willing to announce globally and publically.
- Topo files are placed in a well known URL.
  - (Currently this is the github topology repository.)

Deployment (continued 2)

- **Step 3**: Common Service Definition
  - Agree to the service capabilities that the NSI service(s) will present to users
  - Technical parameters that will recognized within the NSI-CS protocol
  - Other technical service objectives (operations support, monitoring, …)
- **Step 4:** Develop and agree among providers to a security profile for the Connection Services
  - Roles, Information, Operations
  - Users

Deployment (continued 3)

- **Step 5:** The service domains must deploy their respective NSAs (their NSI services)
  - The aggregators will be responsible for path selection and the resulting reservation service tree.
  - Alternatively, the application can determine a workable inter-domain path
    - Then the application can reserve each segment itself.
    - This requires knowledge of the global topology – either learned out of band or by processing the publ global topology files.

Pointers

- The NSI-CS WSDL can be found at:
  - https://code.google.com/p/ogf-nsi-project/source/browse/
- OpenNSA basics can be found at:
  - https://github.com/jeroenh/OpenNSA/wiki

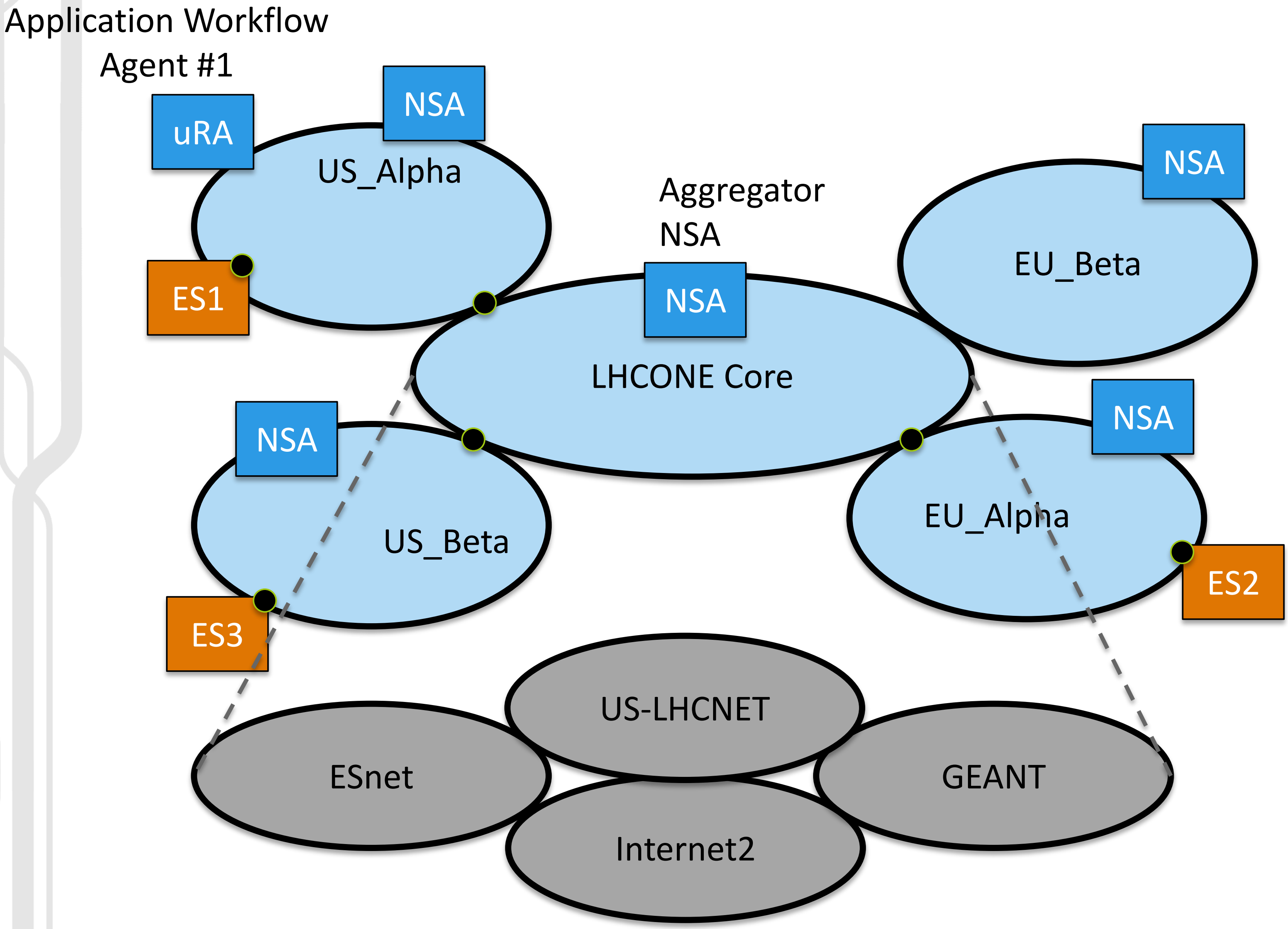
Contiguous Service Regions

- A number of options for multi-domain network organization can be envisioned:
  - A single LHCONE Core NSI CS, with adjacent campus domains
    - Each domain has a Provider Only NSA
    - The LHCONE core NSA acts as aggregator
    - End to end requests are forward directly to core NSA for end-to-end path selection and segmentation…
    - Or each application performs their own tree segmentation
    - Requires a shared authorization space
  - Else… Each domain hosts their own aggregator NSA

NORDUnet

Nordic infrastructure for Research & Education

LHCONE Dedicated Core

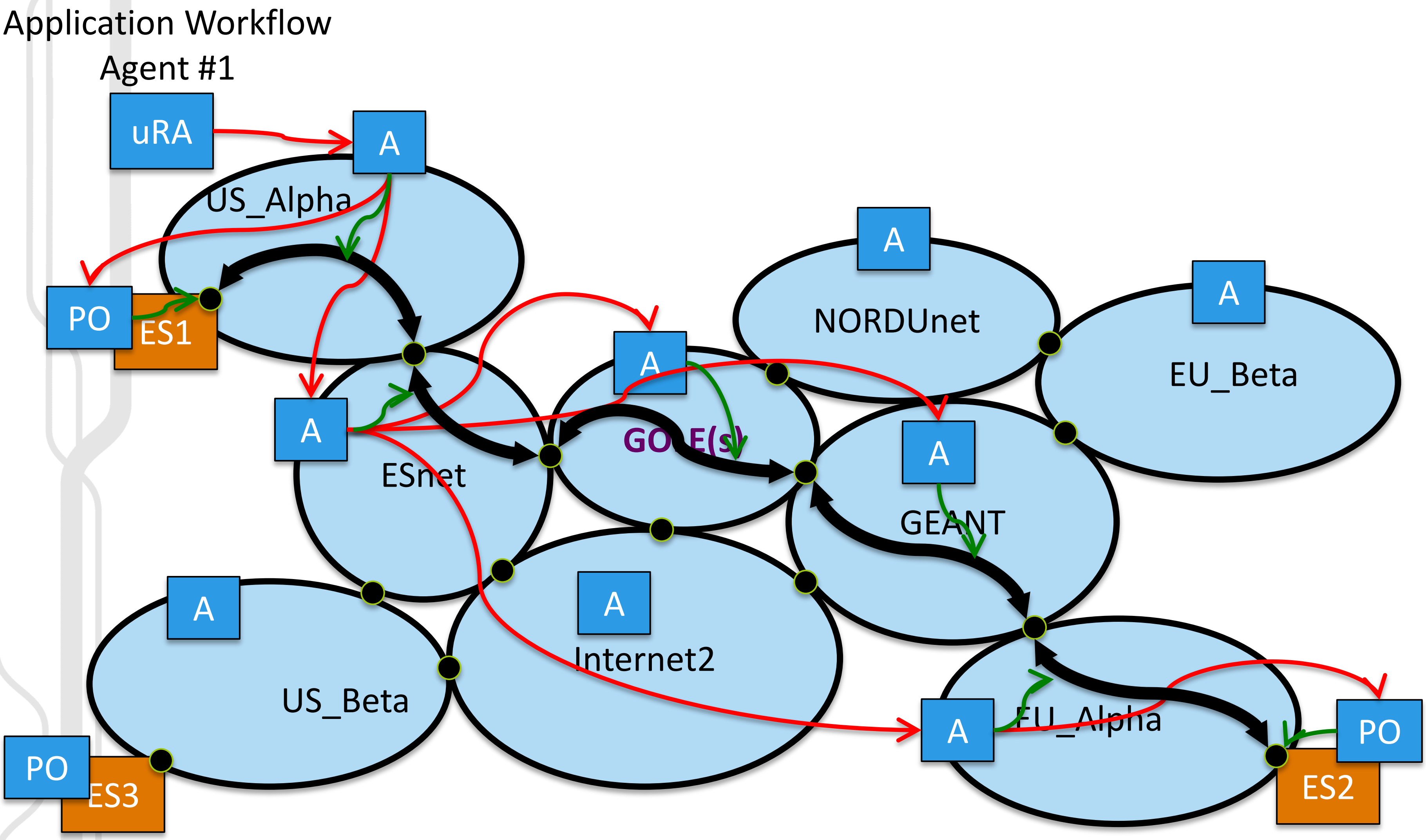


LHCONE Predicated Core

- LHCONE service requests are predicated across other networks
  - Each transit infrastructure deploys and LHCONE compatible service (“P2PCS” ?)
  - Provides a more dynamic multi-domain path capability
  - Does not require an LHCONE specific operational capability
  - Still allows LHCONE to segregate its services and policy

LHCONE Predicated Core

A rich federation of interoperable P2PCS Service Domains



Full Disclosure

- **NSI as a standard and framework is still evolving…and should be expected to continue to do so for next several years**
  - **NSI CS version 2** is being drafted now, and demo'd..
  - We should plan strategically with a long view
  - Incremental and iterative improvement
- NSI v3 requirements/objectives gathering will commence in spring 2013…ETA 2014Q2(?)
  - Dynamic Topology Distribution Service (TS)
  - P2MP
  - Improved message distribution (NSA-to-NSA comms, FW/NATs,..)
  - Improved state processing (SM exploded with Modify())
- Items missing yet to be developed:
  - Better path planning implementations (aggregators, path finders, topology analysis)
  - Better operational control tools (NOC tools)
  - Automated connection performance verification techniques
  - Automated fault processing
  - Logging and Accounting
- Training and educational workshops (!)
- Security/AA policy – *Security and privacy guards must be integral*
  - But we can relax these policies as we ramp up and gain experience

Forward !

- Thanks for listening !
  - Please let us know what we need in v3…

Jerry Sobieski
jerry (at) nordu.net
jerry.sobieski (at) Skype

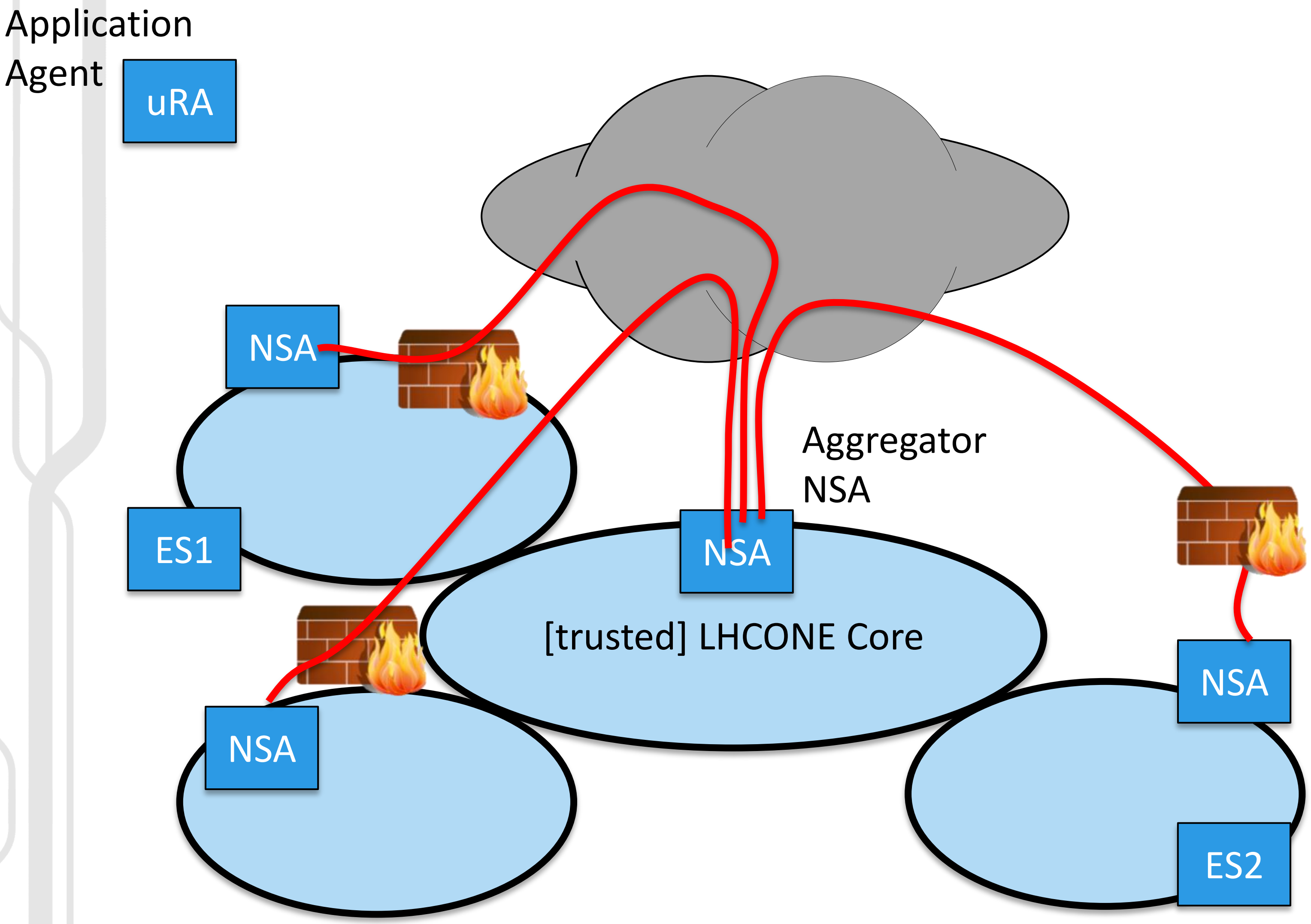
Firewalls and NATs

- Many campus network domains have Firewalls at the edge – preventing external agents from opening TCP sessions to campus based agents behind the firewall
- Likewise, many domains use private non-routable IP addresses within their domain and Network Address Translation (NAT) at their edge
- Both/either of these technologies can prevent NSIv2 from working properly in the control plane layer
  - Most simple request/response messages work.
  - Some unsolicited messaging or long delayed responses – like ProvisionConfirm() or Notify() – are unable to penetrate FWs or NATs without assistance from the network engineers.
  - (This is a topic high on the list for version 3 to make NSI more robust in the face of these types of security/engineering measures
- The NSI protocol expects agents to be accessible in order to exchange messaeges transparently.
- Assigning routable IP addresses and open TCP ports for NSI agents allows NSI to function though FWs, but requires additional planning with campus engineers
  - LHCONE may be able to ameliorate this issue by placing agents in address space that is “trusted” by the LHCONE participatns.
  - This LHCONE Aggregator would provide interaction with networks beyond the LHCONE community – without requiring campus FW holes.

NORDUnet

Nordic infrastructure for Research & Education

NSI with FWs and NATs


Application Agent
uRA
NSA
Aggregator NSA
ES1
NSA
[trusted] LHCONE Core
NSA
NSA
ES2