Bas Oudenaarde, UvA
Madhav SBSS, Nortel
Inder Monga, Nortel
27-05-2005

Grid Network Services: Lessons and proposed solutions from Super Computing 2004 demonstration.

Status of This Memo

This memo provides information to the Grid community. It does not define any standards or technical recommendations. The purpose of the draft is to share our experiences with multi-domain control plane inter-working. We present here key issues identified in multi-domain control plane inter-working, possible solutions and demonstrate an implementation of the system as showcased in Super Computing 2004. Distribution is unlimited. This document is considered living.

Abstract

Provisioning network resources on demand is a key enabler of Grid Computing. There has been tremendous amount of research and development in the Grid community in terms of grid services infrastructure and grid application development. However, there has been very little work done in the area of using network as a first-class grid resource. There is no existing implementation today that can demonstrate the power of exploiting network as a first-class grid resource and the challenges that arise in provisioning end-to-end network connectivity across different control plane technologies spanning multiple administrative domains. In this draft we present a demonstration done in Super Computing 2004 which is the first of its kind, bring out the concept of a service plane and address some of the key technical challenges that we encountered in putting together this demo. In addition, we propose some models to kick off the discussion on issues in inter-working among multiple administrative domains to enable on-demand end-to-end network services.

oudenaar@science.uva.nl, msrimadh@nortel.com, imonga@nortel.com

Bas Oudenaarde, UvA
Madhav SBSS, Nortel
Inder Monga, Nortel
27-05-2005

Contents

Abstract 1
Glossary 3
1. Introduction 4
2. Scope and Background 4
3. Super Computing 2004 Multi-domain Control Challenge 5
3.1 SC04 Demo Summary 5
3.2 Generic Authorization Authentication Accounting 6
3.3 Dynamic Resource Allocation Controller 7
3.4 SC04 Network Services Model 8
3.5 AAA-DRAC Operational Models 9
4. Key Challenges in Multi-domain Service Plane 9
4.1 Abstracting control plane technologies 9
4.2 Resource reservation 9
4.3 Rollback mechanism in resource reservation 10
4.4 Deadlocks and Starvation in Resource Reservation 10
4.5 Network Failure, Restoration and Accountability 10
4.6 Policy Framework - Trust, Roles and Privacy 11
4.7 Generalizing AAA and DRAC 12
4.8 Source Based Routing vs Traditional Routing? 12
4.9 Other Open Issues 12
5. Inter-domain Exchange Point Model vs Peer-to-Peer Model 12
6. Model 2: complex requests 13
7. Security Considerations 13
8. Revision Information 13
Author Information 14
Intellectual Property Statement 14
Full Copyright Notice 14
References 14

Bas Oudenaarde, UvA
Madhav SBSS, Nortel
Inder Monga, Nortel
27-05-2005

Glossary

Term	Description
AAA	Authentication, Authorization and Accounting
Control Plane	Relatively static control mechanisms in the network such as SNMP, ASTN, TL1
DRAC	Dynamic Resource Allocation Controller
Service Plane	Enabler of network services to applications that go much beyond vanilla connections, i.e. high bandwidth low latency connectivity
WSRF	Web Services Resource Framework
Home Organization	The domain where the first application request for network services is made

Bas Oudenaarde, UvA
Madhav SBSS, Nortel
Inder Monga, Nortel
27-05-2005

1. Introduction

A controlled autonomous networking system is likely to have clear interfaces for different control plane entities at each domain. However, to setup end-to-end connections dynamically, we need a common terminology for accessing network services. We see a clear need to abstract the differences in control plane technologies into a programmable interface and introduce the notion of a service plane that can support dynamic network services in a heterogeneous control plane environment. We recognize two important parts of controlled autonomous networking system, e.g. AAA plane and network services plane. Throughout this document we focus on them as complementary services. We will introduce the concept of "service plane" and its interaction with the underlying control plane technologies such as ASTN, SNMP, TL1 and UCLP.

A "service plane" as we view it encompasses two key elements, namely, Authentication Authorization and Accounting and Network Services Middleware. UvA's AAA server and Nortel's Dynamic Resource Allocation Controller (DRAC) are instantiations of the service plane elements. Together these two enable applications to request on-demand network services such as low latency, high bandwidth connection, network knowledge services and third party services. AAA is used to manage trust between the different administrative domains. In the home organization the application or user is authenticated. The home AAA server authorizes the application to use the inter domain control facility. DRAC addresses the challenge of adapting the underlying network to the applications needs. In other words, making today's networks whose underlying transport technologies include, but not limited to, wireless, packet and optical and control technologies include SNMP, TL1, UNI, ODIN, UCLP and so on, adapt to the end user applications by offering them customized dynamic end-to-end network .services. Network service requests may be negotiated out of band over the regular Internet, while the end-to-end data connections are established in a secure private network.

Global Lambda Integrated Facility (GLIF), which is in process of building a flat fee optical network spanning over the globe, is garnering ever growing participation from several institutions. GLIF plans to provide these institutions with a flexible Grid Network infrastructure serving their high-bandwidth, dynamic Grid applications that are not being served well by the current network. We propose adoption of grid application-driven service plane concepts in the GLIF context.

Throughout this document we only explore the (Grid) resources required for enabling end-to-end network services, we will not make any attempt to discuss controlling the Grid Resource attached to those endpoints i.e CPU, Storage.

2. Scope and Background

Grids services require among other aspects, high bandwidth network connections across multiple administrative and network technology domains. For example, an end-to-end connection from Grid application source to destination may require setting up layer 3, layer 2, layer 1 and layer 0 network elements along the end-to-end path as we will see in the section where we discuss our Super Computing 2004 demo. Among network level services spanning multiple layers, we recognize a trend to light up your own dark fibers or join a lambda cloud and create a kind of Internet bypass. Furthermore, we discuss issues that arise in enabling resource virtualization, topology discovery and other network services as mentioned earlier. For example, resource virtualization provides opportunity to choose the "right" layer to do the data transport without the application knowing the nitty-gritty details of the transport plane. These kinds of mechanisms are enabling applications to engage the network to meet their needs in contrast to the applications themselves adapting to the way network behaves.

We have taken these service plane concepts and developed a prototype of the network and the required software modules. This prototype was demonstrated in Super Computing 2004, which is discussed further in the following section.

3. Super Computing 2004 Multi-domain Control Challenge

The demonstration scenario consisted of a grid application in Amsterdam choosing to transfer data using Grid-FTP from a remote server in Chicago and using Grid middleware to allocate network resources to meet the application requirements. This translated into a multi-administrative domain setup and adaptation of the network path to meet the requirements of the grid application.

The challenges exposed consist of leveraging existing control planes developed by institutes or companies which operate at different layers of the OSI stack. Each domain pushes their control plane, which means in multi-domain scenarios different control planes need to be glued together to set up the end–to-end connection. This glue (inter-working) depends on the differences in the operational layer of the OSI stack, or collapsed layers. It makes life easier if there is consensus about the network primitives, control features, addressing and routing information and a common language to communicate. We will first describe the Multi-domain Control Challenge model demonstrated with Nortel's DRAC and University of Amsterdam's (UvA) AAA toolkit at Super Computing 2004 and then use it as a reference point to discuss our service plane architecture.

3.1 SC04 Demo Summary

In our demonstration in Super Computing 2004 we put together a multi administrative domain setup as shown in Figure 1. The demo trail is as follows –

1. A Grid-FTP application requests a large chunk data from a server in Chicago with the highest Quality of Service from the network i.e. high bandwidth, low latecy, dedicated.

2. This request is forwarded to its local (home) AAA server. AAA authenticates the user-application and determines if this user is authorized to use the inter domain control facility. Once the user request has been validated, the AAA server forwards the request to DRAC requesting network service.

3. DRAC constructs a source-routed, end-to-end path and determines the domains that constitute that path. It constructs the path within its domain of authority and then requests AAA to help with constructing the rest of the path by passing this request to the neighboring domain.

4. The home AAA server forwards the user request to the AAA server of the next source routed domain. The user credentials are authenticated and authorization verified by the peer AAA server before the DRAC entity of that domain is requested to setup its portion of the path.

5. This process continues until the full end-to-end path is setup or the request is rejected due to any of the possible reasons. Both these scenarios were demonstrated in SC04. The possible failure scenarios and alternatives are discussed in the sections later in the document.

6. The path was then brought down manually at the Startlight domain to simulate a network path failure. The DRAC entity learnt about the path failure and initiated a inter-domain restore of the path through AAA. The new path consisted of path changes only in the

Netherlight and Starlight domain while the rest of the path remained as before. The Grid-FTP application recovered from the network glitch and continued its transfer.

For this demonstration, the peer network domains were also controlled by AAA and DRAC entities. DRAC dealt with a different control plane technology in each domain. For example, in OMNInet, DRAC interconnects with a network management technology named ODIN. ODIN controls and drives the state of the network resources in OMNInet. In a similar manner, DRAC can interface with several different control technologies such as User Controlled Lightpath (UCLP), TL1, SNMP and O-UNI. This is an example of how DRAC service plane model can support varied control plane technologies underneath it thereby shielding applications from this complexity in networking technologies. The complete inter-working is done in DRAC; individual domains will not notice any difference in operating and controlling the available network resources.

The source based routing approach was chosen to construct an end-to-end path, between two application end nodes. Our experiences with this approach and the Agent control model between DRAC and AAA are discussed in later sections.

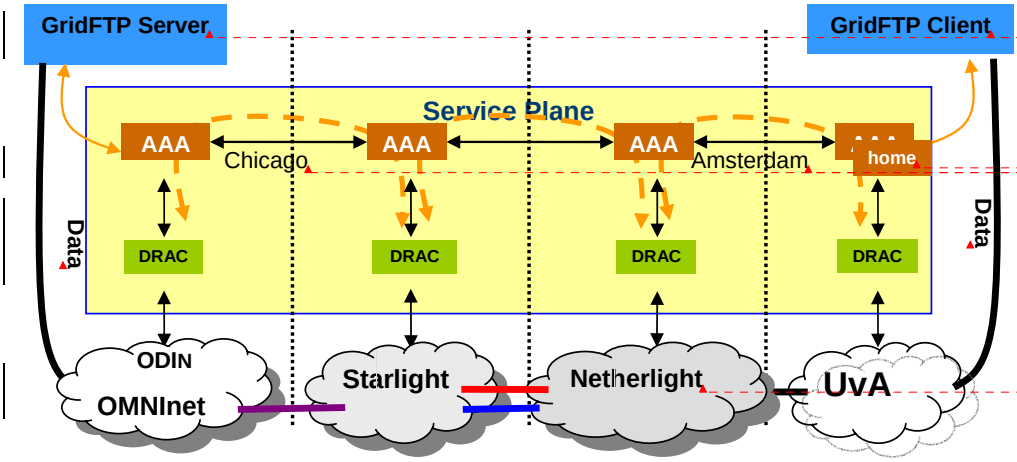

Figure 1 SC04 Demo Network Setup

For communication between the service plane and Grid application we consider XML-based protocol. We modeled network services offered by the service plane into XML-Schemas. The inter-working functionality is developed as a proxy XML object, which could do the translation and deliver functions between the different control software.

In next sections we discuss AAA and DRAC entities and present possible multi-domain path setup models and AAA-DRAC operational models.

3.2 Generic Authorization Authentication Accounting

AAA is used to manage trust between the different administrative domains. In the home organization the application (user) is authenticated. The home AAA server authorizes the application to use the inter domain control facility. The direct authorization cannot be given, since the AAA entities end-to-end are not directly connected to the application when the actions are executed. Specially the resources are not known and certainly not owned by one administrative owner. We modeled the trust mechanism in a peer-model, and applied non-transitive trust. Each

Formatted: Font: 10 pt, Complex Script Font: 10 pt
Formatted: Font: 10 pt, Complex Script Font: 10 pt
Formatted: Font: 10 pt, Complex Script Font: 10 pt
Formatted: Font: 10 pt, Complex Script Font: 10 pt
Formatted: Font: 8 pt, Complex Script Font: 8 pt
Formatted: Font: Bold, Complex Script Font: Bold
Formatted: Font: Bold, Complex Script Font: Bold
Formatted: Font: 11 pt, Complex Script Font: 11 pt

AAA entity has double role, one it is an authority to authorize access to the particular domain, and as requester to ask permission in another administrative domain. This also applies for home AAA server, it determines the authorization, and ask in name of the particular organization to make use of the inter domain control facility node in another administrative domain. The application (user) is not visible during the call setup (privacy-concern).

The peering model should work, since there will be some neighbor domain trust. Otherwise two peer domains are not connected via (logical) link, so we could assume that neighbor do trust each other. This level of trust is not transitive, so if A and B and B and C trust each other it doesn't imply trust between A and C. Each peer will make a call to his neighbor in name of his own organization. In this way we are not restricted to have trust between each admin domain in the chain of the end to end path, which might be hard to arrange. On the other end, we could still have malicious transit domain, which connect two parties who would never authorize the setup if they would have known if a particular party is in the loop. One way to avoid these issues is to keep record of the passed administrative domains and pass the list thru (signed way). At least the home organization might be passed thru to make accounting possible and give the application feedback on the cost of the path. This also opens the way for mutual authorization; the application can decide to use the end to end path.

3.3 Dynamic Resource Allocation Controller

DRAC adapts a decentralized architecture; each network domain may have one or more instances of DRAC middleware running. In case of multiple instances in a domain, a master instance is elected. This DRAC instance manages the domain and the inter-domain connectivity through peer messaging. DRAC core framework includes services, policy engine, topology discovery engine, workflow utilities, inter-domain routing facilities and smart bandwidth management fixtures. The interface to applications is bi-directional, enabling network performance and availability information to be abstracted upward toward the application. DRAC provides applications the means to directly drive their share of network resources within a policy-controlled envelope of flexibility. Network resources include bandwidth, quality of service (QoS), security, acceleration, appliances, sensors and more [].

DRAC provides an API based coupling with applications. Applications can request network services through this API. e.g. "cut-through" (high bandwidth, low latency) service provides applications to bypass layer 3 and directly transfer data over layer 1 connections, this kind of functionality is especially valuable for high data transfer applications in research networks.

As an example we list here how an application can initiate a path setup request from DRAC. This is not the actual DRAC API but only a representation of it, the complete DRAC API may be published at a later date.

```
PathSetupRequest(MsgHdr hdr,
                 String srcAddress,
                 String dstAddress,
                 PathType pathType,
                 PathChar pathChars,
                 SpecialPathServices pathServices,
                 int priority,
                 int failOverFlags);

PathSetupResponse(MsgHdr hdr,
                  ResponseCode resp,
                  int sessionId);

ErrorNotificationRequest(MsgHdr hdr,
```

int pollInterval,
int granularity);

ErrorNoticationResponse(MsgHdr hdr,
ResponseCode resp)

DRAC follows a request and response protocol popularized by the client-server architectures. In the above examples, path setup request includes a common header which has information about the application, session, authentication credential and message id number. The request include source and destination addresses of the end-to-end path desired, path type i.e l3/l1 which is optional, path characteristics such as bandwidth and duration the path should be up and available, priority of this request relative to other requests from this application and failover flags which help in making a decision on what to do when a network failure occurs.

Path setup response includes a response code for the path setup request indicating the status of the request processing and if successful also includes a session identification (sessionId) for use as a handle into all the active application sessions with DRAC.

On the similar lines, error notification request and response provide a way for applications to "pull" network level error information that is relevant to the application performance management.

3.4 SC04 Network Services Model

The AAA service is used as trust-engine to determine if a call from neighbor is authorized or not to pass thru. DRAC determines the best possible route for the end-to-end path. It checks the inner network to see if the request could be satisfied and calculates a sub-solution for the end-to-end path by selecting an egress port to hop to the next neighbor. A sub-solution of the end-to-end path is passed to the neighbor. The sub-solution contains the path till this point and state of the endpoint; this is wrapped in a so called DRAC pseudo object and passed thru to neighboring domain AAA. The neighbor will first check if requestor is authorized (trusted) according the model discussed in the AAA layer. If so, the DRAC object is sent to the local DRAC entity. This repeats until the destination is reached or a failure if the resources cannot be allocated or if a route does not exist. It is a two steps process. First a possible route is determined. If an authorized end–to-end path is found, nothing yet is provisioned, it is only temporary claimed. In the destination peer, the results are passed back to the application domain along the hopped path and backwards a commitment with provisioning is done. If it is possible to do the entire commit on the resources the application could use the end-to-end path. If something fails or resources are occupied, a roll back is done and commitment and provisioned resources are freed up.

Overall, based on determined route by DRAC entities a decision is made by the home AAA entity of the application, which delegates the authorization of used resources to the right domains and collects the answer back to make up the final decision for the application. (ako transaction model). If approved, the enforcement (PEP) is done in each involved domain. The solution is a best path, not the shortest, there are some extra effort needed to avoid loops in the solution. This is still under research. Further more extra investment is needed to avoid deadlocks, starvation of the resources, DOS attacks etc. The logical links are administrated by the two neighbor domains. We also expect an open control model in operation of these logical links, which means links are don't-care from administrative point of view. This means the interfaces are managed in the direction of the flow. Consider domain A and B sharing the logical link, domain A finds a sub-solution via domain B, it claims the resources + link, basically this means it locks temporary the link interfaces. If B receives an incoming request for the same link, it should first test if the link is not locked. This lock check is done in domain A and B before claiming the resource. With OPEN we mean unless it is clearly demarked as to who is the legal owner and the economic owner of the link, 2 parties connected by this link could be becoming 'owners' by locking the link and thus their interfaces, thereby causing the links to be over provisioned.

3.5 AAA-DRAC Operational Models

3.5.1. Peer model (AAA push sequence): AAA and DRAC operate separately. Applications first contact AAA and obtain a token which is then later used in requesting network services from DRAC. A con of this approach is that application developers need to develop two potentially independent interfaces, one to AAA and one to DRAC.

3.5.2. Proxy model (AAA pull sequence): DRAC proxy entity faces the end user or application and accepts requests for network services by consulting with AAA for authentication of application and authorization of network services. Proxy model separates application specific bindings from DRAC core, new applications can be easily supported by DRAC without having to modify DRAC core, but just by either implementing a new proxy interface or by modifying the existing proxy. The signaling between two network domains is done at proxy layer.

3.5.3. Agent model (AAA agent sequence): AAA entity faces the end user or application and accepts requests for network services on behalf of DRAC, authenticates the application, authorizes the request and relays the request to DRAC. In this case, AAA acts as an agent to DRAC for receiving and responding to application requests. The signaling between two DRAC entities of different network domains is embedded (transported) in the AAA – AAA signaling

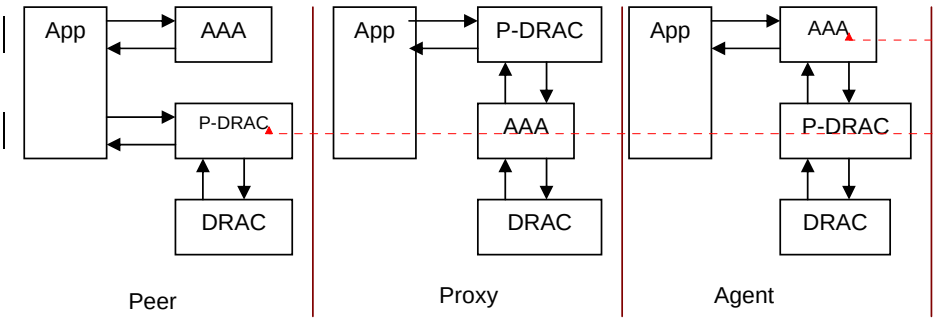

Figure 2 AAA and DRAC control models

We identified several key issues in building and demonstrating an end-to-end multi-domain service plane architecture. In the following sections we will discuss these issues and some possible solutions.

4. Key Challenges in Multi-domain Service Plane

4.1 Abstracting control plane technologies

In the SC04 demo, DRAC developed a separate interface to each of the control plane modules i.e SNMP, TL1, ODIN. The underlying technology can be abstracted out by providing a device driver like API that DRAC core is dependent on, this decouples DRAC core from the network technology. Here are some categories of network elements that we consider: SONET/SDH, DWDM, Gig E equipment. SNMP, TL1, UNI, ODIN, UCLP, ASTN and similar network control mechanisms.

4.2 Resource reservation

Should the resource reservation and commit happen in the forward or reverse direction?

In SC04 resources were provisioned and committed in the forward direction as and when the path was being established however, an end-to-end commit was done in the reverse direction to confirm the availability of resources from app source to app destination.

In general, there are two approaches here.

1. Lazy commit approach – provision forward direction and commit only when you have provisioned all segments of the end-to-end connection. The downside of this approach is that the resources may be taken by another request, no guarantees.

2. Greedy commit approach – provision and commit in the forward direction as and when you can get hold of the resource and roll back in case of failures along the way.

4.3 Rollback mechanism in resource reservation

In SC04, we demonstrated a primitive implementation of the roll back algorithm. In provisioning an end-to-end path if in any domain the connection couldn't be provisioned, a FAILURE message was communicated from the source domain DRAC to all the upstream domain DRAC instances that the connection, identified by a session id, could not be established successfully. In each domain, DRAC instance would then release the committed resources for other application requests.

4.4 Deadlocks and Starvation in Resource Reservation

Two or more applications requests could get into a deadlock situation waiting on a network resource to become available. Similarly, a hungry application may cause all the other applications to starve. Both of these situations could occur in a multi-domain administrative domain unless clear delineation of resource reservation and release policies are in place.

Discuss possible fairness algorithms.

4.5 Network Failure, Restoration and Accountability

How to propagate network failure and restoration to members of the established path(s) in other domains? What information to be shared? Why and when is this necessary?

Each domain maintains its own topology and routing information; In a DRAC-controlled domain, DRAC in that domain will ensure that an alternate path is provisioned appropriately for all the application traffic affected by this failure. However, when an inter-domain network connection failure occurs, there is a need to reroute the data on a different inter-domain path. Affected domains need to reconfigure the edge nodes and DRAC instances may need to exchange further information on which are the new end points of the new inter-domain path. This information can be represented in XML schema as we demonstrated in SC04. An issue that came up during SC04 was how to know which direction traffic is affected and inform only those edge nodes. In the demo we used brute force approach and let upstream and downstream neighbors about the network failure and let them figure out whether or not they needed to do anything in their domain.

Another issue that came up has to do with identifying the specific connection(s) that are affected when a network failure occurs and being able to recall the session information i.e QoS, BW to reestablish a new path.

A third issue was with the topology information sharing between DRAC and AAA entities. The question is should or should not AAA know about the network topology, if it should know, to

what extent? Only to an extent where an AAA instance can identify its domain and other AAA instances in other domains by some domain-AAA mapping?

Fault tolerance, ensuring that partial network or software component failure does not result in a partitioned service plane is of utmost importance.

4.6 Policy Framework - Trust, Roles and Privacy

4.6.1 Privacy

Identity of the user in a particular administrative domain is not distributed over the other administrative domains involved. If an application (in name of the user, impersonation) triggers the controlling autonomous networking service in their home domain, the authentication is done. If the user is authenticated, the domain controlling autonomous networking service uses a token and logs locally the Identity and token. However in signaling request for a sub part of the end-to-end path to neighbor service this token is used. This means the neighbors in general are not aware which user (application) triggers the request. This token might be used in combination with role and call to neighbors is made in name of the administrative domain. The role definition should be agreed on between all involved parties. If there is some fraud involved the originator administrative domain can always trace which user was involved. (Maybe more a European issue, by law it is not allowed to distribute identity information to parties involved.)

4.6.2 Roles

We make calls to neighbor domains to solve a part of the end-to-end part in name of calling domain. (Peer to peer). To have more granularity to authorize resources usage the domain are using roles. However there must be consensus what these roles means in distributed service, and every administrative domain is responsible to map and use the right application and user combination to a certain role.

4.6.3 Policy Framework

A policy framework in general, is a set of rules to control the admission for (scarce) resources. The combination of role definition, authentication tokens and trust tokens are used to determine the authorization of the resource usage (or service). Besides these attributes more general elements could be used in a policy, e.g. time of the day, resource combination.

A rule based engine uses a policy set, and check if a request could be satisfied. It should be easy to add / remove / adjust policies in this set.

Further, DRAC could push policies to policy set; the rule based engine determines the authorization. Also pulling policies could be handy, for example device policies, which determine if avoid physical devices damage.

Example:

Take an example with student, professor, researcher roles and how they can get different levels of network services at different times in the day. For example, student may not be allowed to use the commercial routed network between 8 and 5 pm. In order to accomplish this we need to define network usage policies.

Suppose two administrative domains agreed on the following roles: student, researcher and professor role. Each domain defines rules for these roles to arrange the admission of their resources (which they want to share with others). These rules are forming the policy to protect the

resources. There are different languages to specify these policies to express the rules, below we give pseudo policy to protect a lambda x (10GE link between ingress / egress in the domain).

```
If ( authNToken && TrustDomainToken )
then
   If { ( role == student      && time > 17.00 && time < 08.00  && bandwidth <= 1GE ) ||
        ( role == professor    && time > 12.00 && time < 08.00  && bandwidth <= 5GE ) ||
        ( role == researcher && { ( time < 17.00 && time > 08.00 ) ||
                                  ( time > 17.00 && time < 08.00 )  && bandwidth <=5 GE })
    }
   then
     permit
   else
     assert = 'not allowed, time constain or BW out of range'
     deny
else
  assert = 'not trusted domain request or not valid authN cert'
  deny
```

Note that each domain is free to define their own rules according the roles. The authN and TrustDomain tokens are peer wise. (might be cert iso tokens)

4.7 Generalizing AAA and DRAC

How can we make DRAC and AAA APIs decoupled from their core functionality so that it becomes easier to support multiple applications?

Implementing a proxy design pattern interface with applications and network control plane technologies will push out all the dependencies into the proxy implementation and will ensure that DRAC core itself is not affected by the changes in applications or control plane technologies.

4.8 Source Based Routing vs Traditional Routing?

An example of where a source based routing decision takes importance over traditional routing is the case discussed in section 4.6, a student's application request may not use the commercial network during certain times. In such a situation, the source DRAC may provide an end-to-end route for this application instead of using the traditional mechanisms to find next hop along the path.

4.9 Other Open Issues

5. Inter-domain Exchange Point Model vs Peer-to-Peer Model

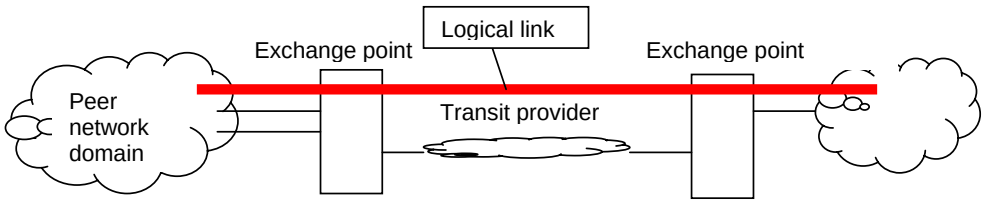

Figure 3 Connectivity between two peer domains as a logical link

With an optical exchange point diverse network domains can be connected, see Figure 3. Especially, to connect exchange point over long distances (Trans-Atlantic) a transit provider (or carrier?) connects the two exchanges. To simplify the complexity of the end-to-end connection, we abstracted the Exchange points in a virtual (or logical) link between two network domains. In each domain ingress/egress ports are connected to those links. In reality, there might be real link between the domains with or without exchange points. The transit provider is a special type of peer domain, only delivers connectivity over large distances (economic reasons). In the peer network domains themselves the internal network is considered to have well-connected ingress to egress ports. DRAC middleware in each domain will abstract the internal network topology, services offered and other internal information into an acceptable standard format such as XML Schemas. For the multi-domain setup, only the ingress and egress ports connectivity is considered. (The peer (logical) links are scarce resources)

Open Issues:
Delegation mechanism of user organization.

1. Authorization in control domains are only based on authentication of user organization. Can we do more? e.g Have user level, application level control?
2. Do we have attributes which we could use in authorization? Scheduling properties, e.g. priority. But this to get better output thru DRAC. (DRACspecific)

In this model the agreements on:

- Controlling of interfaces > openness / different control plane software? UCLP or via originator control software DRAC. Or simulated thru P-DRAC ect
- Addressing, ingress / egress ports (depends on technology? SONET channels ect ect)
- Routing; how are the routes advertised? Or attributes to calculate these routes? Layer dependent L1/L2/L3?
- Control plane; interface (resource) reservation / claiming models; provisioning models (source based routing). Issues deadlocks in distributed environment.

6. Model 2: complex requests.

Lambda centric approach: Dealing with ownership of lambdas the authorization and thus control is slightly different. First the path is calculated from some kind of routing table. In this way all lambda resource + control domains are known.

To build up the path, first permission is asked to all involved lambda owners. If the complete set of lambda (=end2end path) is authorized, the provisioning could start. In each controlling domain which connects a lambda a token is given which gives the admission to make the cross-connects. For transit domains this means that 2 token, needed in able to make the cross-connect. Making this cross-connect might fail, depends if the control plane domains were called during resource reservation in general.

7. Security Considerations

Refer to GGF's GFD.36

8. Revision Information

Author Information

Contact information for authors.
Bas Oudenaarde (oudenaar@science.uva.nl)

Madhav SBSS (msrimadh@nortel.com)
Inder Monga (imonga@nortel.com)
BL60-302
600 Tech Park Drive,
Billerica, MA 01821

Intellectual Property Statement

Full Copyright Notice

References

http://www.nortel.com/drac