Applications Drive Secure Lightpath Creation across Heterogeneous Domains

Leon Gommans[a*], Bas van Oudenaarde[a], Freek Dijkstra[a*], Cees de Laat[a], Tal Lavian [b], Inder Monga[b], Arie Taal[a], Franco Travostino[b], Alfred Wan[a]

[a]Advanced Internet Research group, Universiteit van Amsterdam,
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
[b]Nortel Networks,
600 Technology Park Drive, Billerica MA 01821, US

* Corresponding Authors: lgommans@science.uva.nl, fdijkstr@science.uva.nl

Submitted for IEEE Communications Magazine, Feature topic Optical Control Planes for Grid Networks: Opportunities, Challenges and the Vision.

Keywords: Grid Service, Grid Network, AAA, network control, lightpath

Abstract

We realize an open, programmable paradigm for application-driven network control by way of a novel network plane — the "service plane" — layered above legacy networks. The service plane bridges domains, establishes trust, and exposes control to credited users/applications while preventing unauthorized access and resource theft. The Authentication, Authorization, Accounting subsystem and the Dynamic Resource Allocation Controller are the two defining building blocks of our service plane. In concert, they act upon an interconnection request or a restoration request according to application requirements, security credentials, and domain-resident policy. We have experimented with such service plane in an optical, large-scale testbed featuring two hubs (NetherLight in Amsterdam, StarLight in Chicago) and attached network clouds, each representing an independent domain. The dynamic interconnection of the heterogeneous domains occurred at Layer 1. The interconnections ultimately resulted in an optical end-to-end path (lightpath) for use by the requesting Grid application.

1 Introduction

Independently managed network domains have long been capable to inter-connect and implement inter-domain mutual agreements among service providers. Such agreements are typically reflected in policies negotiated via protocols and interfaces such as the Border Gateway Protocol (BGP) or the External Network-to-Network Interface (E-NNI). In Grid networks, users and applications need to gain greater control of network resources, for them to exploit their atypical traffic patterns and meet their throughput/latency requirements. There is therefore a need to revisit the inter-connect process [1] between domains in the sense that:

a) Adjacent domains are no longer guaranteed to render a transport service at the same OSI Layer. Furthermore, they do not necessarily reciprocate the same peering protocol (whether it is BGP, E-NNI, or proprietary)
b) At the time the network resources are requested, there may not be inter-domain mutual agreements in place other than best-effort transit
c) Some credited users (or software agents on their behalf) are afforded the choice to source-route their traffic across participating domains.

A domain is an independently managed network cloud exposing a set of ingress and egress points associated with Service Specifications. Provisioning an optical end-to-end path while crossing different domains is quite a challenge [2]. As it can be expected in a multi-domain scenario, authentication, authorization, and accounting decisions may differ in method, protocol, and policy among the domains that the end-to-end path crosses. As well, the optical control planes may differ among the multiple domains. It becomes crucial to establish common syntax and semantics for accessing network resources.

In this paper, we present a provisioning architecture that has the ability to integrate different approaches for authentication, authorization, accounting, as well as different styles of control over network resources. We reduce this architecture to practice via a "service plane" — a new software stratum that resides on top of control planes such as ASTN, GMPLS, and JIT. The service plane encompasses software agents for the Authentication, Authorization, Accounting (AAA) and Grid Network Services software agents. Together, these agents allow users (and applications on their behalf) to negotiate on-demand network services such as low latency connection, high throughput transport, network knowledge services, and third party services. At the same time, the autonomous network domains can strictly enforce admission and usage policies and establish necessary trust amongst neighboring domains in order to avoid resource theft.

According to the telecommunications management network (TMN) model, the service plane belongs in the Service Management Layer (SML). Furthermore, the style of AAA chosen conforms to [3, 4, 5].

The paper continues with the description of the provisioning and trust architecture. Throughout section 3, we show how the service plane brings this model into practice, featuring the AAA agent and the Grid Network Services agent. Section 4 focuses on selected, distinguishing behaviors and dynamics for the service plane, such as restoration across domains. Section 5 focuses on the actual trans-continental optical testbed, which was used to prove and showcase the features of the service plane (as shown at the Supercomputing 2004 Conference in Pittsburgh, PA, USA), followed by conclusions and directions for further research.

2 A New Provisioning Model

Grid users are accustomed to allocate and relinquish some virtualized sets of computational, storage, and/or visualization resources. They do so with a high degree of automation, using software feedback loops and schedulers taking the place of GUI portals and operators.

In many Grid scenarios, the network element turns out to be a resource as important as computation and/or storage. As such, Grid users require the same level of control towards subsets of well-defined amounts of network resources for the duration of a specific Grid task.

A chief goal of our service plane is to turn the network into a virtualized resource that can be acted upon and controlled by other layers of software, be it applications or Grid infrastructures (e.g., a community scheduler). In other words, the network becomes a Grid managed resource much as computation, storage, and visualization are.

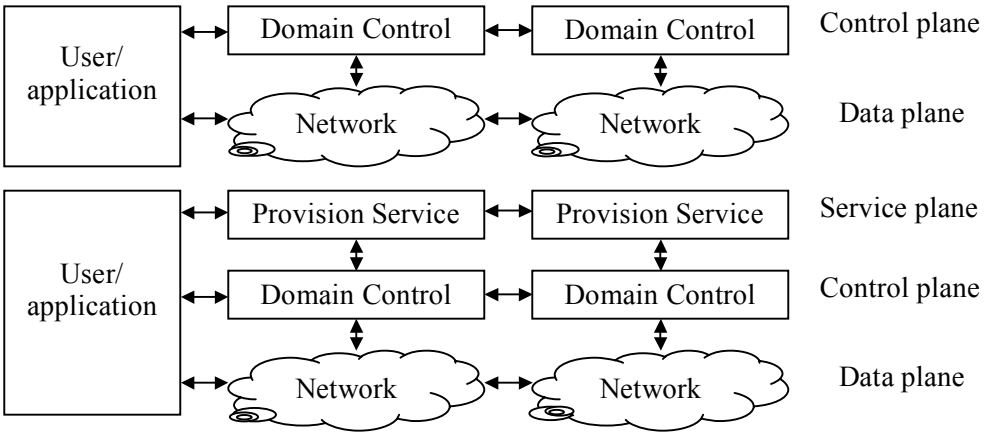

Fig. 1. Top: the current model encompasses a control and network layer, allowing network providers to negotiate network capabilities. Below: The presented provisioning model adds a service plane, which allows additional application driven network control.

Layered upon the (optical) network control plane (see fig. 1.), the service plane is typically concerned with path allocation, optimization, monitoring, and restoration across two or more domains. A service plane must be designed to be extensible from the ground up. It should allow adaptation of various control plane interfaces and abstract their network view or element set into the service plane. Examples of underlying control planes are: ASTN, GMPLS, JIT, etc. Each control domain has exclusive control of its resource and is typically able to signal neighboring domains to create end-to-end paths on behalf of the involved Service Providers.

The AAA and Grid Network Services agents are the two key ingredients of the service plane. For the moment, we assume that each domain has implemented such agents.

A Grid Network Service agent advertises network capabilities to its (trusted) neighbors. In such way, each agent is able to generate a complete topology view in order to construct future optical end-to-end paths. Our agent can optionally support source based routing to select a preferred path through the individual optical clouds.

If a Grid Network Service agent wants the neighboring instances to create a path, it requires a proper authorization token. The AAA part of the service plane obtains authorizations from multiple administrative domains. Before any path can be provisioned, all authorizations will need to be collected. We used our model in a GLIF-like context [6]. Here parties are able join the federation if they allow some user controlled access to their network resources. The end-to-end provisioning is split into a two-phase commitment process, whereby first the authorization is handled and resources (e.g. switches, and links) are reserved. Secondly, the actual commitment follows. Out of many approaches and sequences possible, we have chosen to adopt a RSVP-like signaling mechanism between Grid Network Service instances. The agent sequence is chosen with regard to AAA (per AAA Authorization Framework [4]).

Since the path negotiation transits across multiple domains, the inter-domain trust model is a fundamental aspect to the overall provisioning strategy. There is a peer-to-peer relationship among the AAA servers representing an organization or domain. Furthermore, there is a trust relationship between an AAA agent and the Grid Network Service agent in each domain. Once the User is authenticated and authorized by the AAA agent of the source domain, this AAA agent represents the User during the setup process. Path establishment has been accomplished by means of transitive trust. This model corresponds to the chained partial

control model as described in [7]. The requests are authorized because the requestor is known and trusted and the resource policy conditions are met. In our model, we used a token mechanism to ensure the authenticity of a request. Managing trust by means of distributing corresponding key material can be done in different ways. As we focused on the authorization aspects rather then security aspects, we assumed the existence of some proven mechanism that will ensure safe delivery, storage and usage of keys.

3 Building the Service Plane

The following section will outline the Grid Network Services and AAA agents.

3.1 Grid Network Service agent

In our experiment, Nortel's DRAC (Dynamic Resource Allocation Controller) implemented the Grid Network Service agent. Each participating network domain had one or more instances of the DRAC running. In case multiple instances of DRAC are running in a single domain, a master instance is elected. This DRAC master instance manages the domain and the inter-domain connectivity through peer messaging. DRACs core framework includes services like a policy engine, a topology discovery engine, workflow utilities, inter-domain routing facilities and smart bandwidth management fixtures.

DRAC exposes an API allowing coupling with applications. The interface to applications is bi-directional, enabling network performance and availability information to be abstracted upwards toward the application. Applications can request network services through this API. Applications can for example request a "cut-through" (high bandwidth, low latency) service allowing applications to bypass Layer 3 and directly transfer data over Layer 1 connections. Applications can further specify if they want this service on demand or via a time-of-day reservation. This kind of functionality is deemed especially valuable for the data-intensive applications used in research networks.

3.2 The Generic AAA service agent

An implementation of a service using the University of Amsterdam's Generic AAA toolkit created an agent that was used to determine if a call from an authenticated neighbor is authorized to invoke DRAC. For a complete end-to-end path, the authorization is spanning multiple authorization decisions by autonomous AAA servers. The generic AAA based service agent invokes DRAC via an Application Specific Module (ASM). Such module performs application specific services such as interpreting the meaning of parameters passed to it (see [3]). The service itself may be provided by equipment external to the ASM, in our case DRAC. In such case, the ASM communicates with the service via a well-known protocol (in our case TL-1).

DRAC is consulted to determine the network semantics. In the source based routing approach, a sub-solution of the end-to-end path is passed from neighbor to adjacent neighbor. This sub-solution contains DRAC pseudo objects, which reflects current path and suggested connection point solving the end-to-end path.

We implemented the Generic AAA based Chained Partial Control model ([7]) based upon the concept of a driving policy running in each domain. A driving policy is a nested if-then-else structure executed by a Rule Based Engine (RBE, [3]), which is part of the Generic AAA toolkit. Its main task is to describe which pre-conditions have to be checked before actions, are needed to fulfill an incoming AAA request, and are delegated to ASMs. In general, these ASMs might apply their own policies and protocols. The type of incoming request determines the type of driving policy that the RBE should execute inside the AAA server. Every type of

incoming request is assigned a driving policy. The RBE retrieves the driving policy from a Policy Repository managed by the administrator of a network service domain..

4 Service Plane Features in Focus

In the following section, we explore distinguishing features of our service plane. While the examples given in this section refer to the network layout shown in fig. 2, the principles are generally applicable.

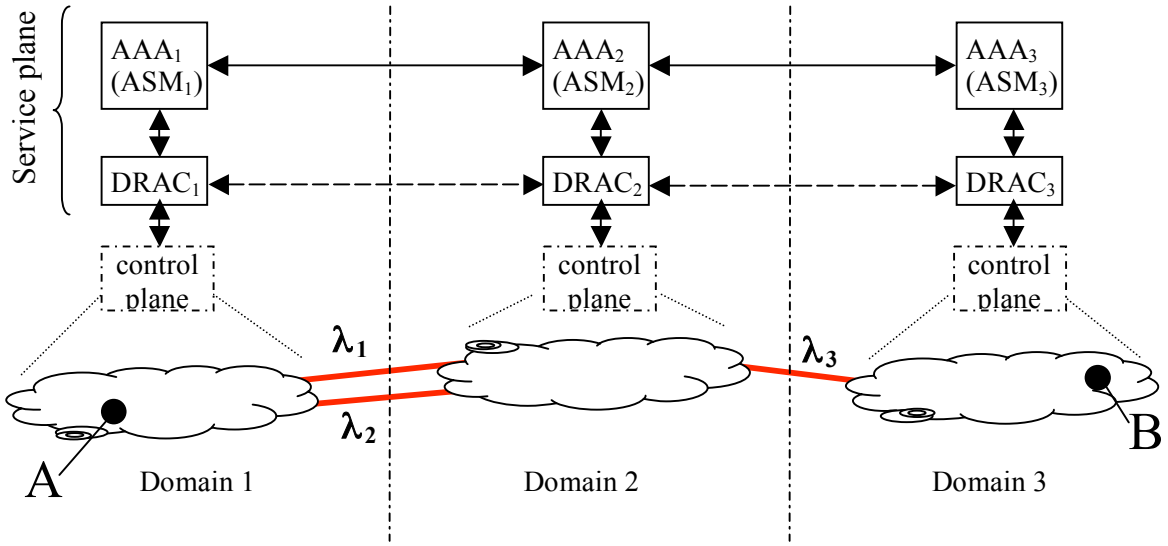

Fig. 2: A network layout extending across 3 domains

4.1 Message exchange in multi-domain provisioning

Without loss of generality, we assume that the AAA_1 server in the domain of a source node A (see fig. 2) receives the request by a User (or an application on its behalf) for a connection between a network node A and B. The request looks like:

```
<AAARequest type="AAA_01" >
  <AAA>
    <Authentication>
      <Name>Joe User</Name>
      <Attribute AttributeId="token">
        SSBhbSBhbiBvcmRpbmFpcnkgdXNlcg==</Attribute>
    </Authentication>
  </AAA>
  <DRAC>
    <Xfer>
      <SrcId>10.1.1.120</SrcId>
      <DestId>10.1.3.130</DestId>
      <Amount>1000</Amount>
      <TimeSpan>3600</TimeSpan>
    </Xfer>
  </DRAC>
</AAARequest>
```

The content of the User's request is divided into two parts. The first part is enclosed between AAA-tags and contains the information concerning authentication/authorization (accounting). The second part enclosed by the DRAC-tags contains the information the DRAC needs. Typical information for $DRAC_1$ are the two network nodes, `SrcID` and `DestId`, the amount

of information to transfer in Mbytes, `Amount`, and the time span within which the transfer must take place, `TimeSpan`.

The provisioning process encompasses a series of actions and messages exchange between AAA servers along a path between source node A and destination node B. Server AAA_1 checks whether it can authenticate and/or authorize the User. When successful, AAA_1 extracts the information between the DRAC-tags in the request and passes it on to $DRAC_1$ by way of the Application Specific Module ASM_1. $DRAC_1$ is asked to define a connection between A and B. The reply from $DRAC_1$ to ASM_1 contains the specification for a request that must be forwarded to the next AAA server in the provisioning process. The message sent to AAA_2 by AAA_1 has a similar structure as the User's request:

```
<AAARequest type="AAA_02" version="0.1" >
  <AAA>
    <Authentication>
      <Name>AAA@science.uva.nl</Name>
      <Attribute AttributeId="token">
        SSBhbSBBQUEgc2VydmVyIDE=</Attribute>
      </Authentication>
    <SessionID>12335</SessionID>
  </AAA>
  <DRAC>
    <DRACXfer>
      <SrcId>10.1.2.121</SrcId>
      <DestId>10.1.3.130</DestId>
      <XferAmount>1000</XferAmount>
      <TimeVal>3600</TimeVal>
    </DRACXfer>
  </DRAC>
</AAARequest>
```

The new attribute for the authentication and the new source node (`SrcId`) are worth noting. The token of the Attribute-tag identifies AAA_1. The new source node determined by $DRAC_1$ represents an ingress point into the next domain, i.e. the domain where λ_1 enters. $DRAC_1$ determines the first part of the path, an intra-domain connection between node A and an egress point, λ_1. This is a decision based on topology information.

The communication among DRAC agents keeps the topology information up to date. When AAA_1 needs to authorize access to λ_1, it is not evident who owns this connection. Here it is assumed that λ_1 is owned by AAA_2 or by a third party that AAA_2 knows about. AAA_1 must therefore first contact AAA_2 to obtain authorization.

This process continues until the last server in the provisioning process, AAA_3, with the help of $DRAC_3$, establishes a connection with destination node B. The reply from AAA_3 travels back along the path established. Upon receiving a positive answer, each AAA server commits the intra-domain connection determined earlier on and makes the necessary preparations for accounting. Finally, AAA_1 commits the connection between node A and λ_1.

4.2 Driving policy

An AAA server's Rule Base Engine is guided by a driving policy that describes the actions to be taken by the AAA server upon receiving a request. This process is governed by a driving policy residing in the Policy Repository of AAA_1 :

```
if ( true )
then (
  // store <Name> and <Attribute> values in local variables
  userName = Request::AAA.Athentication.Name;
  userAttr = Request::AAA.Athentication.Attribute[AttributeId#token];

  // store array returned in local variable K2
  K2 = ASM::Authenticate( userName, userAttr );

  // K2[0]: indicates success or failure
  // K2[1]: new key returned after successful authentication
  if ( (K2[0] == "SUCCESS") &&
       ASM::Authorize( userName, K2[1], “DRAC”, “Setup” )
  )
  then ( // start the setup

    sessionId = AAA::GetSessionID( userName, userAttr, K2 );

    // store array returned in local variable O4
    O4 = ASM::DRAC.Setup( Request::DRAC.Xfer.SrcId,
                          Request::DRAC.Xfer.DestId,
                          sessionId,
                          Request::DRAC.Xfer.Xfer.Amount,
                          Request::DRAC.Xfer.TimeSpan );
    // O4[0]: indicates single or multi-domain solution
    // O2[1], O2[2] internal variables applied by DRAC
    // O4[3]: end point of a lambda determined by DRAC
    if ( (O4[0] == "SUCCESS") ) // single domain solution
    then ( AAA::PrepareAccounting( sessionId, O4 );
           Reply::Answer.Message = "permit";
           Reply::Answer.SessionId = sessionId
    )
    else (
      if ( (O4[0] == "PENDING") ) // multi-domain solution
      then (
        // O4[3]: end point of a lambda determined by DRAC
        destAAA = AAA::GetDomainServer( O2[3] );

        R =  AAA::SendAAARequest(
                            AAA::GetAuthNToken(),
                            O2,
                            destAAA );
        if ( (R == "PERMIT") )
        then (
          // O2[1], O2[2] internal variables applied by DRAC
          ASM::DRAC.Commit( O2[1], O2[2] );
          AAA::PrepareAccounting( sessionId, O2, R );
          Reply::Answer.Message = "permit";
          Reply::Answer.SessionId = sessionId;
        )
        else (
              ASM::DRAC.Rollback( sessionId );
              Reply::Answer.Message = "deny"
        )
      )
      else (
              ASM::DRAC.Rollback( sessionId );
              Reply::Answer.Message = "deny"
      )
    )
  )
  else ( Reply::Answer.Message = "deny" )
)
else ()
```

The first action prescribed by the driving policy is the authentication of the User. Authentication is an example of a service provided by an ASM. Arguments for the function `ASM::Authenticate` are retrieved from the incoming request. The notation refers to the Athentication-tag in the request which has an attribute called AttributeId equals 'token'. This token is passed to the authentication function. If the authentication is successful, the requester is authorized, using function `ASM::Authorize` (see section Authorization for more details).

When all preconditions for the provisioning are fulfilled, a session needs to be created to keep track of the provisioning process. The function `AAA::GetSessionID` creates this session. Provisioning is initiated with the call, `ASM::DRAC.Setup`. In case of a multi-domain setup, the new source node returned by $DRAC_1$ is key information. Beside the data from the User's request, $DRAC_1$ applies the internal topology knowledge to choose the lambda, λ_1. It returns the endpoint of λ_1 in the next domain. This endpoint is used by the RBE to determine the next AAA server in the process, `AAA::GetDomainServer`. Each DRAC agent keeps track of the provisioning process by generating a handle and a session identifier, which are needed in future communication with the DRAC agent.

It is up to $DRAC_2$ to continue the provisioning process from the end point of λ_1. It decides to extend the path via λ_3 to the domain of the destination. This decision is also based on the possibility to arrange an intra-domain connection between λ_1 and λ_3. In the request sent to AAA_3, the new SrcId equals the endpoint of λ_3. The first phase of a successful provisioning process ends with a path between the end node of λ_3 and the destination node B.

AAA_3 arranges accounting and replies to AAA_2. Upon receiving a positive answer, all AAA servers along the path commit the provisioning to their DRAC agent and make arrangements for accounting. Therefore, the policy specifies a call to `AAA::PrepareAccounting`. Making preparations for accounting involves creating a record that contains the session identifier together with the information returned from the call to the DRAC agent.

Other than a reply indicating success, the User is given a session identifier. With such an identifier, the User can intervene on the lifecycle of the session, e.g. by requesting an earlier termination.

4.3 Authentication and Authorization Extension Mechanism

By means of Web Services, a User can find out what kind of authentication data has to be carried in the message (message authentication). Once authenticated, the User needs to be authorized. This is the act of determining whether a particular right can be granted to a requesting entity with a particular credential. XACML (eXtensible Access Control Markup Language), an OASIS standard for expressing and evaluating access control policies, contains a usage model [8] where a Policy Enforcement Point (PEP) is responsible for protecting access to one or more resources.. The PEP sends a request to a Policy Decision Point (PDP), which evaluates the request against policies and attributes and produces a response. The PDP in the XACML model applies its own policies. It is the AAA server's Rule Based Engine that resorts to an Application Specific Model that acts as a PEP in the XACML usage model. This is indicated by the `ASM::Authorize` call in the above listed driving policy.

4.4 Automatic Light path restoration

If somewhere along the optical end-to-end path a link goes down, an alternative link might be arranged automatically in order to restore the lightpath. The control plane often recovers an intra-domain link failure. In all other cases, the link failure triggers an alarm condition inside the DRAC agent. The DRAC agent responsible for the broken link provides its AAA server with information about an alternative link.

The steps the AAA server subsequently takes are rather similar to those of an ordinary setup, except that no new session identifier needs to be generated. Accounting consequences of an alternative link during operation mode might be that the transport over the alternative link is more expensive. A reasonable strategy is to charge the transport over the alternative link according to the prior arrangement.

In case no restoration can be established within a small time span, the initiating AAA server should be informed that the session corresponding with the transfer has ended. A request to stop the transfer between source and destination node can travel backwards to the AAA representing the User. This is facilitated because in a request exchanged among AAA servers the requesting server identifies itself and provides the server down stream with a session identifier.

5 Experimental Setup

The provisioning model was demonstrated during the SC2004 conference, held in November 2004 in Pittsburgh, PA, USA. Three optical domains, NetherLight, StarLight, and OMNINet were part of the experimental testbed, representing Domain 1, 2 and 3 respectively in Figure 2. NetherLight is the optical infrastructure in Amsterdam. The University of Illinois at Chicago manages StarLight, and Nortel, SBC Communications Inc. and Ameritech support OMNINet. In fig. 2 the testbed is depicted showing the three optical networks, each considered as a single administrative domain.

One should interpret the inter-domain lambdas as a collection of optical entities necessary to establish a connection between peer networks. For instance, the connection between NetherLight and StarLight is a collection of SONET based optical Exchange Points [9] and transit provider links for long distances.

In each domain, a single DRAC agent was given responsibility for the setup of intra-domain connections. It also connects ingress points to egress points in the domain under its control. In this testbed, we simulated a link failure by switching off a port on one of switches providing inter-domain connectivity, thus generating an inter-domain failure event. We then measured end-to-end link restoration times across the three domains shown in fig. 2. The elapsed time for both detection and restoration of the inter-domain failure (fig. 3) was in the order of a minute. Although there are several ways to optimize such times, this is already a vast improvement of the usual hours to days required to restore a connection using conventional means such as phone or email.

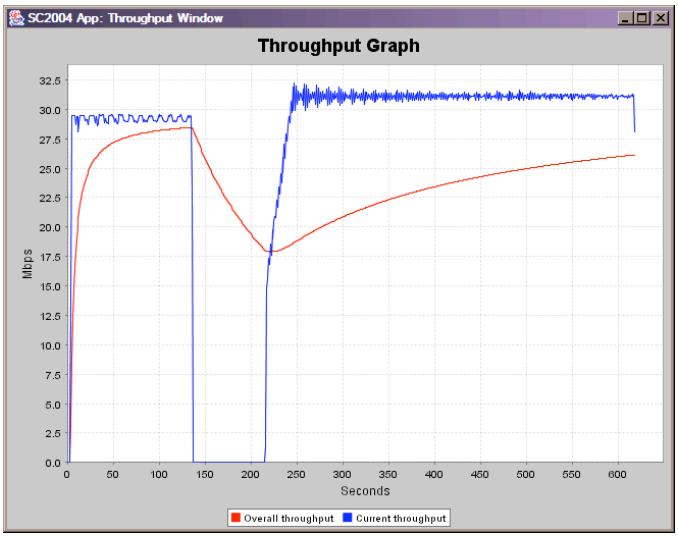

Fig. 3: The throughput line (blue) shows the time interval at [140, 210] seconds that is required for the service plane to detect and recover the simulated inter-domain failure.

6 Conclusions

With a novel provisioning architecture built upon a service plane, we have shown that it is possible to perform programmable, application-driven network control in an open fashion, yet without compromising security. The model has a minimal reliance on adjacent clouds implementing the same control plane standards. We have latched our open approach onto the software evolution curve (Web Services, Service Oriented Architecture) to best exploit dynamic service discovery and feature introspection across domains, without extensive code rewrites. By virtue of a layered approach, we have shown how restoration can be fine tuned to occur at the different places — i.e. the control plane for intra-domain failures and the service plane for inter-domain failures.

The intersection of application-driven provisioning, constraint-based routing, and scheduling of optical resources warrant further research. A chief goal is to mitigate line blocking effects while maximizing user satisfaction and network utilization. Furthermore, the recent advances in workflow language technologies and semantic composition of workflows hold potential to greatly enhance the dynamic binding between applications and the network.

Acknowledgments

The research work was in part funded by the BSIK project GigaPort Next Generation, the Dutch National Research Network (SURFnet) and the EU IST 6th framework program project Nextgrid under contract number 511563.

We like to thank the following colleagues: Madhav Srimadh, Satish Raghunath, Paul Daspit, Bruce Schofield, Chetan Jog, Phil Wang, Joe Mambretti, Fei Yeh, Pieter de Boer, Paola Grosso, Hans Blom and Dennis Paus.

We also like to thank Nortel for their support of this research effort, and the GLIF community for the usage of their lambda's.

References

1: Creating an intelligent optical network worldwide interoperability demonstration; Jones, J.D.; Ong, L.; Lazer, M.A. Communications Magazine, IEEE Volume 42, Issue 11, Nov. 2004

2: An Open Grid Services Architecture Based Prototype for Managing End-to-End Fiber Optic Connections in a Multi-Domain Network; S.M.C.M. van Oudenaarde, Z.W. Hendrikse, F. Dijkstra, L.H.M. Gommans, C.T.A.M. de Laat, R.J. Meijer, High-Speed Networks and Services for Data-Intensive Grids: the DataTAG Project, special issue, Future Generation Computer Systems, Volume 21 Issue 4 (2005).

3: Generic AAA Architecture; C. de Laat, G. Gross, L. Gommans, J. Vollbrecht, D. Spence, RFC 2903, August 2000, http://www.ietf.org/rfc/rfc2903.txt.

4: AAA Authorization Framework; J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, D. Spence, RFC 2904, August 2000, http://www.ietf.org/rfc/rfc2904.txt.

5: AAA Authorization Application Examples. J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, D. Spence, RFC 2905, August 2000, http://www.ietf.org/rfc/rfc2905.txt.

6: TransLight: a global-scale LambdaGrid for e-science; Tom DeFanti, Cees de Laat, Joe Mambretti, Kees Neggers, Bill St. Arnaud, Communications of the ACM, Volume 46, Issue 11, November 2003, pp. 34-41.

7: Authorization of a QoS Path based on Generic AAA; Leon Gommans, Cees de Laat, Bas van Oudenaarde, Arie Taal, iGrid2002 special issue, Future Generation Computer Systems, Volume 19, Issue 6 (2003).

8: XACML 2004 XACML: SAML 2.0 profile of XAMCL, Committee Draft 01, 16 September 2004, http://docs.oasis-open.org/xacml/access_control-xacml-2.0-saml_profile-spec-cd-01.pdf.

9: Control Models at Interconnection Points; Freek Dijkstra, Bas van Oudenaarde, Bert Andree, Leon Gommans, Jeroen van der Ham, Karst Koymans, Cees de Laat, In Press.