Supporting Communities in Programmable Grid Networks: gTBN

Mihai Lucian Cristea†, Rudolf J. Strijkers†*, Damien Marchal†, Leon Gommans†, Cees de Laat†, Robert J. Meijer†*

†University of Amsterdam
The Netherlands
{m.l.cristea, dmarchal, l.gommans, delaat}@uva.nl

*TNO Informatie en Communicatietechnologie
The Netherlands
{rudolf.strijkers, robert.meijer}@tno.nl

***Abstract*—This paper presents the generalised Token Based Networking (gTBN) architecture, which enables dynamic binding of communities and their applications to specialised network services. gTBN uses protocol independent tokens to provide decoupling of authorisation from time of usage as well as identification of network traffic. The tokenised traffic allows specialised software components uploaded into network elements to execute services specific to communities. A reference implementation of gTBN over IPv4 is proposed as well as the presentation of our experiments. These experiments include validation tests of our test bed with common grid applications such as GridFTP, OpenMPI, and VLC. In addition, we present a firewalling use case based on gTBN.**

I. Introduction

Cooperation between organisations, institutes and individuals often means sharing network resources, data processing and data dissemination facilities. Communities are a group of individuals, organisations or institutes that have an agreement about sharing services and facilities, which are accessible only to its members. When user applications need to access and process data from various, possibly heterogeneous, systems and locations, we need to cope with application-specific connectivity, different access policies, and at the same time provide services bound to the community.

The following three scenarios illustrate community-based network services. First, many scientists are allowed to access worldwide digital libraries on behalf of their academic organisation regardless of their location (network source address). Currently, this is only possible within the organisation domain. Second, large-scale experiments by scientific communities require data gathering from multiple sources such as high-throughput sensors, lab equipments, followed by data processing on multiple Grids under different ownership. This needs infrastructure support for sharing computational, storage and networking resources. Third, rules and laws of a country or organization may apply to communities of which their members are located worldwide and interconnected over public and private networks and hence, the network has to guarantee separation of these communities to support judicial territories. The three examples all require a form of traffic identification and control to provide specific services. However, the current Internet model offers only best-effort end-to-end connectivity. In this paper, we address network support for binding specialised, application-specific services to communities and their applications.

An alternative to the Internet model is programmable networks. In programmable networks, network elements become fully programmable devices. By programming the collection of network elements a network can offer specific services, and implement any form of traffic identification and control. Efforts in programmable networks have led to frameworks, such as integrated and discrete active networks [1], and among others resulted in test beds like Tempest [2], Switchware [3] and Capsules [4]. Although less flexible, optical and hybrid networking technologies (e.g., UCLPv2 [5], GMPLS) are now preferred over programmable network solutions to provide application controlled end-to-end connectivity. Alternative to current programmable network projects such as GENI [6], OpenFlow [7] offers a pragmatic, intermediate solution based on flows to allow researchers to experiment with new network services and communication protocols, and also have vendor support. While programmable network architectures in general follow a network centric approach, the User programmable Virtualized Networks (UPVN) [8] architectural framework takes an application centric approach by allowing network services to be defined by distributed and networked applications themselves. The UPVN architectural framework considers the network and its services as software, and defines the elementary components to develop specialised services from applications.

In this paper, we propose an architecture, generalised Token Based Networking (gTBN), that provides binding between distributed networked applications and programmable network services. gTBN uses the concept of Token Based Networking [9] to associate streams with UPVN services by tagging packets with a service identifier. These identifiers are inserted in the process of network resource allocation and management and are independent of communication protocols. We present a reference implementation of the gTBN architecture in IPv4 and a firewall use case as illustration of an alternative approach for domain protection in Grid networks. In a broader context, gTBN allows communities to define their own personal Internet providing network services that match their requirements.

The remainder of this paper is organised as follows. The gTBN architecture is presented in Section II, followed by implementation details in Section III. We evaluate a firewall use case for Grids and show the results of our experiments with streaming, client/server and message passing applications in Section IV. In Section V, related work in the field is compared and discussed, followed by the conclusions in the last section.

II. GTBN Architecture

The Token Based Networking (TBN) architecture was initially introduced to establish lightpaths over multiple network domains [9]. Lightpaths are setup on behalf of authorised applications that need to bypass transit networks. On the one hand, TBN uses a secure signature of pieces of an IP packet as a token that is placed inside the packet to recognise and authenticate traffic. The applications traffic is first tokenised by the *TokenBuilder* of a local domain (e.g., a campus network), after which it is enforced by the *TokenSwitch* at each inter-domain controller along the end-to-end path (see Figure 1). On the other hand, TBN makes use of a separate service and control plane. The control plane consists of a *AAA* server in the push sequence as explained by the Authorisation Authentication Accounting (AAA) framework (RFC 2904). The AAA server acts as an authority that is responsible for the reservation and provisioning of the end-to-end paths, possibly spanning multiple network domains.

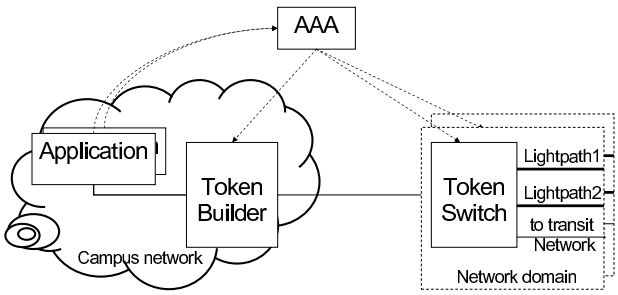

Fig. 1. On behalf of an application, the AAA authority provisions the network resources required for an end-to-end lightpath that may cross multiple domains. At runtime, the application traffic is first tokenised by *TokenBuilder* and then subsequently enforced by each *TokenSwitch* along the lightpath.

Making tokens protocol independent has an important advantage; the token can be regarded as an aggregation identifier to a network service. Generally, we see four types of aggregation identifiers that can be combined, as follows:

- identifier to point a service to the NE (e.g., a multi-cast, or transcoding);
- identifier that defines the service consumer (e.g., the grid application);
- identifier that defines the serviced object (e.g., the network stream);
- identifier that defines the QoS (security, authorisation, robustness, deterministic property, etc.).

First, a token can bind to different semantics and services (e.g., network services for a user, a group of users, or an institute). The semantics that is referred to by a token (e.g., a certain routing behaviour) can be hard-coded into a switch or the token can refer to a stored aggregation identifier that points at the specific behaviour in a programmable network device. Hence, a token provides a generic way to match applications to their associated network services. Second, tokens can be either embedded in the application generated traffic or encapsulated in protocols where embedding is not supported, such as in public networks. Third, tokens can also provide a general type of trusted and cryptographically protected proof of authorisation with flexible usage policies.

A. UPVN

UPVN describes the architectural principles for orchestration and manipulation of network services in programmable networks. UPVNs enable applications to upload code to network elements (NE) in the form of software objects called application components (AC). ACs enable implementation of not foreseen and application-specific services and allow computer programs to access their service interfaces through network components (NC). NCs act as proxies that provide redirection, visualisation or composition of AC interfaces and can be included as part of distributed and networked applications. The architectural framework is shown in Figure 2.

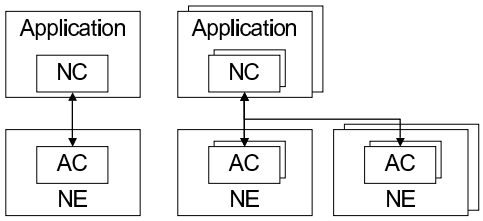

Fig. 2. Relating the UPVN components. NCs are the manifestation of the network in applications. Conversely, ACs manifest as application-specific network services. The interface between NCs and ACs depends on the application domain.

In general, programmable networks differ in how applications interface with network nodes. Basically there are three variants: agents, active messages (also known as active networks) and remote method invocations (RMI). In short, agents are programs that travel from node to node, active messages are network packets extended with application code, and webservices are a good example of RMI. UPVN generalises network/application communication by considering the network as an integrated part of the application. UPVN regards NEs as software components with NCs as their proxy objects, which support agent, active message or RMI communication. NCs can also be part of applications designed for other purposes, such as Mathematica or Distributed Transaction Monitors. This allows creation of new applications from existing ones to provide more features to network users. Such features include authorised resource sharing, fine-grained binding of applications to networks, and building communities over public networks.

B. TBN + UPVN = gTBN

UPVNs enable applications to exploit and integrate the freedom programmable networks offer. TBN provides an authorisation architecture with a clear binding process between applications and network functions or resources. In generalised Token Based Networking (gTBN), the network infrastructure and network interactions are generalised from the classic end-to-end circuits, lightpaths to multipoint-to-multipoint networks.

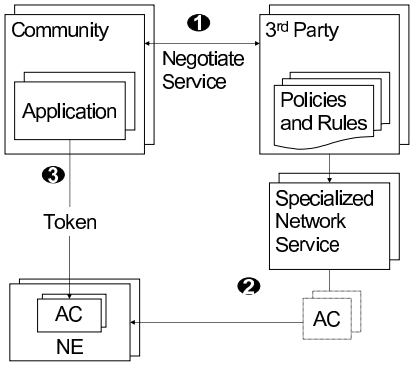

Fig. 3. Applications from a community are associated with tokens according to existing policies ①. A third party manages the network resources and provisions the required services into the NEs ②. At runtime, tokenised traffic sent by applications is recognised and authenticated by the network ③.

The gTBN architecture consists of three parts, as illustrated in Figure 3. First, each community with specific policies and rules needs to be associated with a token ①. A third party implements the negotiation, filtering and reservation of the resources and services using a AAA framework. The implementation of this entity depends on the context, such as resource brokers in Grids or network operators in private networks. Second, specialised network services (e.g., routing, transcoding) for a community are provisioned in the network as required by the applications through their associated tokens. Because the network cannot know in advance which services or combination of services a community may require, specialised services must be uploaded to a network as ACs ②. Last, when a member of a specific community executes an application, the member or its secure execution environment tokenises the produced traffic ③. The network recognises the tokenised traffic and applies the specialised pre-programmed services.

Figure 4 shows an example application of gTBN. Two communities with its member applications are located on different network domains. For example, Application 1 and 2 belong to community B, and Application 3 belongs to community A. The two communities are each associated with a token, gray and white, respectively. Let us assume that the policy of a network domain is that only members of the communities are allowed to be routed through the network. Application 1 and Application 2 are both members of community B and therefore, they can communicate. Correctly tokenised traffic will be routed using default IP mechanisms. However, to communicate with Application 3, Application 1 also needs credentials to access the resources of community A. gTBN supports binding of multiple domain-specific services into a single token. This allows a member of both communities A and B to access each others network domains.

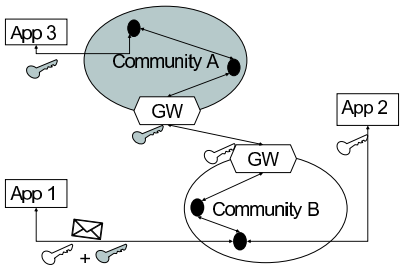

Fig. 4. Application 1 can communicate with Application 2 in community B by using the associated token. To communicate with Application 3, the message needs to contain both tokens of communities A and B.

III. Implementation

The complete architecture is composed of three components, as previously presented in Section II-B and Figure 3. The first component is a programmable network element that implements the network behaviour by recognising the authenticity of the tokenised traffic. Enforcing the authenticity of tokens ensures the correct execution of the intended network behaviour (see Section III-A). The second component runs on the end-user host and binds the token to the application's traffic (see Section III-B). The third component, which is beyond the scope of this paper, associates a token to the policies applied to the communities; interested readers are referred to [10].

Currently, our implementation binds application's traffic to tokens at the IP layer, and enforces the tokenised traffic at the IP layer, too. It is important to notice that according to the RFC 791 the IP option field, we used for tagging, must be implemented by all IP modules. However, in practice we found that the RFC is not respected by all Internet routers. In a future implementation, though, we will put the tags into an IPv6 extension header.

A. Programmable Network Element

Our implementation of the network elements (NEs), called Token Based Switch (TBS), must be able to enforce specific network behaviour (e.g., routing, multicast) on per-packet basis as required by the tokens the packets carry. Currently, it is possible to implement such programmable NEs using specialised hardware (e.g., network processors, FPGAs) or using powerful PCs. We have chosen to use the network processors because we think that such packet enforcement systems, working at the Ethernet layer, will be located at the gateways of the network domains and hence, they need to process packets at multi-gigabit speeds. For example, Intel IXP2850 network processor provides high speed packet handling (up to 10Gbps) and on-chip crypto hardware supporting commonly used algorithms: 3DES, AES, SHA-1, HMAC. Although the current powerful PCs are able to route packets

at multi-gigabit speeds, they still lack of ability to perform cryptographic algorithms at line rates despite the multi-core architecture.

The current Token Based Switch (TBS) implementation uses the dual network processors hardware platform (see Figure 5). Each NPU contains on-chip 16 multi-threaded RISC μEngines running at 1.4GHz, a fast local memory, registers and two hardware crypto units for encryption/decryption. The μEngines are highly-specialised processors designed for packet processing, each running independently from the others from private instruction stores of 8K instructions

As illustrated in Figure 5, the incoming packets are received by the Ingress NPU. These packets can be processed in parallel with the help of the μEngines. The packets are subsequently forwarded to the second NPU. The second NPU can process these packets and then decide which will be forwarded out of the box and which outgoing link will be used.

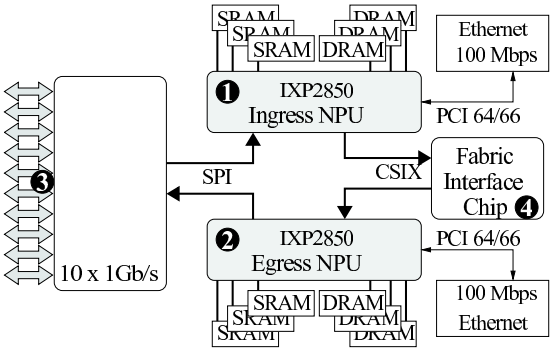

Fig. 5. IXDP2850 development platform uses dual IXP2850 NPUs ① & ②, 10×1 Gbps fibre interfaces ③, a loopback fabric interface ④, and fast data buses (SPI, CSIX). Each NPU has several external memories (SRAM, DRAM) and its own PCI bus for the control plane.

In the current implementation, the TBS uses an authentication application component (AC) combined with a routing network behaviour. In other words, the specific routing service run on each authenticated packet. A packet is authenticated when the built-in token (stored in the IPv4 option field) matches the result of applying a keyed Hash Message Authentication Code (HMAC) algorithm over the entire IP packet, or over part of the packet. Our implementation uses the first 64 bytes of the packet to ensure constant speed processing while the HMAC algorithm creates a one way hash that is a key-dependent (see RFC 2401). In our implementation we opted for a strong proof of authorisation by means of HMAC-SHA1 that is also hardware supported by the IXP2850 network processor.

Figure 6 shows the token creation and checking mechanism. For each received packet, the TBS checks whether the current packet has the appropriate IP option field. On success, a Global Resource Identifier (GRI) field, identifying a specific application instance, is extracted ① to refer to the network behaviour needed to be applied to the packet. For example, the expected network behaviour is authentication and hence, the GRI points to an authorisation table (AuthTable) of an already deployed authentication AC. Next, the authentication AC uses

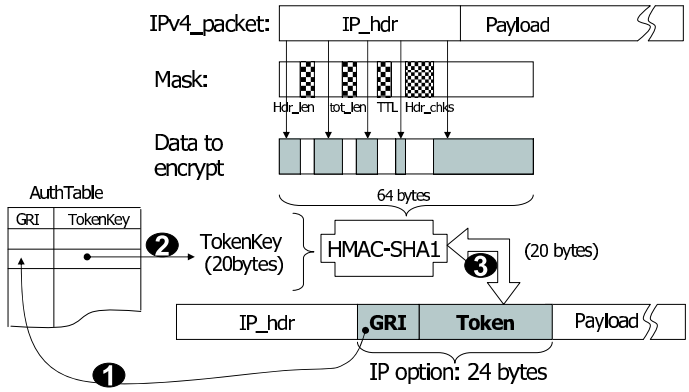

Fig. 6. The token checking mechanism uses the GRI part of IPoption tag to point into AuthTable of authentication component ① in order to extract the TokenKey ② needed to perform the HMAC-SHA1 over the masked packet data. The packet is authenticated when the encryption result matches the built-in token ③.

the GRI entry to retrieve the encryption key (TokenKey), already provisioned in the TBS ② by a AAA authority. Then, the first 64 bytes of the packet data are masked and then are encrypted using the HMAC-SHA1 with the TokenKey ③. The result is compared with the remaining of the option field. When they match, the authentication AC authorises the packet to be forwarded to an adequate port. Otherwise, the packet is dropped. Note that although in the paper we refer to *token* as an entire tag built-in the packet, in our implementation, the tag consists of two parts: a plain-text aggregation identifier (GRI) and an encrypted token.

Summarising, TBS is an implementation of application component (AC) inside programmable NE, which specifically performs packet authentication for access control at multi-gigabit speeds. However, we mention future implementations of other ACs that will perform QoS for the purpose of providing deterministic communication in grid networks (e.g., processing physics experiment data), will assure multi-level security for military networks, will provide robustness in redundant networks.

B. Binding token to applications

In our implementation, the binding of token to an application is done on a per-socket basis in order to offer the fine granularity requested by grid applications. Working at socket granularity allows binding distinct streams in the same application to distinct network services. In order to support the token insertion into the optional field of IPv4 packet it is needed to modify the vanilla Linux kernel[1]. This modification exposes the token to socket binding mechanism through the *setsockopt()* function of the socket API.

From a practical point of view, it would not be accepted an approach where all the applications need to be modified and recompiled to benefit of a specific token based network service. In order to provide seamlessness integration of gTBN

[1]The patch can be retrieved at the address: http://svn.cristomatics.eu/ixp2xxx/token_patch

we developed a middleware using an interposition system such as presented in [11]. An interposition system extends the default socket's function without the need to recompile the application. However, making an unique interposition system working for all imaginable application's scenarios is not possible due to the large variety of socket's usage mechanisms. To overcome this limitation we preferred to support different classes of interposition behaviour, called hijackers. These hijackers are installed in the hosts and automatically selected at application start-up via a rule matching algorithm. The matching algorithm allows users and grid administrators to finely tune how the applications' traffic is tokenised by selecting the proper hijacker. We currently support the following entries in the selection rules: application name, domain, protocol, source/destination ip and port. The rules are checked on a per socket basis; when a rule matches the associated hijacker is started and bound to the socket. We have developed three different hijackers:

- *hijacker_tokeninjector*, in which each socket that matches the activation rule is bound to a token unique for the application. The token is retrieved from an environment variable.
- *hijacker_simpletokenizer*, in which each socket that matches the activation rule is bound to a token that is retrieved from a third party supervisor (see Section IV-B). The token is unique for a pair of *source-ip/destination-ip*. This hijacker supports a per-node traffic management.
- *hijacker_magiccarpet*, in which each socket that matches the activation rule is bound to a token retrieved from a third party supervisor (e.g., a web-server). The token is unique for a pair of *source-ip:port/destination-ip:port*. Such a hijacker allows to differentiate, at the networking level, the different data streams of one application; this behaviour is needed in order to support ftp or GridFTP applications that may use distinct parallel socket streams.

The implementations of these three hijackers are used to evaluate several common applications used in grids, as shown in Section IV-B.

IV. Evaluation and usage

The validation of the approach follows a bottom-up pattern. First we evaluate the performances of Token Based Switch (TBS), our current implementation of gTBN based on the IXP2850 network processor. Then we evaluate the robustness of the middleware by testing commonly used grid applications. In the end we present a use-case in which we applied gTBN to implement a domain firewalling solution for Grids.

A. Token Based Switch benchmark

In order to benchmark our TBS implementation, we built a test bed, as illustrated in Figure 7, composed of four hostPCs (das1 ... das4) interconnected through a TBS. We run the following scenario: das2 send tokenised traffic (generated by iperf tool as shown in Figure 8.①) to das3 through the TBS and at the same time, das1 sends traffic to das4, but this traffic is not tokenised and hence, it is rejected by TBS (see Figure 8.③). Such scenario simulates a case when external 'un-authorised' traffic tries to pass or overload a TBS. We measure the effects of such scenario by monitoring the throughput reported by iperf tool on the tokenised traffic.

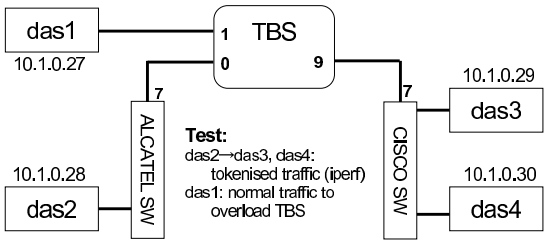

Fig. 7. Test bed setup for testing TBS.

As shown in Figure 8.②, there is no significant influence on the TBS throughput due to the injection of un-authorised traffic. We notice that we can increase the relevance of the evaluation by injecting 'real' traffic such as including random packet sizes, tokenised, un-tokenised, and invalid tokenised. While we could not perform such tests at the moment due to lack of professional traffic generators running at multi-gigabit speeds, in [9] we estimated the outcome by using the Intels cycle accurate IXP simulator. The bandwidth correctly processed by a TBS implemented on the dual IXDP2850 development platform is around 2.5 Gbps.

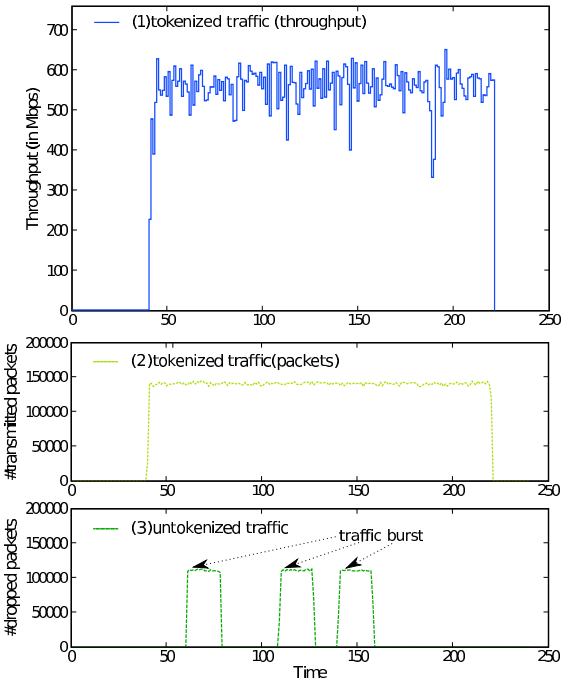

Fig. 8. Cross-domain communication between Das2-Das4 begins at 40s and ends at 300s. The traffic is successfully accepted and transmitted. Additionally, un-tokenised traffic is generated between 80s and 190s and dropped.

B. Interception middleware robustness

We evaluated the ability of our interposition environment to bind applications to tokens, implemented as described in Section III-B, by investigating the behaviour of different applications as regard to their socket behaviour. We run the applications over the interposition system and checked with tcpdump tool that the applications have their traffic properly tokenised. We repeated the tests on a large panel of real-life classes of applications used in grid: client-server, message passing, and data streaming, described as follows:

- **client/server:** A server waits for incoming connections and starts a user specific session when a client connects to it. This behaviour is typically met in case of file transfer applications. The best way to support these kind of applications is to use the *hijacker_magiccarpet* as this hijacker is the only one capable to associate to each of the connected client a unique token based on the pair of end-point of the data channel. We successfully tested the following ftp servers: muddleftp and vsftp with the following clients: netkit-ftp and gftp. We also successfully tested the grid-ftp application from the globus toolkit.
- **message passing:** Message passing libraries are important in grids as they provide a widely used distributed programming paradigm for scientific application. We used the *hijacker_tokeninjector* to bind the token to each of the IPv4 sockets created by the OpenMPI library. By tagging the whole traffic of a distributed application we were able to route the messages through reserved network links, thus providing guarantees on the quality of service.
- **streaming:** The streaming applications, like video streaming or continuously reported data from sensors in live scientific experiment are seldom presented in grids due to the impossibility to guarantee on the quality of service of grids network. However, we experienced the VLC real-time video streaming software with the *hijacker_tokeninjector* and iperf for high bandwidth data streaming. Iperf was run with '-P' option for the purpose to check the hijacker behaviour with sockets used by a typical multi-threaded application.

All of these tests show that it is possible to deploy a working grid environment using a controllable interposition system as a network component (NC) proxy object between applications and network. The impact of this environment on the network performance of the applications is only visible during the socket creation and connection stage.

C. Using gTBN for domain firewalling

It is common to have grids interconnected by high-speed backbones, but in many cases they also offer connectivity to Internet. In this case, it is necessary to use a protection mechanism such as Firewalling, VPN, or dedicated connections to isolate and guard the clusters from undesired users.

As a use case, we describe a simpler solution to the existing firewall problem for grid applications (e.g., GridFTP) that open multiple ports dynamically and make difficult or impossible to filter. Our solution based on gTBN is different than existing solutions by not using any protocol related information on the traffic that must pass the firewall, but it rather uses encrypted tokens built-in the packets based on which the packet is authenticated to pass the firewall.

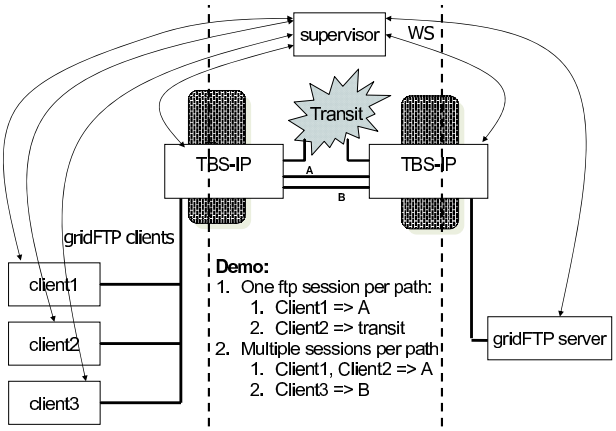

Fig. 9. A firewall setup for grids.

Figure 9 shows the case of two distinct network domains where each is located behind a firewall being interconnected by a public network and dedicated connections (e.g., lightpaths). Each firewall consists of a *TBS* being provisioned by an authority: a web-server called *supervisor*. The test bed also includes four host machines interconnected as follows: three GridFTP clients located in one domain and one GridFTP server placed on the other domain.

The middleware we installed on all hosts uses the *hijacker_magiccarpet* interposition environment in order to tag the traffic of the GridFTP applications. When one host starts a GridFTP client application in order to connect to the GridFTP server, it first gets an authorisation ticket from the supervisor. The authorisation ticket contains a unique identifier for the requested connection, the so-called Global Resource Identifier (GRI) as illustrated in Figure 6, and a TokenKey needed to encrypt part of each outgoing packet in order to obtain an encrypted token. Second, each outgoing packet that belongs to the socket(s) opened by the GridFTP application gets a tag. The tag is composed of two parts, which are concatenated as follows: (1) *GRI* and (2) the encrypted token obtained as a cryptographic result performed over the packet as shown in Figure 6. Next, the traffic passing the TBSes is checked by looking at the tag each packet carries. If the packet is authenticated to pass the firewall, then it will go out to the provisioned path towards the GridFTP server. A similar scenario of traffic tokenising happens on the way back from server to client. Note that when the supervisor received a request for path-setup from client to server, it has sent an authorisation ticket to all systems involved in the connection: both TBSes and server.

Summarising, using this test bed we investigated the data flow and ensure that the un-tagged packets from one domain are rejected at the entry point of the other domain and hence,

only an authorised application correctly bound to a token can enter into a foreign domain. This use case was presented in a live-demo at OGF23 as an alternative firewall solution for grids that authenticates the traffic at the granularity of applications regardless of their distributed hosts. To our knowledge, such a fine-grain authentication is not possible using VPNs and is difficult to achieve with authenticating firewalls where the user application must register its flows from all nodes beforehand.

V. Related work

The need for access control, application-specific services, or QoS support for high performance e-science applications has lead to programmable networks and, more recently, to high performance networks with advanced QoS and path configurability. We consider two main topics related to gTBN. First, current strategies and technologies to increase network flexibility within the existing TCP/IP models. Second, the history of programmable networks from active networks to programmability in ATM networks.

A. Introducing flexibility in networks

Currently, there are several network control technologies to help in building network communities at various network communication layers, some of which combining the flexibility of programmable networks with end-to-end provisioning. Furthermore, some recent hybrid solutions also offer cross-layer circuit provisioning. In order to refer the related work in our context, we categorise the existing network technologies with their control features in three classes: (1) embedded, (2) overlay, and (3) hybrid.

The embedded class consists of all network control mechanisms based on embedding specific information (e.g., tags, labels) into the traffic. One example of the embedded approach is Provider Backbone Traffic (PBT). PBT [12] is a VLAN-based solution enhanced to provide a good degree of control over a cluster network as well as on their interconnection backbones. Another example is Multi-protocol Label Switching (MPLS). MPLS was initially developed to speed up switching time by provisioning route decisions in advance, but is now widely used to establish support circuit switching on packet routed networks, additionally with traffic engineering. It may connect Ethernet or optical circuit-based clients with IP packet-switching clients. Moreover, amongst the latest achievements in the control of IP networks are provided by the extended MPLS solutions to support multi-domain networks based on a common policy system, the so-called generalised MPLS (GMPLS).

The overlay class includes all technologies that offers network services by encapsulation over existing network technologies, such as IP. Virtual Private Networks (VPN) is a well known overlay solution to connect private networks or applications through public, untrusted networks. (VPN) based solutions offer authenticated connections over IP. Furthermore, there were past attempts to build dynamic VPNs with fine-grained control down to the low-level network resources, such as the work in [13]. Another overlay network on top of IP, TOR [14], encrypts and obfuscates routing and connections for building an anonymous network.

The hybrid class contains solutions, which combine the existing technologies into one hybrid framework. Hybrid solutions try to achieve the advantages of both circuit switching and packet routing technologies. Examples of such solutions are, UCLPv2 [5], V-STONES [15] and DRAGON [16].

Although the above classified technologies provide network control and dynamic provisioning of networks, they all limit to end-to-end connections. However, management of dynamic communities is a topic recently tried with MPLS-based VPNs in [17].

In addition to the three classes of control, the middleboxes in a data centre (e.g., firewalls, load balancers) offer a different form of control. Firewalls protects private domains from malicious usage by filtering traffic, only allowing specific ports or types of connections. A firewall needs to be well instructed on the traffic it needs to pass or deny. This is specifically noticeable with packet filtering firewall and statefull firewall as they do block/allow access based on specific protocol-related information of the traffic (e.g., port numbers). In order to facilitate the users to bypass such firewalls several traversal techniques are implemented in standard middleware like in [18]. Authenticated firewalls, such as NuFW [19] and authpf [20], overcome some of the limitations of protocol based firewalls by introducing authentication prior to allowing access. We also notice a recent effort in building of more flexible and easier to deploy middleboxes by using a policy-aware switching layer (PLayer) [21]. This PLayer implements the translated policies as specified by an administrator and consists of specific routing tables and switch instructions. Although using PLayer in dedicated *pswitches* ensures the correctness of a certain possibly complex setup, this approach limits to an existent set of switch capabilities and still use the complex traffic classification on checking various protocol header fields.

B. Programmability

Instead of providing flexibility within the context of TCP/IP, another approach is to consider the network itself as programmable. Active networks is the most notable result of the efforts to develop programmable network architectures. The first trials to build active networks date since the ATM age in 90s, when there was a lot of work involved such as the Tempest project [2], Capsules [4], elastic networks [22], SwitchWare [3], xBind. They tried to introduce QoS into ATM networks by enhancing the networks with programmable devices that would process traffic or would self-program based on built-in tags or programs, respectively. Unfortunately, the research on active networks diminished considerably after the year 2000. Around the same time, optical networks gave enough bandwidth and brought different mechanisms for QoS through the end-to-end lighpaths provisioning. In the end, active networks have not been adopted as a solution to increase flexibility in networks.

In recent years, new developments have led to additional research and new application domains, such as building dynamic communities in networks for the purpose of sharing resources in a secure manner. In the post ATM era, Kindred and Sterne [23] were among the first to describe and address the problem of building dynamically secure network communities over public Internet, though their solution offered a coarse-grain granularity of community members at the level of firewall-protected domains. However, the authors experience shows the difficulty to build an effective community over existing organisations which have different policies and cultures and hence, different ways to distinguish the members from non-members. In other words, communities need a simple way for legacy applications and users to recognise their membership in an existing network infrastructure. Moreover, in the context of grids, a community builds up at the granularity of group members (e.g., user applications or jobs) and eventually different levels of group membership.

VI. Conclusions and future work

The starting point of this work is our believe that in the context of grids, users and communities need the freedom and flexibility to implement their specific network services. To reach this target we introduced the generalised Token Based Networking (gTBN) architecture that combines the programmability of the User Programmable Virtualized Network architectural framework with Token Based Networking. The proposed reference implementation is based on programmable network processors and commodity PCs, and employs an interposition system for seamless integration of gTBN with existing grid applications. A gTBN test bed was set-up and its performance evaluated through a firewalling use case. We also challenged the robustness of the interposition system by executing a set of complex standard Grid applications on our gTBN test bed. These experiments convinced us that multi-gigabit programmable networks is an achievable target for today's Grid networks.

We consider this work as a first step on the road towards a complete Grid implementation of the User Programmable Virtualized Network architectural framework. While still in its infancy, we are now investigating using gTBN for fine-grained access control in dynamically switchable lightpaths of Starplane, the interconnection network of the distributed computing cluster DAS-3 [24]. Furthermore, we are developing application components that implement more advanced network services like QoS and multi-cast.

Future extensions of this work will include improved resource allocation and brokering, and will enable integration of gTBN into the big picture of Grid's middleware resource and service management, such as VLAM-G [25].

Acknowledgements

The authors wish to thank to Herbert Bos and Yuri Demchenko for their useful comments. This work was supported by the EU IST-034115 Phosphorus project.

References

[1] D. L. Tennenhouse and J. M. Smith, "A survey of active network research," *IEEE Communications Magazine*, vol. 35, pp. 80–86, 1997.

[2] J. van der Merwe, S. Rooney, L. Leslie, and S. Crosby, "The tempest-a practical framework for network programmability," *Network, IEEE*, vol. 12, no. 3, pp. 20–28, May/Jun 1998.

[3] D. Alexander, W. Arbaugh, A. Keromytis, and J. Smith, "A secure active network environment architecture: realization in switchware," *Network, IEEE*, vol. 12, no. 3, pp. 37–45, May/Jun 1998.

[4] D. L. Tennenhouse and D. J. Wetherall, "Towards an active network architecture," *Computer Communication Review*, vol. 26, no. 2, 1996.

[5] E. Grasa, G. Junyent, S. Figuerola, A. Lopez, and M. Savoie, "Uclpv2: a network virtualization framework built on web services," *Communications Magazine, IEEE*, vol. 46, no. 3, pp. 126–134, March 2008.

[6] Global Environment for Network Innovations, [Online; accessed 13-August-2008]. [Online]. Available: http://geni.net

[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[8] R. J. Meijer, R. J. Strijkers, L. Gommans, and C. de Laat, "User programmable virtualized networks," in *Proceedings of the 2nd IEEE International Conference on e-Science and Grid Computing*, 2006.

[9] M. L. Cristea, L. Gommans, L. Xu, and H. Bos, "The token based switch: Per-packet access authorisation to optical shortcuts," in *Proceedings of IFIP Networking*, 2007, pp. 945–957.

[10] Y. Demchenko, A. Wan, M. Cristea, and C. de Laat, "Authorisation infrastructure for on-demand network resource provisioning," *Proceedings of the 9th IEEE/ACM International Conference on Grid Computing*, Sept. 2008.

[11] D. Thain and M. Livny, "Parrot: Transparent user-level middleware for data-intensive computing," in *Workshop on Adaptive Grid Middleware*, 2003.

[12] G. Goth, "Major players battle over layers: Layer-2 and layer-3 vendors tussle over metro links," *IEEE Internet Computing*, vol. 11, no. 5, pp. 7–9, 2007.

[13] R. Isaacs, "Lightweight, dynamic and programmable virtual private networks," *Proceedings of IEEE 3rd Conference on Open Architectures and Network Programming OPENARCH*, Mar 2000.

[14] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: the second-generation onion router," in *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, 2004, pp. 21–21.

[15] W. Sun, G. Xie, Y. Jin, W. Guo, W. Hu, X. Lin, M.-Y. Wu, W. Li, R. Jiang, and X. Wei, "A cross-layer optical circuit provisioning framework for data intensive ip end hosts," *Communications Magazine, IEEE*, vol. 46, no. 2, pp. S30–S37, February 2008.

[16] T. Lehman, J. Sobieski, and B. Jabbari, "Dragon: a framework for service provisioning in heterogeneous grid networks," *Communications Magazine, IEEE*, vol. 44, no. 3, pp. 84–90, March 2006.

[17] C. Phillips, J. Bigham, L. He, and B. Littlefair, "Managing dynamic automated communities with mpls-based vpns," *BT Technology Journal*, vol. 24, no. 2, pp. 79–84, 2006.

[18] J. Maassen and H. E. Bal, "Smartsockets: solving the connectivity problems in grid computing," in *Proceedings of the 16th international symposium on High performance distributed computing*. ACM, 2007.

[19] NuFW - An Authenticating Firewall, 2008, [accessed 06-August-2008]. [Online]. Available: http://www.nufw.org

[20] OpenBSD AuthPF, 2008, [accessed 06-August-2008]. [Online]. Available: http://www.openbsd.org/faq/pf

[21] D. A. Joseph, A. Tavakoli, and I. Stoica, "A policy-aware switching layer for data centers," *SIGCOMM*, vol. 38, no. 4, pp. 51–62, 2008.

[22] H. Bos, R. Isaacs, R. Mortier, and I. Leslie, "Elastic networks: An alternative to active networks," *Journal of Communications and Systems*, vol. 3, no. 2, pp. 153–164, Jun. 2001.

[23] D. Kindred and D. Sterne, "Dynamic vpn communities: Implementation and experience," *discex*, vol. 01, p. 0254, 2001.

[24] The Distributed ASCI Supercomputer 3, [accessed 13-August-2008]. [Online]. Available: http://www.cs.vu.nl/das3

[25] V. Korkhov, D. Vasyunin, A. Wibisono, V. Guevara-Masis, A. Belloum, C. de Laat, P. Adriaans, and L. Hertzberger, "Ws-vlam: towards a scalable workflow system on the grid," in *WORKS*. ACM, 2007.