Negotiating On-Demand Connectivity between Clouds and Wide Area Networks

Hareesh Puthalath
Ericsson Research
Stockholm, Sweden
Email: hareesh.puthalath@ericsson.com

João Soares
Portugal Telecom Inovação
Aveiro, Portugal
Email: joao-m-soares@ptinovacao.pt
(joint first author)

Azimeh Sefidcon
Ericsson Research
Stockholm, Sweden
Email: azimeh.sefidcon@ericsson.com

Bob Melander
Ericsson Research
Stockholm, Sweden
Email: bob.melander@ericsson.com

Jorge Carapinha
Portugal Telecom Inovação
Aveiro, Portugal
Email: jorgec@ptinovacao.pt

Márcio Melo
Portugal Telecom Inovação
Aveiro, Portugal
Email: marcio-d-melo@ptinovacao.pt

***Abstract*—Infrastructure as a Service (IaaS) provides the capability to deploy infrastructure on demand, but today there are gaps in the IaaS model for network connectivity, especially when user and/or application require guaranteed connectivity between the deployed infrastructures. This problem is aggravated when virtual infrastructures are geographically distributed, for performance and redundancy reasons. Connectivity is also crucial to acquire and integrate virtual resources in remote cloud datacenters with on-site IT resources. In enterprise environments where connectivity requirements cannot be met by best effort networks like Internet, such distributed infrastructure needs to make use of VPN services for inter-domain connectivity, provided by service providers over their trusted networks. To establish this connectivity, information regarding the interconnection link, such as interfaces and underlying protocols need to be negotiated and agreed upon. Today there is a lack of a technology independent mechanism and interface to perform such negotiation. This is a hindrance to automation of on-demand connectivity. In this paper we present a protocol for dynamic negotiation of connectivity service between domains. We describe how it has been used for negotiations between OpenStack and OpenNebula cloud datacenters and MPLS-based WAN, for establishment of L2/L3 VPN services.**

I. Introduction

Until very recently the network component of Cloud Computing was the most neglected one, however its importance is highly recognized today due to its fundamental role in guaranteeing performance, reliability and security in the cloud.

Today from an administrative standpoint the notion of domain refers to a collections of recources involving hardware and software managed by a single entity, e.g. a data center, or a communication network.

The network component of the cloud may span one or several domains. In other words, we can look at cloud networking within a data center (single-domain), or go further and and bring the Wide Area Network (WAN), i.e. network operators, into the picture (multi-domain).

Infrastructure as a Service (IaaS) provides the possibility for deploying infrastructure on-demand but there are limitations when user and/or application relies on guaranteed connectivity between the deployed infrastructure that is spread across geographically distributed sites or domains. This lack of reliability and performance guarantees over the WAN is today one of the main obstacles against the widespread use of cloud services, particularly in the enterprise market sector. Enterprises usually rely on managed network services such as Virtual Private Networks (VPN), rather than the public Internet (best-effort model), and there is no reason to believe that future cloud services will require a lesser degree of reliability and performance guarantees from the network. However, the current establishment of these services is done through static processes that require a significant amount of manual effort involving network administrators at different domains. Moreover, reconfigurations of these services are supposed to be relatively infrequent.

In order to cope with the cloud, future network services will certainly require on-demand and self-provisioning properties allowing its deployment across multiple domains. However there are challenges on how to negotiate on-demand connectivity between domains. We present in this work a protocol for the dynamic negotiation of connectity services between domains. We describe how the protocol fits the needs and how it can be integrated in current technical solutions.

The paper is organized as follows. Section II highlights the most relevant work and trends in the area. Section III starts with the motivation of our work, a generic use-case description followed by a high-level architectural framing. Section IV explores the interdomain connectivity aspects. It points out the need for on-demand negotiation, presents a technology independent mechanism for on-demand link negotiation. Section V details how the integration/implementation of the protocol can be done. Moreover, it describes a real testbed set-up, using cloud management solutions such as OpenStack and OpenNebula along with MPLS-based WAN. Finally, in section VI conclusions are drawn and some future work is pointed out.

978-1-4673-2798-5/12/$31.00 ©2012 IEEE

II. Related Work

The importance of the network role in the cloud service delivery is increasingly evident. Recent trends and evolvements show that academy, industry, standardization bodies and the cloud market itself are alert and active on the subject.

A. Standardization bodies and industrial efforts

Both standardization bodies and enterprises have been looking into this subject. AT&T and the University of Massachusetts released one of the first studies highlighting the need for more comprehensive control over network resources and security on the cloud service delivery, specially in the enterprise market sector [1]. More recently [2] was published, exploring the on-demand bandwidth reservation for inter-Data Centers (DC) communication. Verizon performed an initial study on the extension of VPNs for Private Clouds and an Internet Engineering Task Force (IETF) Internet-Draft was released [3]. IBM recently published [4], in which they presented CloudNaaS, a cloud networking platform for enterprise applications.

From a standardization perspective, MEF [5] is working on a cloud project to introduce the concept of delivering private cloud services via Carrier Ethernet in WANs. MEF uses cloud broker, an entity that manages the use, performance and delivery of cloud services and negotiates the relationships between cloud service providers, cloud customers and Ethernet based carriers. MEF's cloud broker is the peer of NIST [6] cloud broker that performs the same negotiation and management between cloud service providers and cloud consumers. We consider our work to be very much in alignment with the objectives of both these projects. Open Grid Forum (OGF) is also active in the definition of the Open Cloud Computing Interface (OCCI)[7], and recently a cloud networking extension has been defined by Institut Télécom, Python Open Cloud Networking Interface (pyOCNI) [8]. Distributed Management Task Force (DMTF), the holder of Cloud Infrastructure Management Interface (CIMI), has more recently also been working on network extensions [9].

B. Ongoing projects and Academia

Ongoing EU-funded projects such as SAIL [10], GEYSERS [11] and EURO-NF [12] are examples of the strong work carried out by the european research community. [13] presents the Platform as a Service (PaaS) model for networking, in which the network is abstracted by a single router representation. In the same line of thought [14] proposes two types of network services to cope with the cloud, Network as a Service (NaaS) and Connectivity as a Service (CaaS), and an experimental platform able to deal with cloud infrastructure resources and network services are presented.

C. Related products

On the commercial side, Amazon AWS Virtual Private Coud (VPC) [15] was one of the first solution that looked to the connectivity component of the cloud, however without any interaction with network providers. More recently Amazon has made available the Direct Connect service [16], which brings network operators into the picture. Nevertheless it is important to note that this does not provide a fully automated and on-demand process. Furthermore, AT&T has introduced a VPC solution [17] and IBM offers enterprises a cloud data backup supported by Verizons VPN services. However the establishment of these services are not done either on-demand nor in a self-provisioning manner.

III. A Multi-Domain Cloud Networking Architecture

A. Motivation and Requirements

In the previous section we have drawn attention to various efforts in the cloud networking area. The research community is highly committed to leverage cloud networking, and the wide number of studies and projects are proof of it. Moreover, we have also shown that the commercial viewpoint is not absent, since there are already some initial offers.

We believe that this work is unique with respect tackling the multi-domain connectivity challenge without depending on any particular technology, where IaaS services can be provisioned on-demand and delivered across multiple domains with assured network connectivity guarantees.

B. Use case

Several use-cases could be pointed out, whether to connect users directly to the cloud (e.g. virtual desktop, online gaming) or to create distributed clouds in which cloud resources are spread across domains. In this work we focus on the latter case without aiming at any specific application/service. We take a generic use-case in which the purpose is to deploy cloud infrastructure resources, i.e. virtual machines, in different sites and connect them as illustrated in figure 1.

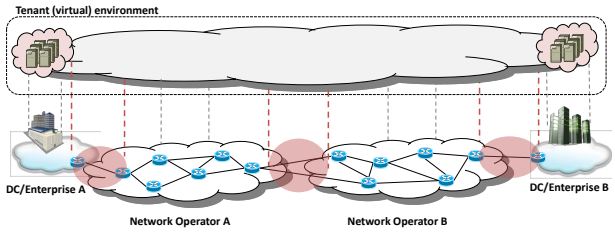

Fig. 1. Use-case

In figure 1 we can see a single tenant (virtual) environment composed by two portions of resources hosted in different DCs connected over a WAN service. Note that this single environment involves four domains. The remaining of this paper is focused on how these domains coordinate among themselves for the establishment of end-to-end connectivity.

C. Architectural Context

Whether the same provider, or different providers, are in charge of the distributed infrastructure (network and data center), there are several logical and physical entities that need to work together in order to establish the connectivity

that spans multiple administrative domains. These entities are the various interfaces, protocols, controllers and input-output processing functions that enable two or more infrastructure service providers to interact and exchange domain specific information needed for the establishment of the connection. Together we refer to them as Interdomain Coordination Framework (ICF) for distributed infrastructures.

We consider that the data centers and the networks in between belong to different administrative domains. In order to communicate between these domains, the solution includes a north bound interface as well as an east bound interface. The request for connectivity comes via the northbound interface. The actual connectivity between domains needs more detailed information such as interface addresses and agreement on routing protocols which are not available in the original request. Figure 2 shows the high level view of these entities.

A composite request (CR) represents a high level description of a Virtual Infrastructure (VI) requested by the user of this system, hereafter called tenant. This request is sent to a Cloud Infrastructure Orchestrator (CIO). The CIO will decompose the CR into a number of sub-requests for each sub-domain in a distributed infrastructure scenario. This sub-request is called a Virtual Infrastructure Request (VIR). Each domain has an infrastructure controller implementing the North-bound interface for receiving the VIR.

The request is parsed by the Message De-serialiser and sent to the Inter-domain Controller, which in essence is a finite state machine acting on the received message based on the actual state. As per defined sequence of actions in case of each message, the next event will be triggered, which might be a new message or an action to the resource management system.

The next message is built and sent out via the Message Serializer to the relevant recipient domains which have the same logical entities (Message (De)Serializers and Inter-domain Controllers).

IV. Interdomain Connectivity

A. Coordination across domains

For requesting connectivity services there is a north-bound interface that can be used by tenants. The interface supports create, read, update and delete (CRUD) operations on an abstract logical switch (for layer 2 connectivity) or a logical router (for layer 3 connectivity). This maps well onto traditional VPN service model, providing a simple and intuitive abstraction of the connectivity service. The details of the north-bound interface is outside the scope of this paper. It suffices for the reader to understand it is a RESTful interface.

To instantiate a connectivity service, a multitude of configurations need to be performed on all affected nodes in the network. This typically requires detailed knowledge about the network equipment in use and the design of the involved networks, for example, topology. Network operators typically consider these as sensitive and business critical information which should not be exposed to the tenant requesting the VI or other domains.

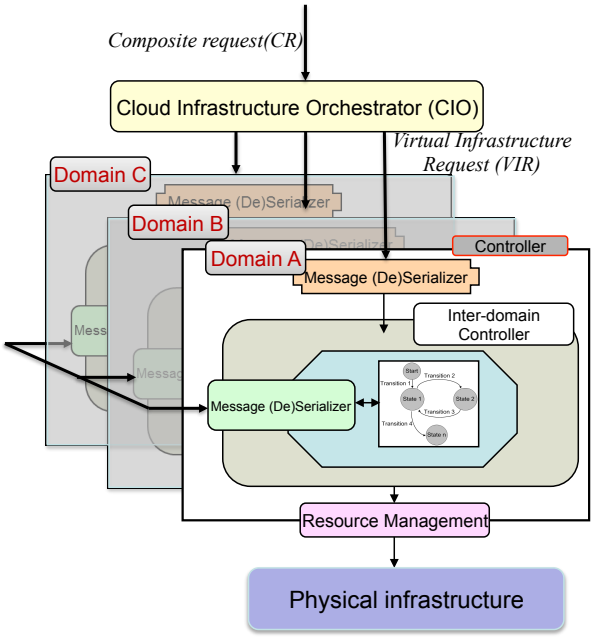

Fig. 2. Architecture

To support this separation of concerns a special interface and associated protocol have been defined allowing negotiation of configuration details. This is the east-bound interface and it is the main focus of the remaining sections of this paper.

When virtual resources, such as VMs with one or several virtual network interface cards (vNICs), are created in multiple clouds, care must be taken so that property values of those resources that must be unique are not assigned colliding values. Examples of such properties can be the MAC address assigned to vNICs, IP address assigned to vNICs or fully qualified host names.

In a static, non-virtual environment collisions are less likely to occur since there is often global coordination, e.g., IEEE assigns equipment vendor unique MAC value ranges. However, in a cloud environment where values of properties like these are generated on-the-fly, overlap is possible across domains. In lack of a global mechanism for ensuring uniqueness, coordination is necessary across the participating domains. In the design proposed here, parameter arbitration and collision resolution are part of the ICF.

B. Link Negotiation Protocol

A VI specified in a CR can span multiple administrative domains. In such cases the individual parts of this VI within each administrative domain need to be connected to its other parts distributed in one or more other domains. Naturally, link is the virtual resource that will cross domains, and therefore will make these connections. For this purpose we have defined the *link negotiation protocol*, which is implemented by the Inter-domain controller. This protocol is responsible for creating one or more virtual links belonging to the same VI

but may be spanning multiple (usually two) domains.

The design of this protocol was done with the following high-level objectives:

- Simple and low level protocol agnostic
- Support of various transport network solutions (L2, L3)
- Agnostic to any particular networking implementation

First of all it is important to highlight and define three terms:

- **Virtual Link**: A link in the virtual infrastructure.
- **Service provider Logical Link (SLL)**: A logical data transmission link from one domain to another. This is usually installed and configured by the service provider. Examples of SLL can include physical layer separation schemes like Provider Backbone Bridging or MAC-in-MAC or tunnelling schemes like IPSec or GRE. Each SLL has a reference to a physical or logical link, which in the latter case means that there might be aggregation happening below this layer at the physical links as well. It is important to note that this work is not concerned with the creation or configuration of SLLs and assumes that they are already in place. It merely uses them via references.
- **Tenant Logical Link (TLL)**: A logical link part spanning between two domains. This corresponds to a virtual link in the virtual infrastructure of the tenant and has a one to one mapping. The main difference is that a virtual link must be unique within one VI, whereas a TLL must be unique across the two domains in all VIs. For example, VLANs may be used for creating TLLs and ensuring isolation, in which case the VLAN number can act as a unique TLL. The TLL can have additional configuration parameters, such as interface addresses of the two endpoints and a routing protocol like in the case of a L3 link.The TLLs are thus virtual slices of an SLL.

The Link Negotiation Protocol offers three functions, viz. Create, Update and Delete. There is another function, route export, for the cases where there is the need to explicitly export a route to a remote domain. The operations are detailed below.

1) ***Create Function*** *:* The process of creating a TLL between two domains (referred in our description as Domain A and Domain B) is illustrated in Figure 3 via a message sequence diagram. Under the natural assumption that both domains have received two ”pieces” of the same virtual infrastructure, the element spanning the two domains is a virtual link (one or several). For that reason we refer to ”Link Request” in the diagram as the request for a virtual link that will cross and connect across these two domains.

From a high-level perspective and depending on the type of link being established, (i.e. whether L2 or L3) the *link negotiation* process can have either one or two phases. For the establishment of a L2 TLL, the process comprises of only the L2 Negotiation phase. After receiving the request, one of the domains will trigger the negotiation process. The decision about which domain takes the initiative is outside the scope of the protocol. (In practice it is usually the network operator).

TABLE I
LINK NEGOTIATION: OVERALL MESSAGE PARAMETERS

Parameters	Description
infra_id	identifier of the virtual infrastructure
transaction_id	identifier of this transaction
msg_type	identifier of the type of message
sender	identifier of the message sender
service_type	identifier of the type of service
virtual_link	Details of the virtual link: virtual link id, in and out bandwidth, TLL information

In our example we assume Domain A to be the ”initiator”. After the trigger, the ”initiator” starts the process by listing for each TLL, the various SLLs able to accommodate it.

This information is then sent to the ”receiver”, Domain B, using the *Link_Offer* message. The remaining parameters of the *Link_Offer* message can be seen in Table I. The parameters in this table are common to all messages in the protocol. *transaction_id* is used to identify this transaction among the set of ongoing transactions at any moment .The *infra_id* identifies the virtual infrastructure in question and the *virtual_link* contains the info about the virtual link crossing two domains. This information is known in advanced by both domains (via the north bound interfaces). The remaining elements, *service_type*, *in_bw* and *out_bw* (in and out bandwidth) are used in the perspective of guaranteeing consistency, since this information is expected to be known in advance by both domains.

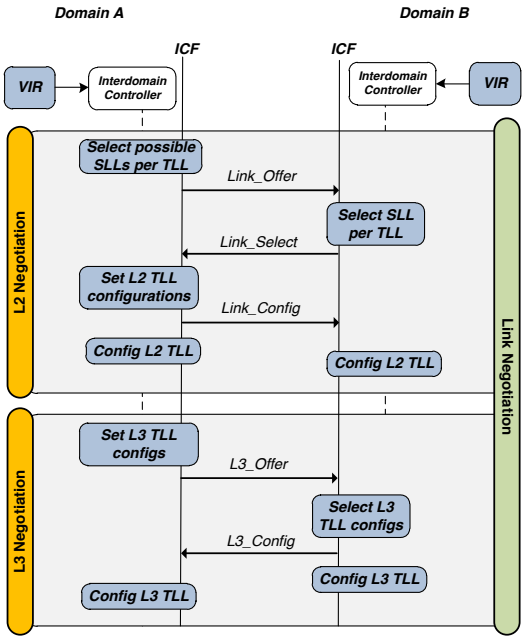

Fig. 3. Creation Function - sequence diagram

Upon receiving the list of SLLs for each TLL, Domain B selects one SLL per TLL, and sends that information in

the *Link_Select* message along with the type of encapsulation scheme used for the establishment of each TLL. Domain A is then responsible for setting the encapsulation scheme configuration attributes of each TLL, sending that information to Domain B using the *Link_Config* message. Table II shows the message parameters. At this point both domains have the necessary information to establish a L2 link.

TABLE II
LINK NEGOTIATION: PARAMETERS FOR LAYER 2 NEGOTIATION

Parameters	Description
TLL_id	identifier of the TLL
SLL_offer	list of SLLs offered to carry the TLL
SLL_id	SLL id selected to carry the TLL
encap_scheme	identifies the encapsulation scheme (type and attributes) for the TLL

If the virtual link in question is a L3 link, the process continues and goes to the *L3 Negotiation* phase. Note that the message sequence does not imply that a L3 link can be configured only after a L2 link is configured, but rather this design choice was taken to reduce the total number of messages. This avoids a new sequence with some duplicate information (like identifying the SLL) in both L2 and L3 configuration. The ”initiator” sends the *L3_Offer* message with the L3 configuration parameters, i.e. the IPs) to be configured in the endpoints of each TLL, and a list of the supported routing protocols by each TLL. The ”receiver” selects the protocol and informs the ”initiator” using the *L3_Config*. All parameters for establishing a L3 TLL(s) is now known and the configuration can be applied. The parameters for the L3 Negotiation phase are presented in Table III.

TABLE III
LINK NEGOTIATION: ADDITIONAL PARAMETERS FOR LAYER 3 NEGOTIATION

Parameters	Description
L3_config	Layer 3 configuration parameters: IPs for the link endpoints (source and destination) and a list of the supported routing protocols.

*2) **Update Function** :* Updating or reconfiguring the parameters are natural actions in the lifetime of a virtual infrastructure. Such expected actions can be triggered by a user specific update of the virtual infrastructure, or by the domain's management policies for reconfiguration. In either cases if the change in virtual infrastructure is associated to a link, then the corresponding TLLs may need to be reconfigured. When the update to a TLL is due to an update on the virtual infrastructure, the domain who initiated the TLL creation process is also responsible for starting the update process. On the other hand, when the update is triggered by an internal domain policy either the ”initiator” or the ”receiver” must be able to start the process. There are two possible scenarios, an update is requested by the ”receiver” (case A) and an update is triggered by the ”initiator” (case B). In the former, case A, the process is initiated using the *Link_Reconf* message, which is a request for reconfiguring one or more TLLs in a certain virtual infrastructure. The ”initiator” will, identical to the creation process, prepare a list of SLLs for each TLL and send it to the ”receiver” using the *Link_Update* (which in terms of parameters is equal to the *Link_Offer*). From this point on the process is similar to the creation process.

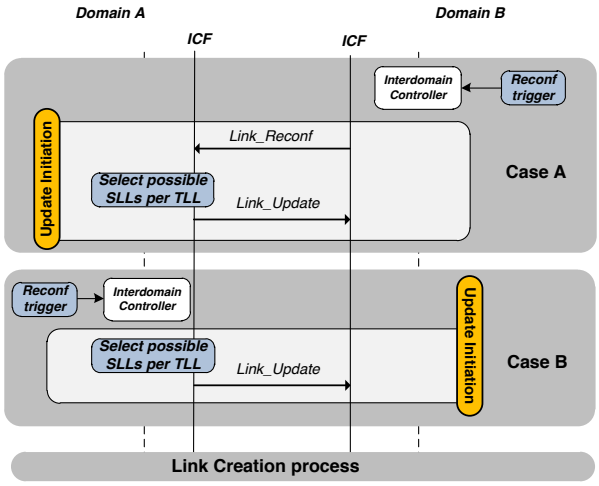

Fig. 4. Update Function - sequence diagram

*3) **Delete Function** :* The delete process for a TLL, shown in Figure 5, is always triggered by the ”initiator”. Since both domains are expected to have received the delete request, the ”initiator” domain usually takes the lead on this process. The *Link_Delete* message identifies the virtual infrastructure and the TLL(s) to be deleted.

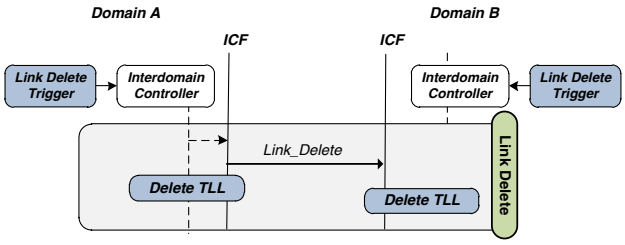

Fig. 5. Delete Function - sequence diagram

*4) **Route Export Function** :* In the case of L3 TLL(s), for some domains, the defined routing protocol maybe static. For such cases, the route export function was defined to cover the need of exporting a route to another domain. To do that the domain wanting to export routes related to a certain virtual infrastructure uses the *Route_Export* message to send one

or more routes to a remote domain. Figure 6 illustrates the process for the case in which Domain B is exporting a route to Domain A.

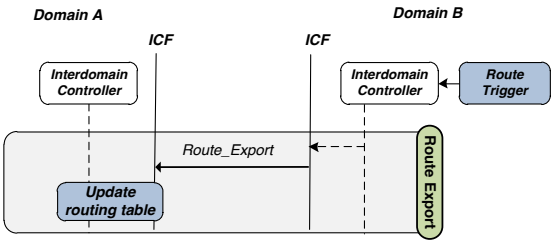

Fig. 6. Route Export Function - sequence diagram

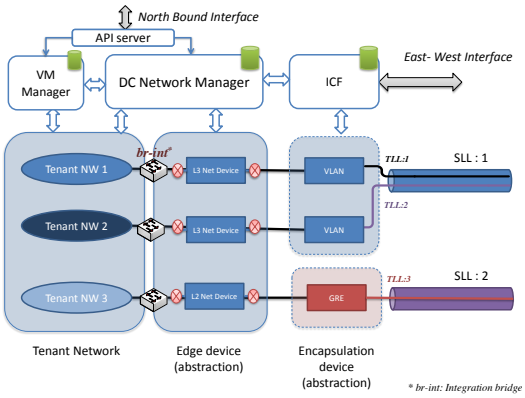

Fig. 7. DC Modules

V. Implementation & Testbed

A. Data Center implementation

Figure 7 shows the various components involved in the link negotiation and setup process in a data center. It also shows a runtime snapshot with some already configured links and their associated internal configuration as well. The data center has modules for implementing the north bound and the east bound interfaces, namely the API server and the ICF modules respectively. The DC Network Manager module is responsible for creating and managing tenant's virtual networks. The VM Manager module is responsible for creating and controlling VMs for the tenant and places them in the corresponding tenant network by interacting with the DC Network Manager module. Each has an associated database for keeping their configuration information. The DC Network Manager module together with the ICF module implements link negotiation (i.e. setup of TLL). In practice, this requires the setup of an intermediate network device for connecting the tenant's virtual network to the WAN.

The DC Network Manager module creates the interconnecting network device. In the case of an L3 VPN this means a router for routing between the two subnets. This functionality may be implemented in software (running inside a VM or a routing service) or by delegating to a hardware based router. In the case of L2 VPNs this means a layer 2 device, namely a switch or even just a link. Here also the implementation may be software based or delegated to an appropriate hardware device. Note that in both cases the edge network devices need to expose two (virtual)interfaces. One of these virtual interfaces are then plugged into the bridge corresponding to the tenant network.

Finally an encapsulation device also needs to be configured. It separates the different tenant links going into the same SLL. This is done by the ICF module and done as per the parameters agreed in the corresponding TLL. Examples of encapsulation include VLANs and tunnelling schemes like GRE or IPSec etc. The 7 shows an example using VLANs for SLL#1 and GRE for SLL#2. Note that the encapsulation type each SLL has already been pre-agreed and known to both domains and only the parameters of that specific flow are set here as agreed in the earlier link negotiation process. Here also, from an implementation perspective, both a pure software only or a combination of hardware and software are equally feasible.

Figure 8 shows the interaction among the modules based on the messages.

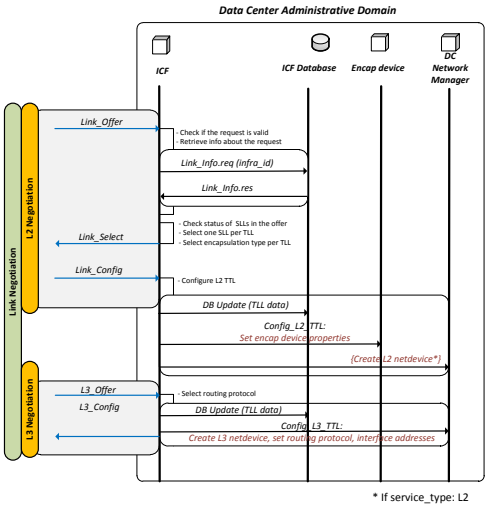

Fig. 8. Link setup - DC network functions interactions

B. WAN implemenation

Similar to the DC, the network management system in the WAN needs to be modified to incorporate the necessary modules and functions needed for link negotiation and setup. Figure 9 shows these modules. This domain follows the architectural model described in Section III-C and has north and east bound interfaces are similar to the DC.

The request for a connectivity resource (a L3 or L2 VPN) is received via the North bound interface, the controller

initiates the negotiation process with the peer domains (DCs in this case) in accordance with the link negotiation protocol (Section IV-B). After the protocol is completed, along with the parameters received via the north bound interface, the controller will have all the necessary information needed to setup the network. Then it contacts a more traditional network management system responsible for provisioning the network with all the necessary information.

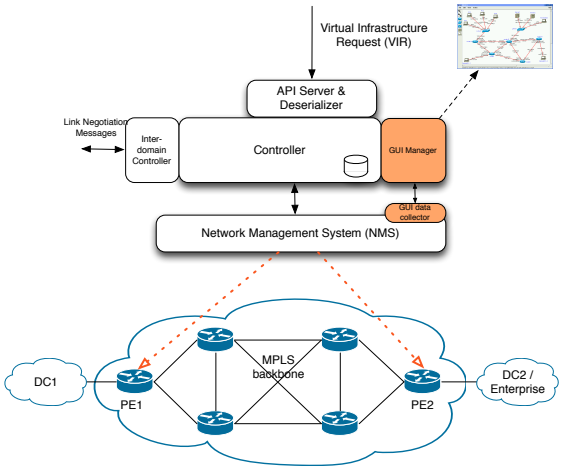

Fig. 9. WAN modules

C. Testbed

We have built a testbed composed of five domains, namely three DCs and two network operators. Figure 10 shows the high level overview of the same. OpenStack [18] and OpenNebula [19] platforms provide the base for the VM Manager and DC Network Manager functionalities in the DCs. Both platforms were extended for the ICF as described in V-A. Our OpenStack implementation is based on 'Diablo' release, while the OpenNebula implementation is based on version 3.4. The WAN testbeds use MPLS as the network technology for L2 and L3 VPN services and are managed by their associated Network Management Systems.

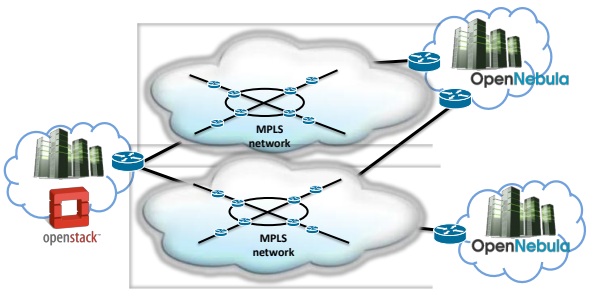

Fig. 10. Testbed

VI. Conclusion

In this work we presented a framework for negotiating on-demand connectivity between domains. This allows the creation of distributed VIs. In order to connect the individual parts of a VI lying in different domains we described a protocol for the negotiation of virtual links between domains.

Moreover we provided an overview of the framework implementation on data center and network operator domains. This general framework was implemented as part of a testbed using popular open source platforms for the management of datacenters (OpenStack and OpenNebula). On the network operator part we implemented a framework with a network management system for MPLS VPNs. Our experiments over this proof of concept show that we can indeed create a VI spanning multiple domains in an on-demand manner.

As future work we intend to present concrete experimental values with respect to the VI creation process, i.e. overall time, link negotiation protocol message exchange time, virtual machine creation time, network service establishment time, and the time for the establishment of connectivity. Also, a more thorough evaluation to verify the possibility of incorporating the proposed protocol within an existing one is necessary.

References

[1] T. Wood, A. Gerber, K. Ramakrishnan,et al, "The Case for Enterprise-Ready Virtual Private Clouds", HotCloud 09, 2009.
[2] Ajay Mahimkar, Angela Chiu, Robert Doverspike, et al. 2011. Bandwidth on demand for inter-data center communication. In Proceedings of the 10th ACM Workshop on Hot Topics in Networks (HotNets '11). ACM, New York, NY, USA, , Article 24 , 6 pages.
[3] So et al, "VPN Extensions for Private Clouds", IETF Internet Draft, February 2011.
[4] Theophilus Benson, Aditya Akella, Anees Shaikh, and Sambit Sahu. CloudNaaS: a cloud networking platform for enterprise applications. In Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC '11). ACM, New York, NY, USA, , Article 8 , 13 pages.
[5] Metro Ethernet Forum, "MEF Carrier Ethernet for Delivery of Private Cloud Services", February 2012.
[6] F. Liu, J. Tong, J.Mao, R.B.Bohm, et al, "NIST Cloud Computing Reference Architecture", National Institute of Standards and Technology U.S. Department of Commerce, September 2011.
[7] Open Grid Forum, Open Cloud Computing Interface (OCCI).
[8] Institut Télécom, pyOCNI [Online]. Available: http://occi-wg.org/2012/02/20/occi-pyocni/ [June. 15, 2012].
[9] Doug Davis (IBM), DMTF, Cloud Infrastructure Management Interface (CIMI) Model and REST Interface over HTTP An Interface for Managing Cloud Infrastructure.
[10] FP7 Project SAIL, http://www.sail-project.eu/.
[11] FP7 Project GEYSERS, http://www.geysers.eu/.
[12] FP7 Project EURO-NF, http://euronf.enst.fr/
[13] Eric Keller and Jennifer Rexford, "The "Platform as a service" model for networking". In Proceedings of the 2010 internet network management conference on Research on enterprise networking (INM/WREN'10). USENIX Association, Berkeley, CA, USA, 4-4.
[14] João Soares, Jorge Carapinha, Márcio Melo, Romeu Monteiro, and Susana Sargento, "Building Virtual Private Clouds with Networkaware Cloud", ADVCOMP 2011, Lisbon, November 2011.
[15] Amazon, AWS Virtual Private Cloud. Available: http://aws.amazon.com/vpc/ [June. 15, 2012].
[16] Amazon, AWS Direct Connect. Available: http://aws.amazon.com/directconnect/ [June. 15, 2012].
[17] AT&T, Introducing AT&T's Virtual Private Cloud. Available: http://www.att.com/gen/press-room?pid=22362&cdvn=news& newsarticleid=33844 [June. 15, 2012].
[18] Openstack, http://www.openstack.org/
[19] OpenNebula, http://www.opennebula.org/