The Future of Multi-Clouds: a Survey of Essential Architectural Elements

Ilya Baldin, Paul Ruth, Cong Wang
RENCI/UNC-Chapel Hill
{ibaldin, pruth, cwang}@renci.org

Jeffrey S. Chase
Duke University, Computer Science
chase@cs.duke.edu

***Abstract*—In this paper we present a vision of an environment composed of multiple independent cloud providers of various sizes, interconnected by programmable networks in which tenants may acquire resources from the providers and interconnect them together to serve a variety of distributed applications. Based on several years of working with GENI and NSF Cloud efforts in the US we present the essential elements of such an architecture and discuss their attributes. These elements perform functions that we claim are minimally necessary in order to realize a multi-cloud environment and include: provider and resource discovery services that rely on flexible, semantically rich description and query mechanisms, common meta-data services for maintaining information state of tenant resource allocations, and programmable interconnect mechanisms to create multi-provider Software-Defined Exchanges (SDXs) that allow tenants to control their connectivity using a declarative authorization logic.**

I. Introduction

The emergence of cloud computing as the dominant architectural paradigm has substantially affected how scientists and engineers think about the network architecture. The Internet, at its roots a distributed decentralized system designed to withstand multiple disruptions and continue to operate to deliver information, has become the means for the billions of users to reach the services of a few 'Internet-scale' providers like Google, Facebook and Amazon, whose resources are concentrated in a relatively small number of datacenters strategically located around the globe. However, we believe that this architectural pendulum is about to swing again in the direction of more distributed services. What is driving it is the rise in performance and the reduction in cost of operation of compute, storage and network resources, as well as the explosion of internet-aware small devices located everywhere around us - the so-called 'Internet-of-Things' or IoT. These devices are generating and consuming larger and larger amounts of data that require low-latency processing and fusion, thus forcing the relocation of computational and storage resources back to the network edge.

This process creates an opportunity to rethink the centralized network-cloud architecture we see today, as a widely distributed multi-provider *multi-cloud* marketplace that allows end-users to pick and choose providers based on cost, distance, jurisdiction, performance or capabilities, picking specific providers for specific tasks within their application stacks. This paper is based on experiences from more than a decade of research and development focused on creating federated widely distributed cloud environments: the development and deployment by the authors of a networked cloud testbed environment called ExoGENI[1], [2], [3], [4], itself part of a federated testbed environment called GENI (Global Environment for Network Innovations) [5] that was created by the US research community with funding from the National Science Foundation (NSF) and our work with Chameleon cloud [6], part of NSF Cloud program.

Working in these programs we had the opportunity to experiment with a variety of architectural solutions that needed to be created in order to enable the desired functionality in these distributed cloud-network environments. Broadly speaking these solutions can be broken up into three distinct categories - (I) distributed information sharing services that support resource discovery and configuration, (II) services that support programmable connectivity between various elements of the ecosystem and (III) distributed trust framework that provides tenant authorization to various services. In this paper we present our vision of what the minimally necessary functions in those categories are, we describe our work with prototype implementations and demonstrate why they are required in order to realize the vision of a multi-cloud network architecture. In Section II we provide the architectural overview and motivation for our ideas, Section III describes the first category of services, while Section IV describes the second. Section V talks about the trust and authorization framework that ties these services together. In Section VI we enumerate some of the related work done by others in this area. In Section VII we summarize our findings and outline some of the directions for future work.

II. Vision and Motivation

Our vision anticipates multiple cloud providers of varying sizes, geographic distributions and capabilities. Some may specialize in serving specific geographic locations, others may be global in nature. They provide resources of different types - bare servers, virtual machines, containers, application stacks, on-demand storage and interconnects. Cloud customers or *tenants* will want to mix and match these resources based on their needs - e.g. their and their customers' locations, required data sets, restrictions on distribution and use of the datasets, locations of devices and scientific instruments that must be connected to processing and storage.

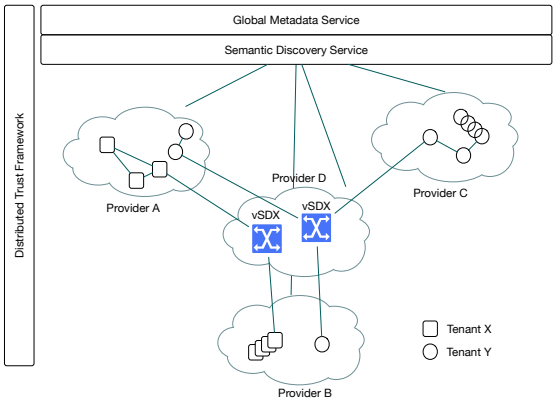

Fig. 1. Architecture Overview

One example could be a company that provides security monitoring and uses advanced real-time video object recognition using high-resolution video feeds from cameras installed on customer premises. Due to latency and privacy restrictions it may want to distribute the processing as close to the premises as possible, and may contract with multiple cloud providers for resources, while interconnecting all the provisioned resources into a single tenant network. This resource arrangement belonging to a single tenant is today commonly referred to as a *slice*. It may offer different tiers of services and may want to introduce traffic prioritization to handle different kinds of customer-related data within the interconnect.

Another example from the world of scientific research may be a distributed group of researchers and engineers that wants to deploy a scientific instrument (a telescope, a network of sensors monitoring a particular region of the Earth, etc) and analyze the large volumes of data coming off the instruments. Depending on the type of research, the processing may have both real-time and non-real-time components, e.g. monitoring the sky for unknown objects and signaling other telescopes to train their sights on a specific sector, while in the background analyzing information for other phenomena and storing it for the long term. This type of tenant may contract with multiple providers, with some resources geared towards real-time lightweight processing and others towards heavy-duty processing that fuses with other data sources and has guaranteed long-term storage, making the processed or even raw data available to other groups. A group like that may contribute their own lab or campus resources into the slice, thus relying on an interconnected combination of leased and owned resources to achieve their goals.

These scenarios are reflected in Figure 1, showing providers A, B, C and D and two tenants, whose resources are distinguished by shape. Providers A, B and C primarily focus on compute and storage resources, while provider D focuses on providing virtual interconnects (more on that later). This is the kind of multi-cloud environments we envision and have been working towards in our projects.

In order to make such environment a reality, however, there are a number of capabilities or functions that must be present and available to the tenants. In Section I we indicated they come in two flavors: *information services* and *interconnect services*. By analyzing the potential requirements of the tenants to this architecture we further break them down into:

- Discovery services - those that enable tenants to locate the desired resources and the providers offering them. Providers can publish information about available resources, while tenants can query for them via well established APIs and resource description mechanisms
- Metadata services - those that allow tenant resources in various providers to store and exchange configuration metadata about themselves or applications running on them, regardless of the provider
- Programmable interconnect services - those that allow to interconnect tenant resources across multiple providers in an authorized manner, while allowing for the interposition of tenant forwarding policies into the data plane

Clearly the discovery services and interconnect services are necessary - without them the tenants would have no way of identifying required resources and interconnecting them to create their slices. What about the metadata services? Many public and private cloud implementations today feature such services e.g. AWS metadata service [7], OpenStack metadata service [8] or AWS SSM [9]. All of them are essential for bootstrapping the tenant software stacks, however their significant drawback is, they are constrained to a specific cloud provider domain or tied to a specific provider authorization infrastructure. For launching applications across multiple domains, tenants must be able to collect and retrieve metadata from resources deployed across multiple cloud providers, hence our assertion that such is service is necessary.

We also emphasize that the discovery and metadata services must be deployed in a *provider-independent* manner, i.e. they must not belong to a single provider. They must allow tenants from any provider to connect in order to use their services and they must rely an authorization model that is separate from the specific provider authorization - not the case e.g. with AWS SSM, which treats external instances connecting to it as second-class citizens compared to instances launched on AWS.

In contrast, the programmable interconnect service can be rendered by an individual provider independent of others, focusing on interconnect resources, rather than more traditional storage and compute. Some of today's telecommunications providers may play this role once their infrastructure is sufficiently virtualized. A provider offering such service instantiates *virtual programmable interconnects* that allow individual tenants to create private networks with programmable dataplanes. Such networks may peer with commodity Internet, or remain as private enclaves, depending on the security requirements of the tenant. The capabilities of such interconnect providers should be discoverable and can use the same metadata services as any other provider.

All three of these services in one way or another must rely on a distributed trust and authorization framework that is based

on endorsements from different trusted entities - providers, information brokers, tenants, consortia etc. - all depending on specific governance models and relationship types, like commercial customer-provider relationships, scientific collaborations or government-regulation based relationships. Discovery services must scope answers to queries, depending on the level of trust between the entity issuing the query and the entity answering it. Tenant X, trusted by provider D may receive a fuller picture of provider topology, compared to Tenant Y, who may only get an aggregated picture. Metadata service must allow read and write accesses to various parts of metadata store depending on the types of principals that are trying to access it - an applications running within Tenant X infrastructure may get different levels of access of different portions of slice metadata (read vs. modify). Finally, elements of Tenant X infrastructure must be properly authorized to connect to virtual SDX in provider D; the situation gets even more complex for vSDXs that allow multiple tenants to exchange traffic *with each other* in an authorized fashion (not shown in Figure 1).

We emphasize that the services we describe in this paper are, in our opinion, *minimally necessary* to enable the envisioned architecture. Other services may provide a significant added value, however may not be necessary for the ecosystem to exist.

In the following sections we discuss the requirements and prototype implementations of the outlined services.

III. Distributed Information Sharing Services

A. Discovery Services

A discovery service must be able to answer questions about resource existence and availability. It must support a multi-cloud environment with many different providers offering a heterogeneous mix of resources. Connectivity between provider resources is itself considered a resource that must be queriable, thus requiring support for various types of *path-finding* and *topology embedding* queries, i.e. determining iso- and homeo-morphic mappings between resources topologies as described by the tenant and topologies as described by providers. It must also be able to scope its answers to the level of authorization of the querier - providing different *views* or *persistent queries* [10] of the same data, depending on the level trust.

These high-level definitions lead to further lower-level requirements - the discovery system must operate using flexible and extensible declarative resource description and query mechanisms, that on the one hand allow providers to describe the resources they have, and on the other, allow tenants to describe their resource needs.

The representation formats designed by the semantic web community (RDF - Resource Description Framework [11], OWL - Web Ontology Language [12]) represent a good starting point for the various uses we outline. They permit to represent many types of both qualitative, as well as quantitative relationships between elements of a networked system. The semantics built into class and property hierarchies add additional expressiveness. The adoption of these formats allows to move away from questions of bit representation of data on the wire or disk and instead concentrate on the more important questions of the appropriate network information models and their use. SPARQL query language [13] provides a rich framework for formulating graph queries.

Semantic descriptions with their complex hierarchies of entity classes and property relationships and standardized vocabularies act as the common abstraction layer to which all other representations can be converted. Critically, these can be extended by individual providers to define classes and properties specific to their environment. The RDFS and OWL entailments allow common resource management and topology embedding algorithms to operate on the shared common classes, thus improving their portability. Considering the main goal of our work of enabling a multi-provider heterogeneous environment, having such a common and extendable way of describing resources is a critical property. Working with other researchers we have defined several *ontologies* or vocabularies of discourse for describing cloud and networking resources [14], [15], [2].

The discovery service may answer the following types of queries to providers:

- Resource types, locations and attributes. e.g. identify resources in geographic proximity of a specific location using inferences from the provided ontologies.
- Connectivity between resources to determine common network providers, links etc.
- Specific attributes of the connections between resources
- Business and other types of relationships between resources and providers to support policy-driven tenant resource requests

Additionally, common networking computation tasks, like path computation and virtual topology mapping, can be modeled as subgraph extractions on the semantic graph [16], that we discussed in detail elsewhere [14], [17]. This allows new resource management algorithms to be built as procedural code heavily leveraging common operations abstracted as standardized queries that are independent of the programming environment and implemented efficiently in common toolsets. Using queries is motivated by similar goals as the development of database management systems to replace hardcoded file processing algorithms: i.e. enabling reuse and automatic optimization.

Rule engines (Pellet, Hermit [18], [19], [20]), typically associated with RDF storage systems can be used to perform additional processing on the resource models in a declarative (using e.g. a constrained version of Datalog), rather than procedural fashion, which makes them more portable and verifiable - a critical feature in complex distributed systems.

Using one such system [21] we implemented a rule-based semantic validity verification for tenant slice topology requests. For example, if a tenant is attempting to embed a vSDX with more than two endpoints that connects multiple provider domains, each domain must be mentioned only once. E.g. it is OK to say, 'I would like to have a connection between nodes belonging to domains A, B and C'. It is

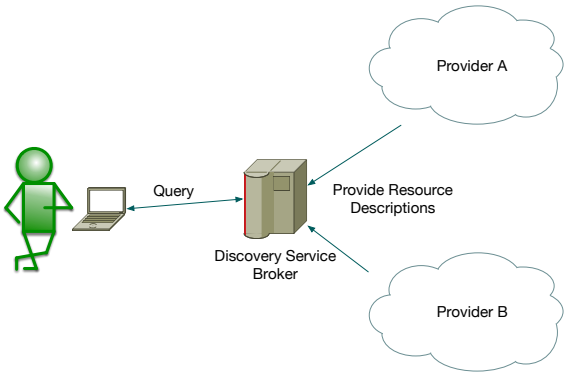

Fig. 2. Discovery with a centralized broker

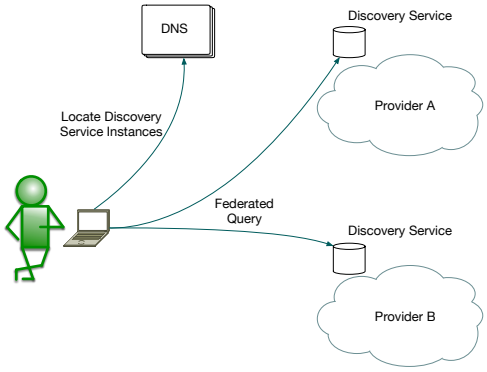

Fig. 3. Distributed discovery with federated query

NOT OK to say 'I would like to have a vSDX connection between nodes belonging to domains A, B and A', since this actually represents a poorly formed request for a point-to-point connection between domains A and B. The tenant must re-normalize the request prior to submitting. The rule expressing this constraint in Datalog is shown below:

(?Z rb:violation error("Domains in vSDX can't be repeated", ?X))
< − (?X rdf:type topo:vSDX), (?X topo:hasInterface ?I1),
(?X topo:hasInterface ?I2), notEqual(?I1, ?I2),
(?A topo:hasInterface ?I1),
(?B topo:hasInterface ?I2),
(?A rdf:type comp:ComputeElement),
(?B rdf:type comp:ComputeElement), notEqual(?A, ?B),
(?A req:inDomain ?D1), (?B req:inDomain ?D2),
equal(?D1, ?D2),
(?X topo:hasInterface ?I3), notEqual(?I1, ?I3),
notEqual(?I2, ?I3),
(?C topo:hasInterface ?I3),
(?C rdf:type comp:ComputeElement),
(?C req:inDomain ?D3), notEqual(?D3, ?D1)

Common syntactic schema-checking mechanisms (e.g. for XML schemas) fail to catch problems like these, however rule-based systems allow to express such rules in a compact, portable and verifiable form.

Architecturally, the semantic resource description information from providers to the tenant can flow through a centralized broker. This is the solution we implemented in ExoGENI [2]. It simplifies the implementation of the client and allows placing complex common pathfinding and embedding algorithms into the broker, however it also has several drawbacks - the broker becomes a single point of failure; it is a trusted intermediary, which decides the level of disclosure of resource topology information to tenants, instead of the provider; finally, when the resource state changes, the broker must be notified and updated, reducing the timeliness of the information available to tenants. This scenario is shown in Figure 2.

An alternative deployment places a discovery service agent with each provider - it operates on behalf of the provider and is trusted by it to make the right decisions with respect to scope of information that is sent in response to queries. However if a tenant is considering an embedding of their slice into multiple providers, complex *federated* queries must be issued by the client against multiple provider discovery service endpoints and it becomes the client's task to perform join or join-like operations on the returned information. This feature is defined in SPARQL 1.1 and implementations exist [22], however they suffer from performance penalties compared to the centralized approach. An additional problem that must be solved is the discovery of discovery services themselves. A tenant wishing to query multiple providers must find out service endpoints at each provider that can be queried. This problem can be addressed by using a modified DNS (Domain Name Resolution) service with custom record types, as shown in Figure 3. Ultimately both types of deployments - the broker-based and the distributed, have their place, depending on the requirements of the particular environment.

In developing the idea of semantic-based discovery services, we found that the performance of these systems can be further improved, compared to SPARQL-based systems, if we limited the set of queries to path-finding queries with *structural* and *semantic* constraints using a Tarjan path algebra [23]. *Semantic constraints* would typically involve link and node type subsumptions, that can be addressed via standard RDFS and OWL entailments. *Structural constraints* may refer to link- and node-disjointedness requirements represented by inclusion or avoidance of specific links or nodes in the query response.

Path algebras [24], [25], [26] have been proposed as a foundation for solving many networking problems. Many such algebras have focused on efficient solutions of variants of shortest path problem. However, encoding a wider range of connectivity problems requires designing a new path algebra, specific to common problems faced, in our case, by SDN controller designers. Further, existing algebras are not concerned with constraints on *semantic properties of networks* which are also critical, and require extending the algebraic approach to support them.

In [17] we explored several methods to storing and querying semantically rich data, restricting our queries to a small number of types. We evaluated query expressiveness and performance using three methods: one based on Semantic Web standards and SPARQL 1.1 query language, another

based on graph databases, specifically Neo4j and Cypher query language, and, finally the approach we developed using path algebras based on path regular expressions. We designed a path algebra and an accompanying approach for efficiently storing pre-processed topology information. Our approach lends itself to decomposition into three distinct phases - pre-processing the data, querying the data and pruning the results. By pre-processing and efficiently storing the data ahead of time we significantly reduce the query time, amortizing the significant pre-processing time over a large number of queries, thus improving the overall responsiveness of the system. We demonstrated the performance advantages of our approach for these types of queries. The novelty of this approach is in identifying common path and topology query abstractions and proposing an efficient way of storing and operating on semantically enriched abstract network topology representations that fit those abstractions.

Our ongoing work is involved in developing a system of *views* of semantic data based on authorization level.

B. Metadata Services

Metadata services must allow storing information about resources provisioned for the tenants by more providers. This information is used by the operating systems and applications running on compute resources within the slice to configure or update their behavior as the slice configuration changes. The service must allow scoping of information for easy and efficient management and it must provide granular authorization control not only for the metadata of different tenants and slices, but even within individual slices, as different principals may have different rights to read and modify the metadata. Importantly, since this service is intended to span many providers and tenants, it must be distributed and scale well.

We define multiple examples of principals that may need to have access to the metadata:

- **Users** - tenants responsible for creating the virtual systems or slices and, also other users authorized by the lead tenant to access individual elements of the slice and configure applications on them.
- **Cloud provider agents** - responsible for provisioning the infrastructure. They originate large amounts of metadata when resources are instantiated.
- **Tenant infrastructure control agents** - these could be specialized applications, like SDN controllers, charged with managing the tenant's virtual infrastructure.
- **OS configuration software in the instances** - this software may fine-tune the configuration of the running instances in a slice by consulting the meta-data.
- **Applications running inside compute instances** - applications may consult metadata created by other principals or use the metadata service to exchange information between their own distributed elements by writing and reading from the service.

The rights to create, read and write the metadata must be carefully and separately managed by the metadata service allowing the principals which create individual elements of metadata to have control over which other principals have the ability to read, write or modify their elements. This approach contrasts with traditional metadata service implementations [7], [8], which allow the tenant to create, read and write metadata for a given instance and any software on the instance has the same rights to read and modify that metadata regardless of ownership. Also, because the access to metadata is controlled based on the IP address of the instance, no other instances can access the metadata, limiting its use for configuring the behavior of distributed applications. Further in this section we provide examples of use for our metadata service.

Similarly, while common implementations, like AWS [7] and OpenStack [8] provide metadata as an unstructured blob, whose interpretation is left to the originator (in this case tenant) and the consumer (in this case software running inside the instance), our design imposes some minimal structure on the metadata. Metadata is created as a set of key/value pairs with values that can be individual strings or, more generally, JSON-like objects. The key name and the interpretation of the value of a given key is left to the originator and the consumer, which in our case can be any of the principals described above.

In order to avoid key name collisions we introduced a two-tier namespace hierarchy, with the top tier typically reserved to individual slices, and the second tier used as an additional typing discriminator. Thus each tenant gets to have a namespace for her slice (slice names are assumed unique and can be represented by e.g. UUIDs [27]), further breaking down the slice namespace by types to e.g. allow multiple applications manage their own key-name spaces within a slice or for semantic convenience. A type could be any unique string, e.g. "network" or use application name as an identifier, e.g. "hadoop".

We implemented a prototype of this cloud-based metadata service as a REST-ful service on top of Accumulo database called COMET [28], the general architecture of COMET is shown in Figure 4. We chose Accumulo as the back-end database, because Accumulo is a 'sorted, distributed key/value store that provides robust, scalable data storage and retrieval' [29] that in addition features cell-based access control, allowing each cell to have a unique credential. This per-cell credential is essential to our implementation because it provides highly flexible user control on a sub-database level. Each Accumulo entry is referenced by a key tuple <ContextID, Family, Key> which map to our notions of <Slice ID, Type, Key> described above and allows to implement a 'bearer token' type authorization for each individual Key/Value pair.

To support the full range of controls over operations on metadata, COMET uses a combination of X.509 client certificates and bearer tokens stored in Accumulo to manage the rights of various principals attempting operations on metadata. COMET defines four REST calls that roughly map into POSIX file semantics and Table I shows how the combination of certificates and tokens is used to decide on rights for each REST call.

COMET API Call	Semantics	Trusted Cert	Read Token	Write Token
write(SliceID, Type, Key, Value, ReadToken, WriteToken)	create a new key and associate a value with it	V	S	S
write(SliceID, Type, Key, Value, ReadToken, WriteToken)	write new value into existing key		V	V
read(SliceID, Type, Key, ReadToken)	read value for a given key		V	
enumerate(SliceID, Type, ReadToken)	enumerate keys in context/family		V	
delete(SliceID, Type, Key, ReadToken, WriteToken)	delete a key/value pair	V	V	V

TABLE I
COMET API CALL ACCESS VERIFICATION. V = VALIDATE, S = SPECIFY

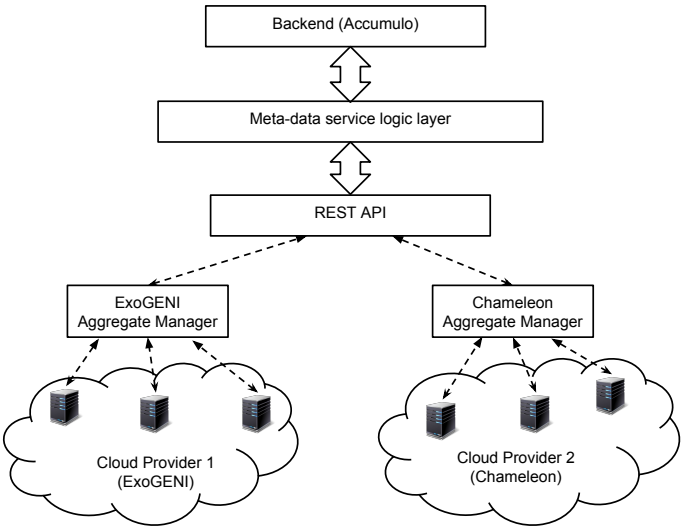

Fig. 4. Metadata Service Prototype Architecture

Note that the *write* call has two separate semantics - *create* and *modify existing*. Creating a new key and associating some initial value with it is considered a highest privilege operation, that requires trust between the principal attempting the operation and COMET. In addition, it requires that trust is established without the principal and COMET having a pre-arranged agreement on tokens. To establish this initial trust, COMET uses X.509 client certificates that can be traced to one of the trust roots configured into it. A client connecting to COMET service must present a valid client certificate in order to be empowered to create new key/value pairs. In this call the principal also must specify two separate secret tokens - a *read token* and a *write token*. The principal then is free to share these tokens with other principals as she sees fit, thus giving her complete freedom of managing access to the individual key/value pairs created by her.

In the implementation, the read token maps directly onto Accumulo cell access token, required to access a given value associated with <ContextID, Family, Key>, while the write token is stored within the value JSON block associated with the key, invisible to users of COMET. Note that tokens are additive - a read operation on a key requires only a read token, while the write-modify operation requires both a read token and a write token.

Enumerate call provides a listing of key names associated with a given <ContextID, Family> or, equivalently <Slice, Type> tuple to which a given read token grants access. Deletion, like write-create is a privileged operation, requiring a trusted certificate, in addition to read and write tokens. This prevents unauthorized deletion of keys by users who were merely empowered to modify the value of a particular key. X509 client certificates are optional for write-modify, read and enumeration operations.

A typical example of using COMET for configuring instance OS behavior may be to specify instance host name and IP addresses of interfaces. In a typical AWS or OpenStack installation DNS and DHCP owned by the provider are used for these purposes, however when a slice is created across multiple providers these solutions are not appropriate - if a tenant creates a private network spanning multiple provider sites, both these services need to be owned and operated by the tenant, which in many cases is a significant overhead. COMET simplifies these operations by offering a tenant a way to communicate this information to the instance, and a simple software agent running on the instance can then retrieve the information and configure appropriate OS services (name resolution and networking stack, in this case).

Critically the credentials and service endpoint to access COMET are communicated to the agent via traditional provider-based metadata services using a simple JSON or INI format with a few key/value pairs: comet end point URL, slice identifier and read token or tokens needed to access specific keys storing the hostname and IP address information. The provider-based metadata service in essence is used to bootstrap the more powerful COMET capability.

More sophisticated examples of using COMET include e.g. distributing SSH keys and populating automatically generated /etc/hosts files inside a cluster that is part of tenant slice, a requirement common e.g. for Hadoop or Condor management systems. A network private to the slice is used to communicate between a head node and worker nodes and requires configuration of trust via SSH and name recognition to operate properly. In this case the metadata (SSH keys, hostnames) can be generated by individual instances, written into COMET and read by other instances to populate their SSH ~/.ssh/authorized_keys files and system-wide /etc/hosts - all tasks that a traditional metadata service restricted to a single

provider cannot accomplish.

IV. Programmable Interconnect Services

A programmable interconnect supporting a multi-cloud can be thought of as a multi-tenant Virtualized Software Defined eXchange (vSDX). This model complements the functionality of a multi-tenant clouds by enabling tenants to request and manage networking resources between traditional cloud sites as well as enabling tenants to provide networking services to other tenants.

Traditional SDXs reside in a single physical location to which clients create physical connectivity. The SDX forwards layer 2 and 3 traffic between clients respecting traffic policies and agreements defined by the clients. In addition, the SDX may provide NFV services, such as security monitoring, on behalf of the clients.

In a multi-cloud environment, tenants combine resources from many cloud and network service providers. These tenants will not have the capability to create dedicated physical paths between a cloud site and an arbitrary physical SDX. Instead, a multi-cloud vSDX provider deploys virtual network connections to each cloud site it supports. Tenants can request *cloud networking resources* from the provider that connects their resources among the cloud sites, instantiating virtual exchange services. Once the cloud resources are connected to the virtual exchange, the tenant can request the vSDX to forward traffic between tenant resources at various cloud sites and even between slices belonging to different tenants.

Examples of services that multi-cloud vSDXs could provide include:

- Direct L2 circuits between sites
- Routing between subnets residing in different cloud providers
- Intrusion detection placed at the edge of the tenant network
- Performance enhancements, like e.g. multi-path, congestion avoidance or resilience
- Tenant-controlled SDN service in the vSDX - interposing complex tenant forwarding policies into the slice network

We have extensive experience both building and using ExoGENI – a testbed used for networking and distributed systems experiments. As part of our project we have produced tools for creating ExoGENI slices that provide vSDX service to attached university campuses and slices belonging to different users using the developed SAFE authorization framework, described in Section V. The vSDX hosted in ExoGENI can create trusted circuits between ExoGENI slices and selected ExoGENI *stitchports* – negotiated peering points with external infrastructure. For example we have demonstrated creating a vSDX linking ExoGENI edge resources to Chameleon NSF Cloud, thus linking resources of the two testbeds providers into a single tenant slice. The virtual SDX model enables SDX users to leverage ExoGENI's automated creation and stitching of L2 paths across wide-area dynamic circuit providers. Requests to use these stitchports is protected by SAFE authorization using exemplary access control models (groups, roles, capabilities, and principal attributes including GENI attributes and slice ownership information).

Our work focuses on advancing support for GENI-like testbeds as a petri dish to culture new approaches to security-managed network services. Our approach is inspired in part by proposals put forward by leading researchers more than a decade ago for "pluralist" network architecture based on deep virtualization. Most notably, the authors of Plutarch [30] and Cabo [31], [32] build network service providers (NSPs) as a software layer over programmable infrastructure: pipes, programmable switching points, and in-network computation. They offer a compelling vision of NSPs that manage end-to-end interoperable connectivity, riding over an architecture-neutral underlay of infrastructure providers.

Live tenant NSPs require performance assurances that are precise and strong at the foundation—the testbed infrastructure service. Recent post-GENI testbeds, including Chameleon, emphasize bare-metal control to achieve high fidelity. Our work uses ExoGENI slices provisioned using open-source virtualization (e.g., KVM), similarly to commercial IaaS clouds. ExoGENI leverages advanced circuit fabrics (I2-AL2S, ESnet) for the cross-site network backplane. Thus, slices may instantiate end-to-end network topologies and evolve them by allocating and releasing VMs and dynamic circuits, with precise resource contracts from the infrastructure providers.

With these ingredients in place, our objective is to enable testbed-hosted NSPs as tenant slices, that benefit real users—for example, to implement secure built-to-order virtual science networks that span campuses. Going further, we strive to support *inter-domain* networking experiments: traffic routing among NSPs that are controlled by different tenants and that peer with one another as in today's Internet.

Our initial efforts focus on three key elements. First, enable controllable packet flow into and among GENI slices. ExoGENI *stitchports* allow slices to establish peering links with other slices at L2 by mutual consent [33], [34]. Moreover, many university campuses have deployed SDN-enhanced edge networks that support *opt-in* redirection of real user traffic into the circuit fabrics and into locally hosted GENI edge points of presence (PoPs). Second, introduce an elastic slice controller architecture for slices to adapt their configurations over time—we might call them "software-defined slices". We recently reported on initial experiments with Ahab slice controllers and elastic NSP slices in ExoGENI [34]. Third, introduce tools for participants to authorize their peering connections, traffic flows, and related interactions. To this end we introduced SAFE: a declarative assertion and policy language—a trust logic—and certificate transport for secure logical trust [35].

Figure 5 illustrates our prototype multi-cloud programmable interconnect called ExoPlex including its structure for hosting end-to-end tenant network providers (NSPs) with in-network services, such as elastic security scanning. ExoGENI is a good foundation for these NSPs because it provides on-demand computation and programmable (virtual) switching/NFV capacity at multiple sites/PoPs on advanced network circuit fabrics linking many campuses and research centers. The

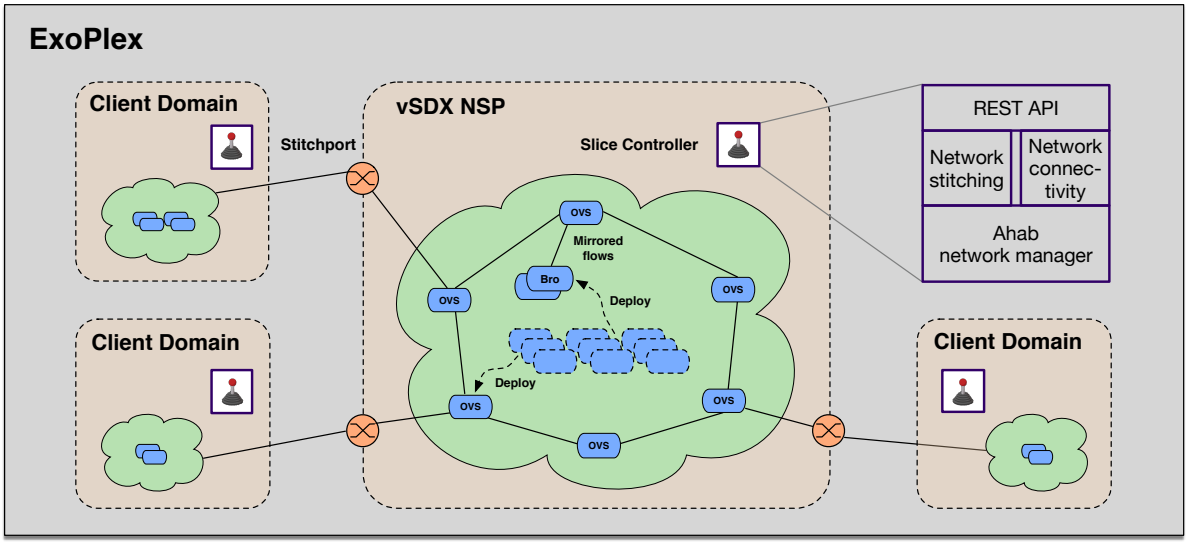

Fig. 5. The ExoPlex network service architecture. A vSDX NSP supported by ExoPlex uses an elastic slice controller to coordinate dynamic circuits and Bro security monitors via Ahab. The controller exposes REST APIs for clients to request network stitching and connectivity and uses a SAFE engine to check each request for compliance with logical trust policies.

customers of a tenant NSP may be subnets in an SDN-enabled host campus network, external networks or testbeds connected through network circuits to edge interfaces (stitchports), or other ExoGENI slices connected via direct L2 peering of the slice dataplanes [33]. For example, the prototype vSDX provides network transport service to the Chameleon testbed using ExoGENI tenant slice controller called Ahab, it also integrates an NFV security function using Bro open-source IDS system [36].

An ExoPlex NSP is a testbed slice containing a dynamic virtual network topology. Each customer of an NSP attaches to it at L2 to form a peering link between at least one pair of interfaces on their network edges, via a circuit attached to a stitchport, or (if colocated at a site) by direct peering over an ExoGENI site interconnect. The customers route traffic onto their peering links, and the NSP transports the traffic onto an egress link to the destination customer—or drops it—according to its configured policies.

V. Distributed Trust

Many of the key problems for multi-cloud environments involve managing trust. Managed trust is essential for acceptance of governing authorities (e.g., federation roots), grounding of global name spaces and address spaces, resource delegation, identity management, group membership endorsements, integrity attestations, data access control, secure connectivity and network traffic filtering, and service trust.

The multi-cloud architecture that we envision occupies a middle ground between today's centrally controlled cloud services, in which trust is based on social trust in the service provider, and "trustless" architectures such as blockchains. Outsourcing and infrastructure markets inherently involve some degree of trust in providers, and even partial trust improves efficiency relative to trustless approaches. However, to the maximum extent possible, trust should be decentralized across independent entities who are accountable for their actions, and expressed in controlled delegations that preserve the well-known Principle of Least Privilege.

We take a common approach to all of these problems by factoring trust concerns out of the implementations and into a common declarative authorization plane, based on the SAFE framework [35] and the Datalog logic language. Principals in the system issue logic assertions to represent delegations, endorsements, and policies. These statements are authenticated by signing under the issuer's keypair or by transmission via SSL-protected channels. SAFE enables participants to execute trust queries against assembled sets of logic statements to validate trust decisions relative to a policy. With SAFE, logical specifications are directly deployable in the implementation.

The design of SAFE was motivated by our experience in applying logical trust in the development of GENI. Although GENI was conceived as a network testbed, it is best understood as a federation of autonomous IaaS providers ("aggregates") linked by various trust relationships and agreements. GENI serves a community of registered researchers with various institutional and project affiliations. Each provider has various policies governing client access. These policies consider endorsements and delegations of trust among the participants, including a root trust anchor that certifies the aggregates and various *authority* services to govern membership and coordination. In this respect GENI is representative of multi-cloud systems in general, although there are differences in terminology.

SAFE synthesizes elements from previous trust logic systems and extends them with additional system support to enable practical deployment. The novel elements of SAFE include a scripting language to insulate applications from logic concerns, and an interface to a shared key-value store (e.g.,

a DHT), which stores authenticated logic content as signed certificates in a native SAFE format. Certificates in the store are indexed by self-certifying links, and can be written only by their issuers. The application trust scripts contain parameterized logic templates to generate certificates easily, and also to link certificates to construct DAGs programmatically as a side effect of delegations.

The linking mechanism simplifies discovery and retrieval of the content relevant to a trust decision. The certificate links also enable pass-by-reference and caching of certificate content at the authorizers. The shared certificate store enables an issuer to update or revoke its certificates by their tokens, addressing common PKI concerns.

The three types of services described: discovery, metadata and programmable interconnect, must rely on a distributed trust and authorization framework such as SAFE in order to fulfill their functions. Recall our assertion that these services must rely on authorization mechanisms external to any provider. The programmable interconnect service described in Section IV is already integrated with the SAFE prototype: each instantiated vSDX uses a tenant-specified SAFE policy (written in Datalog) to decide how the various tenants connecting to it can exchange traffic with each other [35].

Integrating the discovery service and the metadata service with SAFE remains the area of ongoing work for us. We envision SAFE policies being used to define the different scopes of disclosure of provider topology to the tenant queries. Similarly, we anticipate using SAFE policies to determine access to various parts of metadata - moving away from bearer token based authorization we use today to more complex policies that e.g. allow for delegation of rights to certain parts of metadata. In the Accumulo-based COMET implementation the delegation is implicitly implemented by passing the read or write tokens from one principal to another, however it has one significant flaw - once the token is given by the originating principal to another, the original principal has no control over who may get their hands on that token. A principal given a token may decide to pass it to another principal and so on, and this act requires no permission from the original principal. A SAFE access policy would allow us to precisely control the degree of delegation between principals simply based on their identity, without the need to pass tokens around. Thus we see SAFE trust framework as an important pillar of the multi-cloud architecture described in Figure 1.

Beyond the described services the SAFE trust framework that is independent from individual providers can serve as a foundation for a large number of other multi-cloud services, making their deployment significantly easier, and their authorization policies more expressive.

VI. Related Work

Coupling with the rapid growth of cloud computing, the federation of cloud services [37], [38], [39] is becoming a widely discussed topic. To date, most of the existing research works target the same goal of utilizing multiple cloud infrastructures to achieve better reliability, resource provisioning and application scalability. In this section, we review the related work on intercloud architecture from the cloud provider's perspective, which is coarsely divided into global intercloud services, semantic descriptions and software defined exchanges. We also discuss relevant work in distributed trust systems.

Global Inter-cloud Services A paradigm shift is in progress in favor of Intercloud Computing. For instance, 20 approaches to this new challenge are presented in [37]. Within this context, Manno et al. proposed the use of the semantic Federated Cloud Framework Architecture (FCFA) [40] to manage resource life cycles based on formal models. In contrast, the Intercloud architecture developed within the Institute of Electrical and Electronics Engineers (IEEE) Standard for Intercloud Interoperability and Federation (P2302) [41], [42] Working Group uses graphs to describe and to discover cloud resources based on the existing Open-Source API and Platform for Multiple Clouds (mOSAIC) [43] ontology. Both approaches are being considered as domain-specific extensions to our work. In addition, Santana-Prez et al. [44] proposed a scheduling algorithm that was suitable for federated hybrid cloud systems. The algorithm applies semantic techniques to scheduling and to matching tasks with the most suitable resources. The information model is based on the Unified Cloud Interface (UCI) project ontologies, which cover a wide range of details but which cannot handle Intercloud systems. Le and Kanagasabai [45], [46] also proposed ontology-based methodologies to discover and to broker cloud services. They use Semantic Web technologies for user requirements and for cloud provider advertisements, and then apply an algorithm to match each requirement list to advertised resource units. Multiple levels of matching are defined, ranging from an exact match to no match. These methodologies concentrate only on Infrastructure as a Service (IaaS) provisioning. Moreover, they typically neither export their data nor provide a SPARQL Protocol And RDF Query Language (SPARQL) [47] endpoint, thereby hindering reuse of and access to data.

Semantic Resouce Descriptions Semantic descriptions provide a blueprint of the network set up, therefore, it is critical to ensure common description standards in the deployment of federated cloud infrastructures. In this section, we discuss the semantic descriptions for network and cloud infrastructures.

Researchers have been exploring semantic models to address inter-infrastructure issues since 2008 [48]. These concepts were further adopted in the Global Lambda Integrated Facility (GLIF) [49] and GENI communities. The initial building block for our work is an RDFS ontology called NDL (Network Description Language) [50], [51], [52]. The primary use of NDL has been in GLIF, where it is used by individual network providers for sharing the network topology details with each other.

In mOSAIC [43] the authors present a compute ontology based on a collection of cloud taxonomies (NIST [53], OCCI [54]). This ontology is part of a larger effort to create a unified cloud API that is semantically enriched using elements of the ontology. The effort is concentrated on unifying the

views of different cloud providers of varying types (SaaS, PaaS, IaaS) under a single API. Our own compute ontology is also loosely based on NIST and other taxonomies, but is focused only on a single provider type - IaaS, however is much richer in terms of its ability to describe network topologies.

Software Defined Exchange Software Defined Exchange, proposed in 2015 [55], is an application of the highly successful Software Defined Networking concept, which creates Internet exchange points (IXPs) that apply SDN policies to control inter-domain network traffic flows. In [56], Gupta et al. extended the implementation of SDX to support industrial IXPs. To date there have been a number of deployed Software-defined IXPs, such as LightReading [57] and Google's Cardigan SDX controller [58] [59]. In a previous works [60] and [61], we presented a set of efforts toward the implementation of security-managed virtual SDX as an interconnection point to provide high throughput layer-2 link provisioning on national scale cloud infrastructures.

Distributed Trust There is a rich body of work on distributed trust systems. Most importantly we should mention the SPKI/SDSI [62], [63], [64], which is conceptually similar to SAFE, identifying principals by their private keys (or their hashes), allowing for multiple roots of trust. SPKI/SDSI policy language is aimed at encoding ACLs and thus based on first-order logic. Datalog queries used in SAFE can also be reduced to first-order logic.

Among policy languages, we should also mention XACML [65] - an OASIS standard for defining security policies. XACML is more expressive compared to SAFE, however its semantics are not well-defined and verification is possible only on a subset of the full language [66].

VII. Conclusions

In this paper we presented our vision of a multi-cloud networked architecture of the future. We discussed the services, that are in our view critical to the deployment of such architecture and provided examples of implementations of such services from our work. We also discussed the trust framework required to implement authorization policies for these services.

Our future directions include further development of our ideas and prototypes, the integration of discovery and metadata services with SAFE framework and development of standardized complex authorization policies for each service. We are also expanding the capabilities of our vSDX deployments to support more production-oriented services for a variety of science domains and deploying distributed applications that can take advantage of this highly-programmable multi-provider infrastructure.

Acknowledgment

This material is based upon work supported by the U.S. National Science Foundation under grants No. CNS-1526964, ACI-1642140, CNS-1330659, and through NSF's Global Environment for Network Innovation (GENI) program.

References

[1] I. Baldine, Y. Xin, A. Mandal, P. Ruth, A. Yumerefendi, and J. Chase, "ExoGENI: A Multi-Domain Infrastructure-as-a-Service Testbed," in *TridentCom: International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, June 2012.

[2] I. Baldin, J. Chase, Y. Xin, A. Mandal, P. Ruth, C. Castillo, V. Orlikowski, C. Heermann, and J. Mills, "Exogeni: A multi-domain infrastructure-as-a-service testbed," in *GENI: Prototype of the Next Internet*, R. McGeer, M. Berman, C. Elliott, and R. Ricci, Eds. New York: Springer-Verlag, July 2016.

[3] A. Mandal, Y. Xin, I. Baldine, P. Ruth, C. Heerman, J. Chase, V. Orlikowski, and A. Yumerefendi, "Provisioning and evaluating multi-domain networked clouds for hadoop-based applications," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, Nov 2011, pp. 690–697.

[4] I. Baldine, J. Chase, Y. Xin, D. Irwin, V. Marupadi, A. Mandal, C. Heermann, and A. Yumerefendi, "Autonomic cloud network orchestration: A GENI perspective," in *IEEE International Workshop on Management of Emerging Networks and Services (IEEE MENS 2010)*, Miami, Florida, USA, 12 2010.

[5] R. McGeer, M. Berman, C. Elliott, and R. Ricci, Eds., *GENI: Prototype of the Next Internet*. New York: Springer-Verlag, 2016, in production for publication July 2016.

[6] Chameleon Cloud, https://www.chameleoncloud.org/.

[7] AWS Metadata Service, https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html.

[8] OpenStack Metadata Service, https://docs.openstack.org/nova/latest/user/metadata-service.html.

[9] AWS SSM Service, https://docs.aws.amazon.com/systems-manager/latest/userguide/ssm-agent.html.

[10] R. Elmasri and S. Navathe, *Fundamentals of Database Systems*, 6th ed. USA: Addison-Wesley Publishing Company, 2010.

[11] "Resource Description Framework," https://www.w3.org/RDF.

[12] "Web Ontology Language," https://www.w3.org/OWL.

[13] S. H. Garlik, A. Seaborne, and E. Prud'hommeaux, "SPARQL 1.1 query language," http://www.w3.org/TR/sparql11-query/. [Online]. Available: http://www.w3.org/TR/sparql11-query/

[14] Y. Xin, I. Baldin, J. Chase, and K. Ogan, "Leveraging semantic web technologies for managing resources in a multi-domain infrastructure-as-a-service environment," *arXiv preprint arXiv:1403.0949*, 2014.

[15] A. Willner, M. Giatili, P. Grosso, C. Papagianni, M. Morsey, and I. Baldin, "Using semantic web technologies to query and manage information within federated cyber-infrastructures," *Data*, vol. 2, no. 3, 2017. [Online]. Available: http://www.mdpi.com/2306-5729/2/3/21

[16] L. T. Detwiler, D. Suciu, and J. F. Brinkley, "Regular paths in SPARQL: Querying the NCI thesaurus," in *AMIA Annual Symposium Proceedings*, vol. 2008. American Medical Informatics Association, 2008, p. 161.

[17] S. Gao, S. Shrivastava, K. O. (Anyanwu), I. Baldin, and Y. Xin, "Evaluating Path Query Mechanisms as a Foundation for SDN Network Control," in *Proceedings of the 4th IEEE Conference on Network Softwarization*. IEEE, 2018.

[18] B. Parsia and E. Sirin, "Pellet: An OWL DL reasoner," in *Third International Semantic Web Conference-Poster*, 2004, p. 18.

[19] R. Shearer, B. Motik, and I. Horrocks, "HermiT: A highly-efficient OWL reasoner," in *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2008)*, 2008, pp. 26–27.

[20] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical owl-dl reasoner," *Web Semantics: science, services and agents on the World Wide Web*, vol. 5, no. 2, pp. 51–53, 2007.

[21] H. S. W. Programme, "Jena: a Java framework for building Semantic Web applications," http://jena.sourceforge.net/.

[22] "Blazegraph Website," https://www.blazegraph.com/.

[23] R. E. Tarjan, "Fast algorithms for solving path problems," *J. ACM*, vol. 28, no. 3, pp. 594–614, Jul. 1981. [Online]. Available: http://doi.acm.org/10.1145/322261.322273

[24] J. L. Sobrinho, "Network routing with path vector protocols: Theory and applications," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM, 2003, pp. 49–60.

[25] T. G. Griffin and J. L. Sobrinho, "Metarouting," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4, pp. 1–12, 2005.

[26] J. F. Botero, M. Molina, X. Hesselbach-Serra, and J. R. Amazonas, "A novel paths algebra-based strategy to flexibly solve the link mapping stage of VNE problems," *Journal of Network and Computer Applications*, vol. 36, no. 6, pp. 1735 – 1752, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804513000672

[27] P. J. Leach, M. Mealling, and R. Salz, "A universally unique identifier (uuid) urn namespace," Internet Requests for Comments, RFC Editor, RFC 4122, July 2005, http://www.rfc-editor.org/rfc/rfc4122.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4122.txt

[28] COMET github repository, https://github.com/RENCI-NRIG/COMET-Accumulo.

[29] "Apache accumulo," https://accumulo.apache.org/.

[30] J. Crowcroft, S. Hand, R. Mortier, T. Roscoe, and A. Warfield, "Plutarch: An Argument for Network Pluralism," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 4, pp. 258–266, Aug. 2003. [Online]. Available: http://doi.acm.org/10.1145/972426.944763

[31] N. Feamster, L. Gao, and J. Rexford, "CABO: Concurrent architectures are better than one," *NSF NeTS FIND Initiative, http://www. nets-find. net/Funded/Cabo. php*, 2006.

[32] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In VINI veritas: realistic and controlled network experimentation," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4, pp. 3–14, 2006.

[33] Y. Xin, I. Baldin, A. Mandal, P. Ruth, and J. Chase, "Towards an experimental legoland: Slice modification and recovery in ExoGENI testbed," in *International Conference on Testbeds and Research Infrastructures*. Springer, 2016, pp. 35–45.

[34] Y. Yao, Q. Cao, J. Chase, P. Ruth, I. Baldin, Y. Xin, and A. Mandal, "Slice-based network transit service: Inter-domain l2 networking on exogeni," in *Computer Communications Workshops (INFOCOM WKSHPS), 2017 IEEE Conference on*. IEEE, 2017, pp. 736–741.

[35] Q. Cao, V. Thummala, J. S. Chase, Y. Yao, and B. Xie, "Certificate Linking and Caching for Logical Trust," http://arxiv.org/abs/1701.06562, 2016.

[36] V. Paxson, "Bro: a system for detecting network intruders in real-time," *Computer networks*, vol. 31, no. 23-24, pp. 2435–2463, 1999.

[37] N. Grozev and R. Buyya, "Inter-cloud architectures and application brokering: Taxonomy and survey," *Softw. Pract. Exper.*, vol. 44, no. 3, pp. 369–390, Mar. 2014. [Online]. Available: http://dx.doi.org/10.1002/spe.2168

[38] R. Buyya, R. Ranjan, and R. N. Calheiros, "Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services," in *Algorithms and Architectures for Parallel Processing*, C.-H. Hsu, L. T. Yang, J. H. Park, and S.-S. Yeo, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 13–31.

[39] M. Gall, A. Schneider, and N. Fallenbeck, "An architecture for community clouds using concepts of the intercloud," in *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, March 2013, pp. 74–81.

[40] G. Manno, W. W. Smari, and L. Spalazzi, "Fcfa: A semantic-based federated cloud framework architecture," in *2012 International Conference on High Performance Computing Simulation (HPCS)*, July 2012, pp. 42–52.

[41] D. Bernstein, V. Deepak, and R. Chang, "Draft standard for intercloud interoperability and federation (siif)," 2015.

[42] B. D. Martino, G. Cretella, A. Esposito, A. Willner, A. Alloush, D. Bernstein, D. Vij, and J. Weinman, "Towards an ontology-based intercloud resource catalogue – the ieee p2302 intercloud approach for a semantic resource exchange," in *2015 IEEE International Conference on Cloud Engineering*, March 2015, pp. 458–464.

[43] F. Moscato, R. Aversa, B. D. Martino, T. Forti, and V. Munteanu, "An analysis of mosaic ontology for cloud resources annotation," in *2011 Federated Conference on Computer Science and Information Systems (FedCSIS)*, Sept 2011, pp. 973–980.

[44] I. Santana-Prez and M. S. Prez-Hern'ndez, "A semantic scheduler architecture for federated hybrid clouds," in *2012 IEEE Fifth International Conference on Cloud Computing*, June 2012, pp. 384–391.

[45] A. V. Dastjerdi, S. G. H. Tabatabaei, and R. Buyya, "An effective architecture for automated appliance management system applying ontology-based cloud discovery," in *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, May 2010, pp. 104–112.

[46] L. D. Ngan and R. Kanagasabai, "Owl-s based semantic cloud service broker," in *2012 IEEE 19th International Conference on Web Services*, June 2012, pp. 560–567.

[47] C. Aranda, O. Corby, S. Das, L. Feigenbaum, P. Gearon, B. Glimm, S. Harris, S. Hawke, I. Herman, and N. Humfrey, "Sparql 1.1 overview," https://www.w3.org/TR/2012/PR-sparql11-overview-20121108/.

[48] A. K. Y. Wong, P. Ray, N. Parameswaran, and J. Strassner, "Ontology mapping for the interoperability problem in network management," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 10, pp. 2058–2068, Oct 2005.

[49] "Ndl demonstration site at glif," http://ndl.uva.netherlight.nl/.

[50] "Network description language," http://sne.science.uva.nl/ndl/.

[51] J. Pérez, M. Arenas, and C. Gutierrez, "Semantics and complexity of sparql," *ACM Trans. Database Syst.*, vol. 34, no. 3, pp. 16:1–16:45, Sep. 2009. [Online]. Available: http://doi.acm.org/10.1145/1567274.1567278

[52] J. van der Ham, P. Grosso, R. van der Pol, A. Toonk, and C. de Laat, "Using the network description language in optical networks," in *IFIP/IEEE International Symposium on Integrated Network Management*, May 2007, pp. 199–205.

[53] P. Mell and T. Grance, "The nist definition of cloud computing," *Special Publication 800-145, Recommendations of the National Institute of Standards and Technology*, September 2011.

[54] R. Nyren, A. Edmonds, A. Papaspyrou, and T. Metsch, "Open Cloud Computing Interface - Core," *GFD-P-R.183, OCCI-WG, Technical Report*, 2011.

[55] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett, "Sdx: A software defined internet exchange," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, pp. 551–562. [Online]. Available: http://doi.acm.org/10.1145/2619239.2626300

[56] A. Gupta, R. MacDavid, R. Birkner, M. Canini, N. Feamster, J. Rexford, and L. Vanbever, "An industrial-scale software defined internet exchange point," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. Santa Clara, CA: USENIX Association, 2016, pp. 1–14. [Online]. Available: https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/gupta

[57] "ica8 powers french touix sdn-driven internet exchange," http://ubm.io/1Vc0SLE, 2015.

[58] J. Stringer, D. Pemberton, Q. Fu, C. Lorier, R. Nelson, J. Bailey, C. N. A. Corrła, and C. E. Rothenberg, "Cardigan: Sdn distributed routing fabric going live at an internet exchange," in *2014 IEEE Symposium on Computers and Communications (ISCC)*, June 2014, pp. 1–7.

[59] J. Bailey, D. Pemberton, A. Linton, C. Pelsser, and R. Bush, "Enforcing rpki-based routing policy on the data plane at an internet exchange," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: ACM, 2014, pp. 211–212. [Online]. Available: http://doi.acm.org/10.1145/2620728.2620769

[60] Y. Yao, Q. Cao, R. Farias, J. Chase, V. Orlikowski, P. Ruth, M. Cevik, C. Wang, and N. Buraglio, "Toward live inter-domain network services on the exogeni testbed," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2018, pp. 772–777.

[61] A. Mandal, P. Ruth, I. Baldin, R. F. D. Silva, and E. Deelman, "Toward prioritization of data flows for scientific workflows using virtual software defined exchanges," in *2017 IEEE 13th International Conference on e-Science (e-Science)*, Oct 2017, pp. 566–575.

[62] C. M. Ellison, "SPKI," in *Encyclopedia of Cryptography and Security*. Springer, 2011, pp. 1243–1245.

[63] R. L. Rivest and B. Lampson, "SDSI-a simple distributed security infrastructure." Crypto, 1996.

[64] D. Clarke, J.-E. Elien, C. Ellison, M. Fredette, A. Morcos, and R. L. Rivest, "Certificate chain discovery in SPKI/SDSI," *Journal of Computer Security*, vol. 9, no. 4, pp. 285–322, 2001.

[65] M. Lorch, S. Proctor, R. Lepro, D. Kafura, and S. Shah, "First experiences using XACML for access control in distributed systems," in *Proceedings of the 2003 ACM workshop on XML security*. ACM, 2003, pp. 25–37.

[66] M. Matthew Greenberg and C. Marks, "The Soundness and Completeness of Margrave with Respect to a Subset of XACML," 09 2018.