INVITED PAPER *Special Issue on Network Virtualization*

An Architecture For International Federation of Network Testbeds

Robert RICCI†a), **Gary WONG**†b), **Leigh STOLLER**†c), ***and*** **Jonathon DUERIG**†d), ***Nonmembers***

SUMMARY Testbeds play a key role in the advancement of network science and the exploration of new network architectures. Because the scale and scope of any individual testbed is necessarily limited, federation is a useful technique for constructing testbeds that serve a wide range of experimenter needs. In a federated testbed, individual facilities maintain local autonomy while cooperating to provide a unified set of abstractions and interfaces to users. Forming an international federation is particularly challenging, because issues of trust, user access policy, and local laws and regulations are of greater concern that they are for federations within a single country. In this paper, we describe an architecture, based on the US National Science Foundation's GENI project, that is capable of supporting the needs of an international federation.

key words: *testbeds, federation, network, GENI*

1. Introduction

Testbed facilities, which provide experimenters with the ability to run protocols and applications on real or emulated networks, are key resources in the networking and systems research and education communities. Because they are programmable, controlled environments, they give researchers an excellent platform for investigating future directions in Internet architecture. Traditionally, each facility has been built and operated as a stand-alone facility: each testbed is owned and operated by a different entity, and the focus of research on testbed control frameworks has considered problems of designing, building, operating, and improving individual facilities [1]–[4].

This model, however, is changing: as the needs and expectations of testbed users expand, it is increasingly hard to satisfy those needs with a set of isolated facilities. This leads to a *federated* model, in which individual testbeds work together to provide their users with a common framework for discovering, reserving, and using resources. Federated testbeds may be built out of existing stand-alone facilities, new facilities specifically designed for federation, or a combination of the two. A framework for testbed federation must establish trust between members of the federation, but allow each member of the federation to retain *policy autonomy*; that is, the ability to make local decisions regarding which users and experiments are allowed to use it. It should also afford each federated facility *operational autonomy*, meaning that the federate should remain usable regardless of the operational state of other members of the federation.

As federated testbeds grow in scale and scope, they naturally begin to cross national boundaries. This presents a number of new challenges, and strengthens the needs for autonomy among federates: a facility spanning the globe is likely to have more distinct resource owners, with a wide variety of operations models, access policies, and legal considerations. In addition, as the size of a federation increases, the likelihood that every federate is up and operational at any point in time declines, so it is important that failure of remote components does not impact users' abilities to use other parts of the federation. In many cases, participants in an international federation already participate in a national federation: this makes it necessary to support *inter-federation* (federations of federations), allowing individual facilities to participate in more than one federation at a time.

In this paper, we present an architecture for federation of network testbeds that takes into account the needs of federation between nations. We begin by providing background information on the architecture of the US National Science Foundation's GENI project [5]. We then proceed to describe the goals and principles that inform ProtoGENI, our realization of the GENI architecture. In Section 2, we describe how each federated facility can be *operationally independent* while offering users the impression of a single unified facility. In Section 3, we turn our attention to issues of *trust* between federated facilities and federations-of-federations. In Section 4, we put these pieces together to demonstrate how an international *inter-federation* can be constructed using this architecture, combining federations run by different countries into a cohesive whole, while each federation retains its own local operational and authorization policy.

1.1 GENI Architecture

The architecture of ProtoGENI builds on the GENI framework. GENI's architecture is, in turn, based on the "Slice-based Federation Architecture" (SFA) [6], which has been developed by the GENI community. The SFA is so named because it centers around partitioning the physical facility into "slices," each of which can be running a different network architecture or experiment inside. Physical resources, such as PCs, routers, switches, links, and allocations of wireless spectrum are known as "components"; when a user allocates resources on a component, the set of resources they are given

Manuscript received September 14, 2012.
Manuscript revised January 1, 2012.
†The University of Utah, Salt Lake City, UT, USA.
a) E-mail: ricci@cs.utah.edu
b) E-mail: gtw@cs.utah.edu
c) E-mail: stoller@cs.utah.edu
d) E-mail: duerig@cs.utah.edu
DOI: 10.1587/transcom.E0.B.1

comprises a "sliver." This sliver could be a virtual machine, a VLAN, a virtual circuit, or even the entire component. Each sliver belongs to exactly one slice: in essence, a slice is a container for a set of slivers.

There are two key types of principals in GENI:

Aggregate Managers (AMs) are responsible for managing the resources (components) on which users will create networks and run experiments. AMs are responsible for the allocation of their resources, and may make decisions about who is authorized to use them. An individual AM may manage a collection of components, called an *aggregate*; in practice, each facility in GENI runs a single AM that manages all of its resources, and the largest aggregates contain hundreds of nodes and thousands of links.

Users access components from the federated testbed to run an experiment or a service. A user is free to create slices which span multiple AMs, and each user is authorized by one of the facilities in the federation.

Principals and many other objects in the system are uniquely named by a Uniform Resource Name (URN) [7]. The URN scheme that we use [8] is hierarchical—each authority is given its own namespace, which it can further subdivide if it chooses. To maintain *partitioned trust*, each authority is prohibited, through mechanisms described in Section 3 from creating URNs outside of its namespace. An example of a GENI URN is:

$$\texttt{urn:publicid:IDN+}\overbrace{\texttt{emulab.net}}^{\text{Authority}}\texttt{+user+}\overbrace{\texttt{jay}}^{\text{ID}}$$

Because the URN contains the identity of the authority that issued it (in this example "`emulab.net`"), it is possible to tell which authority "owns" the object without resorting to a lookup service.

At a high level, testbeds federate in this framework by forming trust relationships: if facility A trusts facility B, then A is willing to trust B's statements about what users it has, what slices they have created, and what resources B offers. Note that this does not preclude A from having local allocation policies: just because it recognizes B's users does not obligate it to satisfy all requests they might make. Arrangements regarding "fair sharing," etc. can be made as part of the federation agreement. Trust relationships need not be symmetric: A may choose to trust B even if that trust is not reciprocated.

2. Enabling Independent Operation

We build on the basic GENI architecture by adding three new kinds of entities into the federation, and by providing an expanded API for AMs.[†]

Identity Providers (IdPs) attest to the identity of users by forming names that uniquely identify them, then issuing those users certificates that they can use to prove their identities to other services. IdPs can choose to accept responsibility for enforcing arbitrary policies covering users and names.

[†]Here, we limit our discussion to the properties of ProtoGENI that are relevant to international federations; more details on the topics in this section, including a set of principles for distributed facility design, can be found in [9]

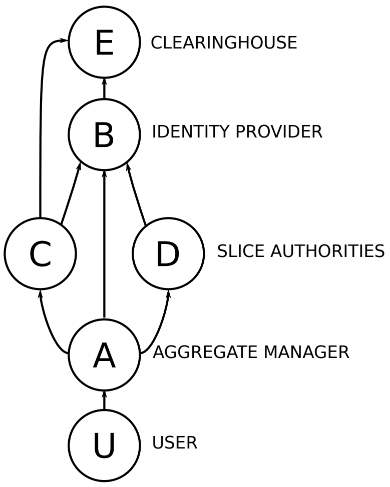


Fig. 1 An example of entities interacting within a federation. The principals are described in Sections 1.1 and 2; the arrows indicate trust relationships, described in Section 3.

Slice Authorities (SAs) are responsible for creating slice names and granting users the necessary credentials to manipulate these slices. By issuing a name for a slice, the SA agrees to be responsible for the actions taken within the slice. An SA may be an institution, a research group, a governmental agency, or other organization.

A user has an account with an SA, called the "home" SA; this SA vouches for the identity of the user, and in most cases, is willing to create slices for the user. The user is, however, free to create slices using any SA that, according to its own policies, is willing to be responsible for that user's actions.

Establishing trust in this pairwise fashion does not scale well to large federations. ProtoGENI's sole centralized service, the **Clearinghouse** (CH), is used to make this process more convenient: it allows federates to publish the certificates that are used to establish trust, and to discover the certificates of other federates. It is important to note that this does *not* mandate specific trust relationships: as described in [10], a federate may choose not to trust some certificates stored at the clearinghouse, or may choose to trust additional certificates that are not registered there.

The clearinghouse also serves a second purpose: it acts as a registry where various objects can be looked up. Notably, users can ask the clearinghouse for a list of all registered federates to bootstrap the process of resource discovery, as described in the next section. In both of these roles, the information provided by the clearinghouse changes infrequently, and can be safely cached for long periods of time (days or weeks).

ProtoGENI has been designed to keep federates as loosely coupled as possible; they do not depend on central services, and the only parts of the system involved in a given operation are those directly affected by it. In the extreme case, if a federate is cut off from communication with the rest of the federation, users who can reach the federate are still able to create slices and slivers on it.

This is possible because ProtoGENI goes to great lengths to ensure that *minimal state synchronization* is required between AMs, SAs, and the CH. This section describes the interactions these services have with each other and with users. We concentrate on where ProtoGENI stores state, how it avoids centralized services, and how failures are managed. Because the full ProtoGENI APIs [11] are too large to cover in depth here, we introduce only the operations necessary to understand state management.

2.1 Slice State

ProtoGENI does not attempt to guarantee a globally consistent view of the state of each slice. Instead, it uses a loose consistency model in which each of the individual authorities and managers maintain their own state.

The authoritative source for user and slice records is the SA that issued them, and the authoritative source for sliver information is the AM on which the sliver exists. Because the URNs used in ProtoGENI encode the issuing authority, it is possible to determine the correct authority to query simply by examining an object's name. If, for example, a AM wishes to find out more about a user who has created a slice on it, the AM may use the user's URN to identify and query the user's home SA.

When a sliver is created, the AM is *not* provided with a global picture of the slice: the sliver request need only contain the resources on the AM in question and any links to directly adjacent AMs that need to be configured as part of the slice. Information about the rest of the slice is not needed for the AM to create its slivers, and maintaining a global view would require that the AM be notified of changes *anywhere* in the slice, even if those changes do not directly affect it.

2.2 Resource Reservation Across AMs

Slices that cross AMs present a dilemma: we would ideally like the process of allocating or updating slivers to be atomic across all aggregates. As a concrete example, consider a slice with existing slivers from two different AMs. We would like to make a change on both slivers, but only if both of the changes succeed. If either one is denied, we want to roll back to the original configuration without losing existing resources or otherwise changing the slivers. However, the loosely-coupled nature of the federation precludes using global locks or global transactions.

Instead, we consider the resource allocation process on each AM to be a separate local transaction, and model the life cycle of each sliver as a state machine. We designed the state machine with *minimal abstraction* in mind, allowing clients or other intermediaries to build a transactional abstraction across AMs on top of our lower-level per-AM API. Each sliver can be in one of four states:

1. The *Null* state, in which the sliver does not exist (has not yet been created, or has been destroyed).
2. The *Ticket* state, in which the user holds a ticket promising resources, but the sliver is not instantiated on the component.
3. The *Sliver* state, in which the sliver has been instantiated, but the user does not hold a valid ticket.
4. The *Sliver and Ticket* state, in which the user has both an instantiated sliver and a ticket.

This state machine makes sliver manipulation a three-step process:

1. Get the list of currently available resources from each AM.
2. Request a new ticket on each AM; this step obtains a *guarantee* of resources, but does does not actually instantiate a new sliver or modify an existing sliver.
3. Redeem the tickets at each AM to "commit" the resource change.

Steps 1 and 2 are not atomic: if other users are simultaneously trying to reserve resources to their own slices, the second step may fail. In a distributed system like ProtoGENI, it is not feasible to *lock* the resource lists for any length of time. Since contention for resources is generally rare in ProtoGENI, a form of *optimistic concurrency control* [12] is employed to both avoid locking and to ensure that users will find out if someone else has already reserved a resource.

2.3 Slice and Sliver Lifetimes

Because authoritative slice state is distributed across SAs and AMs, and we cannot guarantee that they remain in contact throughout the lifetime of the slice, we give each slice and sliver an expiration date. This way, we can be assured that all slivers are eventually cleaned up and their resources reclaimed.

There are important nuances, however, in the relationship between slice and sliver lifetimes. Because each sliver must belong to a slice, the sliver must not outlive its slice. If it did, this could lead to a situation in which the user would lose control of the sliver.

The first consequence of this requirement is straightforward: the expiration time for each sliver is bounded by the expiration time of the slice itself. The second consequence is that a slice cannot be deleted before it expires. It is possible that slivers exist that the SA is unaware of; a AM may have been unable to contact the SA to inform it of the sliver's existence. Therefore, the SA cannot know with certainty that deleting the slice is safe and will leave no orphaned slivers. As a result, slice names cannot be re-used by experimenters before they expire. Since the namespace for slices is effectively unbounded in size, this is not a major concern.

2.4 Behavior in the Face of Failures

ProtoGENI passes as much context as possible in API calls, so that they can be *self-contained*. While this does result in some extra overhead in the calls, the benefit is that the user can continue to make progress in the presence of network or service failures. For example, a user obtains authorisation credentials from his home SA, and these credentials are passed *by the user* to AMs when requesting tickets. As described in [10], the AM receiving this material can verify its authenticity without contacting the issuer. As a result, it is possible for use of the testbed to proceed in the face of many classes of failures of individual components.

3. Partitioned Trust

Our model divides a testbed (or federation of testbeds) into multiple trust domains. Each object within the system belongs unambiguously to a single domain, and each facility has authority only over those objects which fall within its domain.[†] This tight binding is an essential requirement, and guarantees important properties beyond those merely arising from multiple trust anchors. For instance, the trust model used by Web browsers when communicating over TLS/SSL [13] makes use of many trust anchors — browsers typically trust hundreds of root CA certificates — but trust is not *partitioned*, and any CA is permitted to sign any certificate. Therefore, no guarantees about the security of the composite system can be made beyond that of the least trusted CA.

This problem is particularly acute in international federations. The stakeholders in such a federation are likely to have different regulatory and legal requirements, and allowing any member of the federation to assert authority over other members' entities could have serious consequences.

Our model permits a hierarchical structure (that is, "sub-partitions" of trust). It is also transparent: when an object from one domain interacts with an object from another domain, both objects' true identities are exposed, so that policies can be based on the object's full identity, the domain to which it belongs, or both.

When different parts of a system may be owned and operated by different organizations, or need to enforce custom local policies, sophisticated requirements for authentication and authorization infrastructure arise. Our approach is to adopt a decentralized architecture, to support disconnected operation, and to decouple authentication from authorization whenever possible.

The authentication system is based on the IETF PKIX model [14], while the authorization mechanism involves the presentation of cryptographically signed *credentials* (which behave analogously to X.509 Attribute Certificates [15]). When a principal presents a certificate or credential, it provides all of the signatures required to link the certificate or credential with one of the trust roots; as a result, no direct communication with the certifying party is required to validate certificates and credentials. Together, these primitives allow the warranting of identities, the granting and delegation of permissions, and the verification of identity and privilege. Most importantly, all of these operations may be performed by different principals, who need no direct knowledge of each other. Very little global policy is imposed, other than conformance to uniform naming and data representation schemes.

[†]We make a subtle, but important, distinction between having authority over an object and making assertions about it; only authorities can attest to the identity of objects, while assertions (statements relating to that object) may be made by any party, and may be accepted by any entity that trusts the asserter.

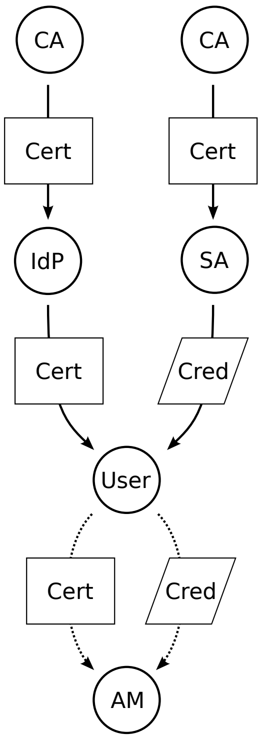


Fig. 2 The transfer of cryptographic certificates ("Cert") and credentials ("Cred") between principals in a federation. Solid lines indicate new material being issued and signed; dotted lines indicate presentation of existing information.

Additional details of this model can be found in [10].

3.1 Certificate Authorities

We deliberately refrain from imposing a single hierarchy on the trust structure (as used in PEM [16], for instance), as that model is inadequate to support local fine-grained trust decisions. On the other hand, an entirely decentralized public key distribution mechanism (such as PGP's "web of trust" [17]) is highly flexible, but tends to introduce barriers to new principals entering the system, as their certificates are unlikely to be accepted by others until they are able to obtain signatures from a sufficient number of existing authorities.

Our public key infrastructure treats each domain as a trust anchor, with its own self-signed CA certificate. Each CA may form subsidiary namespaces and issue corresponding CA certificates over them. This approach yields significant flexibility in trust decisions (at the granularity of domains), although it does introduce the problem of distributing the set of root certificates throughout a federation. Our system makes use of the clearinghouse by giving it the role of a *trusted introducer*: it publishes a bundle of certificates from known domains as a convenience, but it is important to note that this does not detract from any site's autonomy. Each domain is free to add to or delete from the CH's list of certificates (or even ignore it entirely). A CH also aggregates certificate revocation lists [14] from the same set of domains, though nothing prevents domains from communicating their CRLs to each other directly.

3.2 Authenticating Identities

Public key certificates are issued to each principal in the system, including users, services, and components. They must be signed by the authority corresponding to the namespace in which the principal's name belongs: since these namespaces do not overlap, there always exists exactly one certificate authority whose signature will be accepted on any valid certificate.

All requests are made over TLS [13] channels, and both the client and server must authenticate to each other. (This implies that if either peer has decided not to trust the other's CA, then no communication or operation will be possible.)

We have ensured that certificates are self contained: certificates are always presented in conjunction with any intermediate CA certificates, so that any verifier can determine the validity of any certificate with no information other than the set of trust anchor certificates and current CRLs. (TLS and PKIX already provide this property, and we have been careful to preserve it in the conventions and certificate extensions we have added.)

3.3 Issuing Names

Formal structure within names is essential for partitioning trust, so that an unambiguous trust root can be identified for any named object. ProtoGENI has adopted the proposal by Viecco et al. [8] for uniform naming conventions throughout GENI. This name scheme tightly binds each object's name to a particular authority, which is necessary to maintain clear definitions of trust boundaries. By refusing a CA's signature on any certificate whose subject name lies outside the CA's namespace, we are able to guarantee the important property that any valid object name corresponds to exactly one trust root. Although there are certain limitations to this model (for instance, if a principal wishes to associate with a different CA for any reason, then it is forced to change its identity), the benefits are significant: first, each domain has great flexibility in choosing the set of peers with whom it will operate; second, we achieve a reasonable level of fault containment, since even extremely severe faults (e.g. malicious CAs or Byzantine failure of a CA) are unable to affect objects outside their assigned namespace. This property is extremely important, as a single key compromise in a PKI system without naming restrictions can leave the entire system vulnerable [18]. A series of attacks against root CAs in mid-2011 attracted widespread publicity and has required extensive software patches to terminate trust in the affected CAs [19].

3.4 Authorizing Operations

Credentials are used in conjunction with our public key infrastructure to allow secure validation of permissions: X.509 certificates prove that a key is bound to a principal, and credentials prove that permissions have been assigned to that principal (see Figure 2).

It is important to note that our credentials are issued as the result of authorization decisions, and could thus be considered to represent *capabilities*. Another useful class of statement about principals are *assertions*, which can be used as *input* to policy decisions; assertions may take the form of statements such as "X is a student," "Y has a Top Secret security clearance," etc. ABAC [20] (Attribute Based Access Control) combines a system for making signed assertions with a system of formal logic to reason about authorization decisions. Under separate work by the GENI ABAC team, ABAC is being integrated with ProtoGENI.

Almost all services must verify *both* certificates and credentials: relying on either alone is inadequate. (There are a small number of exceptions, such as a ProtoGENI user's own facility, which records state concerning the users privileges over objects under its control, and consequently is able to issue credentials on the basis of a successful key challenge alone.)

4. Constructing an International Federation

We define an inter-federation to be a federation whose members are, in turn, federations[†]. There are two parts to constructing an inter-federation using the ProtoGENI architecture: providing a set of compatible abstractions and APIs to users, and establishing trust relationships between the federates.

In order to ensure operational independence of the federates, the common API should follow the principles described in Section 2. Examples of APIs following this architecture include the ProtoGENI [11] and GENI [21] APIs. In most cases in GENI, this has been done by placing a "wrapper" around the native APIs of existing testbed control software [1], [2], [22]. We expect this to be a popular approach in international federations as well: each nation or funding agency may have developed its own APIs and mechanisms for resource access, or have existing frameworks for testbed control.

The remainder of this section focuses on the second part of inter-federation establishment, the creation of a trust structure. It is important to note that when we discuss *trust* in this context, we refer specifically to the ability to make statements about the existence, identity, and public keys of objects: for example, to assert the existence of a user, a slice,

[†]Without loss of generality, any of these federations could be trivial federations with a single member.

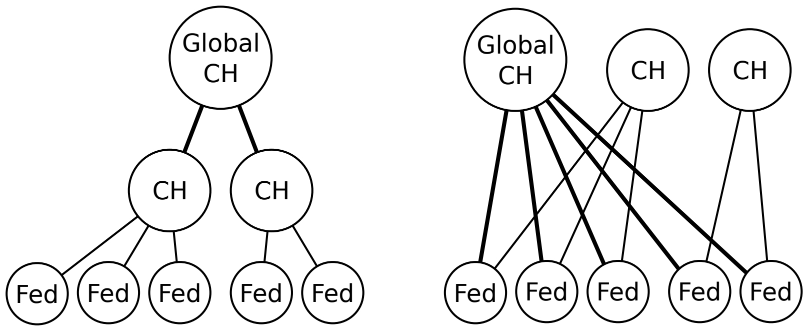


Fig. 3 An inter-federation consists of one clearinghouse for the entire inter-federation ("Global CH"), one clearinghouse per federation ("CH"), and a potentially large number of individual federates ("Fed"). The structure on the left illustrates a single root of trust per federation (see Section 4.1), while the flatter arrangement on the right shows multiple roots of trust per federation (Section 4.2).

or a component manager. By this definition, establishing trust does not, by itself, require members of the national federations to apply specific policies to principals from other federations. The policies that federations apply to other members of the inter-federation will be worked out as part of the inter-federation agreement. Example policies include:

- a national federations may agree to accept *all* slices from foreign Slice Authorities;
- a national federation may agree to accept only foreign slices that are endorsed by a particular foreign Slice Authority, representing experiments that the foreign entity has vetted as appropriate for running internationally;
- a national federation may accept only foreign slices that it has vetted itself for use on its resources.

Models such as ABAC [20] can provide the technical means by which these policies are stated and enforced.

4.1 Single root of trust per federation

We first consider the case in which each national federation traces all trust relationships back to a single trust root, as illustrated on the left side of Figure 3. In this trust structure, the federation creates a root namespace associated with a single authority certificate; this namespace is recursively subdivided, as described in Section 2, with each member of the federation being assigned to a partition. Each federate's certificate is signed by the root for the federation, meaning that trusting the root certificate is sufficient to trust any member of the federation.

Creating an inter-federation in this case is straightforward: a global clearinghouse tracks the root CAs for each national federation, and members of the inter-federation agree to trust all CAs provided by the global clearinghouse. This allows each federation to accept the identity providers, aggregate managers, and slice authorities from the other federation. This, in turn, makes the users, components, and slices belonging to these entities visible across the entire inter-federation.

This case has the advantage of being simple to implement, understand, and maintain. Only a small number of trust roots need to be exchanged between the participants, and that set will change infrequently (only when a federation joins or leaves the inter-federation.) However, it means that each national federation must expose the same trust structure to the inter-federation that it uses internally; this may not always be desirable.

4.2 Multiple roots of trust per federation

Individual federations need not be based on a single trust root: rather than having all federates' certificates signed by a single root, each federate can sign its own root certificate. The act of establishing trust in the federation, then, involves distributing a *set* of root certificates, rather than a single one.[†] This structure, shown on the right in Figure 3, is the trust structure used by the existing ProtoGENI federation. The ProtoGENI clearinghouse maintains a set of root certificates for members of the federation. Facilities supporting the ProtoGENI APIs select their own namespaces, and generate their own self-signed certificates for those namespaces. To join the federation, the facility sends its certificate to the clearinghouse. If the clearinghouse choses to accept the facility as a member of the federation, it simply adds the its certificate to the root set. A new root set is distributed to the federation's members on a daily basis.

To form an inter-federation in this style, a global clearinghouse maintains a set of certificates for individual *facilities* that make up the inter-federation. This is in contrast to the previous case, in which the list was a list of *federations*. In essence, each facility now participates in two different federations: its own national federation, as well as the inter-federation.[††] The global clearinghouse tracks meta-data about which national federation each inter-federation member belongs to. By expressing this meta-data as secure

[†]If this distribution is done by a single trusted entity, such as a clearinghouse, that entity could be considered a single trust root. However, it is not a root in a cryptographic sense: it does not sign the certificates of the federates. It is also possible for multiple entities to serve this purpose, distributing either identical or different sets of federate certificates.

[††]The architecture sketched in Section 2 was specifically designed to support simultaneous participation in multiple federations.

statements (for example, using ABAC [20]), it can be used directly in policy decisions, such as to grant users different privileges based on which national federation they are a member of.

While this federation structure is more complicated, and requires the global clearinghouse to track a larger number of federates, it has a number of advantages. First, it allows scenarios in which not all members of a national federation are presented to the inter-federation: if trust is established by adding a single federation root to the inter-federation, all federates endorsed by that root are recursively trusted. A national federation, may, however, contain members that are not "production quality," allow users who do not necessarily meet the policies of the international federation, or are otherwise unsuitable for inclusion in the inter-federation. Presenting federates individually to the inter-federation allows these members to be omitted. Second, it makes it easier for facilities to join the inter-federation at any time: because we enforce trust partitioning through naming, placing a federate "under" another trust root requires pre-planning. Federates that exist before the establishment of the federation, and therefore have namespaces outside of it, may need to be renamed under such a scheme. In contrast, if the federate is trusted directly by the inter-federation, no such pre-planning or re-naming is needed. Third, it allows each facility to participate in multiple federations simultaneously, each of which may have different trust structures.

It is worth noting that a federation can be formed in this manner even if the national federations have single-trust-root structures: rather than exposing the top-level trust roots to the inter-federation, the root set for the inter-federation can be constructed from the certificates for individual facilities.

5. Conclusion

Federated testbeds provide new opportunities for experimentation, but also raise a number of design challenges. When the federation is international in nature, the issues of operational and policy autonomy are highlighted, and it is necessary for the federation framework to support an organization that clearly delineates the operational and policy boundaries between federates. We have presented the GENI and ProtoGENI architectures, and showed how they can be used to construct federations that cross national boundaries. For the last three years, we have operated a federation using this architecture which now has a dozen federates and 500 users, who have created over 18,000 slivers.

Acknowledgements

Many people have been active participants in the GENI design process, which arrived at the basic design in to which our system is intended to fit. While the total number of contributors to this process is large, we would like to specifically acknowledge the chairs of the GENI Facility Architecture Working Group and heads of the GENI control frameworks: Larry Peterson (PlanetLab), John Wroclawski (TIED), Jeff Chase (ORCA/BEN), and Max Ott (OMF). Others major contributors to the design process have included Aaron Falk, Ted Faber, Steve Schwab, and Ilia Baldine.

This material is based upon work supported by the National Science Foundation under Grant No. 0723248 and the GENI Project Office.

References

[1] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," Proc. of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), Boston, MA, Dec. 2002.

[2] L. Peterson, A. Bavier, M.E. Fiuczynski, and S. Muir, "Experiences building PlanetLab," Proc. of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), Seattle, WA, Nov. 2006.

[3] M. Ott, I. Seskar, R. Siraccusa, and M. Singh, "ORBIT testbed software architecture: Supporting experiments as a service," Proc. of the International ICST Conf. on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom), Trento, Italy, Feb. 2005.

[4] D.G. Anderson, H. Balakrishnan, M.F. Kaashoek, and R. Morris, "Resilient overlay networks," Proc. of the ACM Symposium on Operating Systems Principles (SOSP), Banff, Canada, Oct. 2001.

[5] GENI Architecture Team, "GENI federation software architecture document," March 2012. `http://groups.geni.net/geni/wiki/GeniArchitectTeam`.

[6] L. Peterson, R. Ricci, A. Falk, and J. Chase, "Slice-based federation architecture." `http://groups.geni.net/geni/wiki/SliceFedArch`, June 2010.

[7] R. Moats, "URN syntax," Request for Comments 2141, Internet Engineering Task Force, May 1997.

[8] C. Viecco, "Use of URNs as GENI identifiers." `http://gmoc.grnoc.iu.edu/gmoc/file-bin/urn-proposal3.pdf`, June 2009.

[9] J. Duerig, R. Ricci, L. Stoller, G. Wong, S. Chikkulapelly, and W. Seok, "Designing a federated testbed as a distributed system," Proceedings of the 8th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (Tridentcom), June 2012.

[10] G. Wong, R. Ricci, J. Duerig, L. Stoller, S. Chikkulapelly, and W. Seok, "Partitioning trust in network testbeds," Proc. of the 45th Hawaii International Conf. on System Sciences (HICSS-45), Wailea, HI, Jan. 2012.

[11] ProtoGENI Project, "ProtoGENI API." `http://www.protogeni.net/trac/protogeni/wiki/API`, May 2012.

[12] H.T. Kung and J.T. Robinson, "On optimistic methods for concurrency control," ACM Transactions on Database Systems, vol.6, no.2, June 1981.

[13] T. Dierks and E. Rescorla, "The transport layer security (TLS) protocol version 1.2," Request for Comments 5246, Internet Engineering Task Force, Aug. 2008.

[14] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile," Request for Comments 5280, IETF, May 2008.

[15] S. Farrell, R. Housley, and S. Turner, "An internet attribute certificate profile for authorization," Request for Comments 5755, Internet Engineering Task Force, Jan. 2010.

[16] S. Kent, "Privacy enhancement for internet electronic mail: Part II: Certificate-based key management," Request for Comments 1422, Internet Engineering Task Force, Feb. 1993.

[17] W. Stallings, "The PGP web of trust," BYTE, vol.20, no.2, Feb. 1995.

[18] C. Soghoian and S. Stamm, "Certified lies: Detecting and defeating government interception attacks against SSL." `http://files.cloudprivacy.net/ssl-mitm.pdf`.

[19] K. Dilanian, “Cyber-attack in Europe highlights internet risks,” Los Angeles Times, 9 Sept. 2011. http://articles.latimes.com/2011/sep/09/world/la-fg-cyber-attack-20110910.

[20] N. Li, J.C. Mitchell, and W.H. Winsborough, “Design of a role-based trust management framework,” Proc. of the 2002 IEEE Symposium on Security and Privacy, May 2002.

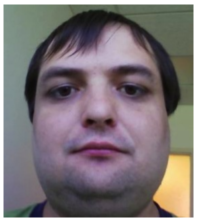
[21] GENI Project Office, “GENI aggregate manager API.” http://groups.geni.net/geni/wiki/GAPI_AM_API, Sept. 2012.

[22] “The ORCA GENI control framework.” http://www.nicl.cs.duke.edu/orca.

Robert Ricci Robert Ricci is a Research Assistant Professor in the University of Utah’s School of Computing. He has been been building network testbeds for over a decade, beginning with the Emulab facility and continuing with the National Science Foundation’s GENI and PRObE facilities. He has done research on a variety of topics regarding testbed design, construction, and use, including resource mapping, control system design, and emulation of realistic network conditions.

Gary Wong Gary Wong is a Research Associate with the University of Utah’s School of Computing, where he works on network testbeds and other systems projects. He has contributed to the design and implementation of authentication and authorisation mechanisms used in GENI, and his previous work covers a range of topics from compilers to distributed systems. He holds a Bachelor of Science degree from the University of Auckland.

Leigh Stoller Leigh Stoller is a Senior Software Developer in the Flux Research Group. He joined the group in 2000 and has been involved in a number of its main research activities since. Leigh is one of the principal architects and developers of the Utah Emulab network testbed [1], as well its companion, ProtoGENI. He plays a central role in essentially every aspect of Emulab’s design, implementation, maintenance, and operation. Leigh has co-authored papers on a variety of topics including operating systems, languages, networking protocols, system architecture, and network testbed design. He earned his MS degree in Computer Science in 1993.



Jonanthon Duerig Jonathon Duerig is a Research Associate in the School of Computing at the University of Utah. He has worked on the ProtoGENI project since its inception and has helped to shape both its architecture and implementation. Jonathon was the lead designer of the standard GENI resource specification format.