A Federated Experiment Environment for Emulab-based Testbeds

Ted Faber, John Wroclawski
USC/ISI
4676 Admiralty Way
Marina del Rey, CA 90292
{faber, jtw}@isi.edu

***Abstract*—We describe an architecture for creating experimental environments across multiple cooperating Emulab-based testbeds, called the DETER Federation Architecture(DFA). The system uses cooperative resource allocation and multiple-level testbed access to create a cohesive environment for experimentation. Testbeds that contribute resources continue to exert their own resource allocation and access policies. The architecture is designed to scale. We describe a prototype implementation.**

Keywords-testbeds, federation, experimentation

I. Introduction

This paper lays out a model for allocating and configuring experiments across multiple Emulab-based[1] testbeds, and describes a prototype. The model employs a multi-level resource allocation strategy that gives both the testbeds and the experimenter influence over allocation decisions. It provides Emulab-like support services to experiments using techniques that both scale and allow testbeds to maintain control of their resources. Federated experiments are administered as system objects and may be queried or operated on by any user of the federation system, subject to access controls. The prototype uses several extensible technologies and is in use federating experiments.

Network testbeds are invaluable for modern research, making experiments more realistic and reliable. They can be used to confirm the dynamics of network simulations and distributed systems, to evaluate the behavior of existing network artifacts (viruses, worms) under controlled conditions, and to examine the interactions between a proposed system and existing infrastructure. Doing this work on physical hardware in a laboratory environment to which others have access improves the quality of research.

Federation - combining the resources of more than one independently controlled testbed - enhances the utility of testbeds significantly. First, experimenters can access more resources, increasing the scale of their experimentation. Furthermore, individual testbeds may include unique hardware or configuration properties that allow experimenters to embark on new kinds of experiments. Finally, because testbeds act as gathering points for experimenters in a given field, combining testbed resources can promote collaboration between those groups. For example, security experts and malware architects can test each other's work in a testbed built partially from each group's home testbed. Such collaboration can be cooperative or competitive.

Two obvious ways of expressing federation of Emulab-class testbeds are to create federated testbeds or to create federated experiments. A federated testbed is a pool of resources that export the interface of a testbed. A federated experiment is an experiment constructed from the resources of many testbeds. The DETER federation architecture creates federated experiments. Our view is that federation should be a common, simple, low commitment operation. Furthermore, testbeds must retain control of their resources for federation to be effective. Composing an experiment from sub-experiments administered locally is a clear way to preserve autonomy while presenting experimenters with a familiar environment.

Federating a few large testbeds is an easy way to create large experiments and is often a focus of federation systems, but the choice of federating many smaller testbeds is also attractive. A federation system that allows many 10-15 computer testbeds to come together may accumulate more resources than one that supports only a few large testbeds. Successful federation systems will scale not only in terms of available resources, but in the numbers of users and testbeds. Scaling along these axes requires decentralized global naming and trust architectures.

We have developed an approach for federation within Emulab-based testbeds, called herein the DETER Federation Architecture. The efforts originated in our Emulab-based security testbed, DETER[2], though a federation system must use other systems. We have developed the system across an

This material is based upon work supported by the National Science Foundation under Grants Nos. CNS-0751027 and CNS-0714770.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of GPO Technologies, Corp., or the GENI Project.

This material is also based upon work supported by the Department of Homeland Security, and Space and Naval Warfare Systems Center, San Diego, under Contract No. N66001-07-C-2001.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Department of Homeland Security for the Space and Naval Warfare Systems Center, San Diego.

internal testbed – part of DETER managed independently for this work – as well using other Emulab-based testbeds on the Internet, including Emulab itself and WAIL[3]. No changes were made to the code of the testbeds we do not manage.

This paper describes the resource allocation and experiment creation aspects of that architecture. In the remainder of this paper we describe the architecture, then discuss in more detail the experiment creation model and how it fits in that architecture, together with our prototype federator implementation, *fedd*. In addition to describing the implementation we describe some of our experience using fedd.

II. The DETER Federation Architecture

The DETER project has created a federation architecture, the DETER Federation Architecture (DFA), that frames the various components needed to interconnect testbeds. The architecture is designed to scale to hundreds of testbeds and thousands of machines. The immediate goal of the architecture is to guide the interconnection of Emulab-based testbeds.

The full federation architecture must provide three capabilities. First, it must provide experimenters and their tools with sufficient information to guide the process of decomposing experiments into testbeds. To accomplish this the architecture must provide scalable channels for testbeds to advertise or respond to queries about the resources they permit to be federated; this information may be filtered based on the identity of the experimenter or abstracted for scaling. Secondly, experiments must be decomposed and embedded into federated testbeds. Finally the architecture must support experimentation across the federated experiment. Part of this goal is to generate a cohesive, scalable experimental environment that may be represented differently to different experimenters. For example, experimenters representing attackers and defenders in a competitive experiment may be provided limited knowledge of their opponents' topology. This paper focuses on the allocation and configuration of resources – testbed nodes, testbed network capacity and other elements needed to connect them - into cohesive, scalable environments for experimentation.

The experiment decomposition and embedding phase of the DETER federation architecture can be viewed from several perspectives - experimenters, the federation system, and the federants all see the architecture differently.

An experimenter creates an experiment using whatever domain-specific creation tools are available. Should the experimenter or the tools decide that the experiment needs more resources than one testbed can provide or that properties generated by federation are key to the outcome, the tools will invoke the federation system. After evaluating experimenter requirements and available resources, the federation system will divide the experiment among testbeds subject to his or her constraints, create sub-experiments on each testbed, and then interconnect them to form the federated experiment. The instantiation and interconnection will be transparent to the user, unless there is a reason to expose it.

For the system implementers the centerpiece of the federation system is the federator. It takes input from experimenters or their tools and creates an experimental environment split across federant testbeds. Specifically, the federator decomposes an experimenter's annotated topology into federable sub-experiments, acquires access to appropriate federants, embeds the sub-experiments in federants, and then connects them into a shared environment. Figure 1 illustrates this architecture.

The architecture is partitioned to separate concerns of the various players. The partitioning of the experiment into pieces suitable to federation depends on the nature of the experiment. This split must be guided by the experimenter using knowledge of the resources provided by the federation system. For example, an experiment used to study throughput of a new protocol must be aware which links are inside a testbed and completely controlled and which are not, to ensure that the unpredictable link performance does not invalidate the results. Collaborative or adversarial experiments will divide along the lines of visibility and testbed administrative boundaries.

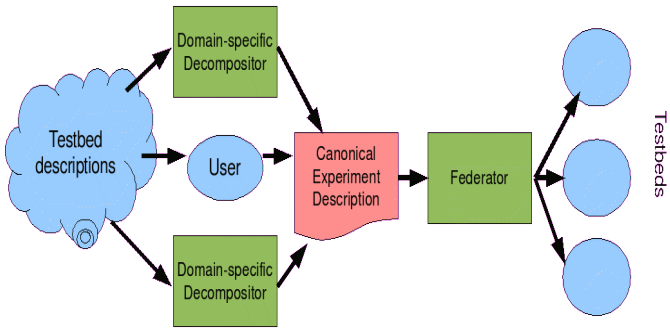

Figure 1. DETER Federation Architecture

The output of this splitting step is a topology description in a standard language, annotated to facilitate the decomposition. The federator accepts these experiment topology descriptions. Currently this language is the Emulab topology description language, based on the ns simulator language. Each node is annotated to indicate the testbed(s) in which it could be embedded. This is a standard but low-level format: we assume that in most cases this description will be generated by higher-level, more sophisticated tools. The division allows development of domain-specific annotation tools to proceed at the same time as the federator is advanced.

On the other end, the federator must communicate with federant testbeds for two basic operations: requesting permission to access resources within the federant and embedding a topology subgraph on that federant. Though testbeds have made a decision to allow their resources to be shared, individual access control decisions are made for each experiment.

The model and mechanism for this access control negotiation has been described earlier[4], and is based on a generalization of Emulab's project and user concepts. We review it briefly here to frame our experiment creation system.

In our design, users, projects, and testbeds are globally named and form the basis for individual testbeds' access control decisions. These names are self-validating in that an entity using a name can prove it has the right to do so and global in the sense that two entities confronted with the same name can be certain it pertains to the same entity. No third party is needed to assure

these properties. Here we extend this name space to include federated experiments.

Though this name space is global, it is decentralized. Names can be allocated using a probabilistic system, such as allocating public keys, or random UUIDs. Testbeds may need to know more than a user's self-generated identifier to make an access control decision, but the identifier space itself does not constrain system growth.

Access is first negotiated with each candidate testbed by specifying the kind and number of nodes sought and the global identity of the requester. The requester can be identified by any combination of a user, project and testbed. Based on that information the testbed may grant access, replying with the information necessary to manipulate resources on the testbed, or may deny the request.

Testbeds represent the access granted as membership in a local project.[1] The membership of a user in an Emulab project controls what local resources that user can access and plays into other local policy decisions. These projects and users may be dynamically created or statically allocated.

When the requester has collected access to sufficient testbeds, also called federants, the federation system begins embedding the experiment.

In the same way that our earlier access control work extended the access control aspects of Emulab projects into a federated environment, this subsystem extends aspects of projects and experiments related to the environment. In this context a project is a set of related researchers who can access experiment resources and support services. This usage is more closely related to the local access control project than the global naming construct. The services of interest include Emulab's loosely coordinated event system and the shared file system used in an experiment.

The process of creating a federated experiment is a departure from the Emulab model. Our system coordinates the resource allocation processes of several testbeds to build the experiment. Any single Emulab is the sole controller of its experiments' resources and the policies for their use. When federated, testbeds retain control of their resource allocation policies, and the federation system combines those allocations into a cohesive global allocation. The federation system knows the topology and allocation of the full experiment, while local testbeds are only aware of their sub-experiment. This partition of information means local testbeds need only administer and understand their own resources, and cannot leak additional information in competitive environments.

Mechanically, resources are allocated by the federator through the Emulab allocation interfaces. Using the resource information acquired during access control, the federator embeds sub-experiments iteratively into appropriate federants until sufficient resources are allocated across multiple testbeds to make the experiment.

1The term “project” is overloaded to mean both a global grouping of related users and a local access control construct. Projects in an Emulab similarly embody several functions.

Different implementations of federators may use different layout and backtracking algorithms. Sophisticated federators may keep track of which testbeds are lightly used or take advantage of seasonal load variations in scheduling embeddings. Other federators may be tuned to split particular types of experiments efficiently. The architecture does not constrain how the federator accomplishes the embedding.

By partitioning the embedding problem into smaller sub-problems, the system scales in terms of experiment size. Each federant only allows embedding of sub-problems it is capable of supporting, so if the federator is able to break up the experiment across them the global embedding will be feasible.

Once created, the experiment has a global identity. Controllers of the experiment can prove they are associated with it and other elements in the system can unambiguously identify it, even if they played no role in its creation. If the researcher who created the experiment allows, others may be able to query and manipulate the experiment referred to by the name.

Like other elements of our architecture, an experiment may reveal different views of itself to different users. The federator implements operations on the experiment, including the modification, termination, and exporting the various views.

We are in the process of instantiating this architecture on DETER and other testbeds. Experiment creation is operational, and higher level functions are being developed.

A. Experiment Creation Model

Creating a federated experiment across federated testbeds must accomplish three tasks. Resources for the experiment must be gathered in accordance with the policies and availability of the various testbeds, the resources must be configured into the global topology and experimental services configured, and appropriate infrastructure access granted to the experiment.

Emulab's abstraction of an experiment has proven to be an effective model for network experimentation. That abstraction consists of a controlled topology in which an experimenter can access support services. The services provide configuration and coordination of the nodes in the topology that support the study being conducted. Our system preserves as much of that model as possible while enhancing it to span administrative boundaries and scale to larger experiments.

The two aspects of an experiment must be handled differently. The configuration of resources into a topology and local software and user configuration – the creation of the experiment's world – is easier to generalize than the support services. Topologies are connected across federants by tunneling traffic from one sub-experiment to another.

Experiment support services are more specialized than traffic forwarding. Support services visible inside an experiment include the ability to access nodes through a control network that is separate from the experiment's world, the configuration of the same set of user accounts on each node and access of those users to a shared file system, and access to a loosely synchronized event-scheduling system. The shared file system

and account access is scoped to the members of a project[2] and the event system is per-experiment.

Finally there are infrastructure features of an experiment that are either invisible from within the experiment, or more conveniently and usually operated from outside the experiment. Examples of this include the ability to control the power to a node or reboot it forcibly. Such abilities are scoped by experiment.

The first class of experiment services - accounts and events - are extended by designating a master testbed that exports its instances of those services into the federated experiment. This has the effect of exporting much of the master's experimental environment into the federated experiment. Users who have accounts on the experiment/project created on the master testbed have accounts on nodes in the experiment throughout the federated experiment. All nodes see the master testbed's shared filesystem. Master services must be properly scaled to work on the entire federated experiment.

A testbed grants access to infrastructure services by granting access to a local testbed account with appropriate local privileges. The access control mechanism grants the federator access to a federant testbed in order to configure an experiment. Part of that configuration is granting the experiment appropriate access to use infrastructure services. Specifically, the federator is granted access to an account that can create experiments in a project with appropriate local permissions, and to an account with the rights to manipulate the experiment once created. The second account may not have the right to create other experiments and access to that account will be made available to the experimenter.

The following subsections describe these operations in detail.

1)Cooperative Resource Allocation

Allocating resources to an experiment is guided by the federator and subject to the policies of testbeds. At allocation time the federator has the annotated experiment description and access to testbeds that have authorized the requesting experimenter to access resources, but no hard allocations in-hand.

Allocation is an iterative process whereby the federator embeds sub-experiments it thinks are likely to be accepted by the federants until the entire topology is created. An embedding may fail because resources are not physically available at a testbed or because the allocation would violate some other local testbed policy.

When confronted with a failed sub-experiment allocation the federator can either adjust the global embedding plan or attempt to embed the same sub-experiment on another acceptable testbed. Which choices are open to the federator depend on the experimenter's preferences, expressed in annotations, and the federator's knowledge of other testbeds' states.

Once the allocation is completed, the federator knows the experiment's virtual topology and the physical realization of it. Both of these become resources accessible to the owner of the experiment and to other experimenters that the owner designates.

2)Infrastructure Access

As described above, each testbed maintains direct control over its infrastructure and resources, even when these are being used by a federated experiment. In order to use the infrastructure services of a testbed, access must be granted by that testbed. Under the Emulab model, a user can exercise control over infrastructure dedicated to an experiment without being able to create or destroy the experiment itself. As part of negotiating access to a testbed, the federator acquires both the rights to create experiments and the rights to infrastructure services.

The rights to perform infrastructure access are passed back to the experimenter rather than being used directly by the federator. While this requires the experimenter (or the experimenter's experiment control software) to be sophisticated enough to make use of these rights, we believe that distributing those rights scales better than making infrastructure service requests through the federator. In addition, it allows experimenters to quickly adopt new infrastructure services testbeds may offer.

Infrastructure access information is a resource of the federated experiment, like the virtual topology or physical instantiation information. Access to that information is controlled by the federator and is provided in line with the principles of least privilege. In a competitive experiment it may be unfair to allow one team to control the other team's power.

Some testbeds will restrict access to experimental nodes' control interfaces, usually by requiring the experimenter to access the nodes from one of the local infrastructure machines. In these cases the testbed can provide federated users access to machines using the same mechanism as granting them access to the infrastructure services.

3)Experiment Environment

Creating the experiment's environment consists of establishing the in-experiment topology and exporting the master testbed environment to other federants. In order to respect local control and to facilitate controlled visibility of the experiment among participants care is taken to avoid information pollution between testbeds. In order to simplify testbeds sharing resources, no new interfaces are added to the Emulab creation interfaces.

When the federator splits an experiment into sub-experiments, it inserts additional nodes that operate as connectors between sub-experiments. Two types of connector nodes are inserted: topology connectors and service connectors. These are logical entities that may be instantiated on the same physical node. Connectors come in pairs, each estabilshing one end of a tunnel to interconnect testbeds. Individual federants can provide hints about how to configure these nodes in their access negotiations. The hints may include particular local images or local configuration scripts customized to allow access to external testbeds. These local scripts also provide a hook for testbeds to customize the federation process to enforce their policies

2 If projects have sub-groups, account access is scoped by subgroup, but the basic scoping is by project.

The purpose of a topology connector is to tunnel experiment-generated packets between sub-experiments on different testbeds. In Emulab terms, these nodes tunnel experimental network interfaces between testbeds. Topology connectors can tunnel links or local area networks transparently, and a single connector may tunnel more than one connection. This places minimal constraints on the federation software and users when splitting experiments; the federator may split an experiment at any interconnection point between nodes.

Service connectors provide services between the master testbed and other federants. In Emulab terms, they forward or proxy services provided on the control network. Unlike topology connectors, service connectors tunnel or proxy traffic selectively. Only recognized services are forwarded. This constrains the services exported from the master and the ability of sub-experiments to disrupt other testbeds infrastructure. Service connectors can also be customized to provide testbed policy guarantees.

The two kinds of connectors impose two name spaces on the experiment. In the experimental topology – imposed by topology connectors – IP addresses are unique through out the experiment and, in principle, can be reached throughout the experimental topology. In contrast the service connectors impose a name space of services. The addresses on the interfaces used to access services (that is the Emulab control network) are not guaranteed to be unique across the federated experiment. Those addresses are chosen by the federants without coordination. Two federants that have chosen control addresses from private address space[5] may allocate the same control address to nodes that are eventually in the same federated experiment.

Each sub-experiment is amended to include enough topology connectors to form the topology and one service connector that will connect to the master testbed. The master testbed is allocated a service connector for each federant, along with the relevant topology connectors. These connectors may map many-to-one onto physical nodes.

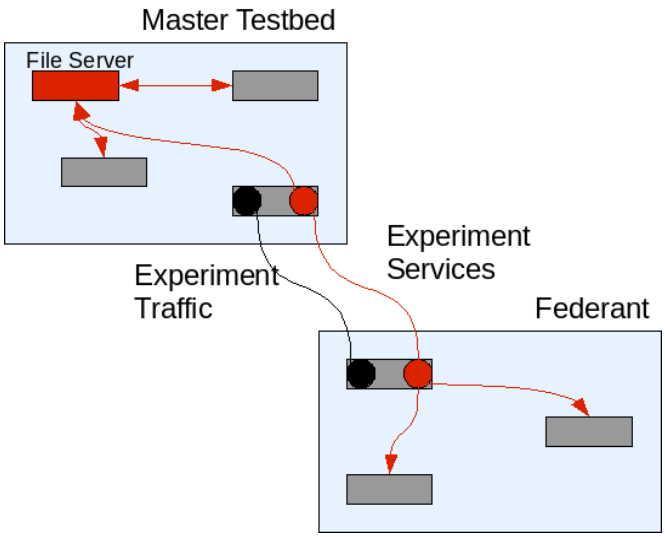

Figure 2. Connectors in a testbed

Figure 2 illustrates connector's role in federating an experiment. That figure shows an experiment federated across one master and one federant. The lines ending in arrows represent the use of infrastructure services and lines without arrows represent experiment connectivity. The master testbed is exporting services from its file server. Each experiment has a physical node acting as both a service connector and topology connector.

Connectors are designated as active or passive when allocated by the federator, with the active end doing the work to form the tunnels, after the passive end has laid any necessary groundwork. Topology connectors can be started in any order, but service connectors enforce a simple sequencing. The master side of each service connector is the active pair, and these connectors initiate their connections only after the master and all federant sub-experiments have been established. As the master services are activated in each federant, that sub-experiment considers the federated experiment to be started. A rough analog in Emulab terms is that experiment start commands can fire when the master services appear.

B. *Model Summary*

This section has described the DETER federation model with emphasis on the model for experiment resource allocation and construction of the experimental environment. The two-level resource allocation allows large experiments to be built up from sub-experiments allocated in accordance with local testbed policies. The experiment support services are comprised of infrastructure services that are under the control of individual federants and the services that create the exported master testbed environment. The testbed composition primarily occurs at topology and service nodes that tunnel experiment traffic and master services, respectively.

The following sections describe our initial implementation of this model.

III. Fedd: A Federation Daemon

Fedd is our evolving federation prototype, introduced in our description of testbed access control.[4] It continues to support global names as self-signed X.509[6] certificates and manage access to testbeds through a generalization of the Emulab project system. This section discusses the implementation of experiment instantiation in that framework. We first describe the resource allocation and access control extensions, then describe the scalable experimental environment in more detail.

The protocols for communicating with the federation system are expressed in the Web Service Description Language (WSDL)[7] and are extensible at several points. Transport Layer Security (TLS)[8] is used for mutual authentication and encryption between the system and the experimenter. Because the runtime system of our implementation also supports XMLRPC[9], fedd exports an identical XMLRPC/TLS interface. The authentication and authorization using TLS and the parameters and their encodings are all the same, but the additional interface makes porting some tools to use fedd easier. Standard tools can generate code to make and serve requests

across this interface using a variety of implementation languages.

A. Creating and Operating on Federated Experiments

When an experimenter, or more likely an experimenter's domain-specific experiment creation application, requests the creation of a federated experiment, it passes parameters in Table 1. As described above, the experiment will be issued a global name, but a fedd may also issue a local, human-readable name to simplify user interfaces; the experimenter may suggest a human-readable name in his or her request, but fedd is free to ignore the suggestion.

Based on user annotations in the Emulab/ns2 description of the experiment and mapping suggestions, fedd selects testbeds for embedding the experiment and gains access to them using the DETER architecture's access control protocol[4]. We have extended this protocol to separate the access granted to fedd to create sub-experiments and for the experimenter to access interface services.

After this access phase, fedd has the right to attempt to allocate resources on a set of testbeds. It this point, it breaks the experiment description into sub-experiment descriptions along lines suggested by the experimenter, considering the resources it can potentially acquire. Fedd generates new sub-experiment descriptions at this point that include topology and service connectors. Some additional transformations are made to establish the experimental environment; these are described below. The sub-experiments also have all experimental IP addresses assigned, to create a uniform in-experiment address space and to avoid inconsistencies in how experimental addresses are assigned by federants.

Field	Purpose
Master	The name of the master testbed in the description
Experiment	The experiment description
Access keys	Keys to use for interface access
Mapping	Suggested mapping of testbed names in description to testbeds
Name	Suggested local experiment name

Table 1: Experiment Creation Request Parameters

Fedd now creates the slave sub-experiments on federants using Emulab interfaces. Allocation of particular nodes and switch interconnections on the federants is invisible to fedd. Should one of these allocations fail, fedd can backtrack and re-split the experiment or substitute another federant. Our current implementation has yet to implement this backtracking, though it will cleanly fail and deallocate any sub-experiments.

Once the sub-experiments are allocated and started, fedd starts the master experiment. When the master sub-experiment has started, the federated experiment is declared to be started, and the experiment metadata is returned to the researcher. The parameters in Table 2 are returned.

Field	Purpose
Vtopo	Federated experiment's virtual topology
Vis	Visualization of federated experiment
Emulab	Federant information (including access keys)
Name	Global name and optional local name of federated experiment

Table 2: Experiment Creation Response Parameters

The current implementation returns the global name as an X.509 certificate and associated key pair. The exchange is encrypted to avoid eavesdroppers acquiring the keys, and the private key immediately destroyed on the fedd. This certificate and key are controlled by the user application, and can be used to act on the federated experiment. Any connection from a holder of that certificate and its private key is allowed to operate on the experiment. Additionally the experimenter who created the experiment may act on it. Passing this certificate around is a convenient way to delegate access to the experiment.

Once an experiment exists at a fedd, researchers with appropriate permissions – creators or holders of the experiment's private key - can query its virtual topology, its visualization information, and deallocate it. Access to local nodes is through the individual testbeds based on the access controls in the master environment.

The infrastructure access information, in this case which ssh key or X.509 certificate is valid on which testbed, is returned to the requester. In addition to allowing users to access federants to use infrastructure utilities, such as power cycling nodes, these access keys may be necessary to access nodes through the control interfaces., as described above.

B. Building a Scalable Experimental Environment

There are many details to get right when creating the experimental environment. We discuss the export of the shared file system and local accounts, custom software installation, and the event system. In each case, the services were chosen because DETER users make use of them commonly to facilitate experimentation. Each of these services is implemented using slightly different techniques, and together they illustrate some of the range of techniques in service export.

In each case the services are provided by a combination of reconfiguring experimental nodes in the federants and accessing master testbed services using service connectors. These actions are initiated using Emulab's *start command* interface. That interface schedules a command for execution when all nodes in the local experiment have reported themselves as configured. When fedd rewrites the experiment descriptions, it adds start commands that establish federated services before invoking any user-specified start commands. These extra start commands are

invisible to the experimenter. The federation start commands on experimental nodes are synchronized by the service connector in the experiment. Service connectors are synchronized by the master testbed. The result is an approximate synchronization across the federated experiment.

Our current implementation uses the secure shell (ssh) to tunnel services between master and federants. The fedd access keys are used to secure the connections. Ssh was primarily a choice of convenience; it is available, secure, and straightforward to configure. We believe that the choice of ssh in this role is not constraining. Any secure packet tunneling system will have the same data management and key distribution problems. Slotting IPSec or another secure tunneling system should be a mater of getting the configuration details right, not of changing the service connector model.

1)User Accounts and File Systems

Each node in a single-Emulab experiment has a unified user name/user ID space and those users share a global file system. This is one of the most commonly used services in Emulab – DETER users make use of it so often that most would characterize it not as a service, but as a fundamental property. It is this combination that allows a user to access each experimental node using the same ssh key installed on the users node. A federation system without it would not really be generalizing an Emulab experiment.

Overlaying the account information and importing the file system makes the experimental machines accessible only to the users in the project on the master testbed. The local federant can still control the node to a degree through its control of infrastructure.

Rather than using a network account management system such as LDAP (the Lightweight Directory Access Protocol) or NIS (the Network Information System, formerly Yellow Pages), Emulab rewrites local account databases on each machine at boot time using data from the local testbed infrastructure; user accounts are imported via the Emulab testbed configuration protocol (TMCD) interface. To import accounts from the master testbed's project, Fedd inserts a script that deletes the local project accounts and repeats the account addition process using data tunneled from the master testbed. The service connector forwards the TMCD connection from the master testbed.

Local Emulabs primarily use the Network File System (NFS) [10] to share files, but the system also export the file systems using Shared Message Block protocol/Common Internet File System (SMB/CIFS)[11]. The DETER federation system uses the SMB/CIFS file system on federant nodes.

Several practical concerns led to this decision, but the upshot is that SMB/CIFS is more straightforward to tunnel. NFS generally makes export decisions based on the IP address of the node making the requests, which means that service connectors would have to perform network address translation on these requests, rather than a straightforward tunneling. Because some testbeds allocate control net addresses from private IP address space, collisions are possible at the master server. Furthermore, NFS is accessed using Sun RPC's dynamic port allocation and portmapper, which is another somewhat complex service to tunnel. Tunneling SMB/CIFS is a matter of encapsulating the packets in the federant and sending them to the master. Only the user's credentials are considered in accessing the file system, not the node's origin.

Unfortunately, SMB/CIFS and NFS are client/server file systems, and Emulab generally uses one file server per testbed. The scaling implications here are obvious and unfortunate: a testbed hosting a large federated experiment or many federated experiments will eventually see its file server overloaded.

In this case, DETER federation chose compatibility over scale. Requiring testbeds to support a new file system is too high a barrier for adoption of federation. The system tries to mitigate server load somewhat by mounting experimental node file systems on demand, rather than all at start up. Fundamental scaling issues remain, however.

In the future those scaling issues can be addressed by using one of the wide-area file systems built by the research community. The architecture could easily support distributing such a system if a testbed exported it. The Andrew File System[12], or its Coda extension[13], are good candidates. Both aggressively cache to achieve better scaling properties than NFS. Coda's disconnected operation mode is particularly attractive, as it may mitigate temporary connectivity failures between federants. Their integration with Kerberos makes deployment somewhat more complex than SMB/CIFS.

2)Custom Software Installation

A powerful feature of the Emulab configuration system is the ability to install software from a tar file or an RPM onto experimental nodes. DETER federated experiments support this feature, and provide it directly and at scale.

Software installation is provided not by tunneling service directly back to the master testbed, but by acquiring the software, distributing it to testbeds during the sub-experiment embedding process and using local testbed installation facilities to install the local copies.

This requires fedd to acquire copies of the software (RPMs may be specified by URL), use the access granted for experiment instantiation to copy the software distribution into federants, and rewrite the requests in the experiment to request the local copies.

This two-tier distribution mechanism is more scalable than a naïve tunneling of file system access for each distribution. All accesses remain local and the software is transferred once for each federant rather than once per node.

3)Loosely Scheduled Events

Emulab provides a simple event system for automating experiment operation. Event delivery can result in changes in link behavior or in commands running or terminating on experiment nodes. Though events can be delivered to multiple handlers, the synchronization is loosely defined. Essentially a publish/subscribe system sends a message and the event occurs at the node or link when the message arrives. Propagation delays may vary, a process exacerbated by retransmissions or delays induced by federation. Though simple, this system provides sufficient synchronization for many experimenters and

some testbed operations. The architecture does not attempt to replicate the event-based testbed operations.

The publish/subscribe system used internally by Emulab has transitioned from the elvin[14] messaging/notification software to their own system with similar features. Emulab also includes a backward compatibility interface to their system than mimics elvin. Emulab messages are collections of attribute/value pairs.

A key feature of these systems is that they allow nodes to act as repeaters. A node can both accept subscriptions and subscribe to other nodes' messages. This feature can be used to build scalable distribution trees.

The DETER federation system starts a publish/subscribe system on each service connector. Each system subscribes to events for its local experiment on its local testbed and to the remote testbed as an event generator. Any events generated by the local testbed are forwarded to the remote side, after addressing attributes are changed to reflect the remote experiment and an attribute indicating the origin testbed has been added. Inter-testbed events cross the connectors once, and internally testbeds use whatever event distribution configuration is appropriate for their scale and conditions.

The event repeater software is a service proxy, not a simple tunnel. It understands the event message format and edits messages to reflect their use in a federated environment.

4)Experimental Environment Summary

This section has described several of the more interesting aspects of creating the shared environment, addressing compatibility, scalability, and utility. The techniques cover tuned use of existing services to install software, simple service tunneling to provide file system access, and application proxies to bridge scalable services between testbeds in our environment.

IV. Experience

Though fedd is a new feature of DETER, we have had experience using it to embed experiments. This section discusses work embedding large experiments that helped assess the interface compatibility and file system scaling properties as well as the work done to provide an interface from DETER's graphical experiment management tool, SEER[15], to fedd.

A. Implementation

The current implementation of fedd has brought together several subsystem prototypes into a unified tool that is more appropriate for general users. The design splits code along the architectural functionality lines. Where necessary, explicit interfaces, described in WSDL, enable a distributed implementation. This distributed implementation is practical as well as aesthetic; different testbed configurations and access policies argue for different placement of fedd functionality on boss and users.

The current implementation of fedd uses the python programming language[16], because it supports clear modularity boundaries without excessive complexity in enforcement. Additionally, python has a broad range of support libraries that simplify and encourage the use of standard interfaces. SOAP remote procedure calls derived from WSDL descriptions are readily supported. With minimal changes these same routines allow XMLRPC access as well. The simple integration of XMLRPC was practically useful as existing Emulab software exports XMLRPC interfaces; SEER makes use of these interfaces and providing XMLRPC access to fedd simplified SEER integration considerably.

The DFA definitions break fedd implementation into modules that manage experiment creation and manipulation, that manage access, and that convert access decisions into concrete testbed configurations. The fedd implementation is comprised of modules that accomplish those tasks. This factoring is visible in the structure of the python classes that make up fedd and in the published interfaces between modules.

The ability to locate different fedd functions on different physical machines allows administrators to install it in their testbed without great disruption. For example, an administrator who is going to dynamically allocate testbed resources must run that function on the machine that hosts the Emulab configuration database, a risk static allocators can avoid.

The various design specifications and source code for fedd, as well as information on configuring and installing it is available from *http://fedd.isi.deterlab.net.*

B. A Large Experiment

One of the earliest tests of fedd was to take a simple experiment being run by a user who was uninvolved with the federation system development and see how well they were able to use it to scale up their experiment.

Our test researcher was able to scale their experiment up to more than 210 nodes and run an experiment larger than any of the three testbeds would have supported at the time. We note that the experimenter was competing against other users for federated testbed resources and that other testbeds were not granting this researcher unusual priority. The 210-node number is not an upper bound on the system's scale.

The key lessons from this large experiment were to make the Emulab notion of non-essential nodes visible in federated experiments, to make fedd's experiment creation process incremental, and to implement the on-demand file system mounting discussed above.

An individual testbed may mark nodes as non-essential to the experiment by using an Emulab command to set that attribute. When such a node cannot be started, the Emulab considers the experiment creation to be a success, though the experimenter cannot access that node. This becomes key when creating large experiments as load on the local experiment creation and experiment management services, e.g., the shared file system, can be considerable.

This feature is useful, but it becomes more so when combined with iterative federated experiment creation. If fedd is instructed to create an experiment that is already instantiated, it modifies the experiment in place rather than removing it from the local testbeds and reallocating them, assuming local testbed policy permits this.

While it can be difficult to capture a large number of nodes from many testbeds in one gulp, an iterative expansion of the federated experiment can be more successful. This iterative expansion was used to acquire the largest set of nodes.

The experiment did not make aggressive use of the exported file systems, but at times the load caused by the large experiment was considerable. An experiment that was more demanding may have disrupted service on the master testbed, which was exporting files to 130 nodes more than it was configured to support. This heavy loading led us to implement the on-demand file system mounting in the federation configurations today.

Overall this experience was valuable in showing the system's ability to scale to useful sizes and that the shared environment conformed closely to the familiar experimental interface. It was also encouraging that the user was able to adopt intuitive methods for allocating a large experiment without prompting. He simply tried to swap in a larger experiment as if on a local Emulab, and the system behaved as he expected.

C. SEER

To DETER users, SEER[15] is an approachable experiment management tool. It allows users to create experiments on DETER, configure traffic, initiate complex actions, and graph traffic in real time. To the federation developers it represents both a complex testbed application to support and a significant improvement in usability of the system. The latest SEER supports federated experiments.

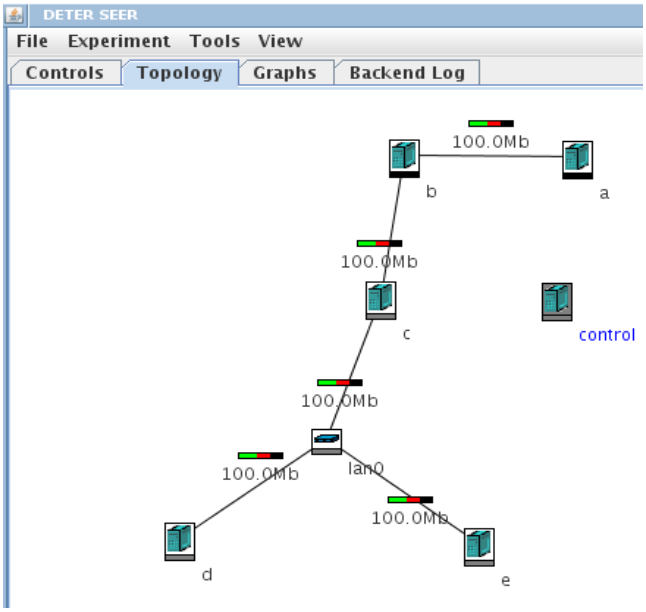

Figure 3. The SEER view of a federated experiment

SEER is the most complex system we have federated to date. It makes aggressive use of the event system, loads support software on all participating experimental nodes, and communicates with the outside world through a controller node in the experiment. It was created before the DETER federation system was planned, so its design made no accommodation for federation.

Fortunately SEER's design was sound. Because the SEER controller used the Emulab event system to coordinate with SEER processes on other nodes rather than creating its own control net protocol, it was amenable to federation.

Before modifying SEER, we were able to make use of basic functionality by connecting it to one of the sub-experiments as though it was a full experiment. We could issue SEER commands to the sub-experiment (and beyond) though only the subnet was visible. Changes to SEER were required to make it a federated experimentation tool.

The issues to be resolved were acquiring information about the global experiment topology and restricting naming to in-experiment names.

SEER uses topology information to draw a representation of the experiment and allow the experimenter to manipulate those elements, e.g., starting or stopping traffic flow or an aspect of the system being studied. This information is available for sub-experiments through an Emulab XMLRPC interface. For federated experiments, SEER needs to contact the controlling fedd and request the information with the local alias or global name. This functionality was added to SEER. For simplicity in early prototyping SEER accesses fedd through the XMLRPC interface, as SEER already has an extensive XMLRPC runtime system.

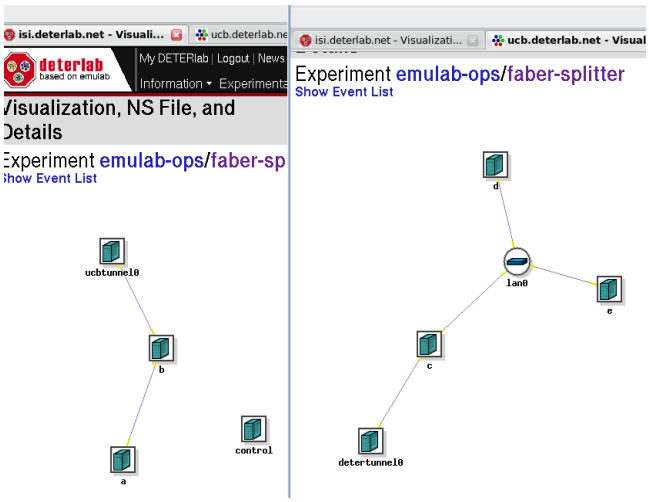

Figure 4. The sub-experiments

A more key change in SEER was restricting the naming to names that were meaningful to the federated experiment. In some cases, SEER addressed events to machines using their control network IP address. As we have mentioned, the IP addresses in the experimental topology are visible across the entire federation, but the control net addresses are not. They may not be routable on another testbed or may be aliased by a local choice. In general, these addresses were used to optimize event routing, not define it, so the federation event proxy was able to remove the incorrect optimization information from the nodes.

As we revamp SEER to function more cleanly in a federated world, we are removing such dependencies on per-experiment information and restricting it to using information in the federated experimental name space. This includes the symbolic names of hosts or global names.

Experimenters can currently use SEER to create, manipulate, and delete federated experiments in ways analogous to single testbed manipulations. Figure 3 shows the SEER view of the two combined experiments shown in Figure 4. Both figures are screenshots, Figure 3 from SEER and Figure 4 from the DETER web interface.

The experiment in Figure 3 is nodes a, b, and c connected directly, with node c sharing a local area network with nodes d and e. The unconnected control node is the SEER controller. It communicates with the other nodes through the event system and has no connections in the experimental network. Only the federated experimental network is displayed by SEER, and federation infrastructure is similarly invisible.

The experiment has been split by fedd between node b and node c. Figure 4 shows the instantiation of the two sub experiments on two Emulabs. The left sub-experiment includes nodes a and b and the control node. The Fourth node is a node hosting the topology and service connectors. The right hand testbed hosts nodes c, d, and e on their LAN as well as a combined topology and service connector.

V. Conclusions and Future Work

This work has described the DETER federation architecture's model for creating federated experiments, including the multi-phase cooperative resource allocation system, the access control levels, and the construction of a cohesive experimental environment. Later sections described our ongoing implementation, fedd, and discussed our experiences creating large experiments and adding support to experiment management tools.

The architecture has shown itself to be powerful and extensible as we have developed and expanded it. The implementation supports large experiments and complex systems, such as SEER.

Future work focuses on realizing more parts of the architecture and expanding the function of existing implementations. The core federation implementation has reached the point where it can support domain-specific experiment design tools that are aware of federation. Such tools are a necessary next step in making testbeds more useful in research and it is key that federation be designed in them from the beginning.

Competitive experiments are an exciting possibility opened up by federation, and expanding the fedd implementation to support multiple experimenters is key to realizing that possibility. The experiment descriptions need to be further annotated so that the federation system can determine the levels of visibility and access control to services. From that fedd's query system and experiment creation systems need to be extended to support experiments that appear differently to the competitors.

While X.509 certificates have been a useful initial implementation of our global namespace, future versions will support more ways for name holders to authenticate themselves. We are working to adapt fedd's interfaces to support other authentication systems.

The point of these various thrusts is to make federation more accessible and powerful in ways that allow new kinds of research to be done in federated testbeds. We believe that the DETER federation architecture will support these new directions and the fedd is promising evidence that the architecture is realizable.

VI. 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," OSDI 2002, December 2002.

[2] T. Benzel, R. Braden, D. Kim, C. Neuman, A. Joseph, K. Sklower, R. Ostrenga, and S. Schwab, "Experience with DETER: a testbed for security research," Proceedings of Tridentcom (International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities), March 2006.

[3] http://wail.cs.wisc.edu/projects.html

[4] T. Faber and J. Wroclawski, "Access Control for Federation of Emulab-based Network Testbeds," In Proceedings of the CyberSecurity Experimentation and Test (CSET) Workshop, San Jose, July 2008.

[5] Y. Rekhter, R. Moskowitz, D. Karrenberg, G. J. De Groot, E. Lear, Address Allocation for Private Internets,, RFC1918, ISOC, February 1996.

[6] ITU-T Rec. X.509: Information Technology Open Systems Interconnection - The Directory: Interconnection framework, June 1997.

[7] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, Web Services Description Language (WSDL) 1.1,, http://www.w3.org/TR/wsdl

[8] T. Dierks, E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.1, RFC4346, ISOC, April 2006.

[9] D. Winer, "The XMLRPC Specification," http://www.xmlrpc.com/spec, June 2003.

[10] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and Implementation of the Sun Network File System," USENIX Conference, USENIX, June 1985, 119-130.

[11] Microsoft Corp., "Microsoft SMB Protocol and CIFS Protocol Overview", http://msdn.microsoft.com/en-us/library/aa365233.aspx .

[12] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West, "Scale and performance in a distributed file system," ACM Trans. Comput. Syst. 6, 1 (Feb. 1988), 51-81.

[13] J. J. Kistler and M. Satyanarayanan, "Disconnected operation in the coda file system," ACM Transactions on Computer Systems, ACM, vol. 10, no. 1, 1992, 3-25

[14] B. Segall and D. Arnold, "Elvin has left the building: a publish/subscribe notification service with quenching," Proceedings of AUUG97, Brisbane, Australia, September 1997.

[15] S. Schwab, B. Wilson, C. Ko, and A. Hussain, "SEER: a Security Experimentation EnviRonment for DETER," In Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test, August 2007.

[16] Python Software Foundation, http://www.python.org.