A DETER Federation Architecture

Ted Faber

John Wroclawski

Kevin Lahey

USC/ISI
1 June 2007

1. Introduction

DETER testbeds[1] are a class of Emulab-based[2] testbed that provide a secure experimental environment through configuration and management. The DETER project[3] at ISI runs a DETER testbed with segments located at USC/ISI and the University of California at Berkeley consisting of more than 250 nodes. In addition to the original Emulab[4], there are several other testbeds based on the same software scattered throughout the US, e.g., Wisconsin's WAIL[5], with different research focus and varying degrees of public access.

Given that each of these testbeds have similar underlying resource and allocation models, but are tuned and managed for specific objectives, it is natural to want to interconnect them. A given experiment may need more resources than a single testbed can offer and interconnecting them can provide more. A testbed may contain hardware useful to an experimenter but lack other resources needed for the experiment; connecting two or more testbeds is a natural solution. An experiment may itself naturally combine the environments provided of two or more testbeds. An experiment that consists of a secure set of nodes and a set of nodes outside the secure area may be best implemented as the connection of a DETER secure testbed and another testbed, rather than simulating the two regimes inside one testbed.

We call the process of deploying an experiment across multiple testbeds *federation*. The DETER project has been able to distribute experiments across multiple sites (ISI & Berkeley) and to temporarily incorporate resources outside the testbed for experimenters. These experiments have convinced us that being able to combine DETER experiments with remote resources temporarily include outside resources is useful. To date, these successes have been very labor-intensive. An automated system for creating federated experiments is called for.

Some of the challenges of federation are well-understood distributed systems issues such as a establishing shared naming, authentication, and access control authorities that the various testbeds understand. In this case, a well understood system is not a trivial one, but there is considerable related work in these areas.

Other aspects of the problem are thornier because the resources of testbeds are not simply pooled, but are managed by the owners of the various testbeds in different ways. The management and use policies are not simply matters of convenience to the owners that can be negotiated away; the critical properties of the testbeds can depend on these policies. Aspects of DETER's containment and security guarantees are based on the administration of the testbed nodes and the configuration of its physical networks. Finding and using resources subject to the use constraints of the component testbeds complicates resource discovery and allocation for federation.

Any useful federation system must be supported by a significant set of the Emulab-model testbeds, and furthermore, resources being scarce as always, the federation system will be developed in parallel by designers of those testbeds. Enabling this shared use and development requires that the players agree in a shared architecture and set of interfaces. The architecture will provide a shared vocabulary for discussion and design and partition function into components that can be orthogonally developed. In addition, testbeds supporting different controllers or maintaining different invariants may implement components in frameworks geared to their environment or using algorithms appropriate to their goals. Fixed interfaces allow testbeds to continue to share resources even as underlying software is changing.

This paper lays out a draft straw man architecture as a starting point for consensus. It is by its nature not complete and should be taken as a draft and

a starting point. Disagreements remain on the straw man even among the authors of this document. This document introduces the framework, but space prohibits a detailed interface description.

This document addresses the key name spaces and components used for creating, manipulating, and tearing down experiments. Though we touch on the need for resource discovery systems and recognize places where trust must be established, specifications of those aspects of the system are out of scope for this paper.

The discussion proceeds as follows. Section 2 lays out the ways in which the resource and allocation model of these testbeds affects the specification of the system, Section 3 lays out the straw man. Section 4 concludes.

2. Distributing the DETER/Emulab Model

Because the testbeds share an underlying model, inherited from Emulab[2], an natural way to proceed is to extend the elements of that model into a more distributed world. This section discusses the significant features of that model and shows how they will fit into the architecture in Section 3.

2.1. Access Control

Testbeds enforce use policies on their resources to maintain the guarantees of the installation. In a distributed environment, testbeds represent the borders of an area of policy adherence and property enforcement.

In the single-testbed model a testbed authorizes *projects* to make use of its resources after the testbed managers vet the attributes of the work to be done and the identity of the proposers. Associated with projects are *experimenters* who can request resources to be allocated and configured into *experiments*. Once created, any *user* configured by a project may use the resources of the experiment. The documentation of Emulab and DETER refers to both users and experimenters as "users," but in discussing federation the distinction between the entities that can request resources, experimenters, and the principals that manipulate them, users, is material.

The relation between experimenters and projects is a little more complex. Each experimenter is in one of three classes of privilege in a project, with the classes forming a strict containment. The least privileged experimenters in a project can make use of experiments created by others in the project, the next class can create experiments as well, and the most privileged experimenters can admit new experimenters to the project and change experimenters' privilege levels.[1]

The project and experimenter together embody the access control. An experimenter can only request an experiment as a member of a project, and the access permitted to the experimenter depends on the experimenter's standing in the project. An experimenter may be unable to allocate any resources on behalf of one project, but capable of managing resources on behalf of another. In principle, if rarely in practice, projects can be given limits to the resources they can access as an aggregate.

Because access rights are granted to projects in single testbeds, it is natural to make projects visible to other testbeds and use them as a basis for access control.

While vesting projects with access control rights provides a mechanism for testbeds to authorize access, more needs to be done to ensure that the testbed granting resources understands the guarantees that the testbed requesting access is making about the project. This negotiation will be a bilateral one, and requires agreements between the owners of testbeds. While the architecture allows such negotiations to be carried on pairwise, scaling to many testbeds may require a more scalable trust model.

2.2. Federation Objects and Scope

Within a local DETER testbed, there are two name and object scopes, testbed and experiment. Projects and experimenters have a testbed scope – they exist independently of a given experiment. When an experiment is created, per-experiment objects are created, e.g., loghole directories and local DNS names. As multiple testbeds begin to name and use each others resources, it is important to clearly lay out the scope of each kind of name and object. Table 1 summarizes DETER elements and their naming scope.

Object	Scope
Projects	Testbed
Experimenters	Testbed
Experiment Nodes	Experiment
Inter-Testbed Connection Points	Testbed
Classes (node & image)	Testbed
File space	Experiment
Experiments	Testbed

Table 1: Objects and Scope

In the federation architecture, the two scopes remain but experiment scope can now span multiple

[1] Projects can be further divided into groups, but that is not a fundamental feature.

testbeds. That is, all the nodes allocated to a given experiment see a shared, unqualified name space; resource names do not reflect the testbed providing the resource. One aspect of federating an experiment is constructing an experiment name space that maps objects from multiple testbed name spaces into one experiment name space.

Each testbed has a unique identifier that is used to disambiguate names in its name space from names in other testbeds' name spaces. The prefix identifies the testbed that has named the object and owns it. If there are a few testbeds that can federate, the DNS name of the testbed's local controller may be the prefix. If more testbeds are envisioned or testbeds appear and disappear commonly, a unique prefix that can be created without coordination is a better choice, e.g., a Universally Unique Identifier (UUID)[6].

An individual testbed may only be able to resolve names scoped by a testbed it trusts. Once two testbeds have communicated, they can exchange prefixes and therefore know how to disambiguate each other's names. Testbed/name space discovery is beyond the scope of this document. Experimenters may acquire the disambiguation prefixes out of band, should they need to specify particular nodes in their configurations.

Experimenters will specify most experiments in terms of classes of nodes and classes of system images, described below. In cases where specific nodes are required, explicit testbed name space disambiguation of the node name can be used. An experiment that requires specific nodes is more difficult to instantiate and less portable than one referring to nodes by class.

When an experiment is instantiated on a single testbed, a local DNS name space is constructed that allows the experimenters to refer to allocated nodes by names picked by the experimenter in the specification. Generalizing this process to map strings to network addresses in the federated experiment is fairly straightforward. These per-experiment names are the primary way experimenters interact with the nodes.

Experimenters associated with the project are granted access to the federated resources, using the same access control mechanism that the federating testbed uses. In cases where that access control method violated use agreements, an alternative system may be negotiated as well. All experimenters associated with a project are granted access to the shared resources, so the identity of the requesting experimenter is not part of the access control decision.

The earlier discussion has mainly dealt with how much of the object space is visible to experimenters in specifying and configuring experiments. Also at issue is how much of the internal name space and topology testbeds expose to one another during experiment creation. This is addressed below, but testbeds will need to name at least the nodes used to interconnect them.

2.3. Classes and Images

Other system identifiers of note in the local testbed are the classes of nodes and system images. Experimenters can request resources by name or class and can request node configurations by image name, though additional configuration parameters are discussed below. The assignments of class and name are local matters, but conventionally nodes are assigned names sequentially (node1, node2, ...) and classes are assigned based on the common needs of the experimenters. For example, if processing power is important, nodes may be grouped into classes by processor clock speed; if communication access is important, they may be grouped by number of interfaces. Node class is a single string, and often multiple attributes are encoded in it. Similarly, images are named to reflect the operating system and application software included on them.

Though nodes are commonly accessed by node or class name, there is a set of attributes common to Emulab testbeds that describe each node. A similar set of attributes describes each image, though less completely. Currently these attributes are a global convention, but the management of the attribute space is addressed below.

Unlike group names, class and image names need not be global, and are primarily a notational convenience. A class is expressed as a named set of criteria; nodes or images meeting

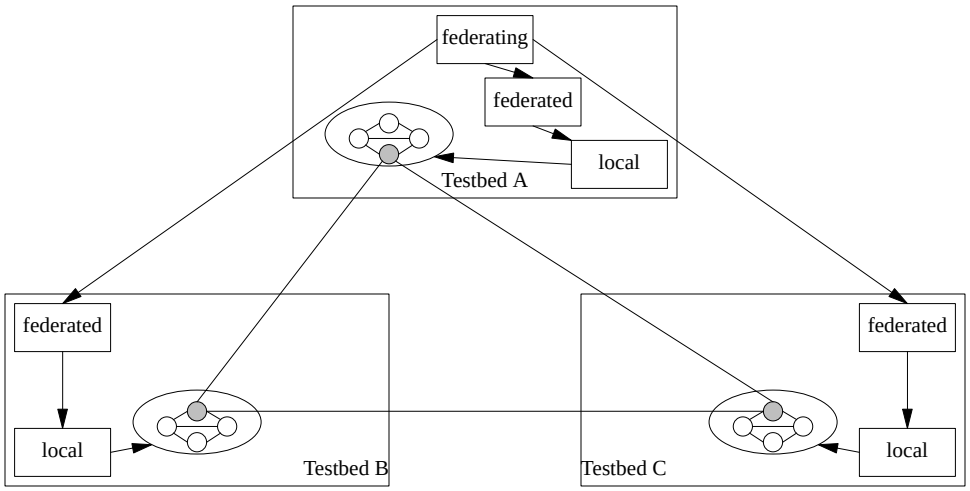

Figure 1: Federation Architecture

the criteria are in the class or excluded. Though class names are part of a testbed's name space, any node or image can be tested for inclusion.

Experimenters will generally refer to resources in their experiments by class names, reducing the amount of inter-testbed detail that needs to be passed around. This does depend on testbed designers agreeing on a set of attributes that describe nodes. In the current simple model where DETER testbeds consist mostly of general purpose computing nodes specialized in software, a consensus is fairly easy to forge. If resources become more complex, class descriptions may need to carry information about the ontology in which rules are expressed and other generalizations.

2.4. Resource Allocation

In each testbed, resource allocation is centralized at a single *local controller* called a *boss*. The local controller takes requests from experimenters associated with projects and allocates and configures experiments. The requests are usually expressed in terms of the classes described above. Configuration consists largely of loading system images onto nodes and establishing proper interconnections on the local network.

Generalizing this process will require decentralizing some of the resource allocation processing, with interfaces to make local resources visible to other testbeds and to allow the remote testbeds to request allocations to projects in the shared name space. Local resource allocation and configuration necessarily remains in the control of the local testbed. This recognizes the reality that local controllers generally control the access to testbed nodes and that local testbeds must enforce their local use policies to protect local experimenters and experiments.

At one end of the spectrum, testbeds may export detailed information to a central allocator that calculates the experiment layout. At the other end is a more decentralized version where testbeds provide some information about their resources and a central allocator creates a high-level plan that is refined by each federating testbed. The distributed nature of administration in federated testbeds argues for a more decentralized approach, but such approaches are more complex. On the other hand, centralized systems require more disclosure.

Though the shape of the resource allocation system is emerging the naming system above supports either. For scaling and to respect the privacy and control of local testbeds, a decentralized system is called for, but the exact division of labor is yet to be determined.

The next section takes these broad generalizations of local testbed operations and connects them to new federation components to form a federation architecture.

3. Architecture

Extending DETER to federate with other testbeds requires the addition of 3 new functional entities, a *federating controller,* a *federated controller, and a federation network connector.* In practice these may be collocated, and co-implemented, with other DETER subsystems.

The federating controller is the central government of the federation, responsible for assembling and configuring the resources as well as ensuring a

consistent experimental environment. A federating controller does the deployment and configuration of the experiment largely through proxy operations actually executed by the federated controllers – the states of the federation. Though a federating controller expects its requests to be carried out, federated controllers remain ultimately in control of their resources. Federated controllers implement their control by interfacing with the existing local controllers.

Once the federated experiment has been established, there is little for the controllers to do, but the network connectors will be busy passing data between the federated testbeds. At their simplest, the network connectors are nodes in the federated experiment that can send traffic between federated testbeds. More sophisticated connectors may also monitor traffic for containment breaches, do traffic shaping, or encrypt traffic between testbeds.

Figure 1 shows an experiment running on resources federated from three testbeds. Testbed A is the federating testbed and is using resources from Testbeds B and C in a federated experiment. The federating controller is coordinating with three federated controllers to allocate and configure nodes. The network connectors, shaded within the sub-experiments on each testbed, are passing experiment traffic between testbeds.

As always when laying out an architecture, the boxes indicate functional components; they may be implemented as extensions to current testbed implementations or as new standalone components.

The remainder of this section will define the requirements on each of the three entities.

3.1. The Federating Controller

The federating controller guides the creation of a federated experiment. It is responsible for discovering resources available to the project requesting the experiment; deciding how to compose the experimental environment, both topology and shared file spaces; coordinating the allocation and configuration of resources; monitoring and maintaining the experiment, including terminating it when appropriate; and collecting measurements or other data after after the experiment terminates. These tasks may well become modules in an implementation of a federating controller.

To accomplish these tasks, the federating controller communicates with the experimenter and the various federating controllers. These interfaces are fixed, allowing for multiple implementations of the components or evolution of the implementations.

3.1.1. Discovering Resources

The process of discovering resources is a negotiation between testbeds about what resources are available and which projects can use them. In the simplest case, the federating controller contacts the federated controllers at testbeds that the federating controller knows. The federating controller asks for resources available to the project (named in its name space) under any additional constraints it may enforce. Each federating testbed would reply with available resources, named in its testbed name space. The federating controller would then proceed with selection.

That approach only scales to a few federations at a time over a few testbeds – where "few" is implementation dependent – so it is important to design the interface here for expansion. A likely more scalable approach is to include resource brokers that assist in collection and allocation of resources.

Though the broker model is different, the interface between broker and federating controller is essentially the same as that between federating controller and federated controller. In either case, a set of resources meeting a set of constraints is requested and provided. The broker/federated controller interface is also similar, whether the federated controller is speaking to a broker or a controller.

Careful design of this interface – and of all the interfaces in this section — supports evolution of the system as a whole.

3.1.2. Experiment Composition and Allocating Resources

This document discussed earlier how the nature of the resource allocator depends partially on the amount of information that a testbed releases through its federated controller during resource discovery. Using either distributed or centralized allocation, the federating controller guides the process, identifying which clusters of resources can be connected and which parts of the allocation can be made at federated controllers. As all these actions are taken on the part of a project creating an experiment, the federating controller's trust in the project entity, perhaps combined with knowledge and trust in the testbed originating it, will be controlling.

Again, it is instructive to think about how the allocation process changes as the number of federated experiments and testbeds grows. With only a few testbeds and a few experiments, the simple resource discovery model is mirrored by a simple resource allocation model. The federating controller breaks the experiment into pieces that can be realized in the few

clusters of federated resources it has found, and asks the local federated controllers to allocate them. Configurations in the network controller connect the allocations.

As more testbeds and requests appear, it becomes clear that more incremental and deadlock-resistant algorithms are going to be required. The possibility of experiments that are composed of small allocations from many testbeds is a challenge case.

In any case the allocations will be distributed among testbeds and the federating controller is responsible for monitoring the experiment creation, perhaps correcting minor faults, and presenting the experimenter with the established experiment.

Picking the right scalable resource allocation algorithm is not the point of this document, but recognizing that the federating/federated controller resource allocation interface must be both small enough to be simple and expressive enough to allow multiple resource allocation implementations is. The interface will need to support requests to create experiment segments that a federating controller can string together into shared experiments. Some form of pre-allocation or locking of resources is also likely to be useful, with deadlock avoidance help as well. Again, the right interface will simplify interposing brokers or other scaling entities.

3.1.3. Configuration of Resources

Discovery, composition, and allocation all manipulate testbed level objects into a topology specified by the user. Configuration of the resources constructs the experiment-level name space, as well as actually creating the local and inter-testbed interconnections. The federating controller's role in this is primarily limited to making the experiment name space available to the various federated controllers as well as providing access to the local resources that will be rolled into that name space.

This interface provides the shape of the experiment name space and the initial contents of the shared file space. The shared file space may be replicated in various testbeds for performance or security reasons and copies of images from this testbed's name space may need to be transferred or staged as well.

We discuss this process more below.

3.1.4. Experiment Termination

When a federated experiment terminates, the federating controller must collect any experiment state left in other testbeds. Such state may include measurements, logs for debugging, or checkpoints. State may come to be on remote nodes directly through being written to nodes' storage resources or being written to parts of the shared file space.

Generally this process is the reverse of the process of providing configuration information. Rather than providing the seed information for node and testbed configuration, the federating testbed is collecting information from those spaces. Each seeding operation discussed above must have a reverse operation to collect changes from the seeded object.

3.2. Federated Controller

Much of the division of function between the federating controller and the federated controller has been addressed above, so this section will focus on the relationship between the local controller and the federated controller. Of particular concern is the creation and management of the experiment name space.

The federated controller interfaces to the federating controller and to the local controller. It is responsible for releasing information about available resources to qualified federating controllers; allocating local resources, especially inter-testbed connections; configuring local resources to create the experiment environment; and determining when an experiment has terminated and cleaning up the local portions of that environment.

3.2.1. Discovery and Allocation

The federated controller acts as a gatekeeper for information about testbed resources during discovery and converts inter-testbed allocation requests into local controller actions during allocation. The federated controller queries the local controller about available resources and uses trust information about the project (and the testbed hosting the project) to determine how much information to release. This requires an interface to the local controller to discover this information, and the previously discussed interface to communicate with the federating controller.

Allocating resources to an experiment consists largely of proxying the federating controller's requests for sub-topologies into local allocation requests. All operations that the federating controller can request must be converted by the federating controller into an equivalent local request. Potentially, an appropriate federated controller could interface a non-DETER-based testbed into a DETER federation if a proper translation could be found.

3.2.2. Configuration

The most intricate process of the creation of a federated experiment occurs at this phase, managed

by the federated controller proxied into the local controller. The federating controller provides the parts of the experiment name space to be managed by this controller, the parts of the testbed name space to be imported into the experiment name space, and the testbed interconnection information to configure the connectors. From this the nodes are configured and the name space created.

There are many entities in the experiment name space, but node names are intuitive and instructive to consider. Each testbed will be constructing part of the name to node mapping that the experimenter will see. Each testbed will be establishing part of the node name space in parallel that must fit seamlessly with the others. Proper configurations of the name space must be made so that users in one testbed can resolve names established by another.

While the node name space is created from the specification, parts of the experiment name space must be imported from the federating controller's testbed name space. The shared file space on the originating testbed may contain software or configuration matter that is used in the experiment. For transparency it should be made available to the federated experiment. A federating controller will import that file space, for example by staging it locally or by accessing it from the originating testbed.

Importing the file space demonstrates the relationship between local controller and federating controller clearly. The federating controller decides how to incorporate the shared space – remote access or staging – and does the work to mesh that decision with the local controller's features. Once the set-up is accomplished, the local controller effects the connection. Different local controller features would require different federated controller implementations.

The federating controller is also responsible for configuring the local network connectors to interconnect this sub-experiment to those allocated in other testbeds. Again, this is done by proxying connector configuration into the local controller. The routing information can contain information from other testbed name spaces – the name or address of the other connector – and there is also experiment functionality to configure.

3.2.3. Clean Up

Local controllers will implement different policies for reclaiming resources; federated controllers must mediate between these policies and the expectations of the federating controllers. A federated controller may occasionally confirm that an apparently idle resource allocated to a federated experiment remains in use, or inform a federating controller of the loss of resources due to error or intervention.

When a federated experiment is being torn down, the federated controller assists in returning experiment data to the federating controller. For example, if the shared file space has been staged and the local copy written to, those changes may need to be propagated back. DETER also implements logging facilities that must have their output returned to the experimenter. Once local data has been returned, the federating controller releases the local resources to the local controller.

3.3. Network Connectors

The network connector has the most straightforward function of the three federation components. It interacts with the inter-testbed name spaces in that its routing configuration may be based on that information, but primarily it forwards, protects, and monitors inter-testbed traffic on behalf of the federated experiment. Its configuration is carried out by the local controller based on information proxied through the federating controller.

In addition to forwarding traffic between testbeds, connectors may protect the experiment from eavesdroppers or protect the network from experiments. Experiment protection may take the form of simple encryption of traffic or of full traffic masking. Network protection can be traffic policing or more sophisticated intrusion detection mechanisms. Because the configuration of the connector is accomplished locally, the range of choices is broad, but must be expressible in the higher level interfaces.

3.4. Architecture Summary

To summarize, the components of the architecture and their interconnections are:

Federating Controller
: Discovers remote resources from federated controllers and formulates the federated experiment layout. Requests and configures resources in cooperation with the federating controllers. Interfaces with federating controllers to request available resources, to receive possible resources, to configure federated resources and to tear down experiments.

Federated Controller
: Communicates with the local controller to turn Federating Controller requests into local resource allocations and configurations. The key interfaces are defined in Section 3.4.

Network Connector

Connects the networks of federated testbeds. Configured by the local controller in response to commands from the federated controller.

4. Conclusions

This document has presented a basic draft architecture on which to hang implementation strategies. While we believe the architecture to be basically sound, the primary purpose is to put the ideas out in the open and to get consensus on a vocabulary and a framework to make forward progress. Specifics of the architecture need to be hammered out and some refinement of the parts presented is inevitable. We argue that defining interfaces that represent consensus between the various testbeds that intend to federate and a commitment to keep those interfaces stable is central to a successful federation effort.

The straw man architecture here provides a set of components that isolate the basic attributes of resource discovery, resource allocation, local configuration, testbed interconnection, and coordinated cleanup. Though these are related, they are isolated enough from one another that forward progress is possible.

References

1. Terry Benzel, Robert Braden, Dongho Kim, Clifford neuman, Anthony Joseph, Keith Sklower, Ron Ostrenga, and Stephen Schwab, “Experience with DETER: A Testbed for Security Research,” *Tridentcom*, Barcelona, Spain (March 2006). available as ISI-TR-2006-613 from http://www.isi.edu/deter/documents.html.
2. Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar, “An Integrated Experimental Environment for Distributed Systems and Networks,” *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, pp. 255-270, USENIX, Boston, MA (December, 2002).
3. University of Southern California/Information Sciences Institute, “The Deter Project,” *http://www.isi.edu/deter* (2005).
4. University of Utah, “The Emulab Project,” *http://www.emulab.net* (2002).
5. University of Wisconsin, “The Wisconsin Advanced Internet Laboratory,” *http://wail.cs.wisc.edu* (2007).
6. *ITU-T Rec. X.667: Information Technology - Open Systems Interconnection - Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier components*, International Telecommunications Union (September 2004).