Current Developments in DETER Cybersecurity Testbed Technology

Terry Benzel[1*], Bob Braden[*], Ted Faber[*], Jelena Mirkovic[*],
Steve Schwab[†], Karen Sollins[§], and John Wroclawski[*]
[*]*USC Information Sciences Institute,* [§]*MIT CSAIL, and* [†]*Sparta, Inc.*
{tbenzel, braden, faber, mirkovic, jtw}@isi.edu, sollins@csail.mit.edu, schwab@sparta.com

Abstract

From its inception in 2004, the DETER testbed facility has provided effective, dedicated experimental resources and expertise to a broad range of academic, industrial and government researchers. Now, building on knowledge gained, the DETER developers and community are moving beyond the classic "testbed" model and towards the creation and deployment of fundamentally transformational cybersecurity research methodologies. This paper discusses underlying rationale, together with initial design and implementation, of key technical concepts that drive these transformations.

1. Introduction

The DETER cybersecurity testbed [3] is a dedicated network testbed facility customized for cybersecurity research. Initally deployed in 2004, DETER's first accomplishment was to provide effective, professionally managed research infrastructure and a shared user community for leading academic and industrial cybersecurity researchers. With this first objective largely met, and building on knowledge gained, the DETER team now aims to identify, understand, and enable new, fundamentally transformative, rigorous and scientific methodologies for cybersecurity research and development. This paper describes key technical contributions of our current and ongoing work in support of these goals.

Effective cybersecurity experiments are challenging to today's network testbeds for a number of reasons. Among these are

- Scale. Experiments that involve complicated composite behaviors, rare event detection or emergent effects may need to be quite large and complex to be accurate or indicative.
- Multi-party nature. Most interesting cybersecurity experiments involve more than one logical or physical party, due to the adversarial nature of the problem as well as the distributed, decentralized nature of the networked systems environment.
- Risk. Cybersecurity experiments by their fundamental nature may involve significant risk if not properly contained and controlled. At the same time, these experiments may well require some degree of interaction with the larger world to be useful.

Meeting these challenges requires both transformational advance in *capability* and transformational advance in *usability* of the underlying research infrastructure. A truly large experiment cannot be carried out unless the researcher has access to a truly large facility, but is also unlikely to be successful if the researcher has to create a twenty thousand node experiment description by hand. A potentially risky experiment is likely to take place only if the experimental platform can control the risk and all concerned can be sure it is doing so. A multi-party experiment will best be supported if the experiment control framework explicitly accommodates multiple parties and their concerns.

This paper describes a suite of significant advances to today's state of the testbed art, which taken together move the concept of "testbed" from simple hardware infrastructure to powerful and effective user-oriented research support facility. Central to the paper is our work in three synergistic areas:

First, our unique model for *risky experiment management* enables researchers to carry out

[1] Authors' names in alphabetical order.

This material is based upon work supported by the Department of Homeland Security and the Space and Naval Warfare Systems Center, San Diego, under contract No. N66001-07-C-2001, and by the National Science Foundation under Grant No. CNS-0751027. 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, the Space and Naval Warfare Systems Center, San Diego, or the National Science Foundation.

experiments that interact with their larger environment while retaining both control and safety. The key benefit of this model is that both experimenter and testbed operator can proceed with assurance in carrying out a wide and interesting range of heretofore unsupportable experiments.

Second, our model and structure for *experiment health monitoring* ensures that the underlying conditions and invariants required for an experiment to be valid do in fact hold. The key benefit of this structure is that experiments receive *active* support from the facility to ensure that they are proceeding as the designer intended.

Third, our model and mechanism for *dynamic federation* allows different testbed facilities to come together on demand to support large-scale, complex, heterogeneous, multi-party experiments. The key benefit of this work is that experimenters ranging from lone individuals to large institutions can bring together rich coalitions of otherwise unavailable resources and unconnected communities, all within our larger context of complex, yet safely and reliably executed, risky experiments.

When considered together, these capabilities allow the cybersecurity researcher to carry out experiments that are, simultaneously, more *useful,* more *reliable*, and more *complex, scalable,* and *realistic* than anything possible in today's testbed environment.

To integrate these capabilities into a coherent and easily accessible facility, we develop new abstractions and functions for our experiment control toolkit, known as SEER [1]. With these new capabilities in place SEER will allow users of widely varying sophistication to easily and effectively make use of the next-generation experimental facilities we describe, for purposes ranging from structured undergraduate education to research experiments with order-of-magnitude increases in complexity over those possible today.

Our work is embodied in extensions to the DETER testbed, a large Emulab[2]-derived facility sited at ISI and UC Berkeley and targeted to support cyber-security research. DETER's existing, successful capabilities and community, together with our long involvement in its design, operation, and ongoing development, provide crucial starting points and create unique insights for the work described here.

2. Risky Experiment Management

Experimental cybersecurity research is often *inherently* risky. An experiment may involve releasing live malware code, operating a real botnet, or creating other highly disruptive network conditions. Realistic replication of such attacks is necessary to thoroughly test detection and defense mechanisms.

The common response to this risk is to implement strict isolation capabilities within a testbed, in an attempt to ensure that no actual damage will be caused by an experiment. Depending on the testbed, containment mechanisms may range from complete disconnection from the outside world to allowing narrowly-controlled console access, and include disk scrubbing before and after each experiment.

But such containment itself is highly limiting. A fully contained experiment is hard to observe, hard to establish, and hard to control, because it must be completely isolated from its environment. Similarly, full containment is hard to create with any assurance. Sneak paths, equipment failures, and design mistakes can render containment ineffective in myriad unexpected ways.

Most importantly, full containment is not very *useful.* An interesting and powerful class of experiments are those that *can* interact with the larger environment (i.e., touch the Internet), but only in carefully controlled and well understood ways. Thus, our work aims to move from risky experiment *containment* to risky experiment *management* as a strategy.

2.1 Risky Experiment Management: Concepts

Our approach to risky experiment management is based on a very simple line of reasoning:

- If the behavior of an experiment is completely unconstrained, the behavior of the host testbed must be completely constraining, because it can assume nothing about the experiment.
- But, if the behavior of the experiment is constrained in some particular and well-chosen way or ways, the behavior of the testbed can be less constraining, because the *combination* of experiment and testbed constraints together can provide the required overall assurance of safe behavior.

We call constraints on experiment behavior *"T1 constraints",* while the corresponding constraints on testbed behavior are called *"T2 constraints".*

The *composition* of T1 and T2 constraints determines the overall safety goal for the testbed. This testbed safety goal is a fixed property, defined by the operational and administrative parameters of a particular testbed. While the safety goals of each testbed will have a common flavor of “no harm to other users, the testbed or the Internet,” each testbed may define notions of harm differently. Unacceptable behaviors may depend on the testbed’s mission and the policies at the hosting institutions. Testbed goals must thus be explicitly defined in detail.

Three points should be noted. First, the separate expression of T1 and T2 constraints in this model represents a separation of concerns. This separation allows experiment and testbed constraints to be framed and expressed independently and in terms directly meaningful to their audiences, experimenters and testbed designers, respectively. Explicit experiment (T1) constraints will allow an experimenter to reason about which constraints are acceptable without affecting the validity of the experiment. Similarly, a testbed designer can reason about how to offer different T2 constraint environments as well-known, robust, and documented services, rather than having to separately determine an operating procedure for each new experiment.

Second, we note that the semantics of T1 and T2 constraint composition to obtain a desired overall safety goal is interesting and rich. As an oversimplified thought example, one might imagine a worm that can only propagate by first contacting a “propagation service” (T1 constraint), composed with a testbed firewall (T2 constraint) that allows access to this service only from within the testbed. The result is to limit the worm’s propagation to the defined bounds of the experiment.

Finally, T1 constraints might be enforced by (1) explicit modification of malware to constrain its behavior, (2) implicit constraints using encapsulation, or (3) simply asserting a constrained behavior that the network can verify.

Constraints are associated with each experiment within a project and are continuously active. Because a given experimental topology can be used for multiple different “runs” [5], ideally constraints would be generated and applied for each run. This however would increase the burden for users whose runs exhibit the same risky behavior, and it is also challenging because we lack means to detect different “runs” from within the testbed. Instead, we can associate constraints with experiments, but will provide mechanisms for users to modify these constraints while the experiment is active.

The T1/T2 concept is only useful if we can ensure that the selected T1 and T2 constraints are met. We must require that all experiment constraints either be shown to be “correct by construction” or be auditable by the testbed. Our approach to auditing is to use the Experiment Health Management infrastructure described in Section 3 to implement monitoring tools that verify each constraint throughout the experiment’s lifetime. When experiment constraints are violated, the Experiment Health Management infrastructure will take corrective actions that may range from emailing the user and testbed operators to terminating the experiment.

2.2 Framework

Our current objectives are to 1) develop a set of T1/T2 constraint sets targeting both practical usefulness to the community and advancement of our understanding of the T1/T2 risk management model; 2) develop and deploy necessary mechanisms and tools to implement this risky experiment management model in the DETER facility; and 3) evaluate the success of our work through interactions with current DETER users and others in the research and education community.

To accomplish these objectives we are developing a risky experiment management *framework*. Our framework addresses the following top-level concerns: the experimenter’s research goals, testbed safety goals, and experimenter privacy goals. We identify useful points in each of these spaces, and develop candidate sets of experiment (T1) and testbed (T2) constraints that together provide these useful behaviors. Presently, we have deployed implementations of a small number of selected constraint sets to our early-adopter users.

Current testbeds assume one liberal set of user privacy goals, but this is not realistic since, for example, commercial users may have very different privacy expectations than academic users. Some monitoring of user actions and traffic by the testbed will be necessary for several purposes: (1) to support health management described in Section 3, (2) to ensure that user constraints are implemented correctly, and (3) to reduce testbed liability in case of malicious incidents.

To successfully support risky but controlled experiments, our framework must capture user needs. We have developed an initial experiment categorization taxonomy based on the type of risky behaviors necessary to an experiment, such as self-propagating malware, high-volume traffic, etc. Our initial taxonomy defines a small list of categories, based on current experimental uses of DETER. We expect this list to grow as work progresses, based on user input.

Users are presently required to specify experiment categorization, privacy goals and appropriate experiment constraints at the time of experiment creation. Testbed constraints are generated based both on the specification input by the user, and the testbed safety requirements defined once by the testbed operators. These constraints are put into action by the testbed and our health management infrastructure ensures that they are continuously enforced.

2.3 Implementation

Our ultimate goal is to develop a fine-grain model for T1 and T2 constraints and a formal structure to reason about their composition. In current work we adopt a simpler approach, as a first practical step towards deploying a useful risky experiment management capability and as an assessment of the value of the concept. We are developing a small selection of matching T1/T2 constraint *sets*, where a) the T1 constraints are chosen to be appropriate and useful for a particular class of experiment; b) the T2 constraints are chosen to be implementable and verifiable in a particular testbed environment, and c) the composition of the chosen T1 and T2 constraints produces an acceptable risk management result. We develop a mechanism to allow researcher and testbed operator to agree on particular sets, implement and enforce both experiment and testbed constraints, and thus obtain the level of risk management required for the particular testbed environment.

We are developing a domain-specific language language called *REALM* for specification and manipulation of T1/T2 constraints as well as operational safety objectives. Although it will be possible for users to write REALM specifications directly, our intent is that REALM be the output and interchange language that a variety of tools use to capture and manipulate constraint information. REALM will be integrated with the SEER toolkit described in Section 5, including guided dialogs for users. The current version is also integrated with the DETER facility's "Create an Experiment" Web page. User input is recorded through these interfaces and translated into REALM specifications associated with the experiment.

We regard each experiment as potentially risky until proven otherwise. We initially address three types of potentially risky behavior: (1) running malware, (2) creating disruptive behavior, and (3) requiring connectivity with the outside Internet. The first two behaviors are intentionally risky, whereas the third behavior may be risky by accident, if experimental traffic with the outside is misconfigured and overloads resources, provokes an external attack on the testbed or creates liability.

To meet the needs of DETER's current research community, we initially address the following risks from experiments: (1) malware traffic may infect testbed hardware infrastructure needed for correct operation, (2) experimental traffic of any sort may overload control plane and shared hardware, (3) disruptive actions may affect control plane and shared hardware (4) in experiments with the outside connectivity, experimental traffic sent to remote machines may infect, overload or disrupt these machines and remote networks, (5) in experiments with the outside connectivity, experimental traffic may provoke retribution toward the testbed (e.g., from the Storm Network [44]) or create liability problems to the testbed, (6) malware may stay resident on machines after they are reclaimed by the testbed and may affect future experiments by other users. Our framework is evolvable so new threats will be incorporated as research objectives dictate; we expect this list of risks to grow as we proceed with our work.

The above risks are contained via experiment and testbed constraints. Our initial list of *experiment (T1) constraints* includes: (1) users limit scanning behavior of self-propagating malware, (2) users limit targets of disruptive actions, such as denial-of-service, to addresses within experimental network, (3) users limit their malware choice to well-known malware contained in the DETER-supplied library, (4) users limit experimental connectivity with the outside world to a set of machines under their control, and to specific protocols, (5) users limit the rate of traffic in their experiments, (6) users limit experimental traffic to the experimental network, (7) users implement signatures or self-terminating behavior in malware they plan to use. Our current prototype of risky experiment management supports definition of constraints 1-6.

Our initial list of *testbed (T2) constraints* includes: (1) isolation of experiments on the control plane using a separate virtual LAN for each experiment, (2) experimental traffic filtering and rate-limiting on the control plane using hardware-specific filters at switches to prevent disruption and overload of shared infrastructure, (3) allowing outside connectivity only via specialized nodes ("tunnel nodes") that connect the experimental network to the Internet, (4) controlling experimental traffic contents and rate with the outside Internet via firewall rules and the Bro intrusion detection system [50] for deep packet inspection, both installed on all paths to the Internet, (5) recording traffic on tunnel nodes, recording of login activity on experimental nodes, and association of traffic, logged users, and experiment names for potential liability reasons. Constraints 1 and 3 are implemented in our current prototype system, with others to be added in the near future.

3. Experiment Health Management

Experiment health management addresses two broad and increasingly important needs within experimental cybersecurity research. First is the need to support order of magnitude greater *complexity* in the creation of realistic cybersecurity experiments. Second is the need to bring greater *rigor and scientific discipline* to the experimental research paradigm. We address these needs through a research facility subsystem based on two observations:

- The validity, accuracy, and usefulness of an experiment depend critically on some set of invariants or *expectations* identified by the experiment's creator being met.
- Any given experiment will have a number of other behaviors that are not invariants, and cannot be predicted by researchers, since experiments are done to study unknown effects.

The rigor and scientific validity of an experiment is greatly increased when the expectations and invariants on which its validity depends are clearly understood by the researcher *and* by others who wish to utilize or build on the results of the experiment. A system that makes these expectations explicit and ensures that they are met during an experiment will contribute greatly to the rigor of future experimental research.

This problem is complicated when experimental complexity is increased, because the maintenance of experimental invariants and expectations becomes exponentially more difficult. Managing the complexity of experiments that involve more than a handful of elements demands system support to assist researchers in understanding, documenting, and maintaining the health of their experiments – the validity of the experiment's assumptions, expectations, and invariants.

The challenge of experiment health monitoring and management is to ensure that the underlying conditions and invariants required for an experiment to be valid are being met by the facility, and to aid the researcher in detecting and modifying errors in experiment design. Here "health" refers both to the behavior desired by a researcher of his experiment *and* that desired of the underlying testbed. Reasons for reduced health include, but are not limited to, mistakes by the experimenter, failures or faults of testbed resources, misunderstandings the researcher has about the testbed, unintended interactions between simultaneous experiments, a security constraint being violated, and so forth. The initial DETER system, as with many similar testbed environments [2][7][12], provided little or no support for either determining whether an experiment is behaving as expected (its current health) or for diagnosing failures and improving the situation (improving its health). Our experiment health management system addresses this missing function.

The experiment health problem is characterized by three key properties. First, in contrast with the "network management" problem of maintaining functional behavior in an operating network, our domain is the very different problem of supporting security-related experimentation on a networking testbed. The consequence is that potential range of expected behaviors is very broad must be user-supplied, because many cybersecurity experiments require and *intentionally create* worst-case conditions of overload, resource denial, host penetration and unreliability.

Second, expectations of behavior will range from extremely low level and concrete "invariants" valuable for educational exercises (such as "node A is up") to composite, complex, perhaps statistical, and much more abstract expectations (such as "service has been denied"). An ideal experiment health system will handle this wide range of invariants.

Finally, usability is critical. It must be possible for users to capture desired invariants and health enforcement actions with minimal overhead and maximum clarity if the system is to meet its objectives.

Our goal is to support, with these properties, experiment health maintenance in testbeds such as DETER, and, by extension, to the federation of such testbeds as described in Section 4. Our experiment health system includes five elements, each with its own research and implementation challenges. We outline the elements here and expand on each in the following sections. The system includes functions to support:

- Expectation capture. Concerns are the sources and expression of data about experiment expectations.
- Data collection and monitoring. Concerns are kinds and sources of data, reasons that the information may be incomplete, and how to provide controlled sharing between these tools and health evaluators .
- Health evaluators. This includes observation of collected data and its comparison against an expectation to evaluate if the expectation is being met.

- Enforcement or repair. In its most basic form, enforcement may simply involve repair or replacement, but because expectations may be quite abstract, there may be several diagnostic steps involved in the process, and a selection of repair or enforcement options.
- Support for sharing of information and expectation data. This problem has two aspects: an information plane to manage the availability and sharing of operating information between experiments and the underlying testbed, and a library for cataloguing and accessing expectation templates, resource definitions, data collection tools, current performance tools and health evaluators.

3.1. Expectations

In operating networks, network management has the goal of maintaining connectivity, distributing load, setting up network configuration, and in general supporting the network mission of delivering traffic effectively between end nodes. This common mission is well understood and agreed upon by all participants. In shared testbeds, management needs to achieve a more complex definition of a desired behavior. For a particular user, desired behavior may be to deny service or disrupt connectivity in the experimental network, to test a new worm or to maintain some long-lived service running reliably.

Users also have goals related to research privacy and the usability they expect from the testbed, and these goals differ from person to person. From the testbed operator's perspective, the desired behavior may be to provide reliable service to users, to control risky experiments, to federate with remote testbeds and to protect the privacy of its users.

Because desired behaviors differ widely across different experiments and depend on the nature of experimentation and testbed maintenance, it is impossible to identify universally appropriate behaviors. Instead we require an explicit expression of individual experiment *expectations*. The two key issues we address are sources of expectations and the language for codifying them.

Expectations may be identified in a variety of ways. First, they could come directly from the experimenter. This could be explicit, or inferred from the researcher's behavior ("if he keeps fixing the DNS system when it breaks, it must mean the DNS system should be working"). Ideally a system could simply learn invariants by looking at a working experiment, but the problem lies in recognizing the non-invariants - things that are unimportant or *should* change from experiment to experiment.

Our initial implementation requires explicit expression of expectations by experimenters and testbed system managers, with minor automation from the testbed. We include design hooks for the system to use additional expectation capture methods in the future. We implement an *expectation capture language* that specifies a set of conditions under which the expectation will be evaluated, a subject for the evaluation, a health evaluator, and a set of responses to the evaluation. Together these items capture an expectation and its enforcement and response methods.

A simple example is to expect a server to be running, verify this by sending a ping once a minute and reboot the server if the ping fails. In a more sophisticated example, the expectation may be of a certain level of traffic among a set of gossiping nodes. The traffic level might be checked every minute and if it is below a threshold, each node might be told to increment by one the number of nodes it contacts during a gossiping episode. A security expectation might lead to allowing a traffic flow from the Internet into an experiment (response), if a particular experiment is running, no other experiments are running, and the traffic is all addressed to a particular port (conditions).

We identify a long list of requirements for expressiveness in invariant capture, including but not limited to: time, location, service quality evaluation; verification of particular actions, continuous states, privacy; higher-level concepts such as restriction on code propagation; dependencies among expectations, coupling of expectations to actions; and composition into higher level expectations. To create the capture language, we build on past work such as Ponder [30] and Tcl expect [28], with a domain-specific user-friendly and higher-level syntax, for easier use.

We briefly discuss key issues with respect to the expectation subjects – things about which an expectation may be expressed. These subjects range from simple base level instances, such as a link or node, to more complex elements such as an entire Gnutella-like P2P system. We differentiate between the type of the subject (e.g., link), and the instantiation of it (e.g., link between A and B). The choice of health evaluation tools and repair functions then depends both on the type and the instantiation of a given expectation, and may lead to decomposition of that expectation evaluation into more primitive evaluations. To capture these nuances we provide parameterized templates for many common expectations in our library, for users to instantiate.

3.2. Data collection tools

To implement health evaluation the system provides monitoring and data collection about what is happening in the testbed as a whole as well as in each particular experiment. There are three key sources of such data: (1) static data such as node allocation to a particular experiment, etc. (2) data collected routinely in all experiments and by the facility itself, such as packet tracing, node liveness, etc., and (3) explicit data collection requested by an experimenter or the testbed management, in context of a specific expectation's evaluation.

There are several challenges to data collection. First, because of scale and system unreliability, available data may be incomplete. Second, the desired information may not be directly measurable, but must be inferred from other measurements that can be gathered directly. Finally, in light of security expectations that relate to privacy, some information may not be accessible to a particular experimenter or portion of an experiment. Either an experiment or the testbed system may withhold information from the other. As an initial step towards meeting these challenges, we provide a base set of data collection tools in our library, which will be extensible by researchers.

3.3. Health evaluation

The job of health evaluation is to determine whether an expectation – in our framing, the static or dynamic behavior of a expectation's subject – meets specified health criteria. There are two aspects to evaluating the health of a subject. The first is to select the particular behaviors of the subject, such as link bandwidth, jitter or loss rate, that are to be evaluated. This will in turn determine one or more tools for evaluating the behavior. The second is to determine the health of that subject by comparing observed and expected behavior. As a simple example, the experiment health may require either a high or a low loss rate, depending on the user's desires.

In the case of a more complex subject, with a rich set of possible behaviors and the potential for a complex user definition of health, we break the problem down with a composite evaluation. One approach is to define the more complex behavior as a composition of a set of simpler behaviors. Then when asked to evaluate the health of the subject, the target behavior is computed as a composition of those simpler behaviors and the result is evaluated for its health. In this approach the composite behavior is completely synthesized and then a single health evaluation is performed.

In an alternate approach, we define the health evaluation of a complex subject as the composition of the health evaluations of simpler components. In this case, the system evaluates the behavior and health of each simpler component and *then* composes the results into a single health evaluation. Because each approach is preferable in different circumstances, we define both approaches and allow the user to reason about tradeoffs.

3.4. Enforcement and repair

Enforcement and repair are two sides of the same coin. *Enforcement* provides some level of guarantee that an expectation continues to be met. Thus, for example, in order to enforce that any reproducing malware does not overload resources, the testbed could rate limit the traffic from experimental nodes. Enforcement is likely to require frequent periodic evaluation of expectations. In contrast, *repair* uses similar mechanisms but aims to correct a detected failure. Since failures are not very frequent, evaluation of expectations that involve repair actions may occur on demand or periodically but infrequently.

A third alternative is to detect an unhealthy situation, but take no action to address it other than notifying the user. This may be necessary in cases when there is no specified repair action, or the repair itself has failed. For example if an experiment expects 50 nodes, is assigned the only 50 nodes available and one fails, the only option is to halt the experiment. On the other hand, if some nodes were optional and some critical, the experiment might continue as long as a critical node did not fail.

We allow the user to specify each of these cases, and the desired enforcement or repair action, using our expectation language. We provide tools for common enforcement and repair operations in our library for easy use. Additional language constructs support sophisticated users that wish to provide enforcement or repair actions that are specialized to the nature of their experiment.

3.5. Sharing

To monitor and enforce expectations, a health system must depend on significant amounts of information about experiment performance. Because DETER and similar testbeds utilize reusable and shared resources, this information must be collected and accessible from several contexts simultaneously. Conversely, the same information may be valuable to more than one monitoring tool either simultaneously or at different times. Each of these situations leads to information sharing.

For example, it may be important to collect packet traces in an experiment for a variety of different uses. Monitoring tools used on behalf of the experimenter may depend on these traces to verify correct experiment behavior, while the underlying testbed system may simultaneously use such traffic information to determine the health of the complete set of resources it is managing. At the same time, these traffic traces can provide an audit trail if an experiment creates a security risk for the testbed, e.g., by running a worm that escapes into the Internet. Other contexts for reuse of the same information are possible.

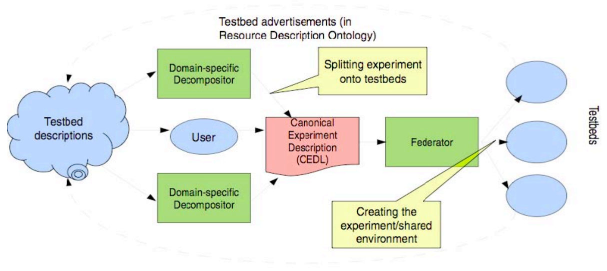
Thus information should be collected once and managed effectively to allow for multiple uses. This requires a common information substrate or *information plane*, with well-defined access rules and contexts. Information collection and access must be designed to reflect the security and privacy expectations that are critical for the whole experiment health management. In this area our development effort draws on and is synergistic with other efforts with this specific focus, particularly the Knowledge Plane activity described in [18].

4. Dynamic Federation

Federation is the task of creating, on demand, a multi-testbed structure to support a single large experiment. The goal of federation is to subdivide and embed a single experiment across multiple testbeds, in a way that meets the objectives, requirements and constraints of both the researcher and the testbed operators. Reasons to federate experiments include scale and realism, access to heterogeneous testbed capabilities, integration of multiple research communities, and information hiding. Of particular interest for large cybersecurity experiments is simultaneously creating federated environments while addressing risky experiment management and health management goals based on the mechanisms of Sections 2 and 3.

The *DETER federation architecture* (DFA) implements federation over Emulab-style testbeds. The architecture breaks the federation task into three steps: 1) decomposing the experiment to be federated into *sub-experiments* to be assigned to individual testbeds; 2) embedding the sub-experiments into individual testbeds and building the necessary connections between testbeds, and 3) operating and supporting the federated experiment.

The architecture recognizes that within the three tasks some functions are dependent on the requirements and characteristics of the particular experiment to be federated, and thus require domain-specific knowledge. Other functions are common across experiments, and can be modularized and generalized. The DFA accommodates this by including 1) elements and interfaces to support common functions; 2) system interfaces and framework for a "plug-in" extensible implementation of domain-specific functions; and 3) an ontology and language to express information used to drive the federation process. Figure 1 gives an overview of the architecture. Key elements are described in the following text.


Testbed advertisements (in Resource Description Ontology)
Domain-specific Decompositor
Splitting experiment onto testbeds
Testbed descriptions
User
Canonical Experiment Description (CEDL)
Federator
Testbeds
Domain-specific Decompositor
Creating the experiment/shared environment

4.1. Sub-Experiment Decomposition

Intelligent decomposition of a single large experiment into per-testbed sub-experiments must of necessity consider two factors: heterogeneity *within the experiment* in one or more dimensions, and *testbed capabilities* along one or more axes. Examples of experiment heterogeneity that may influence decomposition include

- Topology and bandwidth requirements – e.g. embedding densely connected regions of the experiment within a single testbed.
- Specific hardware or software requirements within some portion of the experiment graph.
- Security constraints – in our model, specific T1 constraints that can be offered over some portion of an experiment, and/or specific T2 constraints required from the testbed hosting some portion of the experiment.
- Information hiding – particularly in a composite experiment, such as a red-team/blue-team scenario.

Because the factors that should influence the decomposition of a particular experiment are known only to the experimenter, decomposition is a domain specific function. Thus, the DFA provides for an extensible set of decompositors, together with a well-defined environment for their implementation. DFA provides common information to decompositors using the ontology described in Section 4.3. Inputs or knowledge specific to a single decompositor may be provided by a custom interface, or simply programmed into the decompositor's implementation. As a particularly simple case, the decomposition function can be performed without involvement of higher level software by a human writing directly in the CEDL language described below.

4.2. Canonical Experiment Description Language and Federator

Canonical experiment description language (CEDL) is the output language for all decompositors in the DFA. It can be thought of as an "assembly language" for federated experiments – a common, low-level format that many different tools can generate. CEDL is an extension of Emulab's current NS2-based experiment description language. CEDL describes an experiment as an interconnected topology of nodes, together with a number of *annotations* that guide the embedding of an experiment into its federation of testbeds. Annotations include such information as the target testbed for placing a particular node, and whether a node is critical to the experiment or can be ignored if it cannot be embedded successfully.

An experiment's CEDL description forms the input to the *federator*. The federator is responsible for embedding the sub-experiments within each federated testbed, after creating the additional hidden nodes and links necessary to interconnect regions of the experiment. This task is essentially mechanical, but requires parsing and understanding the CEDL description sufficiently to forward experiment information, security configuration, and user credentials to testbeds, establish the shared experiment support environment of Section 4.4, handle error conditions that may arise, and similar functions.

4.3. Federation Resource Description Ontology

Elements of the DETER federation architecture are bound together by an ontology of information needed to carry out the federation task. This ontology, and its expression in a concrete format, allow the different principals to communicate requirements, needs, and constraints to each other. Statements and requests expressed in this ontology are communicated between testbeds, the DFA federator, and decompositors acting on behalf of potential users, to implement decomposition and embedding functions. We briefly outline the structure and scope of the proposed ontology.

The form of expression is attribute/value assertions potentially attested to by principals or outsiders: (*X* asserts that *Y* is/has *Z*). Some example attributes, values, and meanings are shown in the table.

Attribute	*Value*	*Meaning*
User: PGP_Key	(keyid, server)	User PGP ID
Testbed: Access_Policy	X.509, Kerberos	Acceptable user authentications
Sub_Experiment: T1_Imposed	String	T1 constraint set exp. meets

Multiple attribute assertions can be assembled into descriptions or requests. The operators are grouping, conjunction, and inclusive and exclusive disjunction.

(DETER asserts that its access policy is X.509 certificate AND

(DETER asserts that it has 100 nodes AND DETER asserts that it has 1Gb/s cross-connect) XOR

(DETER asserts that it has 1000 nodes AND DETER asserts that it has 1000Mb/s cross-connect))

The ontology's domain of discourse is testbeds, experimenters, resources, sub-experiments, and attesters. Examples of representable concepts within the ontology are given below, for flavor. Of particular interest is the ontology's ability and requirement to represent our T1/T2 risky experiment management constraints. It follows that the ontology will evolve in this respect as our work in that area proceeds.

The simple semantic model of this ontology is intended to allow the use of a variety of existing representation and constraint matching tools to facilitate reasoning about the federation problem. Our research lies in what must be said and how to interpret it, not in the form of the ontology.

4.4. Scaling the Experiment Support Environment

Centralized testbeds such as Emulab have historically provided a rich *experiment support environment* of functions intended to simplify the job of the researcher by providing useful building-block capabilities. Emulab's experiment support environment includes a shared filesystem, an event system, a simple error management system, and several related functions. When moving to a decentralized, federated environment, the scalability and appropriateness of these functions must be revisited. Several outcomes are possible.

The function may be fully scalable, perhaps with a new implementation, as it is commonly used. The function may be scalable *as defined*, but not *as commonly used*, leading to the need for careful re-thinking. Or, the function may not be scalable at all, leading to its removal from the federated environment, and possible replacement with a more scalable alternative capability.

Different Emulab support functions exhibit each of these properties. As examples,

- The use of a shared filesystem is reasonably scalable in concept, if not always in implementation. A number of researchers have proposed approaches to highly scalable distributed filesystems that could be adopted.
- The Emulab event system is scalable *in concept*, because it does not define either tight real-time semantics or assured ordering semantics. However, *in practice* it exhibits both of these properties in a local testbed, and they have come to be relied on by some experimenters. In a federated environment, two separate mechanisms may be appropriate: one that provides tight real-time constraints on single event delivery, to coordinate the actions of different elements across a highly scaled experiment, and one that provides explicit ordering semantics [42][43] to support complex dependency graphs in highly structured experiments.

In general, the design of an experiment support environment for large federated facilities must carefully and explicitly consider tradeoffs between scalability, robustness, implementability, and usefulness to the experimenter. In our implementation of DFA within DETER, each of the existing Emulab services is analyzed for suitability to the federated environment, and updated or replaced if necessary.

5. SEER as Integration Platform and Usability Framework

The DETER testbed's SEER system [1] is an extensible framework for experiment support and control. We rely on SEER to provide broad infrastructural support for our new capabilities of experiment health maintenance, risky experiment management, and federation.[2] It is important to recognize that without significant advances in this regard, the increased experiment complexity enabled by our other advances could potentially *reduce* testbed usability. For this reason, our goal is to leapfrog existing usability models, creating new experiment creation and control interfaces that directly address this increased complexity. Further, it is useful to address separately the needs of sophisticated experimenters, who frequently must operate at low level and shape their own tools, and for more casual users, who need simple access to complex function.

SEER is structured as a GUI that communicates with a master *controller agent* (CA) for each experiment. In turn, the controller agent communicates with and controls a SEER agent running on each experimental node. The GUI and CA provide the experimenter with access to SEER capabilities through an XML-RPC interface, which allows for interaction with the controller by other programs. For example, even now an experimenter can interact with the controller through a command-line interface. The controller agent contains logic in the form of execution scripts to support potential experimenter requests, as well as the event state necessary to monitor and manage those requests. The controller agent communicates directly with the local *node agents* with requests or commands for local operations and information. This structure provides a framework to support the major functional developments described in this paper. We briefly consider each of these in the context of the SEER agent system.

As described in Section 4, federation of testbeds in support of large experiments is achieved by decomposition of the experiment into partitions, each of which is run on one of the federants under its local control. We reflect that same decomposition in the SEER experiment control, by splitting and distributed the responsibilities of the controller. Under federation, we extend SEER to provide a single *experiment controller*, a set of *federant controllers*, and the requisite *node agent* for each node. In this case, the experiment controller provides high-level oversight and partitioning of the SEER activities, and is capable of partitioning inferior responsibilities to the federant controllers, which in turn interact with the local *node agents.* The experiment controller operates using an experiment-wide event stream, while the individual federant controllers each have their own, partitioned, event state and sequence of events. Notice that this is an example of a situation in which the invoker of an

[2] We note that it is not *necessary* to use SEER to access these capabilities; each is also accessible through new low-level system APIs. It is both possible and expected that additional higher level integration tools will be developed in the future.

XML-RPC on a (federant) controller will be another controller, not the GUI – a new capability that is naturally supported by the modular SEER architecture. With further extensions expected in the longer run, we plan for other situations in which controllers will be invoked by something other than the GUI.

In the case of risky experiment management the challenge is slightly different, but is equally accommodated with an extension to the underlying agent system. The problem for risky experiments is that the interaction with the outside Internet is not based in a "node" of the experiment in the same sense as the traffic and actions of a completely internal experiment. Therefore, in order to manage and monitor that interaction, a *T2 agent* is used to manage the testbed's constraint mechanisms. In terms of overall control of the experiment, this agent is a peer of the node agents. Concretely, this agent is hosted on the same node as the federant's controller. The job of this T2 agent is qualitatively different than that of the node agents because it may have less control over what actually happens, but may require more control over the flow or distribution of information from its "target" to the other components of the experiment. By isolating that control in a separate agent, we increase our ability to make it trustworthy, independently of the other components of the SEER environment.

Our experiment health management architecture maps directly onto the appropriate SEER agent infrastructure for each experiment. By piggy-backing experiment health management monitoring agents onto the existing SEER agent structure, we guarantee that the appropriate data collection, behavior evaluation, health determination, and responses will be managed along exactly the same paths of control as those of the experiment itself. This also allows for policy controls, such as those that may be necessary in the information plane component of the health management system, to follow the structure cleanly, giving each node agent, federant controller, or T2 agent local control over local access. Finally, we extend the SEER GUI itself to provide access to these new capabilities in intuitive ways.

6. Related Work

Our experiment health work is an extension and customization of knowledge plane [18] ideas to network testbeds. Much recent work in this area has concentrated on the sub-problem of supporting an *information plane*. Two approaches are seen. With *Sophia* [34] and the work of Loo et al. [35][36][37], the objective is to provide an all-purpose information plane, in which all information is shared. From our perspective these do not provide the ability to control or limit access to information based on security and privacy policies. The second is more specialized information planes, including *iPlane* [38], which provides path behaviors for managing peer-to-peer overlays, the *Lord of the Links* project [39] which provides comprehensive network topology information, and the 4D proposal [40] for route computation and distribution. In contrast we propose a general-purpose information plane for sharing, but one that permits control or limitation of access to information for policy (proprietary or security) reasons.

There is vast work in network management; a field that is related to our health management effort. The main distinction between network management and our work is that testbed experiments lack generally agreed-upon correctness or performance goals. Namely, what may be regarded as poor performance in one experiment, such as frequent link failures, may be a desired effect in another experiment. Another difference lies in the granularity at which management is done: networks are managed at high granularity of network elements and links, while experiments also need to be managed at low granularity of user actions. Thus, we expect to reuse existing work in network management for coarse-granularity management of the testbed and the experiments, but we extend this field with our fine-granularity management functionalities.

Network monitoring has received significant attention with the advent of grid and cloud computing (to mention just a few publications [14][15][16][31]). We plan to reuse existing monitoring approaches and tools for our health management. The novelty of our work lies in interpretation of the outputs of those tools, and in orchestration of their activity.

Ballani and Francis propose Complexity Oblivious Network Management [17], in which the management interface abstracts much of the underlying implementation complexity, facilitating more effective high-level management. We expect to leverage and extend this work to simplify our management tasks.

There are a number of tools for distributed application management on PlanetLab [12], such as Plush and Nebula [4], PlMan [19], Stork [20], pShell [21], Planetlab Application Manager [22], parallel open SSH tools [23], plDist [24], Nixes [25], PLDeploy [26] and vxargs [27]. With the exception of Plush and Nebula, these tools are all low-level monitoring or management tools that are engaged manually at the setup time of a PlanetLab experiment. They enable parallel execution of multiple tasks, or monitor nodes for liveness, connectivity, and state, and present summarized information to a user. Plush [4] is a toolkit for distributed application configuration, management and visualization. Plush enables users to specify tasks in XML format, then executes them invoking low-level process, file and resource monitoring to detect failures. Plush also provides synchronization primitives and performs resource acquisition and reallocation as needed. The primary distinction between Plush and our proposed health monitoring is that Plush manages for known performance goals (connectivity, process liveness, etc.) that are suitable for continuously running applications, while we additionally manage for customizable performance goals that are suitable for widely varying testbed experiments. Our management thus includes the notion of "expected performance" and covers a wider range of behaviors than Plush.

Emulab's Experimenters Workbench [5] contains support for experiment versioning, cloning via a template, and archiving. These capabilities support pre-packaged experiments, and are complementary to the capabilities provided by SEER. Emulab's workbench does not, however, provide any support for experiment creation and correctness checking, which is the main focus of our health management infrastructure.

In the area of expectation or policy specification languages, we mention two extremes. XACML [29] is declarative and serves as a policy capture framework, expecting enforcement through other means. From our perspective, policy declarations are only a small part of our challenge. In contrast, Ponder [30] [41] is an object based language for declaring not only security and management policies, but time, state, and composite conditions under which the policies should be evaluated, sets of subjects to be evaluated, sets of targets over which some action might be taken, and the actions themselves. All of these can be individuals, composites, or abstractions. In fact, Ponder policies are also objects and can themselves be the subjects of policies. Simpler than Ponder, Tcl Expect [28] is a scripting environment whose syntax enables specification of control flows that depend on controlled program outputs, thus automating system testing. As discussed earlier, our expectation language incorporates such capabilities, with the specific objective of making expectation capture easily accessible, usable, and understandable by a broad set of differently skilled researchers. We find it useful to reuse features of Tcl Expect and Ponder, wrapping them in more user-friendly syntax and/or higher level language constructs.

Though there has been much work on federating databases or constructing meta-computers, federating testbeds is a relatively recent area of research. Emulab-to-Emulab federation remains a topic of ongoing development [51][52][53], though much of this is the foundational work of interconnecting the testbeds at the operational level. Our work extends this to enable experiments that configure the federated environment based on policy considerations such as the risk level of the experiment, and to include distributed monitoring capability. The PlanetLab research community has also begun to federate instantiations of PlanetLab [54][55]. Much of this work centers on splitting a centralized authority between a few entities; our work is more focused on federation without a central authority. The Grid community provides both tools [56] and standards [57][58][59][60] that are useful in addressing several aspects of federation. Primarily, Grid tools simplify exchange of authentication requirements and trust requirements, which are required in a practical federation system but are not central to our research. Adopting these standards and tools frees us to focus on the new aspects of our problem domain.

Honeynets [32] address a problem related to our risky experiment control. Honeynets must allow some malware interaction with the outside Internet, but control it so that the honeynet does not participate in attacks on others [33]. This resembles our goal of allowing experiments to communicate safely with the Internet. However our problem is more constrained since testbed researchers often have some knowledge of the malware they want to study and can describe some aspects of its behavior, while honeynets must support unknown malware and live attackers. Despite these distinctions we find it useful to reuse honeynet practices for our testbed constraint implementation.

7. Conclusion

From its inception in 2004, the DETER testbed facility and community have provided effective, dedicated experimental resources and expertise to a broad range of academic, industrial and government researchers. Now, building on knowledge gained, the DETER developers and community are moving beyond the classic "testbed" model and towards the creation and deployment of fundamentally transformational cybersecurity research methodologies. The risky experiment management, experiment health support, and federation technologies described in this paper simultaneously enable order of magnitude increases in both *scientific rigor* and *realism* for such research, leading to dramatic increases in researcher productivity and quality of results. Further, these DETER advances serve as an incubator for similar capabilities in projects such as GENI and the proposed National Cyber Range, catalyzing dramatic and broad improvement in the nation's cyber research capability.

8. References

[1] 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.

[2] 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”, Proceedings of the Fifth Symposium on Operating Systems Design and Implementation, USENIX, Boston, MA, 2002.

[3] T. Benzel, R. Braden, D. Kim, B. C. Neuman, A. Joseph, K. Sklower, Ron Ostrenga and S. Schwab, “Experience with DETER: A Testbed for Security Research,” In Proceedings of Tridentcom (International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities), March 2006.

[4] J. Albrecht, C. Tuttle, A. C. Snoeren, and A. Vahdat. “PlanetLab Application Management Using Plush,” ACM Operating Systems Review (SIGOPS-OSR), 40(1), January 2006.

[5] E. Eide, L. Stoller, and J. Lepreau, “An Experimentation Workbench for Replayable Networking Research”, Proceedings of NSDI 2007.

[6] A. Bavier, N. Feamster, M. Huang, L. Peterson and J. Rexford, “In VINI Veritas: Realistic and Controlled Network Experimentation,” Proceedings of ACM SIGCOMM 2006.

[7] WAIL testbed, http://www.schooner.wail.wisc.edu/

[8] TCPreplay, http://tcpreplay.synfin.net/trac/

[9] Harpoon traffic generator, http://pages.cs.wisc.edu/~jsommers/harpoon/

[10] J. Mirkovic, A. Hussain, B. Wilson, S. Fahmy, P. Reiher, R. Thomas, W. Yao, and S. Schwab, “Towards User-Centric Metrics for Denial-Of-Service Measurement,” Proceedings of the Workshop on Experimental Computer Science, June 2007

[11] TCPdump, http://www.tcpdump.org

[12] L. Peterson, A. Bavier, M. Fiuczynski, and S. Muir, “Experiences Building PlanetLab”, Proceedings of Operating Systems Design and Implementation Symposium *(OSDI ‘06),* November 2006

[13] T. Faber, J. Wroclawski, and K. Lahey, “A DETER Federation Architecture”, In Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test, August 2007.

[14] I. C .Legrand, H. B. Newman, R. Voicu, C. Cirstoiu, C. Grigoras, M. Toarta, C. Dobre “MonALISA: An Agent based, Dynamic Service System to Monitor, Control and Optimize Grid based Applications, Proceedings of CHEP 2004.

[15] A. W. Cooke et al. “The Relational Grid Monitoring Architecture: Mediating Information about the Grid” Journal of Grid Computing, vol 2 no 4 December 2004.

[16] A. J. Wilson et al. “Information and Monitoring Services within a Grid Environment,” Proceedings of CHEP 2004.

[17] H. Ballani and P. Francis, “CONMan: A Step Towards Network Manageability”, Proceedings of ACM SIGCOMM 2007.

[18] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, “A Knowledge Plane for the Internet,” in Proceedings of ACM SIGCOMM, 2003.

[19] T. Isdal, T. Anderson, A. Krishnamurthy and E. Lazowska, “PlanetLab Experiment Manager”, http://www.cs.washington.edu/research/networking/cplane/

[20] University of Arizona, “Stork”, http://www.cs.arizona.edu/stork/

[21] McGill University, “pShell”, http://www.cs.mcgill.ca/~anrl/projects/pShell/index.php

[22] Intel Research Berkeley, “PlanetLab Application Manager”, http://appmanager.berkeley.intelresearch.net/

[23] Intel Research Berkeley, “Parallel open ssh tools”, http://www.theether.org/pssh/

[24] M. Georg, University of Washington at St Louis, “plDist”, http://www.arl.wustl.edu/~mgeorg/plDist.html

[25] AquaLab, Northwestern University, “Nixes”, http://www.aqualab.cs.northwestern.edu/nixes.html

[26] Intel Oregon, “PLDeploy”, http://psepr.org/tools/

[27] Y. Mao, University of Pennsylvania, “vxargs: Running arbitrary commands with explicit parallelism, visualization and redirection”, http://dharma.cis.upenn.edu/planetlab/vxargs/

[28] D. Libes, “expect: Curing those Uncontrollable Fits of Interaction”, Proceedings of the Summer 1990 USENIX Conference, Anaheim, CA, June 11-15, 1990.

[29] OASIS, “OASIS extensible Access Control Markup Language”, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

[30] N. Damianou, N.. Dulay, E. Lupu, M. Sloman, “The Ponder policy specification language”, Proceedings of Workshop on Policies for Distributed Systems and Networks, 2001.

[31] Y. Mao, H. Jamjoom, S. Tao, J. M. Smith, “NetworkMD: Topology Inference and Failure Diagnosis in the Last Mile”, Proceedings of IMC 2007.

[32] The Honeynet Project, Web page, http://www.honeynet.org

[33] The Honeynet Project, “Know your Enemy”, 2nd edition, Addison-Wesley Profeesional, May 2004.

[34] M. Wawrzoniak, L. Peterson, and T. Roscoe, Sophia: An Information Plane for Networked Systems, ACM SIGCOMM (HOTNETS-II) Proc. 2nd Workshop on Hot Topics in Networks, 2004, 15-20.

[35] J. Kopena and B. Loo, OntoNet: Scalable Knowledge-Based Networking, 4th Workshop on Networking Meets Databases, 2008.

[36] Y. Mao, B. Loo, Z. Ives, and J. Smith, The Case for a Unified Extensible Data-Centric Mobility Infrastructure, ACM SIGCOMM International Workshop on Mobility in the Evolving Internet Architecture, 2007.

[37] W. Zhou, E. Cronin, and B. Loo, Provenance-aware Secure Networks, 4th Workshop on Networking Meets Databases, 2008.

[38] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkatramani, iPlane: An Information Plane for Distributed Services, USENIX (OSDI) Proc. Symposium on Operating Systems Design and Implementation, 2006.

[39] Y. He, G. Siganos, M. Faloutsos, and S Krishnamurthy, A Systematic Framework for Unearthing the Missing Links: Measurements and Impact, Symposium on Networked Systems Design and Implementation, April 2007.

[40] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, and J. Rexford, A Clean Slate {4D} Approach to Network Control and Management, ACM SIGCOMM Computer Communications Review, 35(5), 2005.

[41] N. Dulay, E. Lupu, M. Sloman, and N. Damianou, A Policy Deployment Model for the Ponder Language, IEEE International Symposium on Integrated Network Management, 2001.

[42] L. L. Peterson, N.C. Buchholz, and R. D. Schlichting, Preserving and Using Context Information in Interprocess communication. ACM Trans. Comput. Syst. 7, 3 (Aug. 1989), 217-246.

[43] K. Birman, and T. Joseph, Exploiting Virtual Synchrony in Distributed Systems. *SIGOPS Oper. Syst. Rev.* 21, 5 (Nov. 1987), 123-138.

[44] J. Stewart, Storm Worm DDoS Attack, SecureWorks Research, available at http://www.secureworks.com/research/threats/storm-worm

[45] J. Mirkovic, M. Robinson, P. Reiher, and G. Kuenning, Alliance Formation for DDoS Defense, Proceedings of the New Security Paradigms Workshop, ACM SIGSAC, August 2003.

[46] G. Oikonomou, J. Mirkovic, P. Reiher and M. Robinson, A Framework for Collaborative DDoS Defense, Proceedings of ACSAC 2006, December 2006.

[47] M. Natu and J. Mirkovic, Fine-Grained Capabilities for Flooding DDoS Defense Using Client Reputations, Proceedings of the Large-Scale Attack and Defense Workshop, August 2007.

[48] M. Mehta, K. Thapar, G. Oikonomou and J. Mirkovic, "Combining Speak-up with DefCOM for Improved DDoS Defense", Proceedings of ICC 2008.

[49] J. Li, D. Lim, K. Sollins. The 16th USENIX Security Symposium, DETER Community Workshop on Cyber Security Experimentation and Test 2007, Boston, MA, August 6-7, 2007.

[50] V. Paxson, Bro: A System for Detecting Network Intruders in Real-Time, Computer Networks, 31(2324), pp. 2435-2463, 14 Dec. 1999.

[51] J. Lepreau, R. Ricci, L. Stoller, M. Hibler, "Federation of Emulabs and Relevant New Development," Presentation at the USC/ISI Federation Workshop, December 2006. http://www.cs.utah.edu/flux/testbed-docs/emulab-fed-issues-dec06.pdf

[52] R. Ricci, J. Lepreau, L. Stoller, M. Hibler, "Emulab Federation Preliminary Design," Presentation at the USC/ISI Federation Workshop, December 2006. http://www.cs.utah.edu/flux/testbeddocs/emulab-fed-design-dec06.pdf

[53] J. Lepreau, "The Utah ProtoGENI Project," Presentation at the 1st GENI Engineering Conference, October 2007. http://www.geni.net/docs/Utah_ProtoGENI.pdf

[54] http://www.planet-lab.org/federation

[55] PlanetLab/OneLab Consortium, "Federation Roadmap", Draft White Paper, http://www.cs.princeton.edu/~llp/federation-roadmap.pdf

[56] I. Foster, C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," International Journal of Supercomputing Applications, vol. 11, pp. 115-128, 1997.

[57] "Web Services Federation Language (WS-Federation)", December 2006. http://specs.xmlsoap.org/ws/2006/12/federation/

[58] OASIS Standard, "WS-Trust 1.3", March 2007 http://docs.oasis-open.org/ws-sx/ws-trust/200512

[59] W3C Member Submission "Web Services Policy 1.2 - Framework", 25 April 2006. http://www.w3.org/Submission/2006/SUBM-WS-Policy-20060425/

[60] OASIS Standard, "WS-SecurityPolicy 1.2", July 2007. http://docs.oasis-open.org/ws-sx/wssecuritypolicy/200702