Integrated Environment Management for Information Operations Testbeds

T.H. Yu, B.W. Fuller, J.H. Bannick, L.M. Rossey, and R.K. Cunningham

Abstract Network testbeds are indispensable for developing and testing information operations (IO) technologies. Lincoln Laboratory has been developing LARIAT to support IO test design, development, and execution with high-fidelity user simulations. As LARIAT becomes more advanced, enabling larger and more realistic and complex tests, effective management software has proven essential. In this paper, we present the Director, a graphical user interface that enables experimenters to quickly define, control, and monitor reliable IO tests on a LARIAT testbed. We describe how the interface simplifies these key elements of testbed operation by providing the experimenter with an appropriate system abstraction, support for basic and advanced usage, scalable performance and visualization in large networks, and interpretable and correct feedback.[1]

1 Introduction

Network testbed research has advanced significantly in recent years. Testbeds provide a controlled setting for studying various aspects of a networked environment: hardware, protocols, algorithms, applications, social behaviors, and so on. Public testbeds, such as Emulab (White et al., 2002) and PlanetLab (Peterson et al., 2006), achieve a level of configurability, repeatability, and realism that is well suited for most networking research. DETER (Benzel et al., 2006) is an Emulab cluster with enhanced containment mechanisms designed specifically for cyber-security research.

T.H. Yu, B.W. Fuller, J.H. Bannick, L.M. Rossey, and R.K. Cunningham
MIT Lincoln Laboratory, 244 Wood Street, Lexington, MA 02420, e-mail: tamara@ll.mit.edu, bfuller@ll.mit.edu, john.bannick@ll.mit.edu, lee@ll.mit.edu, rkc@ll.mit.edu

[1] This work was sponsored by the Defense Advanced Research Project Agency (DARPA) and the Department of Defense under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Government.

Since 1997, Lincoln Laboratory has been developing software that generates realistic traffic for testbed networks by simulating users (Lippmann et al., 2000), eventually packaging the software as a tool called LARIAT, or Lincoln Adaptable Real-time Information Assurance Testbed (Rossey et al., 2002). The main distinction between LARIAT and other testbeds is that LARIAT generates a virtual user world that overlays on top of the experimenter's own testbed. LARIAT supports a wide range of software, hardware, and network topologies. It applies various models of user behaviors to directly exercise applications specified by the experimenter, resulting in highly realistic usage of the applications and the underlying operating systems, hosts, and networks.

LARIAT can be used for a broad range of information operations (IO) tests. It is particularly useful for security research and evaluation because it accurately represents vulnerabilities that occur as a result of flaws in the design and implementation of services, protocols, and applications by directly running the software used on the modeled network.

While LARIAT excels in providing host-level realism for tests, early versions of LARIAT were complicated to set up and use. In these first versions, experimenters configured test parameters via direct interaction with a database, manually set up user accounts and applications on each host, parsed log files to determine if the configuration scripts ran correctly, and manually verified that each traffic generator was running. This tedious process limited the size and complexity of networks that could be tested. These early experiments made it clear that efficient testbed management is important.

We have found that certain key tasks of IO experiments – test specification, test control (including software deployment, troubleshooting and validation), and test monitoring – are particularly problematic for experimenters. To address these tasks, we developed the Director, a graphical user interface for managing LARIAT testbeds. In this paper, we present two contributions: (1) an interface that greatly simplifies these key tasks and thereby significantly improves the usability of LARIAT, and (2) a demonstration of how several interface and visualization techniques can be used in large-scale testbed management software.

The remainder of this paper describes the Director in detail. Section 2 surveys related work. Section 3 describes the technical approach of our work. It first presents some background knowledge on the general LARIAT framework in Sect. 3.1, then identifies key design goals in Sect. 3.2, and finally presents the solutions in Sect. 3.3. Section 4 discusses lessons learned and future work. Section 5 closes with a summary.

2 Related Work

There have been extensive studies on testbeds for networking research and development. Solutions span simulation, emulation, overlay, and live networks, providing researchers a wide range of options for experimentation.

Simulation tools such as ns (Bajaj et al., 1999) and Simnet (Kamara et al., 2005) require minimal hardware and are easy to setup, control and use to obtain verifiable results. They efficiently simulate the normal operation of a variety of protocols and arbitrary network topologies, but cannot be used to explore flaws in protocol implementations or unusual application behavior without re-programming the flawed implementation or behavior. To create conditions that closely mimic live networks while maintaining experimental control, researchers turn to emulation tools.

Emulab (White et al., 2002) is a widely used network emulator. It allows experimenters to run the system under test on physical hosts with real operating systems and application software. Testbed assets can be rapidly configured to create any network topology. Hosts can be easily set to a target state for the experiment and reset to a clean state when the experiment is done. Furthermore, Emulab leverages tools like Dummynet (Rizzo, 1997) and ns to introduce synthetic network conditions and traffic between physical nodes. The mix of real resources and simulation provide both depth and breadth for the experimental environment. Emulab traffic is generated at the host level and is protocol-based. It does not model real applications and interactions between users, which can be critical to many security tests.

DETER (Benzel et al., 2006) is an Emulab cluster focused on information security concerns and repeatability of experiments. DETER is unique in that it is an open resource that allows for the running of malicious code. However, the platform's public nature limits the scope of experiments that can be run on the testbed. For strong assurance of security and confidentiality, coupled with significantly lowered hardware cost in recent years, many still prefer private, isolated testbeds.

Unlike Emulab, which operates on local dedicated clusters, PlanetLab (Peterson et al., 2006) is a global platform that runs on an overlay network of nodes on the Internet. It provides a testing environment with live network conditions for large-scale network services such as distributed hash tables (DHTs) (Rhea et al., 2005). PlanetLab users contribute computing assets to the platform in exchange for access to the collective resources. PlanetLab manages the nodes on behalf of their owners and allocates slices of those nodes to the users in a safe, fair manner. While PlanetLab is invaluable for the network research community, it is not for experiments that require direct access to and full control of testbed assets or those that require strict containment.

Many tools seek to effectively manage testbed systems. Plush (Albrecht et al., 2006) provides a framework for deploying applications in distributed environments such as PlanetLab. In Plush, experimenters define applications under test in XML. Based on the definition, Plush controls allocation of available resources, deployment of software, configuration of nodes, and execution of the application. The Plush framework is extensible: it integrates third-party tools where appropriate for distribution of software and monitoring of hosts. The current interface is designed primarily for application developers with area expertise on the definition and deployment tools in use.

The Experiment Workbench (Eide et al., 2007) is a system for managing experiments in Emulab. Using the Workbench, experimenters apply scientific process to testing by running experiments in controlled iterations. Each iteration is comprised

of setting or restoring the test environment to a pristine state, setting or changing parameters, performing the experiment, collecting data, and archiving any material (e.g., code, scripts, and configuration reports) pertaining to the experiment. This allows for easy verification of test results and clear comparison across runs. The Experiment Workbench provides a graphical user interface for easy access and control of the experiments.

SEER (Schwab et al., 2007) is the testbed management tool that is most similar to the Director in environment, goals, and implementation. It provides a user interface for DETER, enabling experiment definition and deployment of traffic generation scripts on the DETER testbed. In contrast, the Director focuses on the definition and deployment of virtual users for testbeds of arbitrary complexity.

Managing a testbed is similar to administering a network. In both cases, the operators have to monitor activities on many machines and detect anomalies, i.e., errors or attacks. Because of this similarity, two network traffic analysis tools are included in this discussion.

Network Eye (Ball et al., 2004; Fink et al., 2005) is a network traffic monitoring tool that provides visualization of the inbound and outbound traffic of a "home" network, allowing users to easily detect any unusual patterns in foreign connections. It further correlates those connections with host processes to aid forensic investigation of suspected attacks. While the aggressor–defender model does not apply to testbed management, Network Eye's techniques for summarizing network status, highlighting anomalies, and mapping low-level observations to high-level activities are relevant and valuable.

TNV (Goodall et al., 2005) is also a visualization tool designed for network traffic analysis. TNV creates timelines of network events by hosts, on top of which it overlays host connections. This allows analysts to view and correlate sequences of events and reconstruct scenarios. The tool provides additional coordinated views of detailed information such as ports and packet contents. The Director has a similar aim of displaying virtual user activities and communications over time for test verification and monitoring purposes.

3 Technical Approach

3.1 LARIAT Overview

LARIAT emerged from the 1998–1999 DARPA off-line Intrusion Detection System (IDS) evaluations (Lippmann et al., 2000). In the first evaluation, Lincoln Laboratory used a 16-node testbed network to emulate thousands of hosts modeling an Air Force base communicating with the Internet. Virtual and real users exercised real applications to generate traffic and launch existing and novel attacks on the testbed. The corpora of network traffic created from the testbed were used to measure the IDSs attack detection and false-alarm rates, resulting in the first formal, repeatable, statistically significant evaluation of IDSs. While the traffic corpora proved

valuable, network characteristics have changed and new devices, operating systems, and applications have been developed.

To provide researchers the flexibility of having their own testbeds, we created LARIAT. In preparation for LARIAT deployment, the testbed staff build the network(s) under test, install the operating systems and applications on the hosts, and deploy network or host-based defensive tools if necessary. On top of this basic testbed, LARIAT deploys virtual hosts and virtual users, creates an emulated Internet, and puts data content on servers to transform the generic network into the specific, desired environment. Driven by Markov models, which are based on patterns of activity derived from real user and network behavior (Boothe-Rabek, 2003), the virtual users exercise applications, access server contents, and interact with other virtual users.

LARIAT tests run on the topology available. The testbed topology generally varies from test to test and may even change during a test, such as when hosts migrate from network to network as a real user of a laptop might. The testbed may include widely disparate hardware – desktops, laptops, and other network devices with different architectures and manufacturers. Individual hosts may be running a wide range of operating systems, including Linux, Unix, and Windows, in multiple configurations. LARIAT supports tests in most of such customized, heterogeneous network environments.

In the LARIAT framework, a host may operate as a client, providing a platform for a virtual user or group of virtual users, or as a server, providing information at the request of one of the clients. Linux and Unix hosts can be configured in one of two modes: single-host, which operates much like a normal desktop, or multi-host, which sources traffic from multiple (10–10,000) IP addresses to emulate multiple hosts. LARIAT Windows servers do not support multi-host virtualization. Similarly, on a Windows client, only one user can act at a time and interact directly with applications (Boothe-Rabek, 2003).

Virtual users communicate via a range of protocols and applications, including e-mail (POP, SMTP, and Exchange) via Microsoft Outlook and mail, file sharing (SMB) via Windows Network and Samba, and web (HTTP, HTTPS, and FTP) via Internet Explorer and Mozilla. They can be assigned different roles, such as supervisors, secretaries, developers, or system administrators. The role assigned to a user dictates their daily usage pattern, the applications used, the way in which they interact with the applications, the types of contents and services accessed, and the pattern of collaboration with other users.

On a LARIAT testbed, servers host data and provide services that need to be accessed by the virtual users. LARIAT automatically configures many Linux and Windows servers, including Apache, Internet Information Services (IIS), Very Secure FTP Daemon (vsftpd), CVS, and Certificate Authority (CA) Server. A substantial repertoire of data contents, such as sample web sites, is provided and can be automatically deployed on the appropriate servers. LARIAT also sets up accounts that allow users to log onto remote hosts via SSH or Telnet. Finally, LARIAT supports emulation of an Internet cloud hosting thousands of web sites and the root

Domain Name Servers, creating the appearance that an isolated testbed is connected to the Internet.

The available options in LARIAT provide an enormous breadth of possibilities. The approach of modeling users of real applications allows for testing of open source and commercially available software, open and proprietary protocols, and the performance of security tools using attacks against actual applications. The challenge is to enable all of these options in a manner understandable to the experimenter.

3.2 Design Goals

LARIAT is a powerful but complex system. When designing a user interface for LARIAT, we must address the following challenges:

- *Appropriate system abstraction*. A key goal of the user interface is to provide functionality without burdening experimenters with implementation details. The user interface should help experimenters design the test they want to run for the network they have and the users and traffic profiles they want to model.
- *Basic and advanced usage*. It is important to identify a set of features common to all tests and make them obvious and easy to use, so that experimenters can focus on correctly designing and configuring the custom components necessary for their particular tests.
- *Scalable performance and visualization*. The user interface must perform efficiently on small and large testbeds. Since LARIAT supports modeling individual and groups of users and individual hosts and enterprise networks, status needs to be provided across a wide range of scales, and interface elements must be able to depict both overviews and details.
- *Interpretable and correct feedback*. The user interface should provide useful feedback on all tasks performed, on all system components under control. Accurate reports of testbed status and detailed troubleshooting information need to be collected in a centralized display tool.

3.3 Interface and Visualization

3.3.1 Director Overview

The Director is a Java-based graphical user interface that serves as the control center of the testbed. It provides a gateway to testbed resources as well as the database, where configuration, status, and log information is stored. It allows experimenters to design and execute tests in a high-level environment. The home screen in Fig. 1 conveys the overall workflow on a LARIAT testbed. Items in the center column depict the basic steps of running a test: describing the *Testbed Environment*, defining the

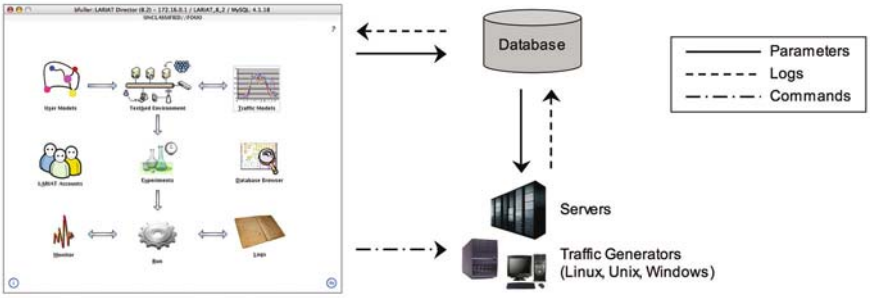

Fig. 1 Data and message exchanges between the Director (*left*) and other testbed components (*right*)

Experiments, and performing experiment *Run*s. Items on the sides provide additional control for advanced users to refine *User Models*, adjust *Traffic Profiles*, and enable access control with *LARIAT Accounts*. As shown in Fig. 1, the Director coordinates actions on the testbed, collects and processes logs, and displays status to the experimenter. The testbed *Monitor* and detailed *Logs* provide further aids in troubleshooting.

The Director is specifically designed to address several key tasks of IO testing: test specification, testbed control, and testbed monitoring. In Sects. 3.3.2–3.3.4, we will describe the visualization components of this interface in detail, focusing on how they meet our design goals in helping experimenters perform these tasks.

3.3.2 Test Specification

A test specification consists of three components: the testbed environment (including both the network topology and the virtual environment overlay), the models that control user actions and shape traffic, and the parameters that define a run.

In LARIAT, the testbed environment is organized hierarchically. At the top is the site, i.e., the testbed facility. Within a site are one or more networks. Networks are broken up into network segments, which in turn contain the physical hosts. Each host has a configuration that describes what the machine represents in the context of a test (e.g., a personal desktop, a public shared machine, an entire organization with many virtual hosts, or a server). Multiple configurations can be specified for a host. This lets experimenters quickly change settings by selecting an alternate configuration. LARIAT automatically generates virtual hosts and virtual users for the machines, which are positioned at the bottom of the testbed hierarchy.

Figure 2 illustrates the hierarchical organization of a LARIAT testbed in the Director. On the left side, the experimenter can see that within the testbed *MIT Lincoln Laboratory* there is a network called *Internet.world.net*, which is responsible for emulating the Internet. This network contains one logical subnet with several hosts: *INT-PC-010*, *hive*, *page*, *mela*, and *spew*. The real host *spew* emulates a set

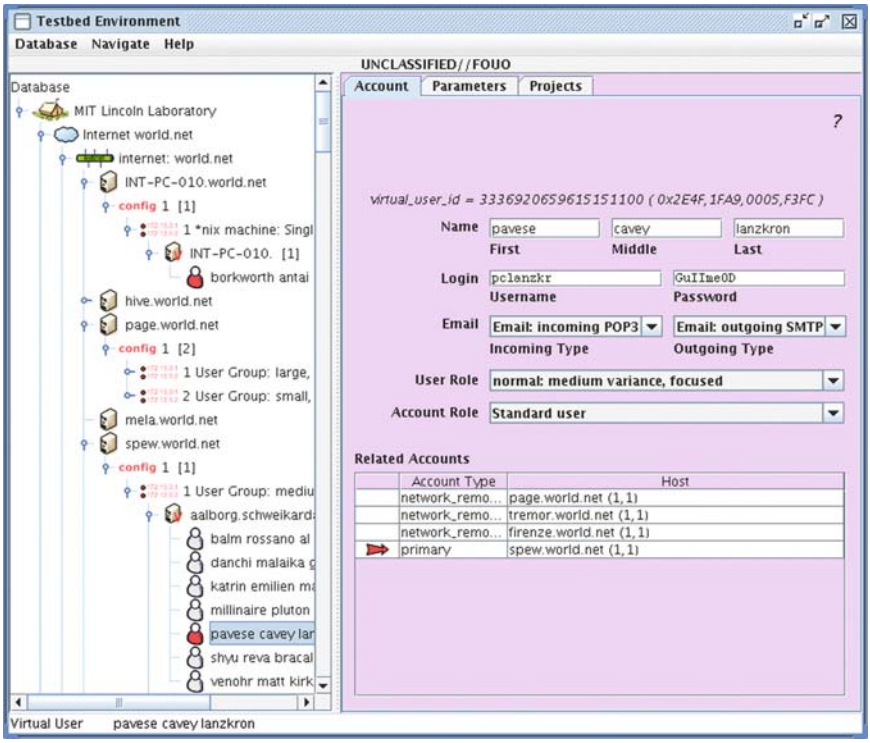

Fig. 2 The testbed environment is displaying part of the hierarchical tree with a virtual user selected

of virtual hosts. Virtual users are assigned accounts on these virtual hosts. In the display, the account of Pavese Cavey Lanzkron is selected, showing his login username and password, his Email client's supported protocols, his user role and account type. *Related Accounts* shows that the user has additional accounts elsewhere on the network. Under the *Parameters* tab, the experimenter can change the user's behavior by adjusting parameters such as the workday hours, the probability of a mistyped command, and the likelihood to access projects. Finally, *Projects* provide the experimenter a mean of modeling collaboration among users, who may share documents and exchange emails frequently.

The testbed environment presents a natural and coherent view of the testbed. The tree structure provides an abstraction of the relations among numerous testbed components of all types. By expanding and collapsing nodes, it allows the experimenter to manage the level of complexity of the testbed configuration exposed, reduce visual clutter, and focus on specific areas of interest. The tree is highly scalable. Even in a large network with hundreds of machines and hundreds of thousands of users, experimenters can still systematically navigate their way around and drill down to any parts necessary. Furthermore, similar to a filesystem browser, the dual-panel interface allows experimenters to quickly navigate the tree on the left and edit

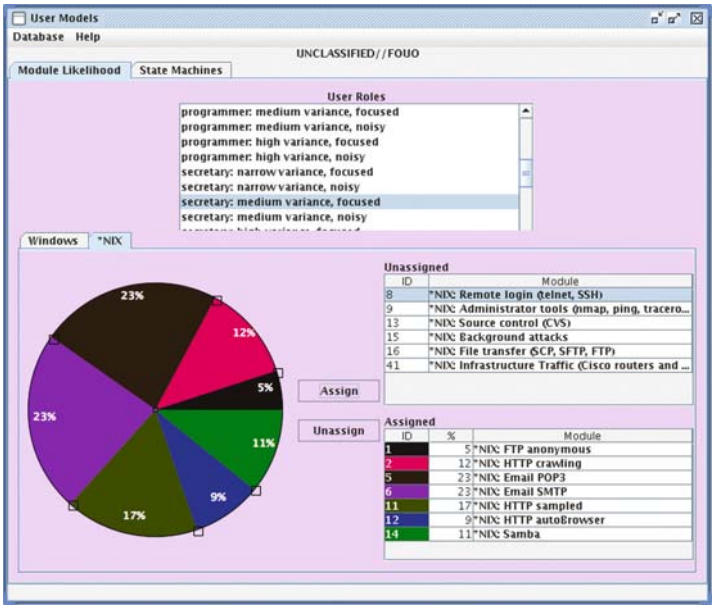

Fig. 3 Experimenters can fine-tune the traffic profiles

the properties of the selected node on the right. Experimenters can add, remove, and search for nodes. Compared to alternative representations such as network graphs, this interface is more compact for viewing large testbed setups and more streamlined for the typical testbed design workflow, which is heavy in edits.

For a basic LARIAT setup, the testbed environment is all that an experimenter needs to configure. LARIAT provides a default set of models that generate reasonable traffic. However, if customization is necessary, the Director also provides more advanced controls. For example, Fig. 3 displays an interface for defining models of virtual users' computer activities as split among different applications and protocols, where each pie segment represents an application or protocol. From this interface, experimenters can quickly gauge the expected traffic composition and adjust it by dragging the handles on the edge of the pie. There are tools for defining other aspects of the traffic models, such as application state machines and daily traffic rates (not shown here). These graphical representations help experimenters understand and modify virtual user behaviors.

Another way to customize the communication patterns on the testbed is through network flows. Network flows are predominately used to represent security policies of organizations. They describe what types of inbound and outbound traffic are allowed in a network as well as any exceptions. Figures 4 and 5 show two complementary views of the same flows on a testbed.

In the graphical view (Fig. 4), directed edges represent traffic flows among network nodes. In this example, most of subnets can communicate with each other, forming the well-connected cluster in the lower left corner. The cluster on the right

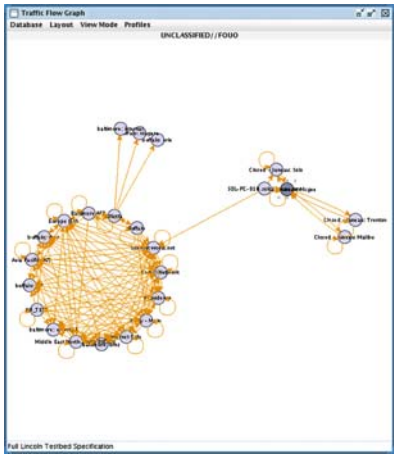

Fig. 4 A graphical view of network flows

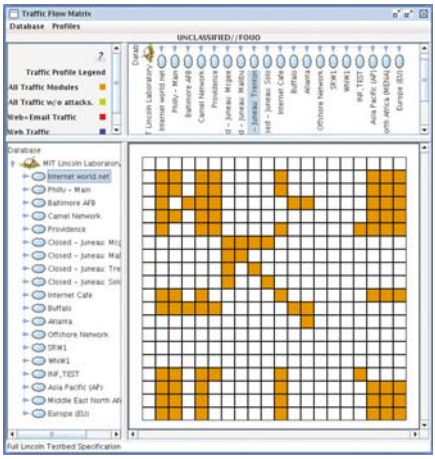

Fig. 5 A matrix view of network flows

represents a small community that talks among themselves but not with anyone outside. An interesting exception here is the host *SOL-PC-010.solo.juneau.gov*, which can go to *Internet.world.net*. The graphical view is an intuitive way of visualizing the flows. Major traffic patterns are easily identifiable, as are certain outliers. However, the graphical view does not scale well. When the number of nodes increases (e.g., when there are more host-level rules), the number of edges could grow quadratically, making the view unreadable very quickly. The graphical view is also relatively difficult to edit.

The matrix view (Fig. 5) solves some of these problems. The traffic flows are presented in an adjacency matrix, in which rows and columns correspond to nodes in the testbed's network topological tree. The cell (r, c) is colored if there is a rule that allows the node in row r to send traffic to the node in column c. Unlike the graphical view, in which the layout could change drastically when some of the flows

change, the matrix view places nodes in an organized and fixed manner. This makes flow editing much easier. The ability to collapse parts of the tree allows this view to work efficiently with large testbeds and lots of flow rules. The main disadvantage is that when a node is collapsed, the lower-level flows are completely hidden. This could result in misinterpretations. We plan to address this in future versions.

In general, the directed graph is intuitive but hard to manipulate; the matrix is space-efficient and easy to edit but poor in showing the "big picture." They can be used to complement each other. Similar dual-representation has been shown in MatrixExplorer (Henry and Fekete, 2006). We adopt this representation because it does not force experimenters to create and verify flows strictly in one view or the other, but allows them to choose the best view for the task at hand.

3.3.3 Testbed Control

Once an experiment is designed, the experimenter has the challenge of realizing the test design on the physical testbed. Setting up hosts individually through manual configuration and ad hoc scripts works in the short term for small testbeds, but the solution is neither repeatable nor scalable. The Director streamlines software deployment and configuration by introducing a well-defined sequence of tasks with precise commands for any host and test configurations. An example is the *Traffic Preparation:All* task selected in Fig. 6, which gives hosts configuration information needed for the experiment. Tasks are organized into phases. In this case, the selected task belongs to the phase *Validation*, which contains tests to ensure the network is properly configured and that hosts will generate traffic as expected. Phases make up the final workflow that the experimenter can step through. In each step, the experimenter can simply select a task, select the hosts to run the task, start the task, and monitor the progress from the Director (Fig. 6).

The Director allows experimenters to execute tasks on testbed hosts in parallel. Based on the task, the host's operating system, and the connectivity of the network, the Director automatically determines how to connect to the host (via SSH, Telnet, etc.) and gives the appropriate command to start the task. The Director uses a thread pool to handle connections to multiple hosts concurrently. To support rapid configuration in large networks and to meet our scalability design goal, the Director can delegate tasks to others. Instead of connecting to all hosts, the Director can connect to one or more group leaders and ask them to either perform the task for all or relay the task order to their group members. Delegation greatly reduces the processing burden on the Director, allowing it to quickly satisfy the experimenter's requests.

As the hosts perform the task, they send status updates to the database. The Director periodically queries the database and displays the progress to the experimenter. The task status display is organized around the hierarchical network topological tree. Each row corresponds to a node. The display shows the overall status for each host followed by pie charts that demonstrate progress in various aspects of the task. The outcome of each step, or wedge, is color-coded: green means okay, yellow means warning, and orange, red, and magenta mean errors at different levels of severity.

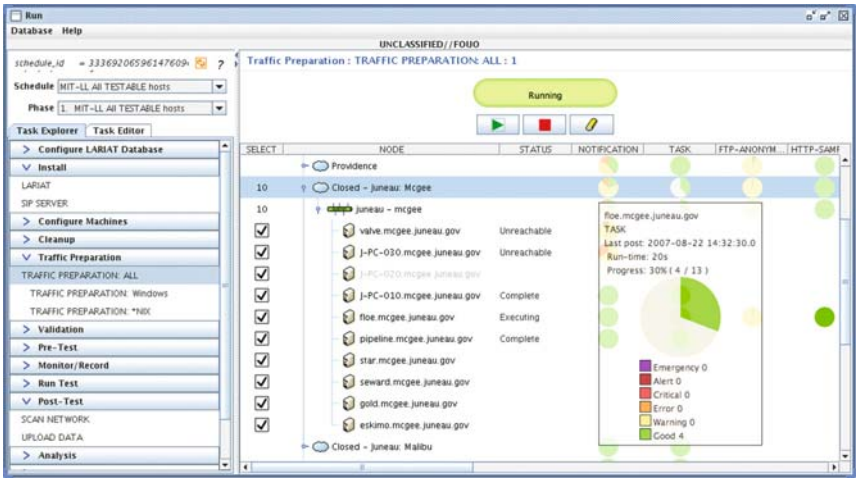

Fig. 6 An interface for running and monitoring tasks on the testbed, which displays the workflow on the left and the task status on the right

Small pie charts are effective in providing the overview at a glance. The extent of problems on the range can be assessed by the amount of red shades shown. The pattern of the colors may also suggest the source of the problem, e.g., whether it is local or at a gateway. If the experimenter wants more details, he or she can mouse over a small pie chart, which would pop up a magnified version. The experimenter can drill down even further by double-clicking the small pie chart to view a set of relevant logs useful for troubleshooting.

In addition to showing host-level status, the task status display aggregates status up to the network segment, network, and site levels. Aggregate status is displayed as semi-transparent to differentiate from host status. The need for aggregation arises from the difficulty of debugging large testbed networks. For testbeds with hundreds of nodes, there is too much information when viewing all hosts at once: experimenters would have to keep scrolling and they cannot focus on a subset of hosts. With aggregation, experimenters can collapse some networks and concentrate on the rest. The faintly displayed aggregate status would not compete for the experimenters attention, but at the same time ensures that any warning or error conditions are still shown. This meets our fourth design goal providing overall status of testbed tasks with the ability to pinpoint specific problems and receive verbose information about errors.

Finally, the Director's task framework is designed to be highly extensible. Instructions for tasks are encoded in database tables and normally abstracted away from experimenters by the interface. However, experimenters have the option of creating custom tasks by adding them to the task tables. Then they can run these custom tasks the same way as normal tasks from the Director. Some examples include tasks to start traffic capture and archive test results.

3.3.4 Testbed Monitoring

While executing tasks, the Director is effective in providing status and troubleshooting information; to meet our goal of interpretable feedback, it is necessary to give information during an experiment as well. Given the vast number of events that occur during a test, this data must be presented in a way that gives the experimenter useful information without being overwhelming. During an experiment, the Director provides status at both the host and user level: a host health monitor (Fig. 7) tracks machines' CPU, disk, and memory usage, and a traffic event viewer (Fig. 8) displays user-driven activities in a timeline format.

The host health monitor (Fig. 7) is a visualization tool designed to allow experimenters to quickly detect and mitigate problems related to system resource usage. The visualization has two parts: a heatmap on the left, which indicates the overall condition of hosts on the testbed, and a set of charts on the right, which provides detailed historical information on a selected host. The heatmap has three columns, which represent CPU, disk, and memory usage. Each row corresponds to a host. The shade of each cell indicates the condition or activity level of a given type of system resources on a given host: green means low usage, black means medium usage, and red means high usage. As the number of hosts increases, the display tries to shrink the row height to fit all hosts. When the number of hosts gets so big that they cannot all be presented in the display in a legible manner, adjacent hosts are

Fig. 7 A dynamic heatmap that represents the health status of testbed hosts, along with details for the selected host

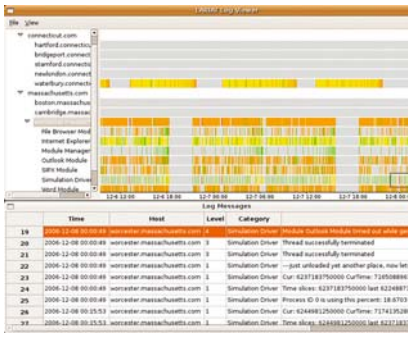

Fig. 8 A timeline of virtual user traffic events expanded by traffic type on testbed hosts

aggregated into one row that displays the worst conditions in the group. In this way, an experimenter can monitor the full testbed at a glance and would never miss hosts in distress even when statuses are aggregated. This display is also highly dynamic. When the experimenter moves the mouse over the heatmap, the rows near the mouse will expand to show individual hosts' status. The experimenter can click on a host to lock the display and then examine the charts on the right more closely. These charts display the recent history of the selected host's health condition, including detailed measurements of load, disk space, swap and paging activities. Together, the heatmap allows experimenters to quickly detect problems and the charts help them diagnose the problems, making this an effective diagnostic and monitoring tool for testbeds.

In any large, complex network, failures are inevitable. Generally, users report these errors to administrators, who then diagnose and correct the problems. LARIAT, at a minimum, must be able to reproduce the error reporting part of the process; that is, it should provide an experimenter with detailed status of the virtual users as they log their actions and results. Figure 8 shows how these logs are presented to the experimenter. Each traffic generator has events displayed in a timeline. The events are painted as vertical bars. The colors of the bars indicate the outcomes of the events: green for okay, yellow for warning, and orange for error. At the high level, the display shows when virtual users generate traffic and whether the traffic is okay. For example, the users on *http://www.waterbury.connecticut.com* are working approximately 8-hour shifts with 4-hour breaks in between. There are a lot of warnings and some errors toward the end of the each shift. The experimenter can zoom in to examine the events more closely. He could also break down the traffic events by applications. For example, traffic on *http://www.worcester.massachusetts.com* is generated from the *File Browser*, *Internet Explorer*, *Outlook*, *Words* among others. In this case, email seems to have a lot of failures, while web browser seems okay. To investigate the problems further, the experimenter can retrieve log messages for the selected events, as shown in the lower part of the display.

The primary use of these monitoring tools is to help experimenters verify that tests are running as expected (at least from the traffic generation perspective) and detect any system or configuration error. LARIAT does not monitor the system under test. Nevertheless, the outcomes of traffic events are directly affected by test events. Therefore, the results shown in these displays must be interpreted in the context of the experiment. For example, traffic failures may be the result of activating defensive tools that block critical pathways. While it is not a complete solution, these monitoring tools can provide valuable situation awareness during test.

4 Future Work

LARIAT and the Director have been used extensively over the last several years at dozens of sites. As with any interface design, the Director must balance expressiveness with ease of use. For example, a network is better represented as a graph

than a tree. However, a graph with thousands of nodes is hard to understand and even harder to manipulate. The simplicity of a tree is attractive, but there are network devices that cannot be easily classified into one network or another. The tree structure is also unsuitable for describing the link layer. We are considering a hybrid approach to provide the best of both worlds.

Over the last decade, we have seen IO testbeds grown significantly, but experimenters are not given any more time to configure their testbeds. Most experimenters already have tools for designing network topologies integrated into their workflow (e.g., Visio). It would save a lot of time if the Director could import data from these tools, so that the experimenters do not have to enter the network information again. The data entry problem can also be alleviated by auto-discovering the topology. However, it is unlikely that all of the information needed by LARIAT can be discovered without modifying existing tools. To this end, it would be useful to let experimenters set properties for multiple hosts simultaneously in a context sensitive environment, so that these missing pieces of information can be filled in quickly.

Furthermore, as testbeds grow, situational awareness becomes even more important. The Director visualizes both host and user-level events, but it is still missing other important aspects of traffic generation, such as the spheres of communication and the paths of traffic. For example, in addition to reporting that a user is browsing a file share, it would be informative to correlate it with other events, such as this user's past activities and other users' actions in this file share. Through correlation, the visualization can start painting a more complete picture of the networks in a test. The challenge is to scale any solution to hundreds of thousands of users, both in designing the visualization and in engineering the system.

5 Conclusions

In previous sections, we described the design and implementation of the Director. The Director dramatically reduces the time and effort required to describe and perform a test in LARIAT. This improvement is due in part to intelligent defaults and the ability of the Director to automatically create the virtual user environment. This allows experimenters to very quickly define their test network at a level they are comfortable with and the Director will populate the virtual user world on top of that topology. For experimenters that desire more fine-grained control, the Director supports editing of user and traffic models, as well as various parameters of instantiated users. As test networks expand to include hundreds of thousands of virtual users, the Director has employed many techniques to address scalability issues. For example, to avoid making too many simultaneous connections, the Director may request a few hosts to relay task orders to others if possible. Task results are posted to a centralized database, where the Director provides an at-a-glance status of the task, showing successes, warnings, and failures. The goal of this work was to help experimenters define and execute tests more accurately and quickly on LARIAT testbeds. We have achieved this goal by providing a unified, easy-to-use environment for controlling

testbeds at both the network level and the user level. The Director acts in concert with other LARIAT software to enable experimenters to create virtual components, configure networks and hosts, and monitor status and activities easily and reliably.

Although the capabilities of LARIAT are unique, the usability challenges are common among large testbed systems. We found Ben Shneiderman's Visual Information Seeking Mantra "overview first, zoom and filter, then details-on-demand" (Shneiderman, 1996) to be a reliable approach in dealing with structural complexity and information overload. We chose visual representation structures that were intuitive and scalable. The hierarchical testbed tree and the dynamic heatmap both handle aggregation well. They allow experimenters to drill down to a specific area without losing sight of the context. Elements are rendered using visual cues such as hue, intensity, and shapes to help experimenters make high-level assessments of the testbed and to direct their attentions to problems. The consistent application of the overview-to-details model, the selection of scalable representations, and the attention to visual features made the Director a successful interface for LARIAT and for IO testbeds. We hope our experience will provide insights for future development in testbed management and interface software.

Acknowledgements The authors wish to thank all contributors to the LARIAT project. We would also like to thank Chris Connelly, Sanjeev Gupta, Kyle Ingols, Shawn O'Shea, Reed Porada, Douglas Stetson, and Seth Webster.

References

Albrecht, J., Tuttle, C., Snoeren, A.C., Vahdat, A.: PlanetLab Application Management Using Plush. SIGOPS Oper. Syst. Rev. **40**(1), 33–40 (2006)

Bajaj, S., Breslau, L., Estrin, D., Fall, K., Floyd, S., Haldar, P., Handley, M., Helmy, A., Heidemann, J., Huang, P., Kumar, S., McCanne, S., Rejaie, R., Sharma, P., Varadhan, K., Xu, Y., Yu, H., Zappala, D.: Improving simulation for network research. Tech. Rep. 99-702b, University of Southern California (1999)

Ball, R., Fink, G.A., North, C.: Home-centric visualization of network traffic for security administration. In: VizSEC/DMSEC '04: Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security, pp. 55–64. ACM Press, New York, NY, USA (2004)

Benzel, T., Braden, R., Kim, D., Neuman, C., Joseph, A., Sklower, K., Ostrenga, R., Schwab, S.: Experience with DETER: A testbed for security research. Second IEEE Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities (TridentCom 2006), March (2006)

Boothe-Rabek, J.C.: WinNTGen: Creation of a Windows NT 5.0+ Network Traffic Generator. Master's thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science (2003)

Eide, E., Stoller, L., Lepreau, J.: An Experimentation Workbench for Replayable Networking Research. Proceedings NSDI, Apr (2007)

Fink, G.A., Muessig, P., North, C.: Visual correlation of host processes and network traffic. In: VIZSEC '05: Proceedings of the IEEE Workshops on Visualization for Computer Security, p. 2. IEEE Computer Society, Washington, DC, USA (2005)

Goodall, J.R., Lutters, W.G., Rheingans, P., Komlodi, A.: Preserving the big picture: Visual network traffic analysis with tn. In: VIZSEC '05: Proceedings of the IEEE Workshops on

Visualization for Computer Security, p. 6. IEEE Computer Society, Washington, DC, USA (2005)

Henry, N., Fekete, J.D.: Matrixexplorer: a dual-representation system to explore social networks. IEEE Transactions on Visualization and Computer Graphics **12**(5), 677–684 (2006)

Kamara, S., Davis, D., Ballard, L., Caudy, R., Monrose, F.: An extensible platform for evaluating security protocols. In: ANSS '05: Proceedings of the 38th annual Symposium on Simulation, pp. 204–213. IEEE Computer Society, Washington, DC, USA (2005)

Lippmann, R.P., Fried, D.J., Graf, I., Haines, J.W., Kendall, K.R., McClung, D., Weber, D., Webster, S.E., Wyschogrod, D., Cunningham, R.K., Zissman, M.A.: Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation. DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00. Proceedings **2**, (2000)

Peterson, L., Bavier, A., Fiuczynski, M.E., Muir, S.: Experiences building planetlab. In: USENIX '06: Proceedings of the Seventh Conference on USENIX Symposium on Operating Systems Design and Implementation, pp. 25–25. USENIX Association, Berkeley, CA, USA (2006)

Rhea, S., Godfrey, B., Karp, B., Kubiatowicz, J., Ratnasamy, S., Shenker, S., Stoica, I., Yu, H.: Opendht: a public dht service and its uses. In: SIGCOMM '05: Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer communications, pp. 73–84. ACM Press, New York, NY, USA (2005)

Rizzo, L.: Dummynet: a simple approach to the evaluation of network protocols. SIGCOMM Comput. Commun. Rev. **27**(1), 31–41 (1997)

Rossey, L.M., Cunningham, R.K., Fried, D.J., Rabek, J.C., Lippmann, R.P., Haines, J.W., Zissman, M.A.: LARIAT: Lincoln adaptable Real-Time Information Assurance Testbed. Aerospace Conference Proceedings, 2002. IEEE **6**, (2002)

Schwab, S., Wilson, B., Ko, C., Hussain, A.: Seer: A security experimentation environment for deter. In: DETER: Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test on DETER Community Workshop on Cyber Security Experimentation and Test 2007, pp. 2–2. USENIX Association, Berkeley, CA, USA (2007)

Shneiderman, B.: The eyes have it: A task by data type taxonomy for information visualizations. vl **00**, 336 (1996)

White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., Joglekar, A.: An integrated experimental environment for distributed systems and networks. In: Proceedings of the Fifth Symposium on Operating Systems Design and Implementation, pp. 255–270. USENIX Association, Boston, MA (2002)