Improving Testbed Experiment Design Through Shifting User Interface Emphasis

Genevieve Bartlett
ISI/USC
Marina del Rey, CA
bartlett@isi.edu

Jelena Mirkovic
ISI/USC
Marina del Rey, CA
mirkovic@isi.edu

Jim Blythe
ISI/USC
Marina del Rey, CA
blythe@isi.edu

***Abstract*—Considerable funding and effort over the past fifteen years have been invested in building large and versatile network testbeds, to support distributed experimentation. Most of the work to date on these network testbeds has been in support of dynamically reserving and configuring the physical resources. This focus on the physical resources of an experiment, and their interconnection, aka *experiment structure*, has lead network testbed user interfaces (UIs) to be centered around *just the structure* of an experiment, and have little to no support to help the testbed user design the *behavior* of her experiment—the actions and orchestration needed to carry out her experiment goals. In our previous work we proposed a new UI for experiment design. Our UI asks users to define the behavior of their experiment, and then derives experiment structure and orchestration automatically from the behavior definition. In this paper we extend our previous work with a survey, to evaluate expressiveness and user-friendliness of our UI. We implement a prototype of our behavior-centric UI, and ask survey respondents to compare it to the traditional, structure-centric UI. We hypothesize that testbed experiment design is improved when users are asked to focus on their experiment behavior, rather than the common practice of defining the experiment structure directly. We use this preliminary survey, to explore our hypothesis, and evaluate next steps in our effort to design better a network testbed UI. Our work presented here is a small step towards identifying fruitful directions in expanding tools and user interfaces to support network testbed users in rigorous and systematic experiment design.**

***Index Terms*—testbed UI, network testbeds, experiment design, user interface, human factors in testbed experiment design, NLP**

I. Introduction

Progress in computer systems, networking and security fields today is driven by industry and academic research. Central to such research is the demonstration of research solutions using distributed, large-scale and complex experiments on *network testbeds*. Additionally, these testbeds serve as training and educational resources [2]. Considerable funding and effort over the past fifteen years has gone into building such testbeds (e.g., Emulab [13], GENI [4], National Cyber Range [5], Deterlab [9], Cloudlab [8], Orbit [11], Fire [10]), however most of the testbed research to date focused on support for efficient allocation and sharing of physical resources.

Far less effort has been spent on supporting testbed experimentation itself, and developing tools and interfaces which support the user in designing rigorous experiments. Today's network testbed user interfaces (testbed UIs) focus on describing an experiment's *structure*—the physical compute machines and interconnections between these machines comprising an experiment's topology. Once such a topology has been instantiated on a testbed, users are on their own to choose tools to define, configure, and orchestrate the experiment's *behavior*—the actions carried out on the structure to test and validate an hypothesis.

This lack of UI for capturing experiment behavior has several negative effects on sharing, building upon, and disseminating best-practices for testbed experiments. First, capturing many of an experiment's details are left up to the user—a process often forgotten—leaving many testbed experiments difficult to reproduce and share. Additionally, this disconnect in support for experiment behavior means that creating experiments is often an ad-hoc and highly manual process, which requires users to become testbed experts to construct and run even basic experiments. The steep learning curve and tedious manual process of constructing a testbed experiment can introduce frustrating errors in experiment design, which prevent the user from successfully instantiating an experiment. Worse, large manual effort in designing and running an experiment can introduce subtle errors, which can lead to incorrect results and conclusions. Ultimately, current testbed experiment UIs stymie scientific advancement.

In our previous work [7] we have proposed a new UI for experiment design, which focuses on behavior and not on experiment structure. From this behavioral description, we then aim to automatically synthesize experimental structure and orchestration scripts, which realize the desired behavior. The paper [7] introduced our behavior-centric UI and a language to express experimental behavior and constraints, Anon-Lang. We designed Anon-Lang to capture the salient details of an experiment's behavior while still remaining human-readable [7].

To illustrate the current, structure-centric approach to experiment design, and our proposed behavior-centric approach, we can use a toy experiment. In this toy experiment, a user wants one machine, machine A, to send ping traffic to machines B and C.

The structure-centric approach would have the user manually enter a topology (e.g., A connects to B and C via LAN) and then manually write a script to send pings from A to B and C.

Our behavior-centric approach would ask the user to first define "roles" of nodes. In our example, there is a "sender"—

currently played by node A—and a "receiver"—currently played by each of the nodes B and C. We would then ask about the behavior of each role. The user could express that the sender sends ping traffic to the receiver. Finally, we ask about any structural "constraints", e.g., if sender and receiver are connected directly or not. In this example, there are no interconnection constraints, but there are two receivers. From this input we would automatically synthesize a topology with three nodes—a sender and two receiver on a LAN. We would also synthesize a script where the sender sends some traffic to the receiver. The user could either give us a pointer to the executable generating traffic (e.g., ping) or we would generate a "template" script, where the path to the executable is left for the user to populate during runtime. It is easy to see how our approach helps users design experiments in a top-down approach, and how it is easy to generate any number of similar structures (e.g., 1 sender and 5 receivers, 5 senders and 2 receivers) from the same behavior description.

In this paper, we extend our previous work, by implementing a prototype UI, which enables users to play with the core concepts of our approach, and we evaluate the direction of our efforts using a limited survey. This survey is the first step towards understanding where to focus our efforts going forward. Through our prototype UI, we had survey respondents compare our behavior-centric UI to a traditional structure-centric UI. For this initial survey, we restricted the traditional testbed UI we compare with to the Anonymized Network Testbed[1], and recruited novice, intermediate and expert users from the Anon-Testbed community to participate. Even though we restrict our comparison to a single testbed, we expect the results of this comparison to hold for network testbeds in general, since today's standard approach to network testbed UI shares the common focus on structure and not behavior. We gather qualitative feedback about behavior-centric and structure-centric UI from survey respondents, and also perform a more quantitative comparison through having respondents produce the necessary inputs to create a basic experiment via both UIs.

We believe focusing on the behavior of an experiment first is more natural to how experiments are formulated in the mind of the testbed user, and that following this flow results in better experiment design. We confirm this belief through our survey. Additionally, our prototype UI and survey are designed to gain preliminary feedback on two different UI methods to capture experiment behavior. First, in our prototype, users can express their experiment's behavior in natural language. Using Natural Language Processing (NLP) we extract details about roles and actions in the experiment, and any ordering of the actions, and convert this into Anon-Lang. Second, our prototype helps users write Anon-Lang directly, by offering predictive suggestions and pointing out errors as statements are written. Our user interface also automates some of the more tedious set up aspects of working with testbeds, and can automatically generate orchestration scripts and topology—a process often done by hand with current UIs.

In the following sections, we detail the user interface in our prototype, and discuss the survey we created to evaluate behavior-centric and structure-centric UIs. We close by discussing the survey responses—both the qualitative responses, gathered from user feedback and the quantitative responses, gathered by comparing the output participants produced.

II. User Interface and Approach to Experiment Design

Our approach focuses on experiment behavior by enabling the user to define their experiment in terms of the *actors*, *actions* and dependencies between these actions in their experiment. The actors are the distinct roles within an experiment, which exhibit unique behaviors, such as "server", "monitor", "attacker", "firewall", "client", and so on. Each actor may be realized as a separate physical machine, virtual machine, a process or a container and multiple actors may reside on the same machine. Each role may be carried out by a single instance or multiple instances. This mapping of actors to the experiment structure is separate from defining the behavior of the experiment. Users first define the actors and their actions and dependencies. Next, users define the mapping of actors to structure by using constraints, such as saying that there are ten instances of the client actor in the experiment, constraining how clients are connected to a server (e.g. via a LAN) or defining what types of hardware or software can be used to realize a client.

Separating out the mapping from defining the behavior means the user can focus on the events in their experiment first. Details about structure and mapping to structure—such as scaling, topology, hardware and software—can fall out of the defined behavior, rather than be first-order concerns.

To emphasize this "behavior first" approach and evaluate this approach, our prototype UI supports four views: two input views (one for natural language input and one for Anon-Lang input), and two output views, where users can visualize their workflow and structure derived from their input. In the next subsections, we describe these views in detail.

A. Natural Language Input

Our NLP view, based on the SpaCy tool kit [3], enables users to write free-form English sentences. These sentences are then processed to extract actors, actions and statements which imply dependency or ordering. For example, "after *A* happens, *B* happens" gives information about the dependency between "A" and "B".

As an example of NLP input, a user can type:

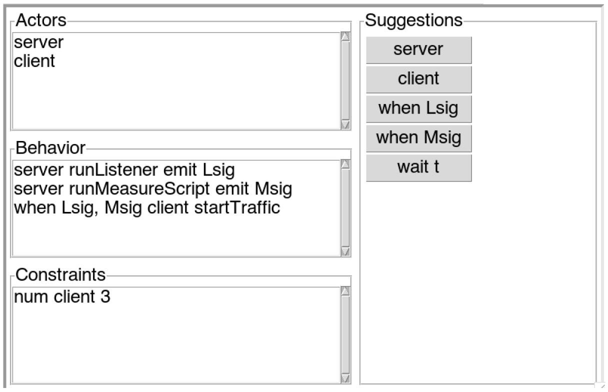
```
After the server starts the listener
and the measure script, the client will
start its traffic.
```

The UI would identify two actors (a "server" and a "client"), three events (starting the "listener", starting the "measure script" and starting "client traffic"), and the condition that the server

[1] To abide by CACOE's Anonymous Submission requirement, we have anonymized the name of the testbed as the authors are closely associated with this testbed. We have also anonymized the terminology for the specific inputs for this testbed (e.g. Anon-File) and our referenced prior publication [7].

```
server runListener emit sListenerSig
server runMeasure emit sMeasureSig
when sListenerSig, sMeasureSig client
 ↪  start_traffic emit sTrafficSig
```

Fig. 1 *Example Anon-Lang input.*



Actors

server
client

Behavior

server runListener emit Lsig
server runMeasureScript emit Msig
when Lsig, Msig client startTraffic

Constraints

num client 3

Fig. 2 *Screenshot of Anon-Lang UI tab.*

must start the listener and measure script, before the client starts its traffic).

B. Structured Input via Anon-Lang

Natural Language Processing is not perfect, and often requires correct spelling and punctuation to be fully utilized. These and other limitations mean that users may want a more direct way to specify their experiment behavior, and can do so through our Anon-Lang. Anon-Lang statements take the form of <optional trigger><actor><action><optional emit stmt>. The actor and action is mandatory, but an optional "trigger" can specify the conditions that set off this action and an optional "emit" statement attaches a label to the action that can be then referred to in other statements in their trigger clause. For example, Figure 1 shows a few lines of Anon-Lang, which specify that after a "runListener" action and a "runMeasure" action have completed on the server actor(s), a "startTraffic" event is started by the client actor(s).

Figure 2 shows our Anon-Lang input view in our prototype UI. This view is populated by any input from the NLP view, but it also allows users to edit and directly type Anon-Lang statements. This enables the user to directly use Anon-Lang, or correct and refine Anon-Lang statements, which were automatically generated from the natural language input. The Anon-Lang tab also has input frames for Actors and Constraints. The Actors frame is also populated by the NLP tab, but can be directly edited and corrected here. Constraints specify how actors should be mapped to the structure. For example, if, in the above example, the user wants three clients, they would enter the "num client 3" constraint. Last, the UI offers a suggestion tab, which helps users with syntax as they type, for Anon-Lang and for the constraints.

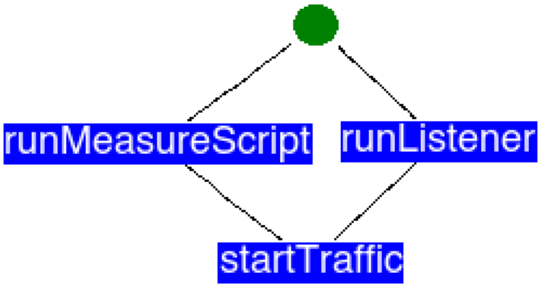


Fig. 3 *Screenshot a Event Dependency Graph.*

C. Output: Visualizing Experiment Behavior

To help users understand how the behavior they described in natural language or Anon-Lang is distilled into scriptable events, we have a Dependency Graph tab. All event dependency graphs are a directed, acyclic graph with a green node indicating the start of an experiment. All other nodes represent an event. Arcs represent that one event is triggered by the connected event's node.

Figure 3 shows the Dependency Graph for the previous example of a client starting traffic after other events have started.

D. Output: Visualizing Structure

Last, our prototype UI has a view to visualize the structure of an experiment. This view is a common visualization in current testbed UIs, but as discussed previously, unlike in current approaches, in our approach the structure is automatically derived from the behavior and constraints on how this behavior is mapped to the structure. This view in our UI helps users identify additional constraints they may need to specify to tune the automatically generated experiment structure.

III. Survey

Our preliminary evaluation survey was approved by our Institutional Review Board (IRB) as an exempt study. It consisted of four main parts:

1) **User Background**: short-answer and multiple choice questions aimed at understanding a user's experience with network testbeds.
2) **Structure-centric UI Task**: an experiment design task, which users were asked to perform through traditional means—using any methods or tools they typically use when working with Anon-Testbed
3) **Behavior-centric UI Task**: the same experiment design task, but users were asked to download and use our prototype UI after being given some background about our approach and how to use our UI.
4) **Feedback**: a mix of open-ended questions and multiple choice questions to gather feedback about comparing and contrasting the structure-centric and behavior-centric UI.

The order of the survey tasks for all respondents was the same. We were not concerned about a learning effect between the two UI tasks (parts 2 and 3 above), as the actual methods to get to the required output for each part were entirely different. In the structure-centric UI (part 2), respondents were asked to write the input for the experiment's structure and scripts for orchestrating behavior. When using behavior-centric UI (part 3), respondents were asked to input a description of the behavior for the experiment through natural language or Anon-Lang statements, or a combination of both. The only presented material, which is repeated between these two parts is the experiment description.

In the following sections, we provide a few more details about each of these survey parts.

A. User Background

Respondents were asked how long they had worked with network testbeds in general and were asked to give a list of the testbeds they had worked most with. To understand each respondent's experience level, we asked for an estimate of how many experiments they had run on network testbeds and we asked them to rate their own experience level as either "novice", "intermediate" or "expert". Since this preliminary study recruited participants from Anon-Testbed users, we also asked respondents about their experience with Anon-Testbed specifically.

B. Experiment Design Task

In both the structure-centric and behavior-centric experiment design task—parts 2 and 3 of our survey—respondents were asked to produce the programmatic encoding for the structure in Anon-Testbed's format and scripts to carry out the behavior for the same basic experiment. We describe our basic "experiment" as a list of straightforward criteria. We chose to describe the design task this way—rather than use a higher-level description, such as describing the experiment's hypothesis and goals—to avoid ambiguity. We wanted the task to be readily understood by novices and experts alike, and ultimately wanted a small enough task that volunteer participants could complete the task in a moderate amount of time. Further, we chose to specify very simple experiment behavior, which could be completed even by novice users.

We asked respondents to design a testbed experiment which met the following three criteria:

```
You have three machines: Machine X, Machine
 ↪  Y, and Machine Z.
Machine X sends a ping packet to Machine Y.
After Machine Y receives a ping packet from
 ↪  Machine X, Machine Y sends a ping
 ↪  packet to Machine Z.
```

The above experiment design task was constructed to be open ended enough, so that it could mimic the open ended nature of research experiments. We also wanted to signal to users that there may be many correct ways to complete the task we specified. For example, a respondent could specify that Machine X, Y and Z are all interconnected via a lan, or she could specify individual links between machines (e.g., X↔Y and Y↔Z).

1) Via Traditional Structure-Centric UI: Anon-Testbed's user interface, like other network testbeds, focuses on the structure of an experiment, not the behavior. To instantiate an experiment through the Anon-Testbed UI, users must write a programatic encoding—called an Anon-File—which informs Anon-Testbed, which resources are needed and how these resources need to be interconnected. We asked respondents to supply an Anon-File which would instantiate the needed topology of three connected physical machines (X, Y and Z) on Anon-Testbed, in addition to any extra resources needed for orchestration. Respondents were encouraged to use any additional tools necessary (such as a syntax checker, which Anon-Testbed's UI provides) to ensure their Anon-File would run correctly.

We also asked users to supply the scripts needed to orchestrate the behavior of the above experiment—a task the Anon-Testbed UI does not directly support. This task must usually be done manually by a user, or through some scripting tools like Jupyter. For example, orchestrating the two events (1. Machine X sends to Y, and then 2. Machine Y sends to Machine Z) could be done by kicking off each event from a separate orchestration node (as might be done when using ssh or tools like Ansible [1] to orchestrate), or through custom scripts on each of the machines and kicking off the events by starting the short chain of events on X. Participants could use the tools and languages they were most familiar with and could reuse any tools or scripting they had used in past experiments.

We asked respondents to take no more than twenty minutes to produce the Anon-File and the scripts needed to orchestrate the behavior of this experiment. Respondents were prompted to upload their Anon-File and all scripts and other artifacts to our survey server. We asked respondents to describe their approach to capturing the behavior of an experiment, the tools they used and the tools they typically use.

After producing the necessary traditional input, respondents were presented with the Anon-File they submitted, and asked to modify the file to scale up the experiment to use six machines, all performing the task Machine Y performed in the original basic experiment. This scaling was done to understand how well users naturally design their testbed inputs to facilitate easy scaling. We did not ask respondents to scale their scripts for orchestrating behavior, since we believed this would be too time consuming for most participants.

2) Via Our Behavior-Centric UI: After specifying the basic experiment through traditional means, we then asked respondents to download and try our prototype user interface. We again asked participants to take twenty minutes to design the basic experiment again—produce the programmatic encoding for the experiment structure and supply scripts to orchestrate the experiment—but this time through our UI. This meant that the users would specify experiment behavior, and our UI would automatically generate the structure and the scripts from user input. Respondents were able to check the experiment structure and behavior dependency graph derived from their inputs, and once satisfied with the results, they were instructed to save the Anon-File and orchestration script outputs.

Though our prototype UI produces a bash script to orchestrate events, the events in this script use user-supplied labels, and do not map to actual executables. Mapping of user labels to executables must be done through a "binding" stage, which we did not yet implement. Nonetheless, the orchestration script produced through our UI captures the necessary steps needed for maintaining event dependencies as described by the user.

C. User Feedback

Users were asked if they found the new UI useful, what features they found promising, which they did not and which they would like further developed.

IV. Survey Responses

We recruited survey participants from the Anon-Testbed user community. We selected this community as we are closely linked to Anon-Testbed testbed and collectively have over 30 years of experience with Anon-Testbed. All respondents had some experience with the Anon-Testbed and Anon-Testbed's UI. In total, twelve individuals responded, ranging from novice, with less than 1 month of experience, to experts with up to twelve years of experience. Nine of the twelve respondents completed the full survey.

In the next sections, we discuss the lessons we can draw from the survey responses. We first look at what we learned from the experiment artifacts produced by respondents through the structure-centric UI and through behavior-centric UI (parts 2 and 3 of our survey). We then discuss the qualitative responses gathered from user feedback.

A. Understanding event dependence is easy with the behavior-centric UI

The first thing we note is that our behavior dependency graph is handy when looking at experiment artifacts. Our approach enables a quicker understanding of event dependence and experiment behavior than the traditional approach of understanding an experiment through viewing scripts and documentation.

When reviewing how respondents designed their experiment behavior with the structure-centric UI, we noted that despite our basic experiment being simple, there was a broad range of experiment behavior design from the respondents. Specifically, all respondents managed to send pings between correct actors but they had varying levels of success when specifying behavior dependencies.

For example, Figure 4 shows an Ansible [1] orchestration script provided by a self-identified "expert" testbed user. Unlike the majority of collected orchestration scripts, this script explicitly takes into account the main event dependency in our simple experiment. Specifically, Machine Y sends a ping packet to Machine Z *only after* Machine Y actually receives a ping packet from Machine X. Figure 5 shows a different respondent's approach using a bash shell script (who self-identified as an "intermediate" testbed user). Under normal circumstances (no network errors or network slow down), this script will carry out the events as listed in our survey instructions, but there is no built-in dependence between Machine Y receiving a ping packet, and Machine Y sending a ping packet to Machine Z. If the ping packet from Machine X to Machine Y is lost, Machine Y will still send a ping packet to Machine Z. Such a missed dependence may lead to incorrect experiment behavior, and potentially incorrect research conclusions.

```
### Ansible file
---
- name:  start ping on serverX
  command: "nohup ping -c 50 10.0.0.2 &"

- name:  listen for ping on serverY
  command: "nohup sudo tcpdump -l src host
    ↪  10.0.0.1 dst host 10.0.0.2 icmp -c 1 &&
    ↪  ping -c 50 10.0.0.3"
  hosts: 10.0.0.2

- name:  listen for ping on serverZ
  command: "sudo tcpdump -l src host 10.0.0.1
    ↪  dst host 10.0.0.2 icmp -c 1 && echo
    ↪  success"
  hosts: 10.0.0.3
```

Fig. 4 *Ansible orchestration from a self-identified expert user for the survey experiment. This orchestration takes into account event dependence, as the ping packet from Machine Y to Machine Z is not sent until Machine Y receives a packet from Machine X.*

```
#!/bin/bash

ssh x "ping -c 1 y"
sleep 2
ssh y "ping -c 1 z"
```

Fig. 5 *Bash orchestration from a self-identified intermediate level user for the survey experiment. This orchestration does not correctly include event dependence as the ping packet from Y to Z is sent regardless of whether the ping from X to Y is received.*

Understanding a testbed experiment's behavior through reading orchestration scripts is fairly straightforward for something as simple as the basic experiment in our survey, so identifying the variants in respondents' approaches was tedious, but doable. However, scripts quickly become complicated for real testbed experiments, and identifying flaws in an experiment design can be difficult if not impossible through reviewing scripts.

In contrast, when reviewing how respondents designed their experiment behavior with our prototype UI, we simply compared the shapes of the behavior dependency graphs. Figure 6 shows the three variants of the experiment design for the respondents who completed designing our basic experiment through our UI. In Figure 6, we normalized the event names using these three labels for the events in our basic experiment:

1) **sendPingY**: Machine X sends a ping packet to Machine Y.
2) **YgetPing**: Machine Y receives the ping packet from X.
3) **sentPingZ**: Machine Y sends a ping packet to Machine Z.

In Figure 6, we can easily understand the three variants without digging into orchestration scripts.

B. The behavior-centric approach is less tedious

We hypothesize that having the users use structure-centric UI is error-prone and tedious. During the structure-centric UI part of the survey, users were encouraged to use any helper tools they typically used in experiment design, and while syntax checkers are available for the Anon-Testbed structure encoding, only one respondent used these tools. Four respondents reported they needed to refer to documentation to complete the structure encoding and five reported referring back to previously defined experiments to copy and modify portions of previously written encodings. All nine who completed a structure encoding the traditional way had to refer to previous examples or documentation or both, indicating that writing the encoding by hand is a tedious task.

When completing the behavior scripting task using traditional methods (i.e. writing scripts by hand, from scratch) only two respondents completed a fully orchestrated experiment. Three were close to a fully runnable experiment, and the remaining four had significant portions missing. In contrast, the automatically generated script from the new UI produced a fully-orchestrated skeletton of the script, with hooks to be supplied by the user for paths to executables.

C. The behavior-centric approach reduces errors

In comparing the structures input by users manually in task 2, with those generated automatically by our UI in task 3, all nine of the automatically generated ones passed a syntax check and produced feasible topologies for the supplied experiment. In contrast, of the manually produced encodings, four out of nine had syntax errors.

D. Users do not write to enable future scaling

When asked to scale their manually-created experiment structure, most respondents simply copied and pasted portions from their original structure, repeating portions over and over again. Two respondents, both self identifying as experts, instead used programmatic functions to scale (e.g. a “for loop”). For these two experts, additional scaling, or modifications to scaled portions would be fairly trivial. For other respondents though, more work would be required to correctly modify and/or scale their experiment structure easily. Ultimately, using a UI to generate structure encodings enables enforcing best practices, such as using programmatic functions to scale.

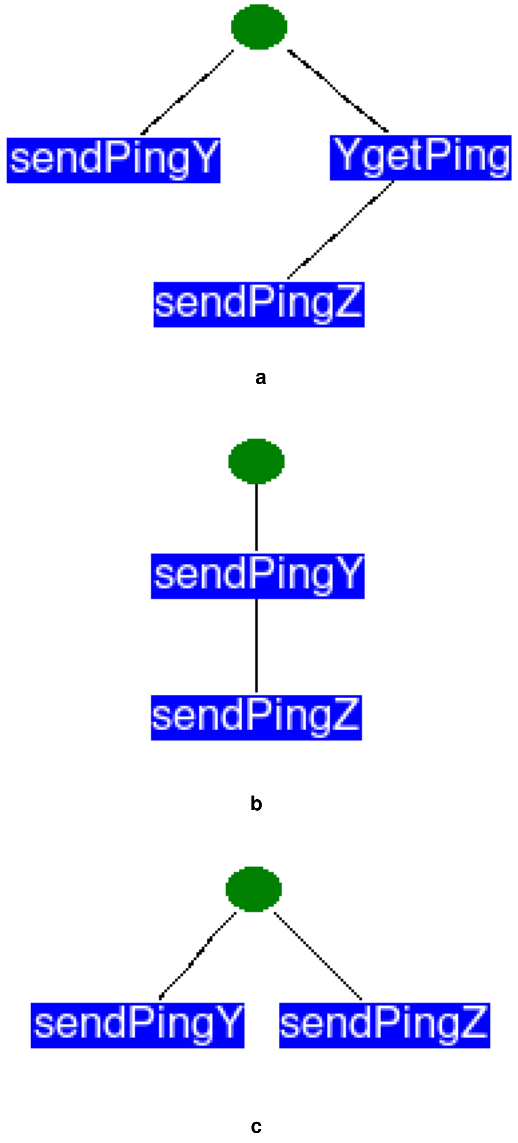


Fig. 6 *Three variations of behavior dependency graphs from respondents: (6a) Most robust and correct experiment behavior: Machine Y does not ping Machine Z until Y receives a ping (“YgetPing”). (6b) Less robust experiment behavior: the action of pinging Machine Z is dependent on having sent a ping to Machine Y, however, if Y never receives the sent ping, the action of pinging Z still happens, which does not fully follow the basic experiment criterion. (6c) Least robust behavior: the action of pinging Machine Z is not dependent on any other action. (Event labels from original respondents’ graphs were normalized to the same names.)*

E. Lessons from respondents' feedback

When asked, all respondents said that they typically design the structure of their experiment first, not the behavior. As one respondent commented,

> *It is structure first simply because all existing testbeds that I know require the structure first and behavior is up to the experimenter to create.*

This reaffirms that designing behavior first is not a testbed supported concept. Yet, the behavior of an experiment is vital to hypothesis testing, whereas structure just enables the experimentation. As another respondent noted,

> *Writing an experiment as a series of actions felt more intuitive.*

Respondents were asked whether they found the UI useful, with possible answers "not at all", "somewhat", "mostly" and "completely". Of 9 respondents, 2 answered 'completely', 4 answered 'mostly' and 3 answered 'somewhat'.

Respondents were asked which features of the UI they found useful. 5 listed NLP and 4 left this section blank. When asked which features of the UI were not helpful, 2 listed the behavior dependency graph and 7 left the section blank. That the behavior dependency graph was seen as not helpful was likely due to the simplicity of the survey experiment. Anecdotally, more complicated experiments benefit from viewing the interdependency of events. Respondents were asked which features were most promising for future versions of the tool. Four respondents mentioned improvements to the NLP feature, and five left this section blank. While NLP-based interfaces can sometimes lead to surprising behavior, and in this case the use of pronouns caused difficulties, all who responded saw NLP as a desirable feature to be improved.

Indeed, in a free text section of the survey, one respondent noted how the NLP approach encourages a description of behavior over structure:

> *"The NLP translation is interesting. Writing a story leads to an event-driven model instead of a structure driven model."*

Another respondent commented on the tool's value for inexperienced users:

> *"As a novice, I found the UI enabled much easier expression of my goals."*

V. Related Work

We believe we are the first to apply Natural Language Processing to network testbed experiment design. However, our goal of capturing behavior of experiments, and not just structure is shared with several other works, in particular, GPLMT [12] and the Experimentation Workbench [6].

The Experimentation Workbench by Eide et al. [6] proposed new constructs to the structure encoding to describe experiment behavior in addition to structure. This work was an inspiration for our work, but it lacks sophistication and user-friendliness that we hope to achieve. The Experimentation Workbench combines structure and behavior, while our approach keeps these separate by using actor roles as an abstraction, and constraints to tie actor roles to structure based only on important features. This makes our approach more easily scalable and more broadly useful, as the same experiment can be used with many different testbeds and topologies.

GPLMT [12] proposes a new experiment-description language, but produces descriptions which are much less readable by humans, more verbose and less structured than those written in our Anon-Lang. Ultimately, our hope for our Anon-Lang is to capture enough information about an experiment to either hand-craft or automate an experiment on multiple testbeds, thus improving repeatability and shareability.

VI. Discussion and Future Directions

Though we view the survey feedback as an indication we are on the right path, we have a long road ahead.

First, many respondents expressed our NLP UI as both useful and frustrating. Ideally, one could automatically generate a testbed experiment set up, complete with behavior orchestrated with the same description one would use for the experiment documentation. However, we expect we need additional tools to create a fully useful NLP interface. Specifically, without some form of supervised machine learning (ML), many details users can express in English, which apply in precise and specific ways to testbed experimentation will be lost. With ML, we potentially can derive meaning from context where otherwise traditional NLP techniques alone would leave interpretation too ambiguous to act on.

Second, the automatic generation of scripted behavior is non-trivial. Our UI can generate scripts based on anything that can be expressed in our Anon-Lang, however, we expect to have to improve our script generation and expand Anon-Lang to capture the complete behavior of more complex and nuanced experiments.

Last, our focus has been on developing for Anon-Testbed. We expect porting to other network testbeds to be straightforward, and involve only back-end development which will not affect our UI approach. Our future work will focus on testing this hypothesis.

Acknowledgment

The authors would like to thank all respondents to our survey. This material is based upon work supported by the National Science Foundation number 1835608.

References

[1] Ansible. https://www.ansible.com.

[2] DETERLab Education Site. https://education.deterlab.net/, 2019.

[3] Explosion AI. Industrial-strength natural language processing—in python. https://spacy.io/, 2018.

[4] Mark Berman, Jeffrey S. Chase, Lawrence Landweber, Akihiro Nakao, Max Ott, Dipankar Raychaudhuri, Robert Ricci, and Ivan Seskar. Geni: A federated testbed for innovative network experiments. *Computer Networks*, 61:5 – 23, 2014. Special issue on Future Internet Testbeds âĂŞ Part I.

[5] Test Resource Management Center. National cyber range. https://www.acq.osd.mil/dte-trmc/ncr.html, 2018.

[6] Eric Eide and Leigh Stoller. An experimentation workbench for replayable networking research. In *4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 07)*, Cambridge, MA, 2007. USENIX Association.

[7] Anonymized Author List. Anonymized title. In *Proceedings of Anonymized*, Anon Location, 201X. Anon Pub.

[8] University of Utah. Cloudlab. https://www.cloudlab.us, 2017.

[9] DETER Project. Deterlab web page. http://www.deterlab.net, 2018.

[10] FIRE Project. Future internet research and experimentation. https://www.ict-fire.eu/tag/testbed/, 2018.

[11] Orbit Project. Open-access research testbed for next-generation wireless networks (orbit). http://www.orbit-lab.org/, 2018.

[12] Matthias Wachs, Nadine Herold, Stephan-Alexander Posselt, Florian Dold, and Georg Carle. GPLMT: A lightweight experimentation and testbed management framework. In *PAM*, volume 9631 of *Lecture Notes in Computer Science*, pages 165–176. Springer, 2016.

[13] 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. In *Proceedings of the Operating System Design and Implementation*, pages 255–270, 2002.