Measuring Student Learning On Network Testbeds

Paul Lepe, Aashray Aggarwal, Jelena Mirkovic
USC Information Sciences Institute
Marina del Rey, CA, USA
{paullepe, jelenami, aashraya}@usc.edu

Jens Mache
Lewis & Clark College
Portland, OR, USA
jmache@lclark.edu

Richard Weiss, David Weinmann
Evergreen State College
Olympia, WA, USA
{rweiss, dweinman}@evergreen.edu

***Abstract*—Engaging students in practical, hands-on exercises on testbeds improves student learning and knowledge retention. However, testbeds may also present an obstacle to learning for students who are not familiar with the environment, or who lack the necessary background to complete their assignments.**

Our research investigates how students learn with testbeds. We instrument a default operating system on the DeterLab testbed and monitor the students' command line input and output, as they perform homework assignments. We use this data to evaluate students' progress, to detect when a student is struggling and to identify common problems.

I. Introduction

Student exposure to practical, hands-on exercises helps awaken interest and internalize concepts taught in networking, systems and cybersecurity classes. Practical exercises also teach students how to use relevant tools and help teach critical thinking. There are many public repositories of homework-type exercises (e.g., [1], [2]) for network testbeds.

Yet, learning with testbeds can also be an isolating experience for students, as assignments may be individual and require work outside of class. This may create a disconnect between students and teachers. Students that struggle with an assignment may spend a lot of time on it, often without making much progress. Students may hesitate to ask for help. Teachers on the other hand cannot directly monitor student progress and may be unaware of problems until the assignment is due and graded. By that time, it is too late to intervene.

We have developed a system, called ACSLE, shown in Figure 1 to automatically monitor and analyze student progress on network testbeds, during well-structured homework activities. We focus on assignments that require terminal-based interaction. ACSLE consists of the Monitor and the Analyzer components. The Monitor component collects students' terminal input and output on all machines in an experiment, and collates it into a single log file per student. The Analyzer processes students' log files and produces individual and summary reports showing students' progress on homework activities, and highlighting common mistakes. This information can help teachers assess how well the entire class is doing, identify students that struggle and identify common problems.

II. Monitoring Student Terminal Behavior

Many student exercises with network testbeds specify tasks that require students to interact with the testbed, often with multiple machines simultaneously, using SSH and a terminal. The Monitor component of ACSLE consists of a single

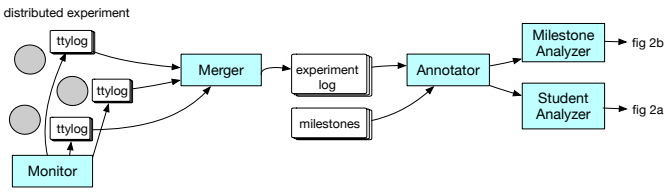


Fig. 1. ACSLE System Architecture.

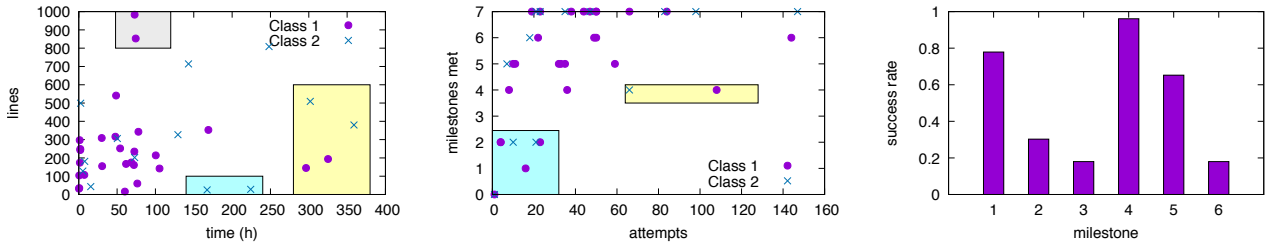
program, which records student input to the terminal, the output that they see, terminal identifier, username, and the time of each interaction. We investigated `script`, `history`, `snoopy` and `ttylog` for monitoring. Of these, only `ttylog` could log both the input and the output of a terminal, in all situations of interest. We modified `ttylog` to also record terminal identifier, username and time of each input. This additional information helps us when merging log files collected from multiple user sessions and/or multiple testbed machines.

III. Analyzing Student Input

The Analyzer component of ACSLE consists of four analysis programs: the Merger, the Annotator, the Student Analyzer and the Milestone Analyzer. We assume that the teacher can identify all experiments that relate to the given homework assignment, e.g., by the time of the experiment or its name. The Merger works on the set of experiments given. It first pulls and collates all the log files, based on the timestamp of each line. Such a merged log is then passed on to the Annotator.

The Annotator takes as input the merged log and a set of *milestones* for a given homework. A milestone is a specific, smaller learning task that a student is asked to complete. We assume that each task requires a specific terminal input or produces a specific terminal output. For example, we can create milestones for tasks where a student needs to run a specific command or create a given effect on the system, that produces a terminal output. In our future work, we will develop support for tasks that produce code, with a given functionality.

To use the Annotator, a teacher would have to encode homework tasks in our milestone format, shown in Figure 3, along with three sample milestones. A milestone consists of node, input and output fields. Each field can contain a wildcard, or have one or more values, separated by "|". The input and output fields can be specified using regular expressions, and should match the log on the specified node.



(a) Command lines vs time (b) Milestones met vs attempts (c) Milestone success rate

Fig. 2. Statistics produced by our Analyzer programs.

```
milestones: milestone+
 milestone: node ',' input ',' output <EOL>
      node: node_name
          | node_name '|' node
          | ''
     input: input_cmd
          | input_cmd '|' input
          | ''
    output: output_cmd
          | output_cmd '|' input
          | ''

node_name : name on the testbed
input_cmd : regular expression to match
            with cmd line
output_cmd : regular expression to match
             with cmd line

client,ip\s+route\s+get|ifconfig,*
client,tcpdump,^((?!denied).)*$
*,^find|^locate,/var/log/136intro-3.jpg
```

Fig. 3. EBNF for milestones, and several milestone examples.

The Annotator attempts to match each line of a student's log file with each milestone. If all three parts (node, input and output) match, the line is tagged as meeting that given milestone. Otherwise, if there is a partial or full match on the input, but the node and/or the output do not match fully, the line is tagged as a failed attempt to meet the given milestone. In all other cases the line is tagged as unrelated to a milestone.

The Student Analyzer produces a summary of each student's success in reaching the milestones as well as the number of failed attempts they made. It outputs the total time each student interacted with the testbed, the command line count, the number of milestones met and the number of attempts. These statistics can help teachers monitor the class' progress and identify students that need help.

The Milestone Analyzer summarizes how many students attempted and met each milestone, which can help identify challenging milestones and implement interventions.

Both programs use student usernames on the testbed. Only teachers can link such usernames to student identities, which protects students' privacy.

IV. Findings

Figure 2(a) shows the number of command lines vs time spent on the same homework assignment by two classes. The Figure is plotted using the output of the Student Analyzer. Most students take up to 4 days ($\approx$ 100 hours) to complete the assignment and produce up to 300 command lines. However, a few students in both classes take much longer – up to two weeks (yellow area in the Figure). Some students also spend a lot of time on the assignment but do not produce many lines of code (cyan area), and some produce almost double the average number of lines (gray area). The teacher could identify these students from our statistics and proactively work with them to ensure that learning goals are met.

Figure 2(b) shows the number of milestones met vs number of attempts, plotted using the output of the Student Analyzer. Most students met 5–7 milestones on this homework. Students that made fewer attempts met fewer milestones, as expected (cyan area in the Figure). Similarly, some students made many attempts but only met a few milestones (yellow area). The teacher could identify these student groups using our statistics and offer additional help.

Figure 2(c) shows the success rate per milestone, calculated as the ratio of successes to all attempts. This output is produced by the Milestone Analyzer for a different homework assignment. Milestones one and four were clearly easy to meet, followed by milestone five. However, students struggled with milestones two, three and six. Such information can help the teacher spend more time in class or produce more written guidelines focusing on the challenging tasks.

By observing the data tagged as attempts by the Annotator, we have been able to identify common mistake patterns. These include: wrong node, misspelled commands, and missing or unnecessary arguments. Currently, we are working on creating an extended Annotator to automatically detect and annotate these common patterns. The Analyzers can then summarize these patterns for the teachers to inform interventions.

V. Conclusions

Testbeds help students gain practical skills related to their learning goals, but they can also present an obstacle to learning. We have presented our approach to measure student learning on testbeds leveraging their command line activity on well-defined practical assignments. Products of our research can help teachers identify students who struggle and offer early help to improve their learning. Our work can also help teachers identify tasks that are difficult for a majority of students and offer more guidance for these.

VI. Acknowledgments

This material is based upon work supported by the National Science Foundation under Grants No. 1723705, 1723714 and 1723717.

References

[1] J. Mirkovic and T. Benzel. Teaching Cybersecurity with DeterLab. *EEE Security and Privacy Magazine*, 10(1):73–76, January/February 2012.

[2] Richard S. Weiss, Franklyn Turbak, Jens Mache, and Michael E. Locasto. Cybersecurity education and assessment in EDURange. *IEEE Security & Privacy*, 15(3), 2017.