PLDetect: A Testbed for Middlebox Detection Using PlanetLab

Paul Kirth[1] and Vahab Pournaghshband[2](✉)

[1] California State University, Northridge, Los Angeles, USA
paul.kirth.155@my.csu.edu
[2] University of San Francisco, San Francisco, USA
vahab.p@usfca.edu

Abstract. Designing, coordinating and deploying repeatable experiments outside of a fully controlled environment pose a serious challenge when conducting network research. In particular, it can be difficult to correctly schedule experiments that collect bulk data using a shared resource. To address this problem, we introduce PLDetect, a simple testbed built on top of PlanetLab which simplifies configuring, scheduling, and deploying large scale Internet experiments for evaluating middlebox detection methods.

Keywords: Middlebox · Detection method · Active measurements · Testbed · PlanetLab

1 Introduction

When conducting network research, it becomes clear that performing live Internet experiments presents a significant challenge: these networks use non-deterministic paths, are subject to a variety of competing administrative policies, and are mostly out of the researcher's control. After overcoming these obstacles, researchers must still provide sound methodologies and test infrastructure to validate their measurements. Overtime, these methodologies and testing practices should become commonplace, widely adopted, and, in some cases, standardized. In practice, researchers often do not have the opportunity to reuse the infrastructure or testing practices described in prior work, even if they wish to use the same methods and verification process. This can be due to several factors, such as the author's not releasing their test infrastructure, lack of sufficient detail in the original work, or an unmaintained codebase. Regardless of the reason, research efforts are becoming increasingly sophisticated, and their tools and methodologies must evolve to keep pace. Reinventing tools and infrastructure that perform similar function is a net loss for the research community, and even within industry the emergence of Dev-Ops has highlighted the need to reduce repeated efforts.

In the past, formalized research testbeds, such as PlanetLab, have provided researchers with superior methods of conducting network research. Previous work often sought to simplify deploying experiments using PlanetLab [2,5], but few, if any, are still maintained. As a result, using PlanetLab is still as challenging as when these works were initially conceived, a state that is further complicated by the nearly decade old Linux distribution on which PlanetLab is based [1]. This environment makes running modern software a challenge, and means that researchers must take infrastructure idiosyncrasies into account when designing their experiments. Even after addressing these limitations, a researcher is often forced to use a custom solution to deploy and manage their data collection activities across their PlanetLab slice.[1] We contend that few researchers should need to implement their own methods for interacting with a research tool as popular as PlanetLab, and furthermore that most should not need to create new infrastructure to deploy proven experimental configuration and validation experiments.

Consider, `NetPolice` [16], `PackSen` [15], and `ChkDiff` [11], three middlebox detection tools that employ similar detection methodologies and were validated using PlanetLab. In each work, the author spent non-trivial effort to implement and validate their detection infrastructure without the opportunity to easily reuse an existing test infrastructure. We point this issue out, not to highlight any fault on behalf of these authors, whose work is exemplary, but rather to illuminate a problem of culture and focus for the larger research community. Many other engineering fields have standardized testing methods for researchers to follow, and, even within Computer Science, test suites, such as SPEC [7], are widely used when validating research efforts. These formalized test suites and methodologies not only simplify the work of the researcher by providing previously vetted and validated tools, but also provide a common language and frame of reference for readers.

To address these challenges, we present `PLDetect`: a testbed for validating middlebox detection tools on PlanetLab that provides a vetted and well explored set of default configurations for conducting data collection and validation measurements for new middlebox detection tools. `PLDetect` is a simple framework that uses a combination of Python, Shell scripts, and configuration files to provide support for long duration experiments that must avoid overlapping measurements to maintain correctness. These can be important criteria for middlebox detection tools, as concurrent measurements can skew results for many experimental configurations. Furthermore, remote system administration tools, such as Ansible, do not yet properly manage temporal accesses with sufficient guarantees for research purposes. Throughout this work we will use the term *middlebox detection experiments* to refer to experiments designed to *validate* middlebox detection tools, rather than those that are trying to detect and map middlebox on the open Internet. This is an important distinction in `PLDetect`'s intended application and typical usage.

The testbed consists of two major components, a configuration utility, that can setup remote PlanetLab nodes, and a central dispatcher responsible for

[1] A slice is a set of allocated PlanetLab resources to which users can add PlanetLab nodes, and through which they are granted access to the nodes in their slice.

initiating experiments and ensuring that they did not exceed their allotted time. This dispatcher controls access to the test server, and will not initiate a new data collection event until the previous measurement has either successfully completed or has been terminated for exceeding its maximum allowed duration. It has, thus far, improved our ability to deploy new middlebox detection experiments across a PlanetLab slice and preserve the timing and scheduling constraints discussed in Sect. 2. `PLDetect` is freely available open source software, and its source code can be found on Github: https://github.com/ilovepi/PL-Detect.

The remainder of this work is organized as follows: in Sect. 2 we discuss the design, implementation, and use of `PLDetect`; Sect. 3 describes our evaluation of the testbed, our experimental configuration, and the results of our experiment; Sect. 4 discuses related work; Sect. 5 outlines the future direction of `PLDetect`; and Sect. 6 concludes this paper.

2 Testbed

`PLDetect` was created to allow researchers to quickly deploy middlebox detection experiments across their PlanetLab slices, and provide simple ways to configure the frequency and duration of data collection. In part, this tool was born out of necessity, because we found no prior solutions that addressed our particular needs for test administration, scheduling, and deployment. While simplicity is one of our driving goals, `PLDetect` must also support potentially complex experimental configurations and requirements. As a result, the bulk of this Section is focused on providing details on how `PLDetect` can be used to configure, deploy, and manage middlebox detection experiments across a PlanetLab slice.

2.1 Design

`PLDetect` is designed with three goals in mind: (1) to simplify the deployment of middlebox detection experiments across PlanetLab, (2) to prevent undesired overlap between experiments, and (3) to be easily extensible. To this end, a combination of Python and Shell scripts have been used to strike a balance between expressiveness and maintainability. We favor simple, direct code and interfaces to ease future development, and when possible we try to accomplish tasks using Shell scripts to better integrate with PlanetLab. In general, we use Shell scripts for managing tasks that are best handled directly via *NIX commands, and Python for more complex tasks and calculations.

2.2 Configuration

The `PLDetect` experimental testbed consists of a set of PlanetLab nodes, a set of Python and Shell scripts, and a set of configuration files. Configuring the testbed requires editing several configuration files prior to deploying the experiment across a PlanetLab slice. The user must configure:

1. Experiment folder: contains all the test scripts, program executable files, and data required to run the experiment on the remote node, including any logging scripts or scripts used to connect to remote databases.
2. `target_ip.conf`: a list of PlanetLab node IP addresses.
3. `install.sh`: an install script that installs the required packages on each PlanetLab node and copies the experimental folder.
4. `schedule.conf`: controls the experimental dates, period, and duration of measurements.
5. `start_experiment.sh`: a Shell script responsible for starting the experiment on the remote host.
6. `sqlinsert.conf`: configures the remote MySQL database.
7. `sqlinsert.sh`: a Shell script used to connect to a remote MySQL database and add experimental results.
8. `functions.sh`: contains a list of user defined functions used during the experimental execution, such as generating experimental parameters or recording metadata.

2.3 Node Configuration

Node Selection. The first file a user should configure is the `target_ip.conf` file. This file allows the user to select a set of PlanetLab nodes available to their PlanetLab slice for use in the experiment. The configuration format for this file is simple: each line of the file should contain only one of the IP addresses of the PlanetLab nodes that should be configured to run the experiment. This list of IP addresses will be consulted by the install script, `install.sh`, that is responsible for configuring the software environment on each PlanetLab node for the experiment to complete, and used in later phases to generate a schedule of data collection events for each PlanetLab node.

Experiment Folder. The next item to configure is the experiment folder, which should contain all of the executables and scripts needed to successfully run the experiment on the remote PlanetLab node. The use of this folder is an important design choice for the testbed. PlanetLab discourages its users from conducting compilation activities on their nodes, and recommends that the users compile their software outside of PlanetLab with an appropriate 32-bit Linux target. For most applications this is favorable because it allows them to compile their projects using a modern compiler with features that will likely not be found on a PlanetLab node. Additionally, most PlanetLab nodes have limited software selections unless the user is comfortable adding updated repositories to all of their PlanetLab nodes. Peterson et al. [9] advise users to create an experimental directory that can be copied to the PlanetLab node using `parallel-rsync`. By moving all of the necessary test materials into a single folder, updating each experimental node becomes a single invocation of `parallel-rsync`, and avoids difficulties with the often outdated software packages available on PlanetLab. The experimental directory can be updated and resynchronized using the install script or by using `parallel-rsync` directly.

2.4 Test Scheduling

After each node has been configured, the user will then select the testing schedule by editing the `schedule.conf` file and running a Python script to create a test schedule. The scheduling script, `ScheduleManager.py`, is used to generate a schedule for each data collection event, and does not allow these events to overlap. It works by consulting a tab delimited configuration file, `schedule.conf`. This file specifies the dates the experiment should run, the frequency of tests, and the commands to be executed at each time the experiment begins.

The output of the scheduling script is a file where each line associates the PlanetLab node IP, the time when a test should run, and the command or script to be run at that time. For example, `experiment.sh` might be scheduled to run at March 15, 2016 to `some.pl.node.edu`. The user can create the `experiment.sh` script that defines how the experiment should run, and have the local operating system invoke that script at the correct time. This script is invoked locally by default, but can be configured to run as a command on the targeted PlanetLab node instead.

By default, the `ScheduleManager.py` creates a schedule where a PlanetLab node takes one measurement per hour for every hour of the day for the entire duration (in days) of the experiment. These parameters can be configured directly in the invoking script, via the command line, or in the `schedule.conf` file. The schedules are created using the `at` command to accommodate irregular and singleton tasks rather than using other facilities, like `cron`.

2.5 Running Experiments

When it is time for an experiment to run, the scheduling machine will `ssh` into the PlanetLab node and invoke the command or script set in the scheduling configuration. The maximum duration for each test is used to set a timeout on each of the measurements to ensure that no two tests will overlap. Once the test program is invoked, the test scripts collect test metadata from the PlanetLab node and invoking context, and write the experimental results to a temporary results file.

2.6 Collecting Experimental Results

After the experiment is complete, the test script on the PlanetLab node logs into the remote MySQL database and records the results of the test along with test metadata by using the `sqlinsert.sh` and `sqlinsert.conf`. Note that the maximum test duration specified in `schedule.conf` should also include the time needed to report experimental results.

2.7 Verifying the Experimental Configuration

For testing purposes the user should provide the scheduling scripts with a single PlanetLab IP and ensure that their experiment and scripts are running as expected before deploying them across the entire list of PlanetLab nodes.

2.8 Limitations

Our testbed is not intended to be a general solution for deploying all types of network measurements on PlanetLab, but is instead focused on correctly scheduling independent measurements at a single site. `PLDetect` was built for a specific class of experimental configuration, in which measurements are conducted at a site that the researcher controls. For middlebox detection, this will often be a switch, or another routing element, that the researcher is trying to detect, and whose behavior they can alter. In this configuration (Fig. 1) each PlanetLab node connects to a target IP behind the middlebox, and data is collected from each of the configured PlanetLab nodes. We further assume that each data collection event should be completely independent from one another, so that traffic from one measurement does not interfere with another. This does not preclude an experiment from using several sources of traffic for each measurement, but will require additional configuration steps when creating the test scripts described in this Section. Additionally, we do not support managing access to multiple test sites, or coordinating multiple experiments at the same site.

3 Evaluation

In this Section, we present the details of our evaluation: the middlebox detection method used, our experimental configuration, and our results. One of `PLDetect`'s goals is to offer a premade validation platform for researchers conducting middlebox detection experiments, and the evaluation experiment discussed in this Section was designed with this use case in mind. Moreover, this experiment provides an example of the testbed's expected use in conducting experiments to validate middlebox detection tools. The goal of these measurements is to infer the behavior of the middlebox based on the detection methodology presented in Sect. 3.1. While we discuss the detection method and describe a typical experiment for evaluating the compression detection tool used in our experiments, we want to stress that the analysis of the data collected is not significant to the evaluation of the testbed itself. Rather, this data serves to demonstrate `PLDetect`'s ability to correctly configure, schedule, and execute data collection activities in the desired manner, regardless of the success of the detection method.

3.1 Detection

To validate `PLDetect` we used a tool for detecting network compression based on the *relative discrimination technique* [15] to infer the presence of a compression link. The tool detects network compression based on significant differences in loss and delay between a *base* flow composed of low entropy data (all bytes are zeros), and a *discrimination* flow composed of high entropy data (all bytes are random). This is congruent with prior work in Pournaghshband et al. [10] that leveraged the difference in processing times for high and low entropy packets for detecting compression links.

3.2 Experimental Configuration

To validate the compression detection tool, we designed an experiment to take independent measurements from several different sites, each trying to connect a test server that was under our control. This allowed us to position the test server behind a middlebox capable of operating as either a compression link or a normal switch so that we could enable and disable the middlebox behavior (compression) as we desired. This topology appears in several prior works aimed at detecting middleboxes [10,15], and is an important part of `PLDetect`'s expected configuration for supporting validation efforts. Figure 1 illustrates an example of our experimental configuration, and highlights the portion of it that are fully within our control.

The testbed in this experiment consisted of five PlanetLab nodes, a middlebox capable of operating as a compression link, and a test server. Each of the PlanetLab nodes tried to correctly infer the presence of a compression link along the transmission path to the test server using the detection method discussed in Sect. 3.1. We selected a geographically distributed set of PlanetLab nodes to capture a wide variety of transmission paths and network conditions. Each node in the PlanetLab slice was configured using `PLDetect`'s install procedure, as outlined in the Sect. 2.3. Our experiment executed one measurement every hour for each of the five PlanetLab nodes in our slice over a period of 30 h. Each measurement had a maximum duration of one minute, and recorded the number of packet losses and transmission latency for each stream, along with a complete packet trace of the measurement. These results are then recorded to the remote database for future analysis. Part of our validation strategy uses bulk data to alleviate any undue influence normal fluctuations in network conditions might have on our measurements.

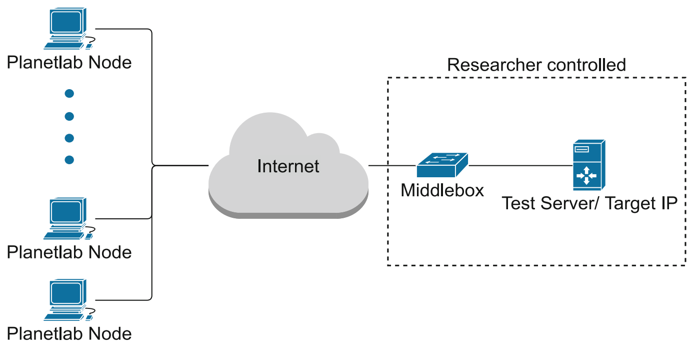

Fig. 1. PLDetect network topology.

3.3 Experimental Results

We collected data using five planet lab nodes, taking measurements once per hour at each site, over a period of 30 h. The entire deployment was configured, scheduled, and recorded using `PLDetect`. Figure 2 contains a plot of the relative differences in transmission times between the base and discrimination flows. We can see that each PlanetLab node recorded its measurements once per hour over the entire 30 h duration, and that the detection tool was able to record significant differences in the loss rates between the base and discrimination flows. By contrast, without a compression link there is virtually no difference between the loss rates or transmission times of the two flows. This large, persistent difference meets the criteria of our detection methodology, and shows the tool's ability to identify the compression link in our configuration. We again stress that what these measurements show is not important, rather they appear here to highlight our ability to manage an experiment and schedule our data collection with `PLDetect` using the procedures outlined in Sect. 2.

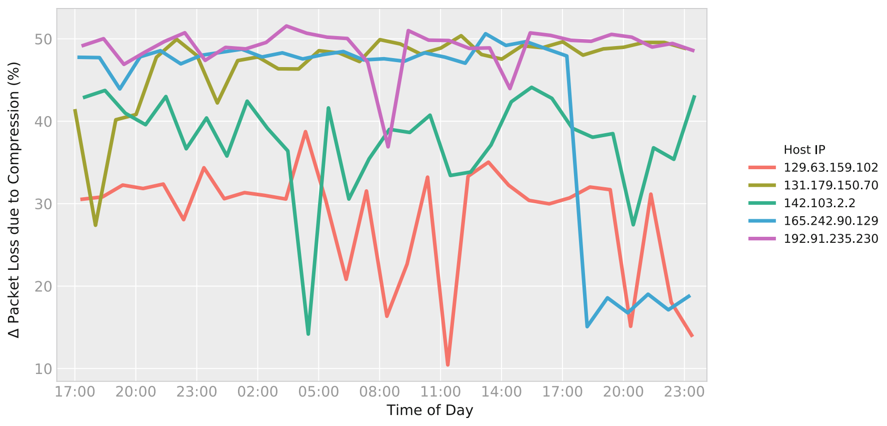

Fig. 2. Percentage of discrimination packets lost when compression is enabled

4 Related Work

`PLDetect` is a network testbed built on top of PlanetLab, and in that regard, it is not unique. Many past works, like Stork [5] and Plush [2], were built to simplify how users create and deploy their experiments on PlanetLab, and while these tools have proved useful, many of them are either no longer maintained or were not made public. `PLDetect` separates itself form these prior efforts in its narrow focus on providing a simple solution for deploying a specific type of measurement, and is made from freely available tools that have thus far performed well on PlanetLab slices. For other platforms, such as GENI [4], the GEE [3] provides an abstraction layer in which the authors facilitate the rapid prototyping, management, and administration of GENI experiments. Our solution does not try to provide the same level of abstraction to PlanetLab as GEE applies to GENI. GENI's resources also are more friendly towards virtualization and the use of containerized environments which plays a key role in GEE's IaaS design. Our tools, by contrast, are much simpler: each script or configuration file plays a limited role in how experiments are conducted and maintained, and we provide relatively primitive forms of configuration management.

Recently, Wachs et al. [14] presented GPLMT, a general purpose testbed framework for managing experiments on PlanetLab. This work does an excellent job of generalizing experimental configuration and abstracting the complicated aspects of managing large scale experiments. However, `PLDetect` and GPLMT have fundamentally different goals and philosophies. GPLMT is a general purpose tool, suitable for managing many types of experiment, while `PLDetect` is focused on supporting middlebox detection with a well known methodology and experimental configuration in addition to managing the experiment. Where `PLDetect` distinguishes itself from existing testbeds, is in its focus on supporting one class of network experiment very well.

While it may seem odd to contrast `PLDetect` with works that are not testbeds, most of its design is focused on supporting the testing of detection tools and methodologies. In particular, we adapt significant portions of our configuration and methods from Pournaghshband et al. [10]. Other significant works in this area, `Packsen` [15], `Tracebox` [6], `NANO` [12], and `POPI` [8], each provided insight and inspiration for the methodology and validation techniques that PLDetect tries to encapsulate.

5 Future Work

Ansible [13] is a popular tool for administering remote systems, and provides two important benefits to its users: simple YAML based configuration and an easy way to share and distribute automation tasks. While YAML based configuration is a superior solution to our simple configuration files, Ansible's main draw is the large repository of community made solutions available through Ansible-Galaxy. As one of `PLDetect`'s design goals is to simplify configuration and automation, we believe that allowing users to easily reuse these existing solutions is the right choice. However, incorporating Ansible into our design will require some care to ensure that our scheduling requirements can still be met. We can integrate Ansible by modifying how we generate scheduling commands, to instead use Ansible playbooks, or by modifying `PLDetect`'s design to provide a new queuing phase for measurements.

As GENI [4] matures, it is becoming an important platform for conducting network research. Future versions of `PLDetect` should incorporate support for the GENI platform, and integration with the more modern services available outside of PlanetLab, such as support for virtual machines and docker containers.

6 Conclusion

In this paper we presented `PLDetect`: a testbed for middlebox detection, discussed its design and intended use, and demonstrated its usefulness as a research tool through an example experiment. To our knowledge it is the only research testbed focused on middlebox detection, and we believe its focus on providing a specialized network research infrastructure can provide a valuable blueprint for others to follow.

References

1. Releases/8/schedule - FedoraProject. https://fedoraproject.org/wiki/Releases/8/Schedule
2. Albrecht, J., Tuttle, C., Snoeren, A.C., Vahdat, A.: PlanetLab application management using plush. ACM SIGOPS Oper. Syst. Rev. **40**(1), 33–40 (2006)
3. Bavier, A., et al.: The GENI experiment engine. ACM
4. Berman, M., et al.: GENI: a federated testbed for innovative network experiments. Comput. Netw. **61**, 5–23 (2014)
5. Cappos, J., et al.: Stork: package management for distributed VM environments. In: LISA, vol. 7, pp. 1–16
6. Detal, G., Hesmans, B., Bonaventure, O., Vanaubel, Y., Donnet, B.: Revealing middlebox interference with tracebox. In: Proceedings of the 2013 Conference on Internet Measurement Conference, IMC 2013, pp. 1–8. ACM (2013)
7. Henning, J.L.: SPEC CPU2006 benchmark descriptions. SIGARCH Comput. Archit. News **34**(4), 1–17 (2006). https://doi.org/10.1145/1186736.1186737
8. Lu, G., Chen, Y., Birrer, S., Bustamante, F., Cheung, C.Y., Li, X.: End-to-end inference of router packet forwarding priority. In: IEEE INFOCOM 2007, 26th IEEE International Conference on Computer Communications, pp. 1784–1792 (2007)
9. Peterson, L.L., Bavier, A.C.: Using PlanetLab for network research: Myths, realities, and best practices. In: WORLDS
10. Pournaghshband, V., Afanasyev, A., Reiher, P.: End-to-end detection of compression of traffic flows by intermediaries. In: 2014 IEEE Network Operations and Management Symposium (NOMS), pp. 1–8 (2014)
11. Ravaioli, R., Urvoy-Keller, G., Barakat, C.: Towards a general solution for detecting traffic differentiation at the internet access. In: 2015 27th International Teletraffic Congress (ITC 27), pp. 1–9 (2015)
12. Tariq, M.B., Motiwala, M., Feamster, N., Ammar, M.: Detecting network neutrality violations with causal inference. In: Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies, CoNEXT 2009, pp. 289–300. ACM (2009)
13. Venezia, P.: Review: ansible orchestration is a veteran Unix admin's dream; ansible and AnsibleWorks AWX bring simplicity and power to Linux and Unix server automation. http://www.infoworld.com/article/2612397/data-center/review-ansible-orchestration-is-a-veteran-unix-admin-s-dream.html
14. Wachs, M., Herold, N., Posselt, S.-A., Dold, F., Carle, G.: GPLMT: a lightweight experimentation and testbed management framework. In: Karagiannis, T., Dimitropoulos, X. (eds.) PAM 2016. LNCS, vol. 9631, pp. 165–176. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30505-9_13

15. Weinsberg, U., Soule, A., Massoulie, L.: Inferring traffic shaping and policy parameters using end host measurements. In: 2011 Proceedings IEEE INFOCOM, pp. 151–155 (2011)
16. Zhang, Y., Mao, Z.M., Zhang, M.: Detecting traffic differentiation in backbone ISPs with NetPolice. In: Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference, IMC 2009, pp. 103–115. ACM (2009)