

Netset: Automating Network Performance Evaluation

Puneet Arora
Department of Computer Science
North Carolina State University
parora@ncsu.edu

Yaogong Wang
Department of Computer Science
North Carolina State University
ywang15@ncsu.edu

Injong Rhee
Department of Computer Science
North Carolina State University
rhee@ncsu.edu

ABSTRACT

Performance measurement and comparison is integral to almost any kind of networking research. However, we currently lack a general framework under which network-based tests can be carried out. Thus researchers tend to design their tests based on their own ingenuity, making it difficult to repeat these tests and compare testing results from different sources. This has seriously stymied innovation in the networking research community. New ideas are difficult to be verified and then widely deployed since different tests produce different or even conflicting results. In this paper, we propose Netset, a software framework to automate network performance evaluation. It makes network-based tests portable, repeatable, convenient and gives realistic and comparable results. It leverages existent models and tools but is also extensible. We have implemented a proof-of-concept prototype of Netset and used it to carry out some TCP tests. Our initial experiences with Netset demonstrate its advantages. We believe that a standard benchmark tool like Netset will facilitate a more objective evaluation of new network protocols and benefit the whole networking research community.

Categories and Subject Descriptors

C.2.0 [Computer-Communication Networks]: General; C.4 [Performance of Systems]

General Terms

Measurement, Performance, Experimentation

Keywords

Automation Tool, Protocol Benchmarking, Network Measurement, Performance Evaluation

1. INTRODUCTION

Performance evaluation is an essential part of networking research. Currently, there are three major methods to quantitatively evaluate the performance of computer networks, namely theoretic analysis, simulation and experimentation. Theoretic analysis is limited in the sense that modern networks are so complex that their behaviors are usually not analytically tractable. Or we have

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PFLDNeT'09, May 21–22, 2009, Akihabara, Tokyo, Japan.

to make crude assumptions to get a tractable solution which may lose realism.

Simulation is widely accepted by the networking research community as a powerful tool for performance evaluation. Tools like ns-2 [1], OPNET [2] etc. are used in all kinds of networking research. Simulation is convenient, portable and repeatable since you don't need to implement the real system and everything is deterministic programs. However, simulation is still modeling real systems and wrong modeling assumptions may lead to completely unrealistic results [3]. Experimentation gives more realistic benchmarks as it is carried out on real testbeds. However, performance measurement of real networks is non-trivial and the results are heavily dependent on the tester's configuration of the testbed. Quite often we see different or even conflicting measurement results from different researchers and it's hard to compare them since they are derived under different testbeds and scenarios.

Thus, it's our view that the current process of inventing, endorsing and adopting new networking protocols is stymied by the lack of standard performance evaluation procedures and tools for evaluating the performance of new protocols. Researchers tend to rely on their own ingenuity in creating testing environments and often spend significant efforts on them. But it's difficult for third party reviewers to validate and duplicate these environments. Also researchers tend to blame each other's test environments for any discrepancy in performance.

We believe that a generally accepted test suite along with an automated evaluation tool can ease this situation. [4] is an effort to achieve the first element for TCP testing. Similar generally accepted test cases may be necessary for other fields of networking research as well. In this paper we are trying to deal with the second element: a framework to automate network performance evaluation. Our proposed tool Netset tries to integrate various network testing resources under one roof. It combines the benefits of simulation and experimentation while eliminating their deficiencies. In summary, Netset has the following features:

1. Realistic: the test is run on real testbeds rather than simulated.
2. Convenient: Netset automates topology generation, parameter setting, traffic setup, as well as data analysis. It provides a user interface nearly as convenient as simulation tools but runs the tests on real testbeds and gives realistic results.
3. Portable: a test can be exported to a structured file (we use XML in our current implementation) and rerun on another testbed.

4. Repeatable: as stated above, tests are fully described by portable files. We can import the files into any testbed and repeat the test. This doesn't mean exactly the same results will come out in each execution of the test (which is the case for simulation). But it ensures that both tests are run under the same scenario.

5. Comparable: since all tests are portable and repeatable, it's much easier to compare the testing results from different parties.

6. Extensible: Netset is designed to be extensible both to the lower layer testbeds and the upper layer tests. The framework can be deployed on different testbeds: either a local testbed or a testbed provided by a third party like Emulab [5]. It could also leverage existent testing tools and accommodate different kinds of tests: either evaluating a new protocol or a new network device driver etc.

The rest of the paper is organized as follow. Section 2 explains in detail the problems with current network performance measurement methods and discusses related work. Section 3 gives our solution: Netset, with detailed information about its design and architecture. We show our proof-of-concept implementation in Section 4 with a sample test and come to the conclusion in Section 5.

2. MOTIVATION

In this section, we first state the problems lying in current network performance measurement methods by a typical example. Then we discuss related research work in this area and their deficiencies.

2.1 Problem Statement

Evaluating a new congestion control algorithm is a typical example of network performance evaluation. The authors have plenty of experiences about this during the evaluation of CUBIC TCP [6]. Although we realize the problem during the evaluation of congestion control protocols, we believe that it applies to many other network-based tests as well.

To test the performance of a new congestion control algorithm, we typically carry out the following steps:

1. Set up the physical network topology on which the test is going to run by plugging wires among nodes.
2. Deploy test software and tools on each node
3. Set related network parameters like bottleneck link bandwidth, delay, buffer size etc.
4. Generate representative and realistic traffic mix
5. Specify parameters of interest that need to be monitored and collected during the test
6. Execute the deployed tools and software to run the test and collect data
7. Analyze test results and generate reports for comparison

These procedures are non-trivial since the performance is sensitive to many factors and a single mistake may lead to unreliable results (e.g. lack of background traffic [7]). But it can be seen that there is a pattern in these steps. It would be redundant for every researcher to repeat them. Also, due to the abundant

details contained in a test, it's very difficult to reproduce the test just based on the descriptions in a published paper. Thus we make lots of duplicate efforts but still cannot get comparable results.

Netset tries to solve this problem by providing a software framework which accommodates a wide range of network-based tests. Under this common model, we can conveniently define portable tests to be run on different testbeds and compare their results.

2.2 Related Work

There are plenty of projects such as Emulab, WAN-in-Lab [8], PlanetLab [9] that offer testbeds for researchers to run network-based tests. They differ in how the testbed is implemented. Emulab provides emulated testbed using software emulation technologies like DummyNet or Netem. PlanetLab offers more realistic networks by deploying hundreds of nodes over the planet and connect them through overlay. WAN-in-Lab is somewhere between the two, providing real networking testbed in a controlled environment.

However, all these projects mainly aim at providing the testbed rather than automating the entire test. As shown in the previous section, network-based tests require more than just hardware testbeds. It also involves "soft" components such as test scenarios, workloads generation, performance metrics monitoring etc. These "soft" components along with the underlying hardware components constitute what we call a test environment. Netset automates the whole test environment setup rather than provides only a testbed. In fact, Netset could make use of these projects for lower layer testbed setup, as will be shown later.

Seawind [10] is a wireless network emulator that tries to automate the whole test rather than provide only testbed. However, fundamentally Seawind is primarily a wireless link emulator although it incorporates some additional features like workloads generation and output analysis. Its emulator origin limits its generality as a framework for network performance evaluation. Netset, on the other hand, contains no testing or emulation functionality in itself. Its pure purpose is to "host" tests and make them portable. Therefore, it could be general enough to host many different kinds of tests and make use of many existent testing tools.

There exist many studies on network modeling [3, 11, 12, 13] as well as many tools for network performance evaluation such as Iperf, Uperf[14], TcpProbe[15] and NetLogger[16]. However, we need an automation tool that leverages these studies and tools by integrating these tools and models and build a general framework under which all kinds of network-based tests can be conducted. That's the main contribution of Netset.

3. NETSET

3.1 Topology Layout

Two key concepts in Netset are Test and Testbed. Test is a collection of all abstract information about a test environment, including topology, required software and tools, command sequences to be executed, data or metrics of interest as well as data interpreter. It's similar to a TCL script in ns-2. Note that the topology information in a Test is abstract and is not mapped into physical topology until the Test is imported and executed on a Testbed.

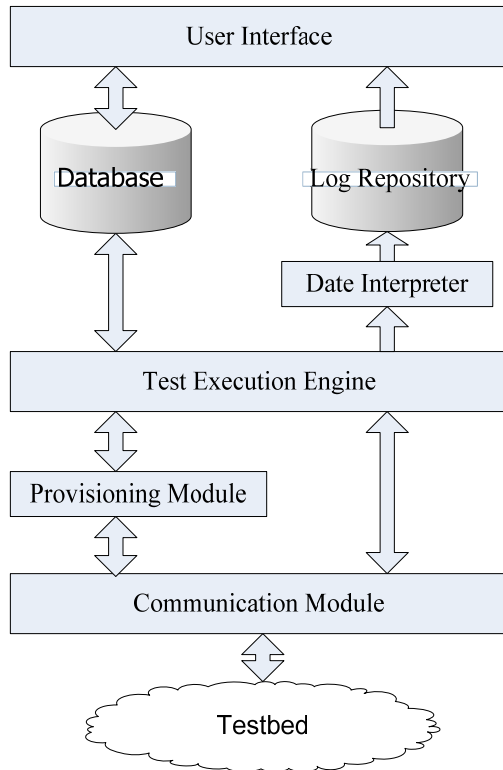


Figure 1. System Architecture of Netset.

Testbed is a description of the physical resources available in the local or remote infrastructure. Each node in a Testbed is a physical machine (host, switch or router etc.) and each connection is a physical link.

When a Test is imported into a Testbed, the provisioning module of Netset will map the logical topology defined in the Test into physical topology in the Testbed. Netset achieves this by first mapping the nodes in the Test into physical machines in the Testbed according to their attributes (For example, a node with 3 data links must be mapped to a physical machine with more than 3 network interfaces). Then Netset tries to set up the connections between the nodes as described by the Test. This can be done via different methods depending on the facilities available in the Testbed. If L1 switches are available, we can configure electrical/physical connectivity by software and achieve the desired topology. If only L2 switches are available, we may use VLAN and subnet to achieve the defined topology. Note that this mapping is not always possible since the physical resources available may not satisfy the needs of the Test. In such cases, Netset just returns and notifies the user that the Test cannot be executed on the current Testbed. Also note that the testbed is not necessarily local facilities. It may be testbeds provided by third party projects like Emulab. In that case, we can just make use of Emulab to achieve the desired topology as long as we define a module to translate between their interfaces and the interfaces defined in Netset.

The focus of Netset is not about how to achieve automatic reconfigurable network topology. There may be many possible methods to accomplish it such as [8]. Also, it's possible to design

very intelligent yet efficient algorithms to do this mapping. But it's not a major designing goal of Netset. We currently use some heuristic approach to implement it and leave the algorithm refinement as future work. The key is that we propose a framework under which portable tests can be defined and executed on different testbeds. How to map logical topology into physical topology could be left to lower layer testbeds, where we can make use of existent projects like WAN-in-Lab.

The separation of Test and Testbed is the pivot of Netset. It is this separation that makes the tests portable, repeatable and comparable. This separation also divides the work of networking researchers and system administrators. The researchers are only concerned with defining the Test and then submit it to certain Testbed to get the results. The system administrators are just responsible for maintaining the testbed and provide a uniform interface to researchers through Netset.

3.2 Test Execution

The overall system architecture of Netset is shown in Figure 1. As explained earlier, provisioning module is responsible for mapping the abstract topology in the Test to the physical topology in the Testbed. For this mapping, the provisioning module needs to communicate with various devices in the testbed and configure them properly. This communication is also necessary for later test execution since we need to upload necessary test software or tools, launch commands and retrieve logs. Therefore, we have a separate communication module for Netset to communicate with the testbed. There may be different ways to access different nodes. It can be telnet, FTP, SSH or SNMP etc. Communication module is responsible for handling these differences and provides a uniform interface to other modules of Netset.

The test execution engine is the main component of Netset. Upon launch, it carries out the different phases of test execution. These include provisioning and layout phase, file transfer phase, configuration phase, running phase and cleanup phase.

During provisioning and layout phase, the test execution engine calls provisioning module to map the abstract topology into physical topology. After that it deploys the required test software and tools on each node through communication module. When all these preparations are done, the test execution engine begins the test which contains three phases: configuration, running and cleanup. These three phases are similar in the sense that they all involve execution of command sequences on different nodes. The purpose of the configuration command sequence is to execute initialization commands for the test like installing kernel modules, configuring router delay and loss characteristics etc. The purpose of the running command sequence is to launch the core test processes. The cleanup command sequence executes clean-up commands like unloading kernel modules, deleting temporary files etc.

Command sequence is an abstraction for executing commands in Netset. A command sequence contains a group of commands. For each command, we specify the command line to be executed and the target node on which the command should be executed. The reason why we let users gain access to command level details is for the sake of extensibility. We keep Netset at a lower level of semantics. It does not understand what traffic means. It only understands commands which when executed on a node result in

traffic. This makes Netset extremely extensible since users could deploy his own software and then run commands to execute it. The alternative to keep users at traffic level will improve ease of use but decrease extensibility. For example, the user may simply define a TCP flow between two nodes rather than specifying concrete commands on each node. Netset then automatically translates it into Iperf commands. This makes the user's job easier but obstructs the user from using customized tools for TCP traffic other than Iperf. To fill this gap, we make a compromise in Netset. For common operations (e.g. start a TCP flow), we write library modules which automatically generate commands for them. But we still keep the command execution functionality so that users can make their own choice. We apply the same philosophy for data analysis. Netset cannot directly understand the output of each command since it could be any arbitrary command defined by the user. Instead we have a component in Netset (data interpreter), which reads the output of specific commands, parses them and generates human readable statistics.

3.3 User Interface

Netset is a database-driven system in the sense that all Test and Testbed information as well as output logs are stored in the databases. When the user defines a Test or a Testbed, the definitions are stored in the database. When a Test is executed, the test execution engine reads the information stored in the database and begins testing. When the test is finished, the output data are fed into the data interpreter defined by the user. The data interpreter will parse the output data and store the results into the database. Then the user could access these results by reading from the log repository.

How to access the databases is implementation-dependent. But one key feature we need to achieve is that Tests are portable, i.e. we can export them from the database into files and later on import the files to execute the Test on another Testbed.

Another concern here may be security. Since the network stack is typically implemented in the kernel, network-based tests usually involve actions that require root privilege. Allowing remote users root privilege to the testbed may be undesired. Our solution is that researchers who defined the Tests do not have direct access to the testbed. They just define the test they want to run and submit it to Netset. Only Netset could access the testbeds directly with root privilege. Since Netset would typically be co-located with the testbed, we eliminate the cross-domain security problem. Netset could check the Test submitted by the researchers before running it. If it's secure, the test is executed on the testbed and the results are fed back to the researcher.

4. IMPLEMENTATION

We implemented a proof-of-concept prototype of Netset and carried out some CUBIC TCP tests upon it. Our initial experiences show that it's more convenient, portable and extensible than our original script-based testing.

The current implementation is programmed mainly in Python and makes use of Django [17] – a Python-based Web development framework. We use MySQL as the backend database management system and provide both a commandline-based and a Web-based user interface.

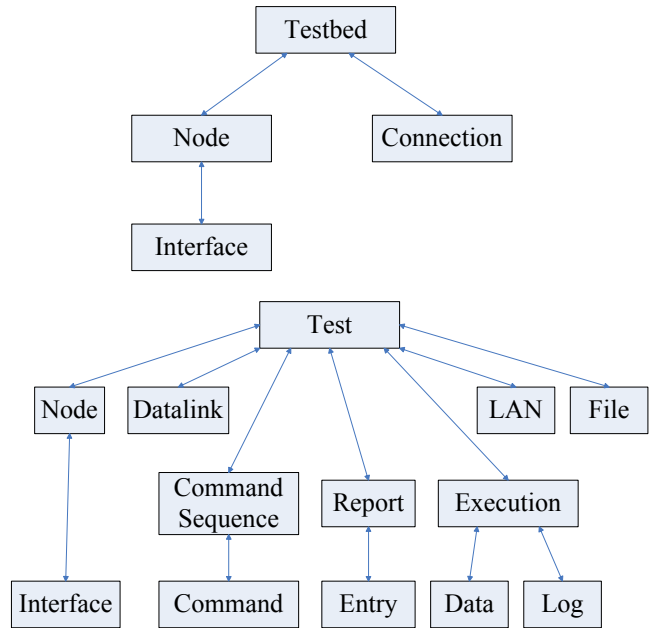


Figure 2. Data Models of Netset.

Netset defines a hierarchy of data models (Figure 2). Each data model is a Python class that is a descendant of Django's Model class that provides Object Relational Mapping (ORM) services. The ORM-enabled Model class provides an abstraction layer between the client application and the database, enabling the use of different databases without modifying client applications.

As shown in the figure, Test and Testbed are the two main top-level models. Test represents abstract topology information by Node, Datalink and LAN models. In addition to topology, Test also contains Files to be deployed on the Nodes and Command Sequences to be executed during the test. Finally, every Test has an associated Report model, consisting of Report Entries. The Report specifies the data interpreters and document templates used to generate a test execution report. A test when executed produces an Execution that stores Execution Log and Execution Data. The log stores timestamps and information about state transitions that the test went through during the execution. The execution data contains the set of values generated during the execution and is used to build the Report.

The Testbed model is used to represent physical testbed on which tests can be run. It consists of Nodes and Connections. A testbed node corresponds to a physical node. It has one or more network Interfaces. Note that this is different from a test node, which is just an abstraction. A Connection in the testbed represents an existing physical path between two testbed node Interfaces.

Note that this is just a brief introduction to these models. Each model has many more attributes and we are still refining them. The difficulty is the balance of comparability and portability. If we define very detailed attributes of a model, then the testing results from different Testbeds are more comparable. But it may make the Test less portable. For example, a Test may require a node to be a Linux box with kernel 2.6.29.1. This accurate specification could eliminate performance discrepancies caused by different kernels but we are unable to run this Test if we don't

have a node with that specific kernel version in our testbed. A less detailed specification like just a Linux box or a Linux box with 2.6 kernel will make the Test more portable but may lead to different testing results due to kernel version discrepancy.

4.1 A Sample Test

In this section, we illustrate how Netset works by a step-by-step sample. Due to space limit, we could not show a complete, realistic TCP test where we have plenty of background traffics and their delays are randomized etc. But this tailored example is sufficient to give you a flavor of Netset.

1. A local testbed is available and its physical topology is shown in Figure 3. All servers are connected to a switch either with one link or multiple links. Some of them are Linux boxes while others are FreeBSD or Solaris. All the information about this testbed is already stored in the database.

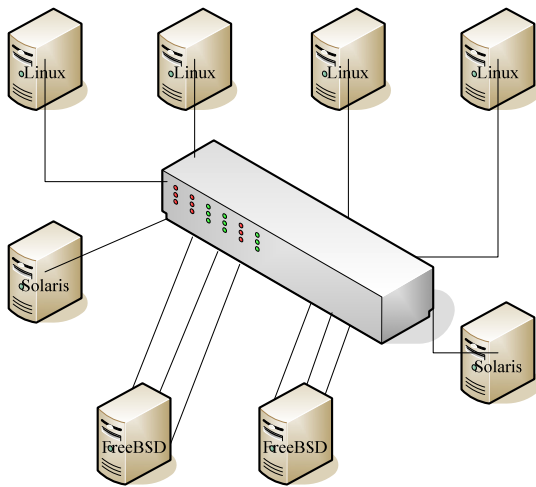


Figure 3. Testbed Topology.

2. We create a new Test, add Nodes and Interfaces to it, and then specify test topology by specifying Datalinks between Interface pairs and/or grouping Interfaces in LANs. The topology specification part of the Test XML file is as follow:

```
<netset>
  <test name='sample' description='a sample test'>
    <parameter key='testbed' value='local_tb' />

    <node name='s1' os='linux' />
    <node name='s2' os='linux' />
    <node name='d1' os='freebsd' />
    <node name='d2' os='freebsd' />
    <node name='r1' os='linux' />
    <node name='r2' os='linux' />

    <IPTopology>
      s1 -- d1
      s2 -- d1
      d1 -- d2
      d2 -- r1
      d2 -- r2
    </IPTopology>
  </test>
</netset>
```

We first specify which Testbed to use since there may be multiple testbeds available. Then we define Nodes and the topology. We don't define Interfaces for each Node explicitly here since Netset could infer them from the topology definition of this simple Test. For more complex tests, explicit definition for Interfaces would be necessary. During the provisioning and layout phase, Netset will map the Nodes to physical servers in the testbed based on their operating system and number of Interfaces. Then Netset uses subnet and VLAN to implement the dumbbell topology defined by the user.

3. We also need to define Files to be deployed on each Node, Command Sequences to be executed and Reports to parse output data. This part is a little tedious and we give a tailored version here.

```
<file name='iperf' targetNode='s1' />
<file name='iperf' targetNode='r1' />
<file name='iperf' targetNode='s2' />
<file name='iperf' targetNode='r2' />

<sequence name='config'>
  <action name='flow1_cfg' module='dummysnet'
  action='dummyPipe' node='d1' from='s1.if0' to='r1.if0' inFlow='1'
  bw='400Mbit/s' delay='200ms' />
  <action name='flow2_cfg' module='dummysnet'
  action='dummyPipe' node='d1' from='s2.if0' to='r2.if0' inFlow='1'
  bw='400Mbit/s' delay='100ms' />
</sequence>

<sequence name='main'>
  <action name='flow1' module='iperf' action='iperf'
  source='s1' dest='r1.if0' time='200' />
  <command name='sleep_25s' type='sleep' cmd='25s' />
  <action name='flow2' module='iperf' action='iperf'
  source='s2' dest='r2.if0' time='175' />
</sequence>

<report name='th_ratio' />
```

We first define which files need to be deployed on which nodes. Then we define command sequences which set up the dummysnet pipes and run Iperf. We also specify data interpreters to parse Iperf output and compute the throughputs of the two flows.

4. After the Test is defined, we can execute the Test through the Web interface and monitor the progress of the test (Figure 4). We could know which phase the execution is currently at, what command sequences are executed right now and even instantaneous statistics.

5. Finally, when the test is finished, we can directly view well-formatted results parsed by the Report (Figure 5). It can even automatically plot graphs if specified by the user.

Note that the Testbed information must be made known to Netset before any Test can be executed. This configuration may be complex for testbeds with plenty of facilities, but it's a one-time effort. Once configured, many different Tests can be executed on this Testbed.

Through this sample, we could see that Netset makes test setup more convenient. At least you don't need to worry about physical topology setup. More importantly, it makes tests portable. Once

you defined a Test, you can distribute it to other researchers via exported files. They can repeat your test and possibly define new Tests based on your Test.

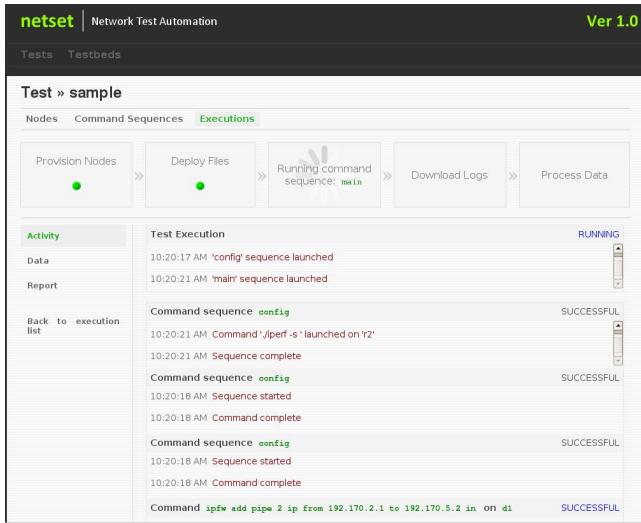


Figure 4. Netset Screenshot while Test Execution.

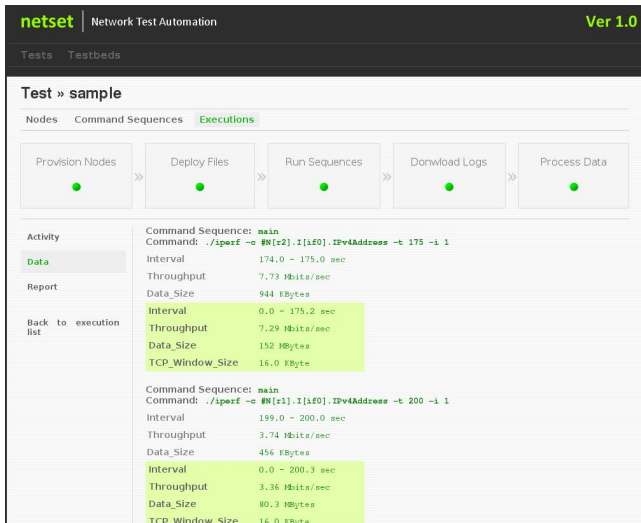


Figure 5. Netset Screenshot after Test Execution.

5. CONCLUSION

We present the design of Netset, a software framework to automate network performance evaluation. It aims to bring existing network testing resources under one umbrella for integrated and extensible use. Netset automates all aspects of network-based tests, from topology layout to report generation and gives realistic and comparable testing results. We implement a proof-of-concept prototype of Netset to demonstrate its portability, repeatability and convenience.

For future work, we plan to refine the implementation of Netset and make it available for public use. We believe that a standard benchmark tool like Netset will facilitate a more objective evaluation of new network protocols and benefit the whole networking research community.

6. REFERENCES

- [1] The Network Simulator ns-2: <http://www.isi.edu/nsnam/ns/>
- [2] OPNET: <http://www.opnet.com/>
- [3] Floyd, S. and Kohler, E. Internet Research Needs Better Models. ACM SIGCOMM Computer Communication Review. 33, 1 (Jan 2003), 29-34. DOI= <http://doi.acm.org/10.1145/774763.774767>
- [4] Andrew, L., Marcondes, C., Floyd, S., Dunn, L., Guillier, R., Wang, G., Eggert, L., Ha, S. and Rhee, I. Towards a Common TCP Evaluation Suite. In PFLDNet'08 (Manchester, UK, Mar 2008).
- [5] Emulab - Network Emulation Testbed: <http://www.emulab.net/>
- [6] Rhee, I. and Xu, L. CUBIC: A New TCP-Friendly High-Speed TCP Variant. In PFLDNet'05 (Lyon, France, Feb 2005).
- [7] Ha, S., Le, L., Rhee, I. and Xu, L. Impact of Background Traffic on Performance of High-speed TCP Variant Protocols. Computer Networks. 51, 7 (May 2007), 1748–1762. DOI= <http://dx.doi.org/10.1016/j.comnet.2006.11.005>
- [8] Lee, G., Andrew, L., Tang, A. and Low, S. WAN-in-Lab: Motivation, Deployment and Experiments. In PFLDNet'07 (Marina Del Rey, USA, Feb 2007).
- [9] Chun, B., Culler, D., Roscoe, T., Bavier, A. Peterson, L., Wawrzoniak, M. and Bowman, M. PlanetLab: an Overlay Testbed for Broad-coverage Services. ACM SIGCOMM Computer Communication Review. 33, 3 (Jul 2003), 3-12. DOI= <http://doi.acm.org/10.1145/956993.956995>
- [10] Kojo, M., Gurtov, A., Manner, J., Sarolahti, P., Alanko, T. and Raatikainen, K. Seawind: A Wireless Network Emulator. In Proceedings of 11th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (Aachen, Germany, Sep 2001).
- [11] Zegura, E., Calvert, K. and Donahoo, M. A Quantitative Comparison of Graph-based Models for Internet Topology. IEEE/ACM Transactions on Networking. 5, 6 (Dec 1997), 770-783. DOI= <http://doi.acm.org/10.1109/90.650138>
- [12] Tangmunarunkit, H., Govindan, R., Jamin, S., Shenker, S. and Willinger, W. Network Topology Generators: Degree-based vs. Structural. ACM SIGCOMM Computer Communication Review. 32, 4 (Oct 2002), 147-159. DOI= <http://doi.acm.org/10.1145/964725.633040>
- [13] Hernández-Campos, F., Smith, F. and Jeffay, K. How Real Can Synthetic Network Traffic Be? (poster abstract). In ACM SIGCOMM'04 (Portland, USA, Aug 2004).
- [14] Uperf: <http://www.uperf.org/>
- [15] TcpProbe: <http://www.linuxfoundation.org/en/Net:TcpProbe>
- [16] Gunter, D., Tierney, B., Crowley, B., Holding, M. and Lee, J. NetLogger: A Toolkit for Distributed System Performance Analysis. In IEEE MASCOTS 2000 (San Francisco, USA, Aug 2000).
- [17] Django Project: www.djangoproject.com/