Adding Emulation to PlanetLab Nodes*

Marta Carbone
Dip. di Ingegneria dell'Informazione
Università di Pisa, Italy
marta.carbone@iet.unipi.it

Luigi Rizzo
Dip. di Ingegneria dell'Informazione
Università di Pisa, Italy
rizzo@iet.unipi.it

ABSTRACT

Network testbeds have become very popular to support research on network protocols and distributed applications. When it comes to reproduce network behaviour, testbeds range between two extremes: use a fully emulated network, as in EmuLab, which yields very reproducible experiments but might be a poor representation of reality; or communicate through the real Internet, as in PlanetLab, resulting in more realistic but less reproducible scenarios. Having both features available in the same testbed, and being able to choose and mix the two at will, is clearly interesting for researchers.

In this paper we present an extension of the PlanetLab testbed to add emulation capabilities to all nodes. The work is centered around the Dummynet emulator, which we ported to Linux as part of this project.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques

General Terms

Experimentation, Measurement, Performance

1. INTRODUCTION

In recent years many testbeds have been deployed to support research on network protocols and distributed applications. The motivation behind these deployments is to make available to researchers a system that, for its size and features, would be not affordable otherwise. The actual target of each testbed varies. Some of them, such as EmuLab [1], are focused on providing a very reproducible environment, in terms of node capacity or network resources. Other testbeds address specific aspects, such as the study of wireless networks (ORBIT [2]), or routing protocols (VINI [3]), or mesh and sensor networks. Finally, testbeds such as PlanetLab [4] are more oriented towards providing a realistic snapshot of the real Internet.

The main contribution presented in this paper is an extension that we developed to add emulation capabilities to PlanetLab. With this work, PlanetLab users gain the ability to configure, independently of each other, the actual features of the network, thus making it possible to run experiments in richer and more varied settings.

*The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n.224263 – Onelab2.

CoNEXT'09 Student Workshop, December 1, 2009, Rome, Italy.
ACM 978-1-60558-751-6/09/12.

2. ARCHITECTURE

PlanetLab is a network testbed made by *nodes*, contributed by participating organizations and distributed across the Internet. Nodes are managed by a central authority called *PLC*, which stores user credentials and other management information. On each node, users can create a virtual server called *sliver*, which provides resource isolation and gives them the illusion of a dedicated system.

The goal of our emulation system is let users define, from within their experiments, emulated links with configurable features, and pass their own network traffic through these links. The architecture of our system is shown in Figure 1. The emulation is implemented by a Linux version of the Dummynet link emulator. The configuration of the emulator is done through the *vsys* subsystem described next.

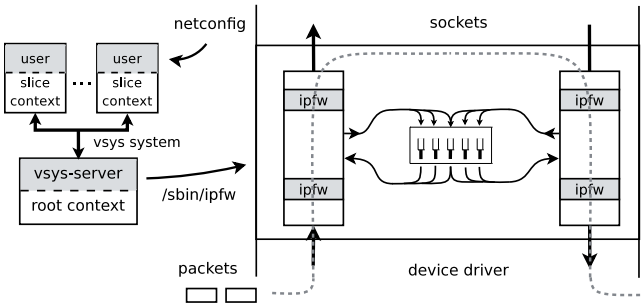

Figure 1: The interaction between slivers, vsys and the Dummynet emulator.

2.1 The vsys subsystem

Slivers run in a so-called *slice context*, where they have limited access to the node's resources. Privileged operations can be run through the *vsys* subsystem, which is made of two parts: a *vsys-frontend*, running in *slice context*, and a *vsys-backend*, running in *root context*, where full access to the node is allowed. A sliver must call a vsys frontend to execute privileged commands; the vsys makes appropriate permission checks and, if allowed, calls the corresponding vsys backend to serve the request.

In our emulator, the frontend, called `netconfig`, is in charge of collecting the parameters to configure an emu-

lated link. The backend, called `ipfw-be`, does the parameter checking and possibly configures the emulator or updates the existing configuration. In turn, the backend talks to the Dummynet kernel module and its control program `/sbin/ipfw` to configure and perform the emulation.

2.2 The Linux port of Dummynet

The actual emulator is a Linux port of the Dummynet[5] emulator and its associated packet filter, *ipfw*, originally developed on FreeBSD. The porting work consisted in the identification and design of suitable replacements for the kernel subsystems used by Dummynet.

A first issue was to hook the classifier and the emulator into the network stack. Our two requirements are to intercept upstream and downstream traffic (Figure 1), and to reinject packets back into the stack after a suitable delay.

On Linux, we have used the *netfilter* mechanism to pass all packets to *ipfw*. For each packet we create a stripped-down `mbuf` structure to adapt the in-kernel packet representation between FreeBSD and the Linux equivalent (`sk_buff`). The mapping does not require expensive data copies nor modifies the packet itself. On return, the external `mbuf` descriptor is simply destroyed, and the packet is reinjected by netfilter completely unmodified into the network stack.

The adaptation of other system services has been done by writing wrappers around Linux functions so that we could export a FreeBSD-compatible API. This includes locking (in the Linux port, we have mapped mutexes and rwlocks to `spin_lock_bh()`); the memory allocator and timer support (Linux offers a similar functionality, with only a different interface and naming); and the infrastructure to support loadable kernel modules.

The result of this porting work is a single kernel module, `ipfw_mod.ko`, containing both the packet classifier and the emulation module, and a control program, `/sbin/ipfw`, which provides the user interface. The current code has been tested on a wide range of Linux versions, including OpenWRT. The full code is available at: `http://info.iet.unipi.it/~luigi/dummynet/` .

3. USAGE

Installing and using the emulator is extremely simple. Vsys and netfilter are already part of PlanetLab nodes, and we have created a new package with the `ipfw_mod.ko` module and associated utilities, which is installed automatically when a PlanetLab node is created or upgraded.

Users must install in their sliver a package containing the frontend program, `netconfig`, which they can run to configure an emulated link even from within an experiment. `netconfig` requires the TCP or UDP port number for the traffic to intercept, plus any other additional parameters related to emulation (bandwidth, delays and so on), e.g.:

```
./netconfig -p <port number> <parameters>
```

This results in the backend program being run, which in turn performs the desired configuration. The following is an example of the *ipfw* rules generated when configuring emulation on port 5678:

```
ipfw pipe 5678 config <parameters>
ipfw add 5678 pipe 5678 src-ip ME src-port 5678 uid S // EXP
ipfw add 5678 pipe 5678 dst-ip ME dst-port 5678 uid S
```

As we can see, we configure two rules, one per direction. One of the rules also stores, in the comment field, the expire time (*EXP*) for the rule, which is used to delete old configurations. Additionally, to avoid interference between users of different slivers, we exploit the *VNET* system. VNET provides virtualized network access on nodes, and among other things it tags packet with the identifier of the slice the packet belongs to. When installing rules in the emulator, the backend also adds a check on the slice identifier (S in the listing) so that slivers can only capture their own traffic.

4. ACCURACY AND PERFORMANCE

The basic accuracy of our emulator equals the resolution of the timer tick, which on PlanetLab nodes is 1ms. While this may seem rather coarse, note that it corresponds to the duration of a maximum-size Ethernet frame at 12 Mbit/s, not to mention that timing uncertainties due to the scheduling of slivers are possibly an order of magnitude higher. Much better resolution can be easily achieved by increasing the resolution of the timer tick (we have successfully run kernels with 25..100μs ticks), or relying on High Precision Timers. Also note that the granularity only affects the error on the timing of individual events (packet transmissions or receptions), but does not accumulate over multiple events.

Another factor that influences the accuracy of the emulation is the CPU load on the nodes. To measure this, we tested the ping response time for a node with 1 and 100 *ipfw* rules, in three different load conditions: IDLE (no activity other than standard system tasks), USER (several CPU-bound user processes consuming all available CPU cycles), and KERNEL (several CPU-intensive kernel tasks). Results (average and standard deviations, in μs) are below:

Experiment	IDLE	USER	KERNEL
IPFW-1	28.1 / 2.82	28.2 / 1.36	55.1 / 2.62
IPFW-100	36.2 / 1.82	36.3 / 1.78	71.0 / 2.60

Compared to the IDLE case, USER load has almost no impact, but KERNEL load can easily add an extra 25-35μs to the packet processing times, so it is rather pointless to increase the timer resolution beyond this limit.

We have also measured the per-packet processing cost of the emulator, which is the sum of two components: a "classification" time, to check the packet against the rules in the classifier; and the "emulation" time, to pass the packet through the emulator's data structures. We measured 100–200ns per rule in the classifier, and 600ns for the "emulation" time, even with 1000 active pipes. These times should be compared with the overall packet processing times, which on our test system were around 3μs. As a result, the presence of the emulator should not impact significantly the network performance of a node.

5. REFERENCES

[1] Emulab. http://www.emulab.net/.

[2] Orbit. http://www.orbit-lab.org/.

[3] Vini. http://www.vini-veritas.net/.

[4] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, 2003.

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