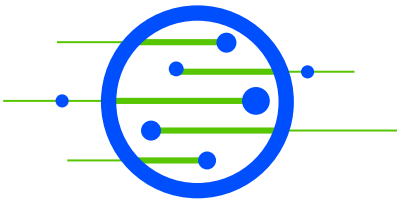

PlanetLab Version 3.0

Marc Fiuczynski, Mark Huang, Aaron Klingaman, Steve Muir,
Larry Peterson, Mike Wawrzoniak
Princeton University

PDN–04–023
August 2004 (updated January 2005)

Status: Ongoing Draft.

PlanetLab: Version 3.0

Marc Fiuczynski Mark Huang Aaron Klingaman Steve Muir Larry Peterson
Mike Wawrzoniak

January 27, 2005

1 Introduction

This document describes the design, interfaces, and rollout of Version 3.0 of the PlanetLab software. The new features included in Version 3.0 are largely a response to discussions on the architecture mailing list over the last year, with many of the issues captured in PDN-04-17. Specifically, Version 3.0 is designed to:

- Support alternative brokerage services, allowing them to acquire resources and create slices with minimal involvement from PLC.
- Support alternative environment services, allowing them to download and manage software packages on behalf of client slices.
- Support stronger resource isolation, including the ability to manipulate resource assignments from both the GUI and the programmatic interface.
- Uncouple the PLC database from the interface used to create and manipulate slices, in anticipation of future decentralization of PlanetLab. We have also taken this opportunity to re-organize the database to better match our current model of roles, sites, services, and so on.
- Support stock ping and tcpdump binaries, meaning that the new implementation of safe raw sockets is compatible with the standard socket API. Network virtualization is also enhanced to support the requirements of new services (e.g., network telescope and GRE tunneling).
- Address the long-term issue of kernel maintenance, including our ability to track new kernel versions with extensive local modifications. We also need to resolve the limitations of depending on the RedHat distribution.

Version 3.0 includes the following major components:

- PLC

  - reorganized database
  - programmatic interface
  - Web/GUI
- Node Manager
  - support for stronger resource isolation
  - support for local sliver instantiation
  - support for privileged operations (Proper)
- Kernel
  - Linux 2.6
  - Vservers 1.9
  - CKRM E15
  - VNET 1.0
- Installer
  - BootCD
  - package management

Note that while some of the new capabilities mentioned above are supported by a single component, in most cases, support for a given feature spans multiple components. For example, support for stronger resource isolation requires changes to the database (to record resource allocations for a given slice), the programmatic interface (to allow PIs and administrators to set a given slice's allocation), the node manager (to control the kernel's allocators and schedulers), and the kernel (to enforce the allocations). Moreover, Version 3.0 defines very little about the policies, and primarily involves providing an flexible framework that can be evolved over time.

2 PLC

Several software components running at PlanetLab Central (PLC) collectively provide an interface to the PlanetLab infrastructure. Few new capabilities, per se, have been added to this software, but it has been significantly reorganized to better support the evolution of the rest of the system.

At the heart of PLC is a database, which is organized into two components: (1) administrative information primarily about sites, users, and nodes; and (2) information about slices running on the infrastructure. For the purpose of this discussion, we treat these as two separate databases, called the AdminDB and the SliceDB, respectively, although in practice they are as a single database.

A major shift in Version 3.0 is that all manipulation of the PLC database will be done through a programmatic interface. The GUI running at the PLC website will use this interface, as will those sites and services that wish to directly manipulate the database.

The interface can be viewed as having two largely independent pieces, corresponding to the AdminDB and SliceDB. Generally speaking, operations that manipulate the SliceDB are available to all PlanetLab users, while operations that manipulate the AdminDB are available to users whose roles allow them. For example, users can modify their own accounts, while PIs can modify any user at their site.

2.1 Slice API

Starting with Version 2.0, users have been able to create slices using a programmatic API (XMLRPC based) rather than through the GUI available at the PlanetLab web site. Among other things, this allows alternative slice creation services to evolve.

Version 3.0 of this interface adds two new features. The first is support for requesting a *ticket* that can later be redeemed at individual PlanetLab nodes to create a local sliver. This allows slice creation services to directly interact with node managers rather than have to create slices centrally through PLC. (PLC still supports a default slice creation mechanism.) The second feature allows users and site PIs to specify additional slice attributes such as resources (e.g., CPU shares, link bandwidth, disk quotas, and so on) or slice-specific initialization scripts to be bound to particular slices. The interface itself is designed to be extensible, with support for new resource types being incrementally added as the rest of the system permits (e.g., as soon as appropriate schedulers and allocators are available on individual nodes).

Note that to enable these features, the programmatic interface had to first be re-implemented. We elected to implement it in python; the original version was written in php. The new implementation also implements a backward compatibility module with the v2.0 dynamic slice API (not discussed here).

Information about the specific slice operations can be found on-line at https://wiki.planet-lab.org/bin/view/Planetlab/Pl

2.2 Admin API

The AdminDB has grown in a rather ad hoc way since PlanetLab was first deployed. With Version 3.0, we substantially reorganized this database to reflect the way PlanetLab is actually administered. Major changes include:

- Support for people being members of multiple sites and having multiple keys.
- Support for *logical node groups*, making it easier to handle custom configurations for particular subsets of nodes; e.g., an Internet2 group, an "alpha node" group, and a "beta node" group.
- Support for organizations, which can span a of groups of sites, rather than just a single site.
- Support for tracking deleted accounts, as well as nodes and sites that are no longer in use.

Moreover, the AdminDB is now accessible via a programmatic interface. The methods used to access and manipulate the AdminDB are currently described on-line at http://www.planet-lab.org/apis/admin_api.html.

2.3 Web GUI

The website interface now interacts with the AdminDB and SliceDB via the programmatic interface. This has several benefits, including:

- All policy and business level logic is now in one place. Before, multiple pages added added nodes to the system, resulting in multiple implementation points for the policy governing new nodes.
- Separating the API from the database makes it easier to modify the underlying tables in the future, since only those methods they effect will need to be updated.
- By separating the interface from the implementation, we introduce the possibility of creating a distributed database, and other decentralized approaches to managing PlanetLab become possible.

3 Node Manager

The node manager is extended in three primary ways. First, it provides richer control over resource isolation. Isolation between slices is enforce by kernel-level schedulers and allocators, but the node manager controls these schedulers, effectively implementing PlanetLab's resource allocation policy. This policy—along with the role played by brokerage services—is not yet defined. The changes primarily provide a framework for evolving the way in which resources are allocated.

Second, the node manager exports a set of local operations that can be used to create and delete slivers on that node. The node manger will continue to provide a default slice creation service, as in Version 2.0, but the new local operations—when used in conjunction with the ticket granting mechanism defined in Section **??**—will enable alternative slice creation services.

Third, Version 3.0 of the node manager includes a facility that allows infrastructure services to manipulate and control "client" slices. This facility, called Proper (PRivileged OPERation), is initially motivated by the desire to support multiple environment services, but we anticipate that it can be used to enable many other infrastructure services [1].

3.1 Resource Isolation

The node manager plays a significant (behind-the-scenes) role in resource isolation. Loosely speaking, it periodically retrieves slices.xml from PLC, creates the necessary local slivers, interprets the resource allocations for those slivers specified by slices.xml, and manipulates the various kernel schedulers and allocators to affect those allocations. This is an implementation detail, and is discussed further in the context of the underlying kernel scheduling mechanisms (see Section 4).

3.2 Local Sliver Creation

In Version 3.0, users (or services operating on their behalf) can retrieve a ticket to create a slice from PLC (think of the ticket as a signed "row" of slices.xml, corresponding to a particular slice), and then instantiate the slice

one sliver at a time by directly calling the node manager on each individual node. Details about the API are available on-line at https://wiki.planet-lab.org/bin/view/Planetlab/NodeManagerApi.

3.3 Privileged Operations (Proper)

Enabling unbundled management means that certain infrastructure slices need a way to perform privileged operations on other slices. For example, an environment service might need to access the filesystem of another slice, and modify the immutable bits. As another example, certain slices need to read, and possibly write, files in the root filesystem in order to access system global information (e.g., mapping from slice ID to slice name). As a final example, a monitoring slice might need to open a true raw socket.

Proper (PRrivileged OPERations) is a daemon running in the root context that exports such capabilities to slices [1]. It is intended to provide fine-grained access control, so that admininstrators can restrict, for instance, a single slice to reading a single file in the root filesystem.

Operation	Description
open_file(name, options)	Open a restricted file
set_file_flags(name, flags)	Change file flags
get_file_flags(name, flags)	Get file flags
mount_dir(source, target, options)	Mount one directory onto another
unmount(source)	Unmount a previously-mounted directory
exec(context, cmd, args)	Execute a command in the context (slice) given
wait(childid)	Wait for an executed command to terminate
create_socket(type, address)	Create a restricted socket
bind_socket(socket, address)	Bind a restricted socket

Table 1: Operations Supported by Proper 0.3

The current implementation of Proper supports a small number of operations, as shown in Table 1—essentially those that service developers have asked be made available. We do not believe this list to be complete, but the fundamental nature of the operations supported suggests that it should suffice to support the majority of applications.

Further information on these operations is provided on-line at https://wiki.planet-lab.org/bin/view/ProperApi. The header file, prop.h, and shared library needed to access these operations from C applications are included in the proper-devel and proper-libs RPMs respectively.

4 Kernel

The Version 3.0 kernel includes several changes: (1) upgrading the Linux kernel; (2) replacing the RedHat distribution with Fedora Core; (3) upgrading to the latest version of the vserver kernel patch; and (4) replacing PLKMOD with two sub-packages (CKRM [2] to provide a framework for scheduling resources and VNET

to support network virtualization). The primary motivation for these changes include better tracking of Linux kernel enhancements and bug fixes, improved performance isolation between slices (vservers), and improved support for network virtualization.

4.1 Linux Kernel and Distribution

PlanetLab V2.0 production nodes have been based on a Redhat 9 distribution running a stock (kernel.org) 2.4.22 linux kernel combined with a number of patches that address bugs, security fixes, and support for vservers and PLKMOD.

PlanetLab V3.0 switches to a Fedora Core distribution, largely due to the end of life of Redhat distributions. Moreover, we are moving to a 2.6 based kernel that closely tracks the Fedora Core release of the kernel. This eases our ability to track bug and security related kernel patches.

The source code to the kernel is freely available via anonymous CVS from cvs.planet-lab.org. Moreover, it is possible to upgrade a standard Fedora Core 2 system to the PlanetLab kernel, and install a few additional packages from the yum respository, resulting in a full-fledged PlanetLab 3.0 development system. Please contact PlanetLab operations if you are interested in doing this.

4.2 Vservers

In moving from Linux 2.4 to Linux 2.6, the vserver patch to the kernel also moved from the 1.22 stable release to the 1.9.2 development release [1]. This change should be completely transparent to users.

One visible change to users inside a slice is the default shell. PlanetLab users may change their shell in their slice's /etc/passwd, which will be used when they log into the slice via SSH. Otherwise, the overall vserver related upgrade should be transparent to users.

4.3 CKRM

We are using the Class-based Kernel Resource Manager (CKRM) in place of PLKMOD as a framework for enforcing resource isolation amoung slice. See ckrm.sf.net for further information. Keep in mind that users (and their slices) do not interact directly with CKRM; resource limits and shares are set by the Node Manager. The following discussion is intended to sketch the implementation that's under the covers.

The table below summarizes the current set of resources that the V3.0 kernel manages. The first column lists the resource types, while the following two columns identify the mechanisms used to enforce either fair sharing or limits. For completeness, the table also lists other resources that V3.0 manages using either vserver or linux resource management solutions.

As shown, V3.0 continues to use the same kernel knobs to manage disk space and outbound network bandwidth. All other resource management support has been subsumed by CKRM. The design document for the CKRM resource controllers is available in the Linux 2.6 documentation tree.

[1] Please do not be confused by the version numbering scheme. The vserver community uses a 1.Xz numbering scheme for stable releases and a 1.Y.z for development releases.

	V2.0	V3.0
Disk Space	Disk Limit (vserver)	Disk Limit (vserver)
outbound Net bandwidth	Fair Share (HTB)	Fair Share (HTB)
CPU	Fair Share (PLKMOD)	Fair Share (CKRM)
Task	N/A	Max Task limit (CKRM)
Disk I/O	N/A	Enhanced CFQ (CKRM)
Memory	N/A	Fair Victim + Max limit (CKRM)

The integration between CKRM and Vserver consists of exposing the *Context ID* to CKRM's rule-based classification engine. With this in place, the above resources (cpu, phys. memory, tasks, and disk I/O) are isolated based on the context ID to which they belong.

A new package, called resman, is used to twiddle the knobs that the kernel exposes for each resource listed in the above table. The scripts provided by resman include: disklimit, bwlimit, cpulimit, memlimit, tasklimit, and iolimit. Each script exports the following commands: init, setlimit, getlimit, on, off, and default. The common case sequence of commands used to change a particular limit consists of a getlimit, setlimit, on sequence. The following example with cpulimit illustrates how to increase the cpu allocation by 4 shares for a slice called v_slice:

```
# cur=$(cpulimit getlimit v_slice)
# let cur=$cur+4
# cpulimit setlimit v_slice $cur
# cpulimit on v_slice
```

A similar recipe can be used for all the other limit scripts. The following table summarizes the units and default values of the getlimit/setlimit commands for these scripts:

command	unit	default share/limit value
disklimit	1K disk blocks	5000000 (i.e., 5GB)
bwlimit	1kbps	1
cpulimit	share	32
memlimit	4kb page	32768 (i.e., 128MB)
tasklimit	process count	256
iolimit	Not yet	

4.4 Network Virtualization

The VNET kernel module replaces the PLKMOD kernel module, as the piece of PlanetLab infrastructure code that virtualizes network access. VNET employs strategies similar to PLKMOD for hooking into the Linux kernel, tracking per-slice network usage, and redirecting packets, but does so in a more transparent and extensible manner. In particular, VNET does away with the *safe raw sockets* interface—the PLKMOD API for sending and receiving raw IP datagrams—in favor of simply supporting standard raw IP and packet sockets.

VNET enhances PlanetLab's compatibility with Linux and contributes additional features unavailable in previous versions of PlanetLab. Most users of PlanetLab 3.0 will be unaware of the presence of VNET's network virtualization efforts, as it should be.

4.4.1 Transparent IP Isolation

VNET provides *transparent IP isolation* between slices. Isolation is achieved by ensuring that traffic sent from one slice does not interfere with traffic sent from other slices, and that slices only receive traffic intended for them. Transparency is achieved by enforcing policy in the VNET code itself, rather than through a user-visible API.

4.4.2 Outbound Policy

By tracking `bind()` calls, VNET ensures that slices do not send datagrams with source ports bound by other slices. This outbound policy is not new with VNET. PLKMOD also tracks `bind()` calls and enforces a similar outbound policy, but VNET does not require that ports be bound to the same socket used to send datagrams, as PLKMOD does.

To send a raw IP datagram with a particular source port on a VNET system, first `bind()` the port to a regular IP socket. As on regular Linux, the act of binding a port reserves it and suppresses ICMP unreachable errors. After binding, datagrams with that source port may then be sent through any number of other raw IP or packet sockets.

4.4.3 Lazy Binding

Alternatively, a slice may skip the `bind()` step and rely upon a VNET feature called *lazy binding*, to send arbitrary IP datagrams. The datagrams may not interfere with other slices' traffic. VNET associates every IP datagram with an IP connection tuple, and tracks the current owner of every connection. If a connection is new or unbound—that is, if its source port is not bound to an open socket—then the last slice to send a datagram with that source port becomes the new owner of the connection. If the datagram is malformed, cannot be tracked, or the connection with which it is associated is owned by another slice, the datagram is disallowed.

Lazy binding enables stock `ping` and `traceroute` to work without modification. Because Linux does not export an interface for binding ICMP identifiers, stock `ping` and `traceroute` send ICMP messages with random identifiers through raw IP sockets. If two different slices send ICMP messages with the same identifier to

the same host within the same period, the last slice to send its message will become the owner of the connection, and the only slice entitled to receive replies.

4.4.4 Inbound Policy

Associating datagrams with connections and connections with slices is how inbound datagrams are demultiplexed. If a port has been bound by a slice, that slice is entitled to receive all datagrams associated with that port. Slices are always entitled to receive inbound datagrams associated with connections they initiated.

Unlike PLKMOD, VNET does not require that the same socket used for binding a port, be the one used to receive datagrams. As long as a port is bound to a socket opened by a slice, that slice may receive packets on any number of other raw IP or packet sockets.

4.4.5 Specific Restrictions

Packet sockets may only be bound to the *vnet* virtual interface. Note that sending IP fragments is not allowed, even through raw packet sockets. Despite its advertised MTU, the *vnet* interface is not a real Ethernet interface and is simply an alternative interface for sending and receiving raw IP datagrams. Ethernet header space must be included in the socket buffer if the raw packet socket type is `SOCK_RAW`, but the source and destination MAC addresses do not have to be valid.

A datagram sent through a raw IP or packet socket will be rejected if any of the following conditions are true.

- It is not a well-formed IP datagram.
- The protocol is not trackable (currently, only TCP, UDP, and ICMP protocols are trackable).
- The datagram is an ICMP error message.
- The source address is not local to the system.
- The source port in the IP header (or the identifier field in the case of ICMP) has already been bound to a socket created by another slice.
- The destination address has been blacklisted by PlanetLab Central.

4.4.6 VNET Extensions

Slices may obtain the identity of loopback peers, by calling `getsockopt(SO_PEERCRED)`. VNET fills the `gid` field of `struct ucred` with the slice ID of the peer when a new loopback connection is established.

Socket ownership may be transferred to another slice by calling `setsockopt(SO_SETXID)` with an integer specifying the slice ID of the new owner. If `SO_SETXID` is not defined, set it to the value of `SO_PEERCRED`. Currently, only the root slice is allowed to transfer socket ownership.

4.4.7 Proxy Sockets

Recent versions of PLKMOD (and now VNET) support *proxy sockets*, special raw packet sockets that allow slices to utilize unused IP address space on PlanetLab node subnets. On nodes with access to such address space, slices may bind to a virtual interface that proxies for the space. The network interfaces may be used as if they were anonymous (i.e. unconfigured for IP) network interfaces in promiscuous mode, connected to a subnet servicing the unused address space.

The primary applications for the interface are *network telescopes* and *honey farms*. Because proxy sockets do not require any special configuration, stock applications like `tcpdump`, `honeyd`, and `ethereal` may be used to implement such applications. Other possible applications include user-level IP stacks and implementations of IP anycast.

All standard `PF_PACKET` behavior and semantics are preserved, with three restrictions:

- The `PF_PACKET` socket protocol must be `htons(ETH_P_IP)` or `htons(ETH_P_ALL)`. Fundamentally, no other packet types besides IP can be transmitted or received on the dummy interface anyway, because the packets traverse the kernel IP routing table. To support `tcpdump` and other similar programs, `htons(ETH_P_ALL)` is accepted but ignored as far as packet capture is concerned. Malformed or unroutable IP packets are tossed according to the rules of the IP stack. The `PF_PACKET` socket type may be either `SOCK_DGRAM` (link-layer headers not included) or `SOCK_RAW` (link-layer headers included but likely to be garbage).
- Sockets must be bound with `bind()` to a proxy device. Only one slice at a time is allowed to `bind()` to the device; i.e., it provides all-or-nothing access.
- The setting of the `IFF_PROMISC` flag via `setsockopt()` is not allowed.

5 Installer

The PlanetLab node install program, Alpina, has been heavily modified to support installing PlanetLab 3.0 while running in the BootCD 2.0 environment. The most important changes that were made to Alpina include:

- Elimination of the BootstrapRPM step.
- Elimination of the BuildVServer step.
- Elimination of the CreateAccounts and InstallPLRPM steps.
- Smaller bandwidth requirement.
- Preparation for BootCD 3.0.

5.1 BootstrapRPM

Because the BootCD 2.0 environment does not provide `rpm` or `yum`, the Alpina installer must bootstrap the installation of these utilities.

Previously, Alpina would download and unpack (in a memory-backed `/tmp` directory) a tarball of a root filesystem that contained enough code to run `rpm`. Alpina would then download a manually determined set of RPMs to provide enough functionality to run `yum`. These RPMs would be installed in the temporary filesystem. The total in-memory usage of the temporary filesystem exceeded 200 MB.

Because of binary incompatibilities between different versions of the RPM database format, as well as a desire to use the same versions of `rpm` and `yum` to install nodes, as are installed on the nodes themselves, it was necessary to either update the temporary root filesystem tarball and the set of RPMs required by `yum` to their Fedora Core 2 versions, or change how the bootstrap process works.

The latter option was chosen. Alpina now downloads a tarball of a Fedora Core 2 root filesystem that already contains enough code to run `yum`. The command used to build the filesystem is essentially `yum install yum` on a vanilla Fedora Core 2 system. As a result, the filesystem's RPM and YUM databases are already initialized and ready to update. The filesystem is unpacked directly onto the hard drive as it downloads, eliminating the large temporary memory and bandwidth requirements. Installation is thus as simple as executing `yum groupupdate PlanetLab` inside the unpacked filesystem on the hard drive.

5.2 BuildVServer

The VServer reference image, from which all slices on a node are cloned, is now installed by an initialization script and is no longer part of installation. Moving installation to initialization time enables the VServer reference image to be updated when necessary. Previously, the image was created once at installation time and never updated.

5.3 CreateAccounts

The last of the legacy "static slices" that were part of the system core, have been removed, so the CreateAccounts step is no longer necessary.

5.4 InstallPLRPM

PlanetLab RPMs now list appropriate dependencies in their RPM spec files, so a separate InstallPLRPM step is no longer necessary. The "PlanetLab" yum group is now a subset of Fedora Core 2 "Base" plus mandatory PlanetLab RPMs.

5.5 Boot CD

A more comprehensive Boot CD environment, and a re-architecture of the installer, are on the todo list for version 3.0 of the Boot CD.

References

[1] MUIR, S., FIUCZYNSKI, M., BAVIER, A., AND PETERSON, L. Proper: A Priviledged Operation Service for PlanetLab. Tech. Rep. PDN-04-022, PlanetLab, Aug 2004.

[2] NABAH, S., FRANKE, H., CHOI, J., SEETHARAMAN, C., KAPLAN, S., SINGHI, N., KASHYAP, V., AND KRAVETZ, M. Class-based prioritized resource control in Linux. In *Proc. OLS 2003* (Ottawa, Ontario, Canada, Jul 2003).