# 33. The PlanetLab Platform

Timothy Roscoe (Intel Research, Berkeley)

PlanetLab is an extensively used, global, community-maintained platform for researchers to develop, deploy, and evaluate widely-distributed applications such as Peer-to-Peer systems. Because of PlanetLab's shared nature, and its unusual design goal of continuous replacement of components by the research community, it can also be viewed somewhat as a Peer-to-Peer system (and an ongoing research project) itself. This chapter describes PlanetLab's goals and origins, and discusses in detail the design principles that have governed its development and growth so far. It also discusses some of the methodological issues in performing research using a platform like PlanetLab – what can be learned from experimentation on PlanetLab, and what research claims can be validated by the system.

This overview of PlanetLab's architecture and development is necessarily brief. More details can be found in a series of documents called PlanetLab Design Notes, or PDNs, which are available on the PlanetLab web site at `http://www.planet-lab.org/`.

## 33.1 Introduction and History

The PlanetLab platform for distributed applications arose in March 2002 from the intersection of several trends in computing research.

First of these was a sense of frustration among the networking research community that the Internet and its protocols had become "ossified": so much infrastructure, commerce, and communication by now depended on the Internet that experimenting with radically new approaches at scale was impossible. At the same time, routing protocols like BGP had become so embedded in the network infrastructure that changing them even in an incremental manner became a delicate issue which most Internet Service Providers have shied away from.

The corresponding effect on networking research was a reluctance to explore radically different approaches on the one hand, and an reluctance to build real networks and systems on the other, with researchers relying instead predominantly on analytical results, simulations from programs like `ns2` [592], or network emulation using platforms like EmuLab [621].

This issue was captured succinctly in a report commissioned by the U.S. National Academy of Sciences entitled `''Looking Over the Fence at Networks''` [442]. The report also suggested a potential way out of the impasse: networking research could use the concept of an *overlay* to deploy, and attract users to, a new network architecture without needing to explicitly change the underlying Internet. This naturally led to discussion of how researchers could deploy overlay networks at sufficient scale to gain both experience and real users.

The second trend was rather more optimistic: the emergence of Peer-to-Peer systems both as a major research agenda, and at the same time the source of a significant proportion of traffic on the Internet. Early 2002 saw the end of the "Internet boom", and there was a feeling that client-server applications over the wide-area was now a mature and well-understood area, and Peer-to-Peer systems provided a new direction in research.

However, this sense of excitement with a rich new research agenda encountered similar problems to the classical networking community: how to try out and validate ideas with such systems at scale. As with networking, ideas such as structured overlay networks were investigated mostly by simulation; in the rare cases that a real deployment took place, it involved a very small number of distributed hosts. For example, the original Chord paper [575] reported experience deploying the system on only 10 sites.

Thirdly, a variety of related technologies had matured at approximately the same time. These ranged from techniques for managing systems composed of computers and networking elements, in the fields of cluster-based computing and data center provisioning, to operating system virtualization techniques and resource control mechanisms. It was felt that these mechanisms could form the basis of an approach to moving research agendas in networking and distributed systems out of the current impasse.

This was the context in which David Culler (of U.C. Berkeley) and Larry Peterson (of Princeton University), incorporating an earlier idea from Tom Anderson (of the University of Washington), met in early 2002 to discuss building a community-maintained, shared, distributed platform for deploying and evaluating wide-area services such as Peer-to-Peer systems. An informal workshop was convened at Intel Research in Berkeley in March of 2002, with enthusiastic attendence from systems researchers at U.S. universities, to discuss the scheme.

The result of this meeting was a vision of a collection of machines spread around the globe, contributed by participating institutions who provide local server hardware, bandwidth, power, and cooling, in return for a share in the worldwide platform.

A position paper outlining the PlanetLab vision followed [485], and Intel Research provided initial seed funding for the project. This consisted of donating the first 100 machines, shipped to about 40 universities worldwide, and providing operational support for the platform until 2004, when responsibility for maintaining PlanetLab shifted to the PlanetLab Consortium, based at Princeton University in New Jersey.

From the beginning, the target application area for PlanetLab has been *planetary-scale systems*, Peer-to-Peer applications prime among these. Planetary-scale applications are characterized as involving computation spread geographically across a wide area, for some subset of the following reasons:

- **Removing latency:** to serve a large, dispersed user population and still provide fast end-to-end response time, computation must be moved towards users to reduce the round-trip time for messages between users and the service. One obvious example of such services are content distribution networks (CDNs) like Akamai, who are in the business of moving content closer to a worldwide user population.
- **Spanning domain boundaries:** the service executes in many geographical locations so as to have a presence in many physical areas, legal jurisdictions, financial domains, etc. Examples of this kind of requirement include censorship-resistant systems like FreeNet [124], and federated archival storage systems like that envisioned by Oceanstore [368], which is intended to survive the physical destruction or financial dissolution of any participating service provider.
- **Multiple vantage points:** the application needs to process and correlate data in real time from many physical locations. For example, network mapping and measurement applications were among the first services deployed on PlanetLab and continue to be major users of the platform. PlanetLab has also been used to deploy distributed crawlers of Peer-to-Peer networks like Gnutella.

PlanetLab was enthusiastically taken up as a research vehicle, and has rapidly become a canonical part of how large-scale networked systems research is performed. At time of writing, PlanetLab consists of over 500 machines worldwide at more than 250 sites, with a significant presence in North America, Europe, China, Russia, Brazil, India, and elsewhere.

## 33.2 Architectural Principles

While supporting Peer-to-Peer systems, and wide-area distributed systems in general, is the main goal of PlanetLab, the community model by which sites donate local resources in return for a fraction of the global testbed means that PlanetLab itself can also be considered a Peer-to-Peer system. Indeed, from the outset PlanetLab has aimed at the twin goals of *decentralized operation* (reducing or removing entirely any centralized management facility) and *decentralized development* (encouraging the community to contribute the software infrastructure required by the platform).

Taken together, these requirements are somewhat novel for a large-scale distributed system, even a Peer-to-Peer system. It means that PlanetLab has to be designed not so much as a single artefact, but as a continuously evolving social *process*.

This is complicated by the tension between two usage models for PlanetLab. The first is as a testbed for experiments: a system is implemented and then deployed on PlanetLab for the purpose of obtaining measurement results for a paper. The second is as a platform to support long-running services in the wide area: several applications such as CoDeeN [468] and OpenDHT [340] have been running continously on PlanetLab and attracting real users for over a year, at time of writing.

To cope with these tensions, the PlanetLab team adopted a number of design principles to follow as the platform evolved [486]. Some of these only became clear during the course of the initial PlanetLab deployment, but others were identified at the original March 2002 workshop, including the three main principles of *application-centric interfaces*, *distributed virtualization*, and *unbundled management*.

### 33.2.1 Application-Centric Interfaces

A central concern in the design of any platform for supporting applications is the *execution environment*: what set of APIs must applications be written to? The prior experience of some of the PlanetLab design team (who had been involved in attempts to standardise interfaces for "active networks") led to a strong desire to keep application interfaces as conventional as possible.

There were several motivations for this. One was encouraging uptake: experience had shown that an unconventional API was less likely to be adopted by developers. But there was a more practical, short-term issue: a strong desire to get something working as fast as possible, since PlanetLab had an eager user community well before the first node came online.

Linux was chosen from the outset as the initial execution environment. The choice greatly facilitated development, since applications could be debugged on laboratory machines or clusters before deployment. A Linux execution environment was also easy to deploy - the first version of PlanetLab was indeed based on an unmodified Linux distribution.

As PlanetLab has evolved, new functionality has been added to the environment. For example, users were provided early on with a virtualized Linux machine to themselves, rather than explicitly sharing a physical machine, for a greater degree of isolation between projects. As a second example, it was long recognized that access to raw networking sockets was desirable for implementing new transport protocols, performing network measurements, etc.

In both these cases, functionality has been provided by modifying the kernel *underneath* the interface to the execution environment (the Linux application binary interface or ABI in this case), without changing the interface in any fundamental way. In the case of raw sockets, a user obtains such a socket in the conventional Linux manner, but the precise behaviour of the socket depends on privilege granted to that user's project by the organization administering that PlanetLab node.

Not all PlanetLab functionality can be presented through the operating system interface. For example, the ability to change the resource allocation for a project has no clear representation or analogue on a Linux machine. Consequently, an *orthogonal* interface to the PlanetLab-specific software running on a machine provides this access without polluting the operating system ABI. This interface to the "node manager" (discussed below) is not required by conventional users, but is essential for implementing what PlanetLab calls "infrastructure services": services which perform useful functions as part of the normal running of PlanetLab (for example, resource location and brokering).

Longer term, the principle of keeping the execution environment as conventional as possible will remain, though Linux will cease to be the only option. By replacing the Linux kernel with a virtual machine monitor such as Xen [179], different operating system execution environments can be employed depending on user needs. Keeping the interface to the node manager orthogonal to the execution environment allows each to evolve, and diversify, separately.

### 33.2.2 Distributed Virtualization

The key abstraction in PlanetLab is the *slice*: a distributed collection of virtual machines, in which an application or service executes. A PlanetLab virtual machine is often referred to as a *sliver*, and forms a resource container [56] for the application on a particular physical node.

PlanetLab faced a difficult choice as to the node virtualization mechanism – the node software that presents a virtual machine abstraction to slices. Like most of PlanetLab, this has evolved over time and is expected to change further in the future. The key requirements are those of any operating system: multiplexing the node resources, resource isolation (ensuring that one sliver does not gain unfair use of resources allocated to another on the same machine), security (ensuring that one sliver cannot gain unauthorized access to the state of another sliver), and abstraction (presenting the execution environment discussed in the previous section).

Initially, PlanetLab slivers were simple accounts on an unmodified Linux system. Each participating PlanetLab institution was allocated the same 10 user ids on every PlanetLab node. These simple Unix accounts were rapidly replaced by virtual Linux kernels using the "VServer" patch [391], which provides an illusion of multiple Linux machines over a single underlying, modified kernel. In time (and after some experimentation with different approaches), the static allocation of 10 slices was replaced by a fully dynamic system where slivers are created on demand on the nodes selected by the slice owner.

In addition to the VServer modifications, the PlanetLab Linux kernel has undergone a series of other modifications over time. Throughout, however, the strategy has been to prefer externally supported (and so relatively "standard") code to custom-written modules and patches, so as to reduce the burden of support and development. At the same time, modifications made by the PlanetLab support team (some minor changes to the VServer patch for example), have where possible been fed back to the original developers. The resource isolation mechanism was provided for some time using a version of the SILK (Scout paths In the Linux Kernel) [64] module, which has now been replaced by the more widely used Linux Class-based Kernel Resource Management (CKRM) extensions.

Modifications to the Linux kernel were by no means the only virtualization option on the table at PlanetLab's inception. Full-scale virtual machine technologies like VMware are attractive because they provide the ability to run multiple unmodified operating systems, including custom kernels. However, full virtualization does not currently provide the scalability required: it is not unusual for around 100 slices to be active on a PlanetLab node, in particular shortly before a major conference deadline.

Another promising direction was *isolation kernels* [350]. Isolation kernels are very small hypervisors, making verification of isolation and security properties much easier. Rather than fully virtualizing the whole ia32 processor architecture and PC hardware, systems like Denali and Xen [179] perform *paravirtualization*: they present a virtual hardware abstraction which is close to the underlying architecture but much more efficient to virtualize.

Paravirtualization solutions were not mature in early 2002, but Xen has become widely used since, making PlanetLab's transition to a Xen-based node architecture increasingly attractive. While this allows several different execution environments, as discussed above, it would not entail changes to the PlanetLab node manager interface.

PlanetLab encountered unexpected complexity in virtualizing the network on a node. Since PlanetLab nodes exist on the public network and their resources are donated by participating institutions, they typically have a single globally-routable IP address each. Consequently, services running in slices must share the space of available port numbers for listening sockets, just like any set of processes on a single machine. However, PlanetLab's target applications include network measurement infrastructures and routing overlays, making access to raw sockets from many slices in a safe, efficient, and controlled manner an important requirement for PlanetLab.

Even simple port contention can be a challenge. More than one Peer-to-Peer-based replacement for the Domain Name Service has been deployed on PlanetLab, and for compatibility with DNS all of them want to listen on UDP port 53. A user-supplied DNS multiplexer, running in a slice, provided a solution to this dilemma.

### 33.2.3 Unbundled Management

A final design principle that characterizes PlanetLab as a platform is *unbundled management*. Like any large, decentralized, federated system, PlanetLab faces significant management challenges. In addition to the two roles for PlanetLab mentioned above (supporting short-term experiments and long-running services), PlanetLab is also a research project in its own right, indeed, it is a research project that is being pursued by a number of the participating institutions simultaneously.

For PlanetLab to succeed as a community-built artifact and true Peer-to-Peer system, therefore, it is important that the research community is able to contribute functionality to the platform in the form of long-running *infrastructure services* that perform essential management functions (resource discovering, slice creation, etc.), *and* that different approaches to implementing these services can be tried out "for real" by different groups simultaneously. We call this principle *unbundled management*: management of the platform is as much as possible carried out by multiple, competing services contributed by users.

As a side effect, this principle has also enabled PlanetLab to become operational extremely quickly, albeit with a centralized management infrastructure. Since the intention has always been that all of PlanetLab's management functionality will be replaced by the community, this freed up the initial implementors to rapidly deploy a simple, provisional management architecture, as long as it was all replaceable in time.

The long-term goal, however, is for PlanetLab to evolve into a fully decentralized Peer-to-Peer system itself, with little or no need for the centralized "PLC" (or "PlanetLab Central") database currently used to manage the physical nodes themselves (installing the system software), slices, and user accounts.

What kinds of system design decisions does the principle of unbundled management lead to? The most immediate implication is that interfaces to system information must be as low-level as possible, while remaining sharable: multiple management subsystems on a node (themselves likely structured as Peer-to-Peer systems) must be able to use the same interface instances. Interfaces for querying PlanetLab management state are typically based on a standard format [524] which supplies tuples with a minimum of preprocessing.

A consequence of exporting management interfaces at as low a level of abstraction as possible is that the concepts that such interfaces deal with on a node are purely local to the node. Any distributed abstractions, including slices themselves, require some external service to implement them.

The decision has so far been reasonably successful: discovery services like SWORD [461] and monitoring services like CoMon [474] are now widely used by PlanetLab users, despite not being maintained by the core PlanetLab support team.

A more complex example is that of resource allocation. While the PlanetLab consortium has laid out a “framework” for resource allocation [490], this largely focuses on the representation of resources on nodes in the form of *tickets*, which represent promises of future resources, and *leases*, which represent resources currently bound to a sliver. Details of how tickets may be exchanged, and how the set of resources required by a slice are to be assembled, are again left for 3rd-party services to define and implement. The intention is to foster a number of different resource allocation schemes coexisting alongside each other; as well as the initial one (PLC), at least two others have been built: the Emulab portal [617] and SHARP [235].

## 33.3 PlanetLab Methodology

While some small-scale wide-area distributed testbeds existed previously, PlanetLab is the first such platform which is widely available to researchers in many countries with broad coverage over the globe. It has had, and continues to have, a significant influence in how much work in distributed and Peer-to-Peer systems is carried out: it is now possible to deploy many more large systems and evaluate them alongside each other “for real” on the Internet.

However, there has yet to appear consensus within the research community about what results obtained on PlanetLab actually mean, or how to use PlanetLab to derive valid experimental results. This section reviews the current state of thinking about how best to use broad-coverage testbeds in general, and PlanetLab in particular.

### 33.3.1 Using PlanetLab

We start by briefly describing what is, at time of writing, the way most researchers and students use PlanetLab: creating and controlling slices through the PlanetLab Central (PLC) web interface, or a command line tool which talks to PLC. Other interfaces exist, for example slices can be created via the Emulab portal [617].

Users log into PLC and create a slice, giving it a name, for example `ucb_p2ps`. The first half of the name identifies the creating institution, while the second is an arbitrary name for the slice. Having done this, users then add nodes from anywhere on PlanetLab to the slice. Having added a node, say `planet1.berkeley.intel-research.net`, to the slice, the user can log into the machine via secure shell with a command like:

```
$ ssh ucb_p2ps@planet1.berkeley.intel-research.net
```

What the user sees when logging in looks like a networked Linux machine. She can install programs, run code, `su` to root, create new user accounts, run programs like `tcpdump`, etc.

This lack of restriction on what users can do in slices leads to great flexibility in the code that can run, as well as providing a familiar programming environment (PlanetLab applications are usually compiled on users' desktop machines and then deployed on PlanetLab).

That said, there are significant differences between the runtime environment of a program running on a PlanetLab node and one running on a workstation or server in a lab: network conditions are very different, and the machine is always being shared with other projects. This has led to some debate about the kinds of experimental results that PlanetLab can provide, and the kinds of claims about system designs that can legitimately be made based on such results.

### 33.3.2 Reproducibility

Can PlanetLab be used to obtain reproducible results?

Naturally, the answer to this question depends a lot on what "reproducible" means. PlanetLab has never been intended as a testbed suitable for quantitatively reproducible experiments, in the sense that `ns2`, EmuLab, or ModelNet is. A number of factors contribute to the basic unpredictability of results obtained on the Internet, and PlanetLab in particular.

As numerous measurement studies have shown, observed conditions on the Internet can change quite radically over a wide variety of timescales. Furthermore, contention for resources (CPU, network bandwidth, disk activity) on PlanetLab itself varies considerably: despite the increasing provision of resource isolation on the platform, it will always be possible for one slice to "notice" the load imposed by others. Consequently, experiments on PlanetLab where, for example, different design choices for a Peer-to-Peer system are compared by running each choice consecutively for an hour or a day and measuring the performance of each, are unlikely to convince peer reviewers in the research community.

However, it is still possible to make valid comparisons between the performance of different systems running over PlanetLab. Running services over a long period of time with extensive measurement can provide clear evidence of what options or algorithms can make a real difference to performance, a methodology used by a number of projects (for example, the CoDeeN content distribution network). In many cases, this approach works best in conjunction with simulations, which we discuss below.

Another, complementary approach to providing some reproducibility of results on PlanetLab has to date not been explored in detail, but presents an interesting area of research in its own right. If conditions both on Planet-

Lab and the Internet in general can be measured continuously (for example, by some kind of "weather service"), it may be possible for the state of the environment at the time of a particular experiment to be sufficiently *characterized* that the results obtained can be rigorously compared with those of other experiments which are found, after the fact, to have been conducted under the same conditions.

There is, however, a different sense in which PlanetLab provides for reproducibility: the functionality of systems can be verified, and indeed used, by peer research groups. This has, of course, always been the case for small-scale (in terms of deployment) systems like compilers and operating systems, but the availability of PlanetLab now means that large-scale distributed systems built by research groups can also be taken up and used by other teams. Of course, this tends to impose a different standard to that by which such systems have traditionally been evaluated, a topic we return to below.

### 33.3.3 Representivity

Alongside the issue of how, and in what sense, experimental results from PlanetLab are reproducible is the question of the extent to which they are *representative* of reality. Like the issue of reproducibility, this has a number of aspects.

PlanetLab nodes are situated in a wide variety of places, including commercial colocation centers, industrial labs with commercial ISP connections, universities with dual-homed connections to academic and commercial networks, and DSL lines and cable modems. Consequently, PlanetLab machines provide an excellent set of vantage points from which to observe the Internet as a whole. PlanetSeer [640] is an example of a service leveraging this: observation of user traffic to the CoDeeN proxy network from all over the Internet is used to detect, triangulate, and diagnose Internet routing anomalies.

At the same time, however, the actual locations of PlanetLab nodes themselves is heavily skewed. Almost all nodes have much more bandwidth than a typical domestic U.S. or E.U. broadband connections, and the overwhelming majority are connected to lightly loaded academic networks with very different traffic characteristics to the commercial Internet. Banerjee, Griffin, and Pias [55] were to the first to point this out in print, and analyze the situation in some detail, but the immediate practical implication is that measurements of Internet paths *between PlanetLab nodes* are not likely to be representative of the Internet as a whole.

It is suggested in [55] that PlanetLab node locations might be chosen in the future so as to converge PlanetLab's network coverage to be representative of the Internet. However, resource constraints make this unlikely: PlanetLab is primarily maintained by sites, which are usually universities, hosting machines locally in exchange for access to the global platform.

This kind of representative coverage is important for empirical measurements of the Internet as it currently is experienced by ordinary users. The issue is very different if we consider PlanetLab as a way of testing out services for a future Internet or, more radically, a testbed for network architectures (including Peer-to-Peer services) which may supplant or replace the Internet. It is also worth remembering that many applications may not be targeted at networks like the Internet. Enterprise IP networks, for example, are often architected very differently to an ISP.

Networking aside, it is important to remember that PlanetLab nodes are shared between many projects, and this is a rather different deployment scenario to commercial services, which at time of writing typically are hosted on dedicated servers. In this respect, Peer-to-Peer applications to be deployed on domestic machines (which are presumably used for other applications as well) are closer to PlanetLab's situation.

A comprehensive study of PlanetLab's availability has yet to be undertaken, but it is clear that PlanetLab is significantly less stable at the node level than a well-run commercial hosting service. This is both a challenge and an opportunity: it is much harder to rely on any single PlanetLab node being available, but plenty of services have been deployed that demonstrate an impressively high degree of reliability despite fluctuations in the underlying platform.

Ultimately, a researcher has to be clear about the resemblence or otherwise between PlanetLab as a deployment environment and the motivating application environment for his or her research.

### 33.3.4 Quantitative Results

It is important to realize the value that PlanetLab brings to a Peer-to-Peer systems project, and avoid the trap of naively transferring experimental methodologies based on repeatable simulations and idealized models to an inappropriate environment. PlanetLab's significance is that, perhaps for the first time, researchers have access to the experience of having to cope with everything that a real systems environment can throw at any design.

PlanetLab can be used to *validate* simulation results. Modulo the representivity caveats above, if a system does not behave on PlanetLab as it does in simulation, there is likely to be some aspect of the real environment not captured by the simulator. If the disparity is large, this casts doubt on the validity of the simulation results. Conversely, if the measured behaviour of a system on PlanetLab matches closely the simulation results, this provides a supporting argument for simulation-based claims about how the system scales beyond what can be tested directly on PlanetLab.

In this way, PlanetLab can be valuable in designing simulators, since it provides an excellent source of phenomena, situations, and conditions that a simulation might need to take into account. Furthermore, long-term deployment of a service on PlanetLab can be used to quantitatively measure the effect of algorithmic or implementation changes in a way not possible with simulation or emulation environments. Finally, PlanetLab deployment may be used to characterize and model workloads to drive simulation design.

### 33.3.5 Qualitative Experience

The drawbacks and challenges mentioned in the sections above illustrate, to some extent, the tensions involved in any form of computer systems research, including Peer-to-Peer systems. On the one hand, there is the scientific aspect: experimental results should be reproducible by the experimenter, independently verifiable by the scientific community, and an accurate portrayal of nature. On the other hand is the engineering aspect: systems research is about creating new kinds of computer systems, and understanding not simply how they work, but understanding how to build them in the first place.

Ultimately, perhaps PlanetLab's greatest value is that it provides an opportunity to learn from the real world, demonstrate the *qualitative* feasibility of a system, attract real users, and provide a longer-term deployment platform for genuinely useful services.

There are great challenges in building robust Peer-to-Peer systems that can operate even at the scale of PlanetLab (about 500 nodes), when exposed to the full reality of the Internet. Nodes fail, and the resources available to a PlanetLab sliver (CPU, network bandwidth, etc.) can vary suddenly and dramatically over even quite small timescales. The network itself is not homogeneous even in connectivity: some pairs nodes on PlanetLab can never directly communicate over IP, even though they both have excellent connectivity to a variety of other PlanetLab nodes. Many assumptions implicitly made by laboratory-based evaluations are violated by PlanetLab: packets are seen to be duplicated, nodes are not fail-stop, etc. Almost all researchers who have deployed a service on PlanetLab report their first experience of seeing their code, which tested fine in the lab, fail for some unknown reason when run on PlanetLab.

Despite this, a remarkable number of highly robust services have been created and deployed on PlanetLab. Furthermore, it's likely that many of the insights gained by the researchers who have built and maintained these applications could not have been produced with a large-scale deployment.

## 33.4 Effects on the Internet

As numerous studies have shown, Peer-to-Peer applications such as BitTorrent, Gnutella, Kazaa, etc. have had a profound effect on the traffic mix observed on the Internet at large. However, it has been hard to capture the implications of such systems for the future design of the Internet itself, since by their very nature these applications, and the effects they have on the network, are not tracked in any detail.

The experience of running applications on PlanetLab – even though such deployments are at a much smaller scale than a successful Peer-to-Peer file-sharing application, for example – has led to a number of insights about how Peer-to-Peer applications interact differently with the Internet. Three features of Peer-to-Peer applications lead to this difference in behaviour.

### 33.4.1 Many-to-Many Connections

Peer-to-Peer services are, for the most part, *many-to-many* applications: a particular node running a component of a Peer-to-Peer service can be expected to contact a large number of peer nodes in a small amount of time, in contrast to clients in a traditional client-server application.

The many-to-many communication patterns of PlanetLab nodes running many Peer-to-Peer systems simultaneously has caused problematic interactions with IP routers, particularly low-end hardware which implements a "caching model" of network forwarding. Such routers have a hardware engine which can forward packets at line speed, using a cache of flows (based on packet 5-tuples). Flows which "miss" in the cache are handled by the route processor (typically a small embedded 32-bit RISC processor). PlanetLab overlay applications have been seen to generate new source-destination pairs at a sufficient rate to overwhelm the processor on such routers, causing the router to reboot (losing network connectivity in the process).

The Internet architecture, of course, strongly enshrines the idea that any node should be able to address a packet to any other node, at any time. What has happened in this case is that router vendors have optimized their hardware for a "common case" (the Web, email, and other TCP-based applications) which does not include the communication patterns of Peer-to-Peer systems. If Peer-to-Peer systems continue to grow as a proportion of Internet traffic, this common-case assumption will become increasingly untenable.

### 33.4.2 Many Alternative Routes

A component of a Peer-to-Peer system running on a single node (a sliver, to use the PlanetLab terminology) often has a large number of alternative nodes to contact in order to fulfil some particular function. For example, a filesharing system typically has many replicas of desired data scattered across the network, and a routing overlay typically has many, roughly equivalent, intermediate nodes that it can route traffic through (indeed, many DHT implementations by design keep multiple node addresses for each entry in their routing table). In contrast, a traditional web client (for instance) typically has few or no alternative addresses to contact in the event of a failure to contact the server.

This opens up a new design space for node-to-node communication protocols. For example, if one is interested in minimizing message latency in the presence of failures, as in DHTs like Bamboo [510], it pays to have very aggressive timeouts on the hop-by-hop exchanges by which messages are routed through the DHT. Even if a node is simply being a little slow, it's probably worthwhile to reroute the message around the node and along an alternate path, as long as this does not unduly increase network congestion.

Such fine-grained control over, and rapid reaction to, message timeouts is not possible with TCP as it is implemented in a mainstream operating system kernel. Furthermore, TCP's policy of reliable, in-order delivery of messages is not appropriate for many, if not most, high-performance Peer-to-Peer systems. Consequently, almost all the Peer-to-Peer applications deployed on PlanetLab today use UDP-based, TCP-friendly custom transport protocols rather than vanilla, kernel-based TCP. This is in stark contrast to traditional Internet applications, which are mostly TCP-based.

### 33.4.3 Overlays and Traffic Correlation

While only a few Peer-to-Peer applications claim to provide overlay networks to their users, in effect some kind of overlay network is at the heart of every Peer-to-Peer system. This leads to a *correlation* of traffic between nodes that is qualitatively different from traditional point-to-point TCP connections. For example, a flow traversing several overlay hops appears to the underlying IP network as a series of highly correlated point-to-point connections, whose path may bear no resemblance to the IP routing tables in operation at the time. Peer-to-Peer systems whose implicit overlay networks exhibit multicast behaviour further complicate this issue.

Some researchers are beginning to study the effects of such overlays on underlying ISP-based networks, for example [350]. While such work is currently at an early stage, it does appear that the traffic characterists of Peer-to-Peer overlays do not interact with typical ISPs traffic engineering policies in the way that traditional applications do.

## 33.5 Long-Term Goals

As was its intention, PlanetLab continues to evolve as a platform for deploying broad-coverage Peer-to-Peer services. As the platform grows, there is a trend towards the decentralization of control: the functionality unique to PlanetLab

Central (PLC) is expected to decrease, though PLC itself will most likely remain as one resource broker among many.

Beyond merely supporting distributed and Peer-to-Peer applications, however, recall that an explicit motivation for PlanetLab was to break the impasse facing Internet researchers, in not being to able introduce architectural changes in the Internet. Overlay networks using the underlying Internet were suggested as a way out of the problem.

Recently, the term *network virtualization* [559] has been coined to describe the use of overlays above a network like the Internet to provide Internet-like functionality themselves. By providing the ability to run multiple virtual networks with real users, the argument goes, alternatives to the Internet can be explored at scale without replacing the current infrastructure.

There are two broad schools of thought as to where this might lead. One says that by experimenting with alternative network architectures, the networking community in the broadest sense (researchers, carriers, governments, etc.) can select a new network architecture with properties preferable to the Internet, and then continue to use network virtualization as a way to incrementally deploy it.

The other, slightly more radical, school of thought is that network virtualization *is* the next architecture, in other words, future networked applications will operate in the main by setting up per-application virtual Peer-to-Peer networks, which then connect with other applications at many points.

In any case, investigating issues such as these requires the ability firstly to place computation at many points in the world (for routing and forwarding calculations), and secondly to acquire network paths between such points whose resources are guaranteed in some way, possibly probabilistically. PlanetLab provides the former, existing commercial ISPs' virtual private network (VPN) services or optical switched wavelength paths could provide the latter. A combination of the two holds real possibilities for implementing the successor to the Internet.