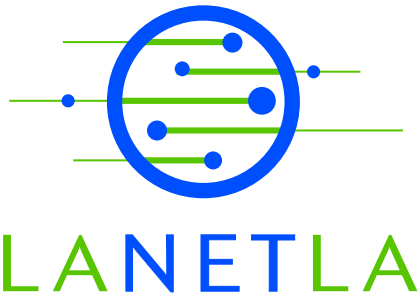

Port Use and Contention in PlanetLab

Jeff Sedayao
Intel Corporation

David Mazières
Department of Computer Science, NYU

PDN–03–016
December 2003

Status: Ongoing Draft.

Port Use and Contention in PlanetLab

Jeff Sedayao, Intel Corporation
David Mazières, Department of Computer Science, NYU

December 10, 2003

1. Introduction

This document describes TCP and UDP port number usage issues in PlanetLab. It covers a range of possible solutions, and includes a long-range vision of how PlanetLab will evolve. A road map for implementing that vision is also included. This document assumes a working of knowledge of TCP/IP.

2. Problem Description

Because particular ports are associated with well-known Internet services, PlanetLab experiments and service may require use of these well-known ports. Most well-known TCP and UDP ports are numbered below 1024. Ports in this range are considered "reserved" by UNIX, and cannot be bound by unprivileged software. Several critical utilities bind reserved ports, for instance the SSH server which listens for connections to TCP port 22. If more than one service wants to use such a port, the contention needs to be resolved. One example is the use of the DNS [1] port (UDP 53). While the PlanetLab maintainers would like to run a local caching DNS service using port 53, PlanetLab experimenters have been running a service using that port.

One solution to this particular problem would be to run one DNS service on the IP address of the hosts and another on the localhost interface (IP address 127.0.0.1). At writing of this PDN, that is not possible because of restrictions implemented by SILK, the network resource manager used in PlanetLab. Moreover, this approach works in the particular case of DNS only because PlanetLab's DNS server communications exclusively with local processes. In the more general case, or in the event that multiple experiments need the DNS port simultaneously, the key problem remains how to allocate and share well-known ports.

3. Exploring the Solution Space

In this section, we explore the possible ways to solve the problem. We start with simple solutions that are immediately usable and then cover ever more complex and contentious scenarios.

3. 1. Encourage Use Other Ports if Possible

Some services and protocols allow you to specify a port, which only slightly complicates the user interface. This eliminates the need to use a specific, well-known port. For

example, while the standard HTTP port is 80, URLs can specify a port with the format http://hostname.com:port-number/index.html.

People constructing services for PlanetLab should consider using DNS SRV [2] records to allow DNS to specify a port number as well as an IP address. For some applications, such as SSH, one can build a small wrapper script that fetches the port number from a DNS SRV record and feeds it to the official client on the command line. This doesn't work for services that must work with unmodified clients.

Some protocols do not function well living on nonstandard ports. While a DNS server can be configured to accept requests on a port other than the standard port 53, it is not possible to refer recursive client DNS resolvers to servers on non-standard ports.

3.2. Usage Scheduling

If ports cannot be shared, then another solution would be scheduling usage of the port by some method or using a resource allocator like SHARP [3] to arbitrate usage of the port. This is sufficient for short-term usage of a port as long as there is no planned usage of the port by PlanetLab itself. This is not sufficient if port usage is desired for long periods of time. Another solution would be to schedule the ports on some hosts to one experiment and the ports on the remaining PlanetLab hosts to another. If two experiments running simultaneously need to have geographical coverage, since there are typically at least two PlanetLab hosts running in each hosting site, experiments could be divided among the hosts at each site and still provide geographical coverage. This approach is unlikely to scale above two simultaneous experiments, and fails if PlanetLab itself requires use of the well-known port across all hosts.

3.3. Using the LocalHost Port

For the case where there are two long running uses planned for a port, it may be sufficient that one of the services listens only on the localhost interface. For example, the planned use by PlanetLab of the DNS port can work by listening only on localhost. This would allow a researcher to run only on the external IP address. The PlanetLab port management software, however, prevents it from being used by two services, even if one is listening only on the localhost.

Even if the port management software is fixed to allow sharing of a port between the external IP address and the localhost interface, some experimenters want to run a DNS server on the localhost port but only in their particular slice. The port management software needs to be able to handle this particular case. Even if fixed, this approach fails if experiments will not work using the localhost port.

3.4. Using an Address Pool

A better way to arbitrate the use of ports is to have a pool of IP addresses associated with a PlanetLab host and allow a slice that wanted to use a well-known port to use that port

on an IP address in the pool. The PlanetLab port management software would need to evolve to permit this, and we would need a resource manager to arbitrate the allocation of the port and the IP address. This solution works if there are spare IP addresses that can be allocated and if the demand for ports is low enough so that the contention for the IP addresses is not an issue. It is likely that some of the most "interesting" places to use PlanetLab (i.e., outside of North America) will not have many IP address available. Otherwise, a resource allocator or some other kind of scheduling will need to allocate use of the port across different hosts.

3.5. Protocol-specific Packet and Connection Multiplexing

Should demand for well-known ports exceed the number of IP addresses available and make allocation of ports across a number of different hosts impractical, the last approach that should be taken is to write a program that multiplexes requests based on the content of the packets. UDP based services need to demultiplex each packet received to a vserver's daemon. The daemon will need the ability to look at the complete contents of incoming packets and send arbitrary packets. Using the DNS example, a number of servers could be running on a variety of ports, and a program would run that would forward the request to one of the servers based on the characteristics of the packets. The demultiplexer for TCP based services should peek at the initial bytes transmitted by the client, and based on this data, hand the connection off to a vserver daemon (which has been slightly modified to work with the demultiplexer). Protocols that use both TCP and UDP (e.g., DNS and H.323 [4]) might require maintaining enough state information to permit coordination of UDP and TCP demultiplexing.

This approach only works for protocols in which the client transmits enough information to demultiplex before receiving any response from the server. Fortunately, many protocols have this property. For example, HTTP requests include a "Host:' header [5] specifically for the purpose of having multiple virtual web servers on the same IP address. In the case of DNS, the domain name in a client's request can be used to demultiplex requests. In other cases, it may be difficult or impossible to determine from the client how to demultiplex a connection. With SSH, the client expects a public key upon connection setup and never expresses what particular service or host that it is expecting to reach. SSL has similar issues with certificates.

One complication of demultiplexing ports is that some experiments delve deeper into the IP layer than typical applications. For example, one service looks at the IP TTL (time to live) in the IP packet that forms a DNS query, sends multiple replies with different TTLs, and maps out the network with any ICMP responses it receives to packets with short TTLs. In such a case, simple DNS proxying is not sufficient--the demultiplexer needs to include IP headers and forward any related ICMP messages to the vserver daemon as well.

While the final solution of using protocol-specific demultiplexing would accommodate any number of servers on a given port, it has a number of drawbacks. The approach requires careful parsing of packets to make sure that multiplexing works correctly. This

choice should only be done if the demand for a particular well-known port proves high enough to invalidate all of the previous approaches.

4. Well-known Port Usage Vision and End State

In the long term, PlanetLab will offer several techniques for resolving port contention problems. In order of decreasing preference, use of non well-known ports is encouraged first. If that solution does not work and smart scheduling/resource allocation will not work, then the Address Pool method should be used will be used. If contention for well-known ports and addresses remains high, the protocol specific multiplexing will need to be implemented.

5. Implementation Roadmap

Below is the series of changes (in recommended but not mandatory sequence) that need to be implemented to achieve the long-term vision described above.

a. Modify SILK to enable use of a well-known port on an external interface and on localhost interface
b. Implement fix to enable multiple vservers listening on localhost interface
c. Implement port reservation as part of PlanetLab resource allocation
d. Implement address pool for multiple services listening on port.
e. Implement generic multiplexer for multiple services listening on port if address pool technique is not sufficient for port contention

REFERENCES

[1] DNS RFC. P. Mockapetris. Domain Names – Concepts and Facilities. RFC 1034. November 1987.

[2] A. Gulbrandsen, P. Vixie, and L. Esibov. A DNS RR for specifying the location of services (DNS SRV). RFC 2782. February 2000.

[3] Y. Fu, J. Chase, B. Chun, Steve Schwab, and A. Vahdat. Sharp: An Architecture for Secure Resource Peering. In Proceedings of 19th ACM Symposium on Operating System Principles. Bolton Landing, NY, October 2003.

[4] "Packet-based Multimedia communications systems", ITU-T Recommendation H.323. November 2000.

[5] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. June 1999.