Supercharging PlanetLab – a High Performance, Multi-Application, Overlay Network Platform

Jon Turner
Washington University
+1-314-935-8552
jon.turner@wustl.edu

Brandon Heller
Washington University
+1-314-935-6160
bdh4@arl.wustl.edu

Jing Lu
Washington University
+1-314-935-4658
jl1@arl.wustl.edu

Patrick Crowley
Washington University
+1-314-935-9186
pcrowley@wustl.edu

Fred Kuhns
Washington University
+1-314-935-6598
fredk@arl.wustl.edu

Michael Wilson
Washington University
+1-314-935-6160
mlw2@arl.wustl.edu

John DeHart
Washington University
+1-314-935-7329
jdd@arl.wustl.edu

Sailesh Kumar
Washington University
+1-314-935-6160
sailesh@arl.wustl.edu

Charles Wiseman
Washington University
+1-314-935-6160
cgw1@arl.wustl.edu

Amy Freestone
Washington University
+1-314-935-6160
amf4@cec.wustl.edu

John Lockwood
Washington University
+1-314-935-4460
lockwood@arl.wustl.edu

David Zar
Washington University
+1-314-935-4876
dzar@arl.wustl.edu

ABSTRACT
In recent years, overlay networks have become an important vehicle for delivering Internet applications. Overlay network nodes are typically implemented using general purpose servers or clusters. We investigate the performance benefits of more integrated architectures, combining general-purpose servers with high performance Network Processor (NP) subsystems. We focus on PlanetLab as our experimental context and report on the design and evaluation of an experimental PlanetLab platform capable of much higher levels of performance than typical system configurations. To make it easier for users to port applications, the system supports a fast path/slow path application structure that facilitates the mapping of the most performance-critical parts of an application onto an NP subsystem, while allowing the more complex control and exception-handling to be implemented within the programmer-friendly environment provided by conventional servers. We report on implementations of two sample applications, an IPv4 router, and a forwarding application for the Internet Indirection Infrastructure. We demonstrate an 80× improvement in packet processing rates and comparable reductions in latency.

Keywords. PlanetLab, overlay networks, network processors, Global Environment for Network Innovation (GENI)

1. INTRODUCTION

Network overlays have become a popular tool for implementing Internet applications. While content-delivery networks provide the most prominent example of the commercial application of overlays [DI02, KO04], systems researchers have developed a variety of experimental overlay applications, demonstrating that the overlay approach can be an effective method for deploying a broad range of innovative systems [BH06, FR04,RH05, ST02]. Rising traffic volumes in overlay networks make the performance of overlay nodes an issue of growing importance. Currently, overlays nodes are constructed using general purpose servers, often organized into a cluster with a load-balancing switch acting as a front end. This paper explores an alternative approach that combines general purpose server blades and high performance *Network Processor* (NP) subsystems into an integrated architecture designed to support multiple applications concurrently.

To provide a concrete target for the research, and to facilitate the system's deployment and use by others, we have chosen to focus on the design of a high performance node for the PlanetLab overlay network testbed [CH03, PE02]. In the roughly five years since its inception, PlanetLab has become a popular experimental platform and deployment vehicle for systems researchers in networking and distributed systems. PlanetLab nodes are implemented using conventional PCs, running a modified version of Linux. This provides a familiar implementation environment and is inexpensive and easy to deploy. At the same time, it does have significant performance limitations that have become increasingly apparent as the usage of PlanetLab has grown, and as researchers have sought to deploy long-running services that carry significant volumes of traffic. Because PlanetLab applications run as user-space processes, their packet forwarding rates are typically limited

This work supported in part by NSF (grants 0520778 and 0626661).

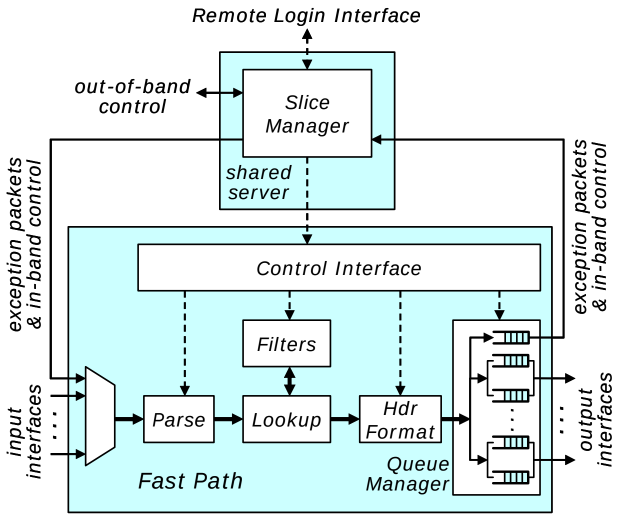


Figure 1. Generic application structure showing fast path (implemented on a network processor) and slow path on a general purpose server.

to under 50K packets per second, which translates to less than 100 Mb/s for average packet lengths of 250 bytes. Applications that do significant processing of packets (rather than simply forwarding them) can have substantially smaller packet forwarding rates. In addition, applications running in PlanetLab are subject to high latencies (tens of milliseconds per hop), high delay jitter and poor performance isolation. These characteristics are caused by the coarse-grained time-slicing provided by the operating system, and the failure to properly account for OS-level processing on behalf of different application processes.

To address these issues, we have developed an experimental system that can serve as a high performance PlanetLab node. Our *Supercharged PlanetLab Platform* (SPP) integrates general purpose server blades with performance-optimized NP subsystems, into a platform that delivers the flexibility and ease-of-use of a conventional PlanetLab implementation, while delivering much higher levels of performance. By supporting a simple and familiar fast-path/slow-path application structure, we make it straightforward for researchers to map the high volume part of their applications (which is typically fairly small) onto the NP resources, while enabling them to implement the more complex parts in the programmer-friendly environment offered by a general-purpose server. We report on the implementation of two applications running in this environment, that demonstrate packet forwarding rates of 4.8 million packets per second for a single NP subsystem; this is sufficient for throughputs of 5 Gb/s for average packet lengths of just 130 bytes. We also report latencies that are consistently less than 200 μs. It should be noted that while we focus on PlanetLab as the implementation context for this work, our broader objective is to understand the design of such platforms for more general contexts, such as future commercial overlay hosting services that are likely to be far less resource-constrained than PlanetLab. So, while some aspects of the architecture exceed current requirements for PlanetLab, they can be important in other settings.

Section 2 of the paper provides an overview of the system, setting the context for the more detailed presentation in later sections. Section 3, provides some background on network processors generally, and the IXP 2850, in particular. Section 4 describes the software framework that enables the fast path processing of multiple PlanetLab slices to co-exist within a single network processor. Section 5 briefly discusses our strategy for improving the performance of the general purpose processor blades. Section 6 describes the overall control architecture of the system and explains how it fits within the PlanetLab framework. The results of our evaluation are presented in Section 7, where we report on experiments with both an IPv4 router application and an implementation of a router for the Internet Indirection Infrastructure [ST02]. We finish with a short discussion of related work in Section 8, some alternative approaches in Section 9 and closing remarks in Section 10.

2. SYSTEM OVERVIEW

2.1. Objectives

Our principal objective for the SPP is to enable PlanetLab applications to achieve substantially higher levels of both IO performance and processing performance, while making it reasonably straightforward for PlanetLab users to take advantage of the capabilities offered by high performance components, such as network processor subsystems. We also require that legacy PlanetLab applications run on the system without change. While unmodified applications will experience limited performance gains, the ability to support existing implementations can make it easier to migrate to higher performance implementations that take advantage of the network processor resources.

To enable multiple PlanetLab applications to use the network processor resources concurrently, the system supports both multiple NP subsystems and sharing of individual subsystems. Since modern NPs are not designed to be shared, this creates some challenges. To accommodate the limitations of the NP environment and to simplify the porting of applications to the NP, we have chosen to provide support for the generic application structure shown in Figure 1. In this structure, applications are divided into a *Fast Path* (FP) that runs on an NP and a slow path for control and exception processing that is handled by a separate *Slice Manager* (SM), running within a vServer [VS06] on a general purpose compute server. The SM can control the fast path through a generic control interface. A remote user can control the application by logging into the vServer hosting the SM. Slices can forward control messages “in-band” by sending them to the fast path, which inserts them in the appropriate outgoing queue, or they can send them “out-of-band”.

2.2. Node Abstraction

The abstraction provided by the node seeks to mimic the PlanetLab node abstraction as closely as possible, while providing some additional features. The PlanetLab node abstraction seeks to give each vServer the illusion that it is running on a dedicated machine. The illusion is imperfect, because practical limits on IP address availability force the different vServers to share the same IP ad-

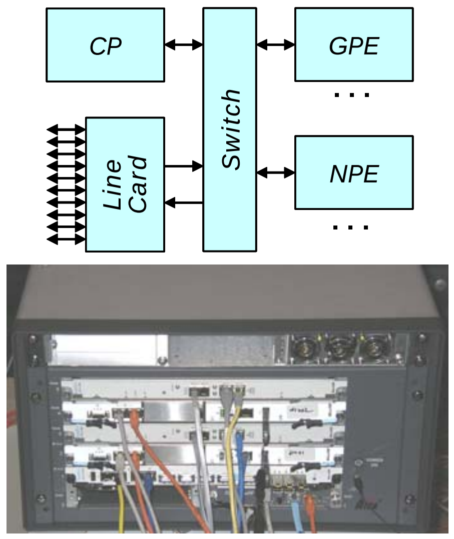


Figure 2. System organization showing Control Processor (CP), General Purpose Processing Engines (GPE) and Network Processing Engines (NPE); photo of current development platform

dress, which in turn means that they must share common sets of TCP and UDP port numbers. PlanetLab also does not support any explicit “virtual link” concept. While slices may obtain a reserved allocation of network bandwidth, this allocation simply applies to the *network interface bandwidth* and does not imply any reserved capacity from node to node. Typically, PlanetLab nodes have a single network interface, so the bandwidth reservation simply gives the node the illusion of a dedicated network interface with a specified capacity.

SPP nodes must operate under the same constraints and provide the same abstraction as a conventional PlanetLab node. However, since an SPP node can support multiple physical interfaces, we allow slices to reserve a share of each of the available interfaces. We also allow slices to associate multiple queues with each interface, and to divide their share of the interface bandwidth among their different queues. Through the fast path control interface, an application may install filters that map packets to specific queues and output interfaces. The control interface also allows the application to specify each queue’s share of the outgoing interface and its capacity.

As mentioned above, PlanetLab slices running in the same node share the same set of TCP and UDP port numbers. To enable remote users to send packets to a PlanetLab slice, users need to know which port number to use to get the packet to the correct slice. Since nodes are shared, slices cannot count on a specific port number being available on a particular node and Planetlab applications must be prepared to cope with this. Of course, this issue also arises in the SPP, but it is further complicated by the fact that an SPP node includes within it, multiple general purpose and NP subsystems. While the SPP node will often have multiple IP addresses (one for each of its physical interfaces), the number of IP addresses need not match the number of internal subsystems, so there can be no direct mapping. This means that not only must externally visible port numbers be shared among vServers within a given physical server, but they must also be shared among vServers in different physical servers and NP subsystems. We will elaborate on the implications of this in Section 6.

2.3. System Components

Figure 2 shows the main components of an SPP node. All input and output occurs through the *Line Card* (LC), which is an NP-based subsystem with one or more physical interfaces (our current development platform has 10 gigabit Ethernet interfaces, as shown in the diagram). The LC forwards each arriving packet to the system component configured to process it, and queues outgoing packets for transmission, ensuring that each slice gets the appropriate share of the network interface bandwidth. The architecture can support multiple LCs, but since the deployment contexts for PlanetLab nodes generally constrains the available bandwidth, PlanetLab provides little motivation for systems with multiple LCs. The *General Purpose Processing Engines* (GPE) are conventional dual processor server blades running the PlanetLab OS (currently Linux 2.6, with PlanetLab-specific extensions) and hosting vServers that serve application slices. The *Network Processing Engines* (NPE) are NP subsystems comprising an Intel IXP 2850 NP, with 17 internal processor cores, 3 banks of SDRAM, 3 banks of QDR SRAM and a Ternary Content Addressable Memory (TCAM). The NPEs support fast path processing for slices that elect to use this capability and each provides up to 5 Gb/s of IO bandwidth. There are two NPE subsystems on each physical NP blade. The *Control Processor* (CP) is another conventional server blade that hosts the software that coordinates the operation of the system as a whole. It can also host vServers serving application slices. The switch block is actually two separate switches, a 10 Gigabit Ethernet switch for data, and a separate 1 GE control switch.

Figure 2 also includes a photograph of the current development platform for the system. This configuration includes a switch plus two NP blades; one implements the Line Card functions (the IO interfaces are at the rear of the chassis) and the other implements two NPEs. The configuration also includes two server blades, (one CP and one GPE). The system architecture is designed to support larger configurations. In particular, these same components can be used in a 14 slot chassis, allowing for up to 12 conventional server blades or NP subsystems in a single PlanetLab node. Multi-chassis configurations are also possible.

3. NETWORK PROCESSOR ISSUES

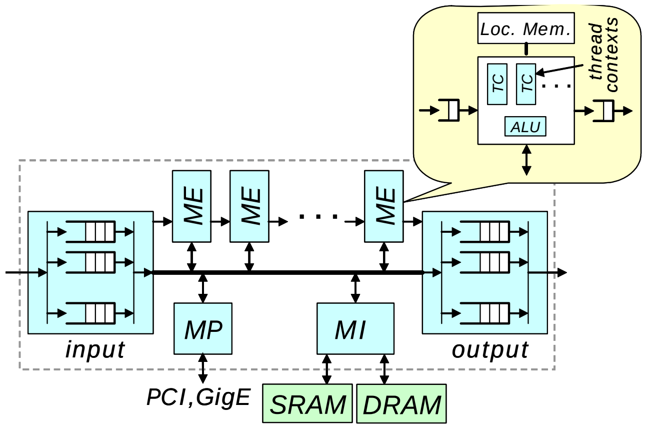


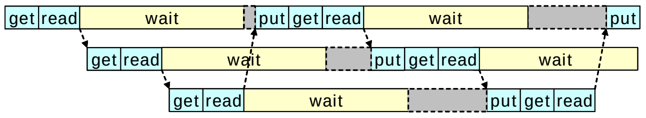
Figure 3. IXP 2850 block diagram showing the 16 Micro-Engines (ME) and the Management Processor (MP)

To appreciate some of the NPE design issues it's helpful to understand a little bit about Network Processors and the Intel IXP 2850, in particular [IXP]. First, NP products have been developed for use in conventional routers, as replacements for the Application Specific Integrated Circuits (ASIC) that have typically been used to provide high throughput packet processing. Products, like Cisco's CRS-1 use proprietary NPs to perform all the line card packet processing functions [CI06], and the IXP family of network processors is used in a wide variety of products made by multiple system vendors. Because NPs are programmable, they enable more rapid development and more rapid correction of design errors.

To enable consistently high performance, the IXP 2850 is equipped with 16 multi-threaded *Micro-Engines* (ME) that do the bulk of the packet processing, plus several high bandwidth memory interfaces (see Figure 3). In typical applications DRAM is used primarily for packet buffers, while SRAM is used for implementing lookup tables and linked list queues. There are also special-purpose on-chip memory resources, both within the MEs and shared across the MEs. An xScale *Management Processor* (MP) is provided for overall system control. The MP typically runs a general-purpose OS like Linux, and has direct access to all of system memory and direct control over the MEs.

As with any modern processor, the primary challenge to achieving high performance is coping with the large processor/memory latency gap. Retrieving data from off-chip memory can take 50-100 ns (or more), meaning that in the time it takes to retrieve a piece of data from memory, a processor can potentially execute over 100 instructions. The challenge for system designers is to try to ensure that the processor stays busy, in spite of this. Conventional processors cope with the memory latency gap primarily using caches. However for caches to be effective, applications must exhibit *locality of reference*, and unfortunately, networking applications typically exhibit very limited locality of reference, with respect to their data.

Since caches are relatively ineffective for networking workloads, the IXP provides a different mechanism for coping with the memory latency gap, *hardware multithreading*. Each of the MEs has eight separate sets of processor registers (including Program Counter), which form the MEs hardware *thread contexts.* An ME can switch from one context to another in 2 clock cycles, allowing it to stay busy doing useful work, even when several of its hardware threads are suspended, waiting for data to be retrieved from external memory. Multi-threading can be used in a variety of ways, but there some common usage patterns that are well-supported by hardware mechanisms. Perhaps the most commonly used (and simplest) such pattern involves a group of threads that operate in a round-robin fashion, using hardware signals to pass control explicitly from one thread to the next, as illustrated below.



In this example, the first thread starts by reading a data item (e.g. a packet pointer) from a shared input queue, then issues a read request before passing control to the second thread, which then reads the next data item from the shared queue, issues its own read request and passes control to the third thread. By the time the third thread issues its read request, the first thread is ready to continue. Notice how this allows the processor to stay busy, in spite of the long memory latency. Also, note that the round robin processing ensures that data items are processed in order. This technique works well when the variation in processing times from one item to the next is bounded (which is commonly the case in packet processing contexts), and is straightforward to implement.

There are two other aspects of the MEs that are important to understand. First, each has a small (8K), dedicated program store, from which it executes. This limits the number of different functions that can be implemented by a single ME, favoring programs that are divided into smaller pieces and organized as a pipeline. The MEs support such pipeline processing by providing dedicated FIFOs between consecutive pairs of MEs (*Next Neighbor Rings*). A pipelined program structure also makes it easy to use the processing power of the MEs effectively, since the parallel components of the system are largely decoupled from one another.

4. SHARING THE NPE

To support the generic application structure in Figure 1, we have developed software for the NPE that allows it to be shared by the fast path segments of many different slices. The organization of the software and its mapping onto MEs is shown in Figure 4. Packets received from the switch are copied to DRAM buffers by the *Receive* (Rx) block on arrival, which also passes a pointer to the packet buffer through the main packet processing pipeline. Information contained in the packet header can be retrieved from DRAM by subsequent blocks as needed, but no explicit copying of the packet takes place in the processing pipeline. At the end of the pipeline, the *Transmit* (Tx) block forwards the packet to the output. Packet pointers (and other information) are passed along the pipeline primarily using FIFOs linking adjacent MEs. Pipeline elements typically process 8 packets concurrently using the hard-

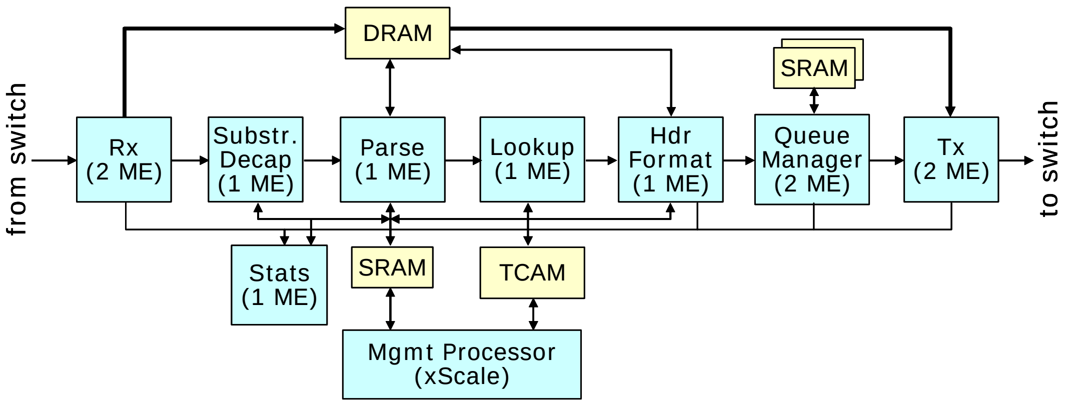


Figure 4. NPE software structure showing the use of memory by various software components, and the mapping of components onto Micro-Engines (ME)

ware thread contexts. The performance of individual pipeline stages can be further increased by distributing the processing across multiple MEs. The *Substrate Decapsulation* block determines which slice the packet belongs to, by doing a lookup in a table stored in one of the SRAMs. It also effectively strips the outer header from the packet by adjusting the packet pointer before passing it along the pipeline.

The *Parse* block includes slice-specific program segments. More precisely, *Parse* includes program segments that define a preconfigured set of *Code Options.* Slices are configured to use one of the available code options and each slice has a block of memory in SRAM that it can use for slice-specific data. Currently, code options have been implemented for IPv4 forwarding and for the Internet Indirection Infrastructure (I3) [ST02]. New code options are fairly easy to add, but this does require familiarity with the NP programming environment and must be done with care to ensure that new code options do not interfere with the operation of the other components. The primary role of *Parse*, is to examine the slice-specific header and use it and other information to form a *lookup key*, which is passed to the *Lookup* block.

The *Lookup* block provides a generic lookup capability, using the TCAM. It treats the lookup key provided by *Parse* as an opaque bit string with 112 bits. It augments this bit string with a slice identifier before performing the TCAM lookup. The slice's control software can insert packet filters into the TCAM. These filters can include up to 112 bits for the lookup key and 112 bits of mask information. Software in the Management Processor augments the slice-defined filters with the appropriate slice id before inserting them into the TCAM. This gives each slice the illusion of a dedicated TCAM. The position of filter entries in the TCAM determines their lookup priority, so the data associated with the first filter in the TCAM matching a given lookup key is returned. The number of entries assigned to different slices is entirely flexible, but the total number of entries is 128K.

The *Header Formatter* which follows *Lookup* makes any necessary changes to the slice-specific packet header, based on the result of the lookup and the semantics of the slice. It also formats the required outer packet header used to forward the packet to either the next PlanetLab node, or to its ultimate destination.

The *Queue Manager* (QM) implements a configurable collection of queues. More specifically, it provides ten distinct packet schedulers, each with a configurable output rate, and each with an associated set of queues. Separate schedulers are needed for each external interface supported by Line Cards. The number of distinct schedulers that can be supported by each ME is limited by the need to reserve some of the ME's local memory for each. Each scheduler implements the *weighted deficit round robin* scheduling policy, allowing different shares to be assigned to different queues. When multiple NPEs have schedulers configured to send to the same Line Card physical interface, the sum of their output rates is configured to be no larger than the physical interface rate. The rates used by the different schedulers can be statically configured or can be dynamically adjusted by distributed scheduling processes (this borrows ideas from [PA03]). Each slice has an associated set of queues that it can map packets to. When a slice's control software inserts a new filter, it specifies a slice-specific queue id. The filter insertion software remaps this to a physical queue id, which is added, as a hidden field, to the filter result. Slices can configure which scheduler to associate with a specific queue, the effective length of each queue and its share of the scheduler bandwidth.

The *Statistics* module maintains a variety of counts on behalf of slices. These can be accessed by slices through the Management Processor, to enable computation of performance statistics. The counting function is separated from the main processing pipeline to keep the associated memory accesses from slowing down the forwarding of packets, and to facilitate optimizations designed to overcome the effects of memory latency. The counts maintained by the Statistics module are kept in one of the external SRAMs and can be directly read by the MP.

5. ENHANCING GPE PERFORMANCE

While our main focus is on boosting application performance using the NPEs, the system also provides opportunities to boost performance of applications that run only on the GPEs. The GPEs can improve throughput over typical PlanetLab nodes in two ways. First, they use higher performance hardware configurations than is usual for PlanetLab. In particular, our current GPEs are

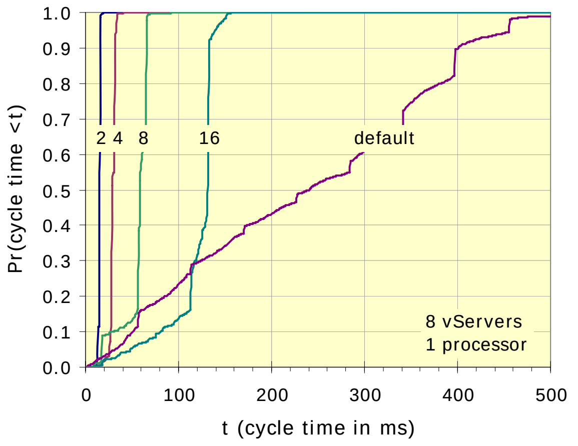


Figure 5. PlanetLab scheduler behavior for compute bound task; chart shows cumulative distribution function of "cycle time" for various choices of scheduling parameters.

Intel NetStructure MPCBL0040 server blades with a pair of 2 GHz Xeon dual-core processor chips, four gigabit Ethernet interfaces and an on-board disk.

In addition to improving throughput, we would like to improve the latency of PlanetLab applications. Because PlanetLab is built on top of a conventional operating system (Linux), it inherits the basic coarse-grained scheduling paradigm that characterizes such systems. The newer versions of the PlanetLab OS actually make significant modifications to the standard Linux scheduling, but the default scheduling parameters still produce coarse-grained time slices, resulting in latencies that can be unacceptably high for some applications. The PlanetLab scheduler is token-based, with each token representing 1 ms of computation. Each vServer is supplied with tokens at some specified rate (32 tokens/second, by default) and has a specified maximum number of tokens it can hold (100). In addition, before a vServer can be scheduled, it must have a specified minimum number of tokens (50). This leads to time slices of 50-100 ms, so in a system where *N* vServers are competing for the CPU, a vServer can be pre-empted for as long as 100(*N*–1) ms at a time, meaning that a packet arriving for a suspended vServer can wait a very long time before being processed. This issue is acknowledged in [BA06] where the authors address it by artificially changing the process priority of a particular slice of interest, but this is clearly not a general solution.

To gain some insight into the scheduler behavior, we ran a simple experiment in which eight identical vServers ran a simple compute-bound program, competing for a single processor core. Each vServer continuously checked the system time to detect periods when it was pre-empted and we recorded the distribution of pre-emption times. We then estimated the "cycle time" (the time required for all eight vServers to complete one scheduling round) as 8/7 times the pre-emption time and plotted the resulting cumulative distribution function, shown in Figure 5. The default PlanetLab scheduling parameters produce cycle times ranging up to 500 ms with very high variability. For the other cases shown, each vServer was allocated tokens at a rate of 120 per second (slightly less than one-eighth of the processor) and the minimum and maximum token allocations were set to the same value, with this value being varied from 2 to 16. We note that by making the minimum and maximum token values equal, we get much more consistent scheduling behavior and we note that the median cycle times are roughly equal to the product of the number of vServers and the number of tokens, which is what one would expect based on an idealized analysis.

6. CONTROL ARCHITECTURE

Figure 6 is a block diagram of the system, showing the control components of the architecture. First, note that the system provides a control network that is independent of the switch that carries data traffic (the control net is actually implemented on the same switch blade as the main data switch, but the control traffic is logically and physically separate). The control network is accessible only to the control elements of the architecture. In particular, vServers hosting user slices have no direct access to it.

The system's Control Processor (CP) obtains slice configuration data using the standard PlanetLab mechanism of periodically polling the PlanetLab Central database (PLC). Slices that are configured to use the system are assigned to one of the GPEs by the *Global Node Manager* (GNM) and a corresponding entry is made in a local copy of the Planet Lab database (myPLC). The *Local Node Managers* (LNM) on each of the GPEs periodically poll myPLC to obtain new slice configurations.

Once a vServer has been assigned to a slice, a user of that slice may login to it, in order to set up the application on the vServer. To applications that don't use the NPEs, this process works much like it would on a conventional PlanetLab node. However, there are some configuration steps that must be implemented under the covers, to make this as seamless as possible.

To allow slices to reserve externally visible port numbers, we provide an interface to the LRM that relays the reservation requests to the GRM. The GRM keeps track of all externally visible port numbers that have been assigned, and if the requested port number is available, it makes the appropriate assignment and configures the Line Card so that when packets are received with the specified port number, they will be forwarded to the right GPE. To make this process transparent to the slices, the interface to the LRM is hidden inside library routines that are used in place of the standard IO libraries. Users do need to link their applications against these libraries, but are otherwise unaffected.

To allow outgoing TCP connections to be handled correctly, the system must implement Network Address Translation (NAT). This is handled by the Management Processor in the Line Card which intercepts outgoing TCP control packets and maps the source port number to an available value from a pre-allocated range. It then configures tables in the Line Card so that IP addresses and port numbers are appropriately re-mapped for all subsequent data packets. NAT processing is also performed for other protocols that require it (e.g. ICMP echo packets).

A slice that elects to use an NPE for its fast path must request an NP slice from the *Local Resource Manager* (LRM), which

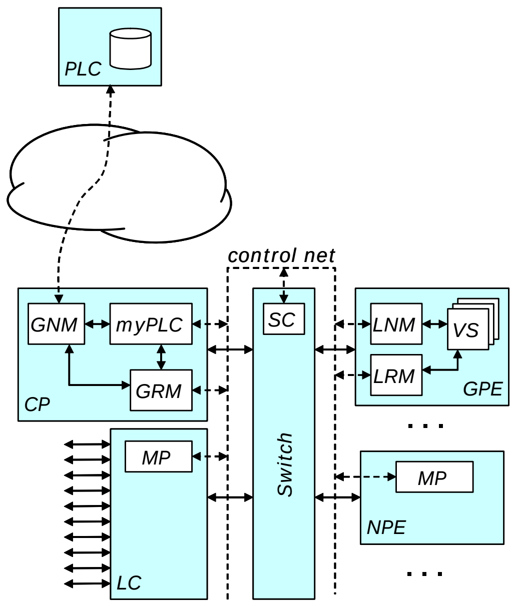


Figure 6. Control architecture showing global and local Node Managers (GNM, RNM) and Resource Managers (GRM, LRM).

forwards the request to the GRM. The request specifies the NPE required code option and various resource parameters, including the number of required filter table entries, queues and packet buffers, the amount of SRAM space it needs, and the total reserved bandwidth it requires. The GRM selects the most appropriate NPE to host the slice and returns its id to the LRM. The LRM then interacts with the MP to complete the initial configuration.

Once the initial setup is complete, the application running in its vServer can perform additional configuration steps. In particular, it can request that UDP ports on any of the system's external interfaces be configured to forward packets between the external interface and its NPE fast path. As part of this process it may request a specified share of the external interface bandwidth (both incoming and outgoing). Once a slice has a configured UDP port on a physical interface it can associate one or more of its fast path queues to a packet scheduler for that interface (in the NPE queue manager). It can also configure the lengths of individual queues and their individual shares of the interface bandwidth, as well as filters in the lookup table. In addition, it can write whatever slice-specific configuration data is appropriate in its SRAM memory space. This is accomplished through a generic memory read/write mechanism provided by the NPE's MP. All these control interactions take place through the LRM, which serves as an intermediary between the vServers running on the GPE and the NPEs.

As mentioned above, users login to their vServers to configure their application, in much the same way that they do on PlanetLab today. However, this is complicated by the fact that there are multiple GPEs and each user's session must be directed to the appropriate GPE. In an ordinary PlanetLab node, the remote client opens a connection to the node's SSH server, which goes through the authentication process, and if that succeeds, forks a process running in the appropriate vServer, which transparently acquires the connection, along with its associated TCP kernel state.

In order to emulate this process, without requiring major changes to the PlanetLab OS kernel, we redirect all incoming SSH connections to the CP (using filters in the Line Card), which does the login authentication. The ideal solution at this point would be to fork a process, and then migrate that process to the GPE hosting the correct vServer. However, in the absence of a general process migration mechanism, we have chosen instead to use a *relay process* that runs in the CP on behalf of the slice. This relay process opens a second SSH connection to the SSH server on the selected GPE and forwards traffic at the application level.

We recognize the obvious performance drawbacks of this approach, but consider them acceptable given the relatively limited amount of traffic that must be relayed on behalf of users' login sessions. While there are alternative approaches that avoid application level forwarding, they are not transparent to users, and we consider user transparency the more important consideration here.

7. EVALUATION

To evaluate the system, we implemented two different applications and studied their performance, relative to conventional PlanetLab implementations. The first is an IPv4 router that we developed from scratch. The second is a port of the Internet Indirection Infrastructure (I3). In this case, we restructured the system into fast path and slow path sections and mapped the fast path onto an NPE.

7.1. IPv4 router

Our IPv4 router uses the NPE to implement normal packet forwarding and uses a vServer running on one of the GPEs to implement control functions and exception handling. The TCAM in the NPE allows us to implement both conventional IP routing and general packet filtering, allowing arbitrary subsets of packets to be mapped to different queues. We compare the performance of the NPE-based forwarder to a forwarder using Click [KO00], running in a vServer.

For the experiments reported here, the IPv4 router is configured with five externally visible UDP ports, which are mapped to five different physical interfaces on the LC. The LC demultiplexes received packets based on their UDP port numbers and forwards packets for the router to the NPE. The NPE's packet schedulers are rate controlled to limit their sending rate to 800 Mb/s. Figure 7 shows the results from a basic operational test that demonstrates the operation of all the major subsystems and verifies the fair queueing mechanisms. Each of the five inputs sends traffic into the router at times that are offset from one another. All the traffic is destined for the same output, and the traffic from each input is mapped to a different queue at that output, with each queue getting a different share of the output bandwidth (the shares are 30%, 25%, 20%, 15% and 10%). The top chart shows the rates from each input as solid lines and the output rates from each queue in incremental form as dashed lines (the incremental form means that the top dashed line represents the sum of the rates coming from all the inputs). The bottom chart shows the lengths of the

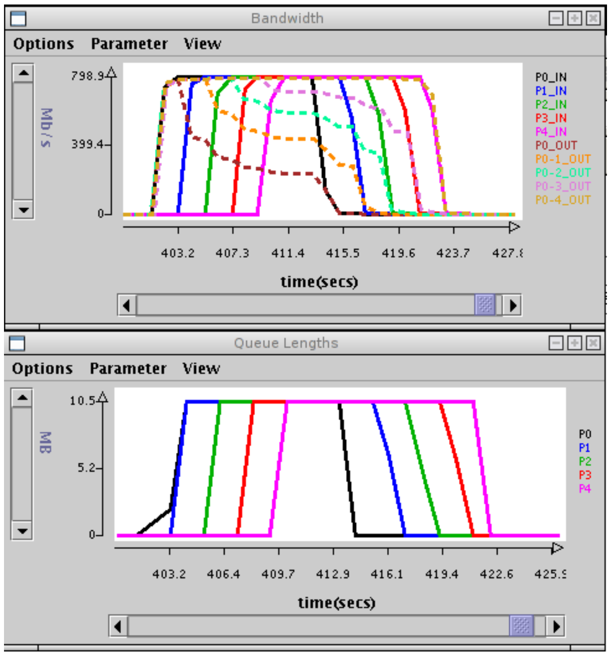


Figure 7. IPv4 router basic operational test showing bandwidth used at inputs and outputs (top) and queue lengths

queues. Since the scheduler is work-conserving, the output rates change as new inputs turn on and as queues drain. These charts are real-time performance measurements obtained by polling the statistics counters maintained in the LC and NPE and accessed through the MP.

Figure 8 shows a somewhat more complex scenario involving traffic to all five outputs. The five traffic sources send the same mix of traffic at all times. In each phase, one of the outputs receives 1.6 Gb/s of traffic, one receives no traffic and the remaining three receive 800 Mb/s of traffic (with each output receiving an equal amount of traffic from each of the five inputs). Each output accumulates a backlog during the period that its input rate is 1.6 Gb/s, and this backlog is cleared when the input traffic to that output turns off. The chart shows the input rates (in incremental form) as solid lines. The output rates (also in incremental form) are shown as dashed lines. Note that for a short period after the second and third input rate transitions, the output rate briefly rises to 4 Gb/s as the previously accumulated backlog for the newly idle output is forwarded, along with the traffic to the other four outputs. These periods show up as brief blips on the display, which is sampling the traffic counters every 200 ms.

Figure 9 shows the results from a large collection of throughput measurements on both the NPE-based router and the Click router. Both routers are configured with five externally visible UDP ports mapped to five different physical interfaces, and input traffic is distributed uniformly across the five ports. The numbers labeling the curves are the sizes of the payloads carried by the packets. The lengths of the frames carried on the external link are 98 bytes longer (this includes two UDP/IP headers and Ethernet overhead, including preamble, VLAN tag and inter-packet gap). The bandwidths are the actual link bandwidths, including all

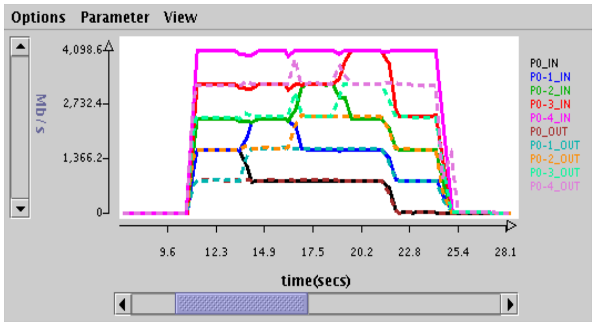


Figure 8. Basic IPv4 router throughput demonstration showing input rates (solid) and output rates (dashed)

overheads. Note that the chart uses a log-log scale. We observe that for 0 byte payloads, the maximum throughput that the Click router is able to achieve is under 50 Mb/s and that its performance deteriorates dramatically as loads increase beyond its maximum capacity. The NPE-based router is able to keep up with the input up to a rate of 3.7 Gb/s. For larger payload sizes, the NPE router can sustain the full 5 Gb/s. For Click, the maximum packet processing rate is about 59 Kp/s, while for the NPE router, it approaches 4,800 Kp/s, an 80 times improvement. The highest throughput for the Click router is 540 Mb/s.

Most of the Click results are for a single processor core, even though the server blade has two dual-core processors. We also show results for 400 byte packets using all four cores, with traffic being distributed across four vServers, each running its own Click router. While one might expect this configuration to achieve four times the throughput of the single core, in fact it only achieves roughly twice the throughput of the single core. There are several possible explanations for this deficiency. We think that the most likely explanation is that at high loads, a large share of the processing capacity is being used by the operating system, and there is insufficient parallelism in the OS to take full advantage of all four cores. Note that since the NP blade contains two independent

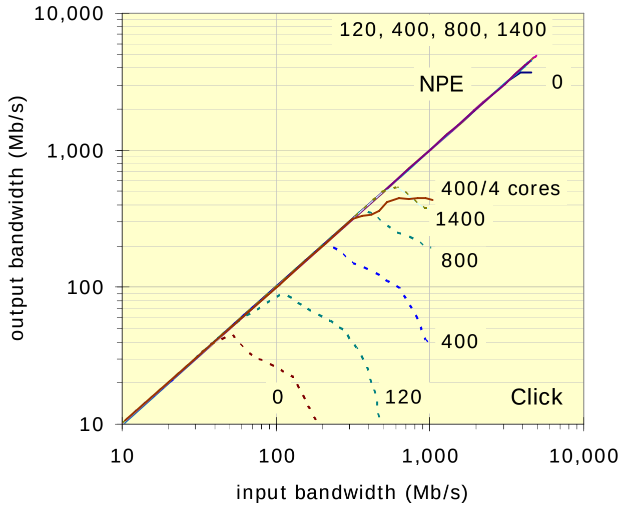


Figure 9. Comparison of throughput for various payload sizes (add 98 bytes to include all overheads); most Click results are for single core; ones uses 4 cores.

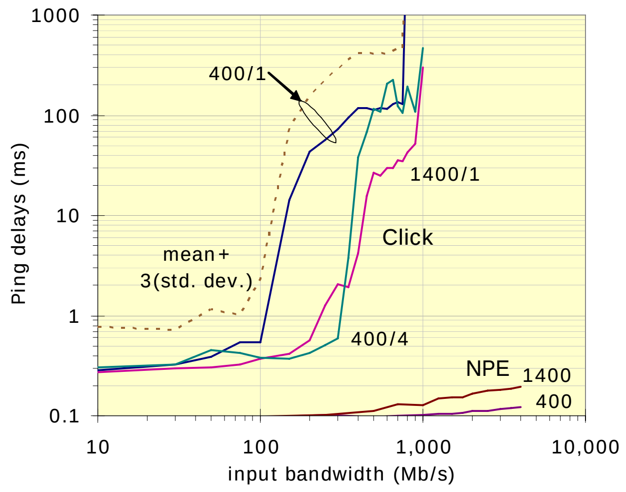


Figure 10. Comparison of latency for payloads of size 400 and 1400; results for 1 and 4 cores for 400 byte payloads; mean+3(standard deviation) shown for 400/1

NPEs, it can achieve a maximum packet processing rate of 9,600 Kp/s, while the server blade has a maximum packet processing rate of about 120 Kp/s, when using all four cores to process minimum size packets.

Figure 10 shows the results of a series of latency measurements. We are particularly interested in understanding how the sharing of a component (GPE or NPE) among multiple PlanetLab slices affects the latency. Consequently, we configured eight instances of the router application on an NPE and compared this with the eight Click routers running in separate vServers on a GPE. For each data point in Figure 10, the routers were supplied with a background traffic load with the total input rate and payload sizes shown in the chart (with each of the eight routers receiving one eighth of the input traffic). We then sent ping packets through the loaded routers using separate logical interfaces (so the ping packets were not subjected to queueing delays in the routers). Each data point on the solid curves is the average of 2000 measurements. We show results for 400 and 1400 byte payloads, and for the Click router, we show results using just a single core and using all four cores for the case of 400 byte payloads. For the 400 byte case with a single core, we also show the mean plus 3 times the standard deviation of the delay, in order to show the variability of the delay. These data indicate that there is a non-negligible fraction of the traffic that experiences delays that are as much as 5 times the mean delay. For the NPE-based routers, the average round-trip ping delays never exceeded 0.2 ms and the standard deviation was generally a small fraction of the mean. In the NPE case, the ping traffic shares queues with the background traffic within the fast path. This is why we observe larger delays when the background traffic has 1400 byte payloads. For the Click routers, the average ping delays remain small until the input rate starts to approach the maximum rate that can be sustained. It then rises sharply, with average delays well above 10 ms. We note that while four cores does lead to better throughput, it does little to limit the latency, once the throughput limit has been reached.

The data in Figure 10 use the standard PlanetLab scheduling parameters. We also experimented with different choices, expecting that as we reduced the number of tokens allocated to each vServer, the latency would drop as the cycle time of the scheduler dropped. We found, to our surprise, that the scheduling parameters had a negligible effect on latency (or throughput), under the traffic conditions used in this experiment. While we have not been able to confirm it, it appears that when the system enters overload, the Click router is not appropriately balancing the time devoted to input processing with the time devoted to output processing. In addition, a significant fraction of the processing time is being taken up by the IP stack in the operating system, which must move arriving data from the network device driver queues into the socket buffers used by the vServers. Since ping packets can get delayed behind the packets that make up the background load in the device driver queues, they are directly affected by the background load, even though they have separate socket buffers and pass through separate queues within the Click routers.

7.2. Internet Indirection Infrastructure

The Internet Indirection Infrastructure (I3) [ST02] is a novel network architecture that explores the use of indirection as an underlying mechanism for giving users greater control over the traffic they receive. Instead of traffic being sent directly to a destination address, it is sent to a user-defined identifier, called a *trigger*. Triggers are defined within a flat identifier space, and the responsibility for handling packets labeled by different triggers is distributed over the I3 routers. The I3 routers use Chord-style forwarding [ST01] to deliver each packet to the node responsible for its trigger. In the simplest (and presumably most common) case, when a packet reaches the router responsible for its trigger it finds a single filter matching its trigger value, which specifies the address of the destination to receive the packet. By having their packets sent indirectly through triggers, users can more easily shield themselves from unwanted traffic and can shield their communicating peers from changes in their actual address, making support for mobility very straightforward. I3 also allows packets to match multiple filters, facilitating multicast; it also supports more complex trigger processing, including packets with “stacks” of triggers and user-defined “remapping” of trigger identifiers.

I3 has been implemented on PlanetLab and we used the publicly available I3 implementation as the basis for the results reported here. We first installed, configured and verified the operation of the standard I3 implementation on our GPE, and took a set of baseline performance measurements of this configuration for comparison purposes. We then created a hybrid implementation, with the I3 fast path running on the NPE, and the slow path on the GPE. The fast path does all the Chord-level forwarding. So, if a router receives a packet with a trigger that lies outside the range of values that the router is responsible for, the router does a lookup in the fast path’s Chord finger table to forward it to the next I3 router. The fast path also handles simple trigger processing. So, if a router receives a packet with a trigger that is within the range of values it is responsible for, and the fast path has a

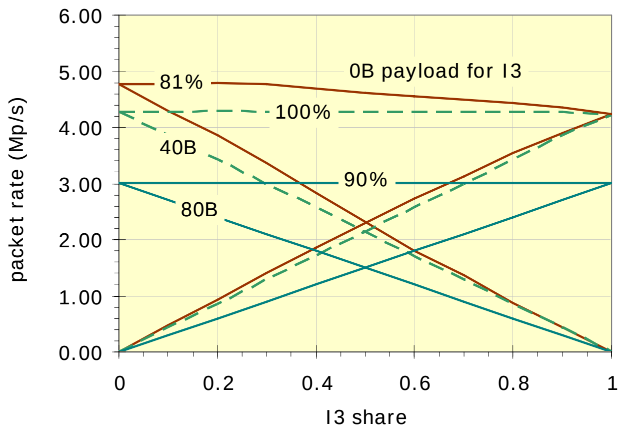


Figure 11. Throughput for combined I3 and I4 traffic with I3 payload lengths of 0, 40 and 80. Input rate held constant at 5 Gb/s, while IPv4 share is varied.

single matching filter in its TCAM, the packet is forwarded directly by the NPE. In our current implementation, all other packets are handled by the GPE. In particular, packets matching multiple triggers or requiring more complex trigger processing are forwarded to the GPE for processing. Some of these more complex cases could also be shifted to the fast path, but in this initial set of experiments, we chose to limit the NPE's role to only what we expect to be the most common cases.

For these experiments, we have focused on the datapath, so only minimal changes to the supplied I3 code were needed (about 350 lines were added to deal with the interaction with the NPE). A complete system would also require the control mechanisms needed to allow the GPE to configure the Chord forwarding tables in the fast path and insert filters to match triggers. For the experiments described here, both of these were manually configured.

We started by performing a set of baseline throughput and latency tests similar those we did for the IPv4 application. The results are qualitatively similar, although the version of the I3 router that runs on the GPE achieves throughput that is generally 30-40% higher than the Click router. We next did a test to verify the operation of the NPE when hosting both the IPv4 and I3 applications. For these tests, we held the total input traffic at 5 Gb/s and varied the fraction of traffic for the I3 application. The results are shown in Figure 11. There are 3 sets of curves, one for I3 payload lengths of 0 bytes, one for 40 byte payloads and one for 80 byte payloads. The IPv4 packets had slightly larger payload lengths, in order to match the overall packet length of the I3 packets (the I3 header is 7 bytes longer than the IPv4 header). For each set, we show the output packet rate for the I3 application, the IPv4 application and the sum of the two. Each of the "sum" curves is labeled with the percentage of the input rate that is achieved in each case. For 0 byte payloads, we can keep up with 81% of the input rate when all the traffic is IPv4. For 40 byte payloads, we can keep up with the full input rate in all cases and for 80 byte payloads, we are handling 90% of the input rate. This last result is somewhat anomalous and further study is needed to explain it.

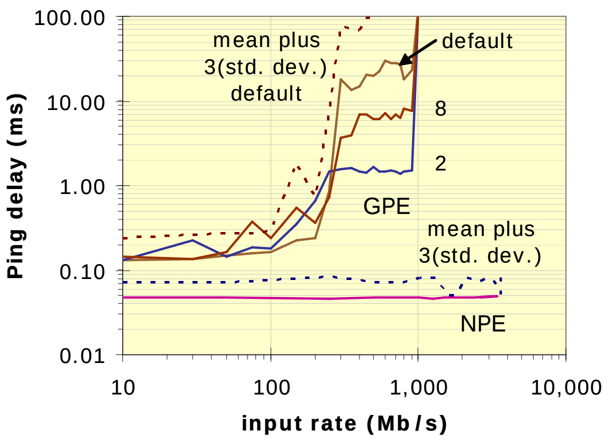


Figure 12. Latency results for I3 using separate instance of I3 application for ping traffic and comparing effect of scheduling parameters on GPE latency

Figure 12 shows results of a set of experiments to evaluate the latency in the I3 case. These experiments were structured differently from the earlier ones for IPv4. Specifically, we used a completely separate instance of the I3 application to handle the ping traffic (for both GPE and NPE), in order to isolate the ping traffic from the background traffic as much as possible. The background traffic consisted of packets with 400 byte payloads and was distributed across two instances of the I3 application. In the GPE case, we used a single processor core (as noted earlier, using all four cores improves throughput, but has little effect on latency).

The most striking difference between these results and the IPv4 results presented earlier is that the delays are substantially smaller for both the GPE and the NPE. This appears to be largely due to the isolation of the ping traffic. For the NPE, the average delay is essentially constant at just under 50 µs. For the GPE, the default PlanetLab scheduling parameters result in average delays of 15-30 ms when the input rate exceeds 300 Mb/s. Reducing the PlanetLab scheduling parameters (*minToken*, *maxToken*) to 2 reduces the delay by more than a factor of 10. Allowing it to rise to 8 gives a delay of 6-8 ms. We found that reducing the scheduling parameters had just a modest impact on throughput, reducing the maximum forwarding rate by less than 10%. Note that as the number of background instances of I3 is scaled up, the delays in the GPE case can be expected to scale up in direct proportion.

8. RELATED WORK

There are two main categories of previous work that are most closely related to this research. The first concerns high performance implementations of overlay networks and the second concerns high performance, programmable routers, particularly those based on network processors. In the overlay network space, commercial organizations have led the way in developing high performance implementations, but relatively few published descriptions are available. Reference [KO04] describes Akamai's system for delivering streaming audio and video, which includes a description of its cluster-based architecture for the overlay nodes. These systems use general purpose servers linked by an Ethernet

LAN, which is used to multicast streams from the servers handling input processing to the servers forwarding packets to next hops and/or end-users.

In the programmable router space [KA02, SP01] describe an extensible router that uses an IXP 1200 for the fast path processing and reference [CH02] describes a system that places a general purpose plugin processor at each port of a gigabit hardware router. There is a much larger body of work relating to active networking and extensible routers generally, but the vast majority of this work is primarily concerned with other issues than achieving high performance. A more recent piece of related work is [TU06], which describes a proposed design of a backbone router for NSF's GENI initiative [GE06].

9. ALTERNATE APPROACHES

It's natural to ask what other approaches might be available to improve the performance of overlay networks. PlanetLab is already shifting to higher performance servers with dual processor chips and two cores per chip. Four core processor chips are now available and eight core chips are expected soon. As we have seen, while multi-core processors can boost throughput, the increase is not necessarily proportional to the number of processors. Memory and network bandwidth must also be scaled up, and even with appropriate hardware scaling, limited locality-of-reference in networking workloads may lead to poor cache performance, limiting the gains. In addition, the parallelism in the workload must at least match the number of cores, if linear speedup is to be achieved. A PlanetLab node hosting many slices that each require just a small fraction of the system capacity, can potentially achieve such parallelism. However, if the workload places substantial demands on the OS, then the OS must also be highly parallel. In particular, the network stack must take advantage of the multiple cores to avoid becoming a bottleneck. Similarly, slices that require a larger share of the system capacity will have to be written to take advantage of multiple cores if they are to benefit from their presence.

Of course, even if multi-core systems are properly engineered and operating system and application code is structured for parallel execution, there remains the issue of user-space overhead for IO-intensive applications. Such overheads contribute significantly to the limited performance of typical overlay platforms. The most promising approach to overcoming such overheads is to decompose applications into separate fast path and slow path segments, and push the fast path down into the OS kernel or even the network device driver. This can be highly effective, but does trade-off protection and ease of software development for performance. We have chosen to accept that trade-off to take advantage of NPs, and argue that a similar choice must be made to get the most out of multi-core server blades. To ensure safe operation in this environment one must be prepared to place limits on how the fast path is programmed, possibly through the use of specialized languages such as PLAN [HI98].

We should also note that a multi-core server blade, even with 8 or 16 processor cores does not provide a scalable solution to the general challenge of high performance overlay network platforms (although it may suffice for PlanetLab in the near term). A scalable solution requires an architecture that supports systems with tens or even hundreds of server blades. One way to approach this is to use a scalable, shared memory multiprocessor. Such architectures are common in supercomputers designed for scientific computing, but these systems are typically not engineered for the IO-intensive workloads that characterize overlay networks (although they certainly could be).

Another approach is to use a cluster of general purpose server blades, connected by a high bandwidth switch. Low cost 10 gigabit Ethernet switches are now available [FO07] and server blades will soon be routinely equipped with such 10 GbE interfaces. This approach is quite similar to the architecture developed here. While we have chosen a more integrated approach using ATCA components, this difference is mainly a matter of physical implementation, rather than architecture. The more significant difference between the two approaches is our emphasis on the use of network processors for the fast path processing. While we argue that at the moment, NPs offer significant performance advantages for the fast path processing, future improvements in general purpose server blades and operating systems could close the gap.

10. CLOSING REMARKS

This work demonstrates that overlay network platforms with substantially higher levels of performance can be implemented using an integrated architecture that combines general purpose servers with modern network processors. NP systems can outperform general purpose servers by a surprisingly large margin. This is partly due to the richer hardware resources available in the NP, but a large part of the difference comes from the operating systems used on general purpose servers, which were developed and optimized for much less IO-intensive workloads than are found in both network routers and in most overlay network contexts. As the number of processor cores in general purpose systems increases to over the next few years, it's likely that general purpose chips will be able to compete more effectively with NPs. However, reaping the full benefits of such systems in overlay network settings will almost certainly require operating systems that are more IO-oriented and will require that applications be programmed to exploit the parallelism provided by the hardware.

Our experience implementing the IPv4 and I3 routers using the fast path/slow path application structure was very encouraging. In particular, we found it very straightforward to restructure the I3 code to conform to this pattern and we expect that many other PlanetLab applications can be similarly modified. The framework provided by the NPE made it straightforward to add the I3 code option. The C source files required to implement the fast path total less than 2,000 lines, and it took two graduate students less than one week to write the code and verify its operation. While we have not implemented the control software to allow the GPE-resident software to configure the Chord routing tables and insert filters to match triggers, we expect these additions to be fairly routine, since they mainly require the addition of code modules to the xScale, which provides a reasonably friendly Linux-based programming environment.

It's worth noting that one of the key factors that made the re-targeting of I3 so straightforward was that we had a well-engineered existing implementation as our point-of-departure. Our experience suggests that before attempting to build an application for the NPE environment, it is wise to develop a fully functional version for the GPE environment. This can serve as a reference point guiding the decisions for exactly what functions to shift to the fast path. It also facilitates an incremental development strategy with small, easy to manage steps, producing intermediate versions of a system that can be useful on their own.

It's natural to ask how suitable our approach is for applications that are different from the two we have considered here. We believe that while it is likely to be more useful for some applications than others, the approach is widely applicable, since many applications lend themselves to decomposition into a simple, high traffic volume fast path and a more complex subsystem to handle exceptions and control. For applications such as content-delivery networks, the need for disk storage places limits on the role the NPEs can play, but even here, packets may be forwarded across multiple hops before reaching the location storing the information of interest. It seems likely that the associative lookup mechanism provided by the TCAM can be a powerful tool for making the required routing decisions. For network measurement applications, the ability of the NPE platform to eliminate the large and highly variable delays found in general-purpose servers promises more accurate measurements with lower computational effort.

We plan to make the SPP system available as a node in the public PlanetLab infrastructure, once we complete our implementation of the control software. Some additional refinements to the software for the LC and NPE need to be implemented before the public release, but the core functionality is now complete and we expect to have the system available for general use by late 2007.

REFERENCES

[BA06] Bavier, A., N. Feamster, M. Huang, L. Peterson, J. Rexford. "In VINI Veritas: Realistic and Controlled Network Experimentation," *Proc. of ACM SIGCOMM*, 2006.

[BH06] Bharambe, A., J. Pang, S. Seshan. "Colyseus: A Distributed Architecture for Online Multiplayer Games," In *Proc. Symposium on Networked Systems Design and Implementation* (NSDI), 3/06.

[CH02] Choi, S., J. Dehart, R. Keller, F. Kuhns, J. Lockwood, P. Pappu, J. Parwatikar, W. D. Richard, E. Spitznagel, D. Taylor, J. Turner and K. Wong. "Design of a High Performance Dynamically Extensible Router." In *Proceedings of the DARPA Active Networks Conference and Exposition*, 5/02.

[CH03] Chun, B., D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. "PlanetLab: An Overlay Testbed for Broad-Coverage Services," *ACM Computer Communications Review*, vol. 33, no. 3, 7/03.

[CI06] Cisco Carrier Routing System. At www.cisco.com/en/US/products/ps5763/, 2006

[DI02] Dilley, J., B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. "Globally Distributed Content Delivery," *IEEE Internet Computing*, September/October 2002, pp. 50-58.

[FO07] Force 10 Networks. "S2410 Data Center Switch," http://www.force10networks.com/products/s2410.asp, 2007

[FR04] Freedman, M., E. Freudenthal and D. Mazières. "Democratizing Content Publication with Coral," In *Proc. 1st USENIX/ACM Symposium on Networked Systems Design and Implementation*, 3/04.

[GE06] Global Environment for Network Innovations. http://www.geni.net/, 2006.

[HI98] Mike Hicks_ Pankaj Kakkar_ Jonathan T_ Moore_ Carl A_ Gunter_ and Scott Nettles. "PLAN, A packet language for active networks," In *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming Languages*, 1998.

[IXP] Intel IXP 2xxx Product Line of Network Processors. http://www.intel.com/design/network/products/npfamily/ixp2xxx.htm.

[KA02] Karlin, Scott and Larry Peterson. "VERA: An Extensible Router Architecture," In *Computer Networks*, 2002.

[KO00] Kohler, Eddie, Robert Morris, Benjie Chen, John Jannotti and M. Frans Kaashoek. "The Click modular router," *ACM Transactions on Computer Systems*, 8/2000.

[KO04] Kontothanassis, L. R. Sitaraman, J. Wein, D. Hong, R. Kleinberg, B. Mancuso, D. Shaw and D. Stodolsky. "A Transport Layer for Live Streaming in a Content Delivery Network," *Proc. of the IEEE, Special Issue on Evolution of Internet Technologies*, 9/04.

[PA03] Pappu, P., J. Parwatikar, J. Turner and K. Wong. "Distributed Queueing in Scalable High Performance Routers." *Proceeding of IEEE Infocom*, 4/03.

[PE02] Peterson, L., T. Anderson, D. Culler and T. Roscoe. "A Blueprint for Introducing Disruptive Technology into the Internet," *Proceedings of ACM HotNets-I Workshop*, 10/02.

[RA05] Radisys Corporation. "Promentum™ ATCA-7010 Data Sheet," product brief, available at http://www. radisys.com/files/ATCA-7010_07-1283-01_0505_datasheet.pdf.

[RH05] Rhea, S., B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica and H. Yu. "OpenDHT: A Public DHT Service and Its Uses," *Proceedings of ACM SIGCOMM*, 9/2005.

[SP01] Spalink, T., S. Karlin, L. Peterson and Y. Gottlieb. "Building a Robust Software-Based Router Using Network Processors," In *ACM Symposium on Operating System Principles* (SOSP), 2001.

[ST01] Stoica, I., R. Morris, D. Karger, F. Kaashoek and H. Balakrishnan. "Chord: A scalable peer-to-peer lookup service for internet applications." In *Proceedings of ACM SIGCOMM*, 2001.

[ST02] Stoica, I., D. Adkins, S. Zhuang, S. Shenker, S. Surana, "Internet Indirection Infrastructure," *Proc. of ACM SIGCOMM*, 8/02.

[TU06] Turner, J. "A Proposed Architecture for the GENI Backbone Platform," In *Proceedings of ACM- IEEE Symposium on Architectures for Networking and Communications Systems* (ANCS), 12/2006.

[VS06] Linux vServer. `http://linux-vserver.org`