Can Software Routers Scale?

Katerina Argyraki[1], Salman Baset[2], Byung-Gon Chun[3], Kevin Fall[4], Gianluca Iannaccone[4], Allan Knies[4], Eddie Kohler[5], Maziar Manesh[4], Sergiu Nedveschi[6], Sylvia Ratnasamy[4]
[1] EPFL, [2] Columbia University, [3] ICSI, [4] Intel Research, [5] UCLA, [6] UC Berkeley

ABSTRACT

Software routers can lead us from a network of special-purpose hardware routers to one of general-purpose extensible infrastructure—if, that is, they can scale to high speeds. We identify the challenges in achieving this scalability and propose a solution: a cluster-based router architecture that uses an interconnect of commodity server platforms to build software routers that are both incrementally scalable and fully programmable.

1. INTRODUCTION

A major factor in the Internet's supposed architectural ossification [12, 18] is the inflexibility of routers near the core of the network: it is simply too difficult to deploy any change that might impact core-router behavior. Some of this difficulty is due to providers' natural conservatism, but much of it is because hundreds of millions of dollars of deployed hardware would require an upgrade on any change. Software routers, in contrast, perform all significant processing steps (*i.e.*, per-packet protocol processing, route loookup, forwarding) in software running on commodity servers. Commodity PCs are attractive for several reasons, including low cost due to large-volume manufacturing, familiar programming environment and operating systems, and widespread supply and support chains. Programmable architectures built around custom network processors and switch fabrics lose many of these advantages.

Unfortunately, today's software routers do not scale beyond the 1-3 Gbps range, while currently available carrier-grade equipment starts at 40 Gbps and scales as high as 92 Tbps [3]. As a result, software routers are reflexively written off as irrelevant for anything larger than a small enterprise network. But is this performance cap fundamental or just a consequence of current software-router architectures? Motivated by recent advances in the I/O capabilities of general-purpose platforms [2, 4], we revisit this question, *i.e.*, we ask whether it is feasible to aim for a general-purpose network infrastructure that is easily extensible and yet capable of scaling to high speeds.

A basic limitation of today's software routers comes from the fact that they adopt a "single server as router" architecture: given the evolution of general-purpose platforms, a single-server architecture is unlikely to ever reach carrier speeds and, hence, the extensibility enabled by software routers is unlikely to impact the Internet's vast provider infrastructure. All is not lost, however. In this paper, we propose a *clustered* software-router architecture that uses an interconnect of multiple servers to achieve greater scalability.

Consider a router with N ports, each with line rate R bps. As N and R increase, router *per-packet processing* such as lookups and classification must scale with $O(R)$, while internal *switching* from input to output ports must scale with $O(NR)$. Current carrier-grade equipment typically uses special-purpose hardware (network processors and switch fabrics) for these tasks.

Instead, we aim to build such an N-port router using a cluster of server-class PCs. Each server handles one incoming line, and the N servers are interconnected to switch packets from input to output lines. Scaling this architecture to high N and R thus faces two challenges:

- the per-packet processing capability of each server must scale with O(R)
- the aggregate switching capability of the server cluster must scale with $O(NR)$

In this paper, we present preliminary work that explores the feasibility of meeting these two scaling challenges: in Section 2, we measure and analyze the per-packet processing performance of current and future commodity-PC platforms; in Section 3, we examine the switching problem and propose server-cluster topologies and routing algorithms drawn from the parallel-computing literature. As a first step, we consider forwarding and route lookup and postpone considering more advanced router features (multicast, ACLs, QoS, and advanced buffer management) to future work.

Our preliminary results are encouraging: back-of-the-envelope analysis combined with experiments on an unoptimized PC platform show that cluster-based software routers could indeed offer the scalability required of carrier-grade equipment; however, careful software engineering (of network drivers and the OS network stack) will be required to actually achieve this scalability. Moreover, important issues like power usage, form factor, and reliability remain to be studied before we can draw conclusions about the viability of our proposal. These are all non-trivial issues, often ignored in the context of research proposals; yet we believe they are worth studying, as their successful resolution could enable a network infrastructure that is programmable and extensible in much the same sense as desktops and servers are today.

2. SCALING PER-PACKET PROCESSING

The feasibility of our cluster-based architecture depends on whether the packet-processing capability of a single PC can scale with line rate. Surprisingly, there appears to be no conclusive investigation of this question in the literature. Most prior research within the systems community has tended towards a "run it and see approach," reporting performance in the context of a specific platform [15, 22]. While detailed performance modeling is common within the computer-architecture community, these studies have not typically focused on router-like workloads [7]. Standard benchmarks such as TPC-C are not relevant since they differ significantly in characteristics like I/O load and cache locality.

PRESTO'08, August 22, 2008, Seattle, Washington, USA.

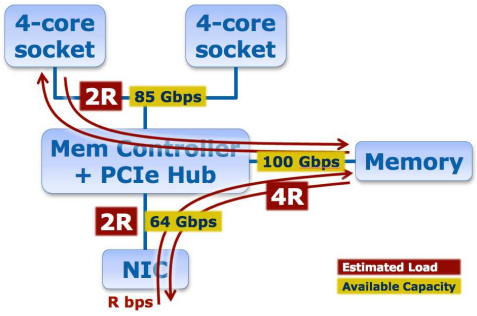

Figure 1: *Traditional shared bus architecture*

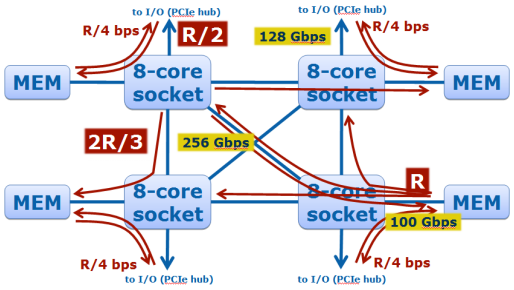

Figure 2: *Point-to-point architecture*

A long-term goal of this work is to close this gap by developing a model of packet-processing workloads that, given a certain PC architecture, *predicts* packet-processing performance. A key challenge is identifying the right modeling granularity—one that requires the minimum number of parameters, yet leads to accurate predictions (*e.g.*, within a factor of 2 of reality).

As a first step towards addressing this challenge, this paper explores two extreme approaches: first, we construct a highly simplified model using back-of-the-envelope analysis over a high-level view of PC architectures; second, we experimentally measure the packet-processing rate achievable on a current PC platform. Together, these approaches provide achievable-performance bounds: the back-of-the-envelope analysis is wildly optimistic and provides an upper bound that indicates whether line-rate scaling is at all plausible; by contrast, the experimental analysis relies on a currently available, unoptimized platform and, hence, provides a lower per-packet performance bound.

2.1 A Strawman High-level Model

Figures 1 and 2 depict at a high level two different PC architectures: (1) a shared-bus architecture, *e.g.*, [1], and (2) a recent multi-processor mesh architecture, *e.g.*, [4].

We start with the shared-bus architecture of Figure 1. A front-side bus (FSB) is used for communication between the CPU(s),[1] memory controller, and primary I/O bus controller—PCI Express (PCIe) hub. This high-level view ignores many details such as the cache hierarchy, the discrete buses that form the FSB (*i.e.*, address and data bus), the presence of multiple memory channels, and the actual memory layout (*i.e.*, ranks and banks); we use it as an example of a coarse-grained model of the hardware architecture.

To estimate this architecture's packet-processing capability, we consider the following *minimum* set of high-level operations, typically required to forward an incoming packet:

1. The packet is DMA-ed from NIC to main memory (incurring one transaction on the PCIe and memory bus).
2. The CPU reads the packet header (one transaction on the FSB and memory bus).
3. The CPU performs additional packet processing including route lookup (CPU-only, assuming no bus transactions—we return to this assumption later).
4. The CPU writes the modified packet to memory (one transaction on the memory bus and FSB).
5. The packet is DMA-ed from memory to NIC (one transaction on the memory bus and PCIe bus).

Based on this model, forwarding a single packet results in 4 transactions on the memory bus and 2 on each of the FSB and PCIe buses; thus, a line rate of R bps leads to a load of $4R$, $2R$, and $2R$ on each of the memory, FSB, and PCIe buses.[2] Currently available technology advertises memory, FSB, and PCIe bandwidths of approximately 100Gbps, 85Gbps, and 64Gbps respectively (assuming DDR2 SDRAM at 800MHz, a 64-bit wide 1.33GHz FSB, and 32-lane PCIe1.1); in the context of our model, these numbers suggest that a current shared-bus architecture should sustain line rates up to $R = 10$ Gbps, and that the packet-processing bottleneck is the memory bus.

We now consider the multi-processor mesh architecture from Figure 2. This architecture replaces the FSB by a mesh of dedicated point-to-point links, removing the bottleneck for inter-processor communications and allowing higher operating frequencies than the traditional multi-load FSB. Moreover, the point-to-point architecture replaces the single external memory controller shared across sockets with a memory controller integrated within each CPU; this leads to a dramatic increase in aggregate memory bandwidth, since each CPU now has a dedicated link to a portion of the overall memory space. On a 4-socket system as shown, the incoming packet stream is spread across the 4 memory controllers integrated in the CPUs. Hence, each packet contributes 4 memory-bus transactions, 4 transactions on the inter-socket point-to-point links, and 2 PCIe transactions; since we have 4 memory buses, 6 inter-socket links and 4 PCIe links, a line rate of R bps yields loads of R, $2R/3$, and $R/2$ on each of the memory, inter-socket, and PCIe buses respectively. If we (conservatively) assume similar technology constants as before (memory, inter-socket, and PCIe bandwidths at 100Gbps, 85Gbps, and 64Gbps respectively), according to our model, emerging server architectures should scale to carrier-grade line rates of 40Gbps and higher.

CPU resources. In addition to bus loads we need to consider whether the available CPU resources suffice to handle packet processing at line rates. We first consider the number of cycles available for packet processing based on the offered line rate and server architecture. Assuming 40-byte packets, packet interarrival time is 32ns and 8ns for R =10Gbps and R =40Gbps respectively. Current shared-bus architectures offer up to 8 cores running at speeds up to 3 GHz, *i.e.*, a budget of 3072 and 768 cycles/pkt respectively. Assuming a cycles-per-instruction (CPI) ratio of 1, this suggests a budget of 3072 (768) instructions per packet for line rates of 10Gbps (40Gbps). Similarly, multi-processor mesh architectures are projected to offer up to 32 cores at similar speeds, *i.e.*, a budget of 12288 and 3072 instructions/pkt for 10Gbps and 40Gbps respectively. Hence, current shared-bus architectures may have sufficient CPU resources to scale to 10Gbps but not 40Gbps, while emerging servers may scale even to 40Gbps.

Given the above cycle budgets, IP route lookup cannot afford multiple cache misses while accessing the forwarding table. Fortunately, increasing cache sizes should help here—existing server caches are already at 8MB/socket, and emerging servers are pro-

[1] We use the terms CPU, socket, and processor interchangeably, to refer to a multi-core processor.

[2] This estimate assumes that the entire packet is read to/from the memory and CPU for processing.

jected to have as much as 16 or 24MB/socket [4]. Caching can be exploited in two ways: (1) to store the entire forwarding table as described by Gupta *et al.* [11] or (2) simply to store the routing entries for popular prefixes. Multiple studies [10,20] have shown that the vast majority of traffic is destined to a small fraction of prefixes and, hence, we expect that caching popular prefixes should allow most route lookups to be resolved from the cache.

Clearly, our analysis is overly optimistic: it assumes 100% efficiency in bus usage, the ability to exploit multicore to its fullest, and no interference from the operating system during packet processing. In the following section, we use experiments to estimate the extent to which these assumptions lead us to overestimate performance.

As a final note, we address a possible source of confusion caused by the difference between traditional router platforms and PCs. Scaling analyses for router performance have traditionally been concerned with memory *access times* (as opposed to memory bandwidth), as these improve at a significantly slower rate than link speeds [5]. Memory latency refers to the time it takes a CPU to read or write a given memory location; current latencies are in the order of 100ns, leading to a maximum bandwidth (for the same location) that is two orders of magnitude slower than the maximum memory bandwidth. Access time, however, is not a major concern on general-purpose processors that pipeline memory operations: CPUs can maintain a large number of in-flight memory requests to hide access delays, while memory request queues are sized to allow full use of the available FSB bandwidth.

2.2 Using experimentation

We now turn to experimental evaluation to bound the inaccuracy due to our simplified model from the previous section. For our experiments, we use a mid-level server machine running Click [15]. Our server is a dual-socket 1.6GHz quad-core CPU with an L2 cache of 4MB and two 1.066GHz FSBs. With the exception of the CPU speeds, these ratings are similar to the shared-bus architecture from Section 2.1 and, hence, our results should be comparable. The machine has a total of 16 1GigE NICs. We generate (and terminate) traffic using similar servers with 8 GigE NICs. In the results that follow, where the input rate to the system is under 8Gbps, we use one of our traffic generation servers as the source and the other as sink; for tests that require higher traffic rates each server acts as both source and sink allowing us to generate input traffic up to 16Gbps.

We instrument our servers with Intel EMON, a performance monitoring tool similar to Intel VTune, as well as a proprietary chipset-specific tool that allows us to monitor memory-bus usage. As before, our goal for now is to understand the fundamental capability of the overall platform to move packets through without considering more sophisticated packet processing. Hence, we remove the IP forwarding components from our Click configuration and simply enforce a static route between source and destination NICs. We have 16 NICs and, hence, use 8 kernel threads, each pinned to one core and each in charge of one input/output NIC pair. We disable NIC interrupts and use interface polling. This setup is admittedly artificial, but our goal is merely to take a first step towards relating hardware specifications to achievable software performance—we recognize that much additional work remains for a comprehensive performance analysis.

Bottlenecks. The first question we explore is what forwarding rates we can feasibly achieve with the existing software/hardware and what are the likely bottlenecks. For this, we look at the loss-free forwarding rate the server can sustain under increasing input packet rates and for various packet sizes. We plot this sustained rate in terms of both bits-per-second (bps) and packets-per-second (pps) in Figures 3 and 4 respectively. We see that, in the case of larger packet sizes (1024 bytes and higher), the server scales to 14.9 Gbps and can keep up with the offered load up to the maximum traffic we can generate. However, in the case of 64 byte packets, performance saturates at around 3.4 Gbps, or 6.4 million pps.

Tracking down the bottleneck that limits forwarding performance for 64 byte packets turns out to require a fairly involved sleuthing process; we detail this process in [6] and only summarize our key findings here. In [6], we look for the bottleneck by examining the load on the four major system components—CPUs, memory bus, FSB and I/O bus. Not unexpectedly, we found that the immediate bottleneck to system performance occurs at the memory system. Surprisingly, however, the bottleneck is *not* in the aggregate memory bandwidth the memory system can sustain but instead due to an unfortunate combination of packet layout and memory chip organization. We were able to partially remedy this issue by modifying the linux memory allocator and this allowed us to improve performance by approximately 30%, for a forwarding rate of 8.2Mpps. Once past this initial bottleneck, the next bottleneck we hit appears to be due to the FSB. Under the covers, the FSB comprises an address and data bus and our tests at 8.2Mpps revealed that, while the utilization levels on the data bus are fairly low, the *address* bus is 70.3% utilized. Prior work [24] and discussions with architects reveal that, as an engineering rule of thumb, an address bus is considered saturated at a utilization of approximately 75%.[3] These results suggest that it is unlikely we could scale this server platform to 10Gbps *as-is*. However, as we discuss in what follows, this performance could be substantially improved through fairly non-radical software and hardware architectural changes.

Loads. The second question we look to answer is: how far off is the naive model from Section 2.1, and why? Again, a detailed evaluation of this question is presented in [6], and we only summarize our key findings here. To understand the per-packet overheads, we measure the load in bits/sec on the FSB and memory bus. The load difference between these two buses give us an estimate for the PCIe load; this is a reasonable approximation since our experimental setup is such that we see little inter-socket communication. Recall that our model from Section 2.1 was only considering the load due to moving packets around. Not surprisingly, we found that the loads we actually measure are significantly higher, indicating that all three buses incur an extra per-packet overhead. We quantify this overhead as the number of per-packet transactions (*i.e.*, transactions that are *not* due to moving packets between NIC and memory) performed on each bus. We compute it as follows:

$$\frac{measured\ load - estimated\ load}{packet\ rate \cdot transaction\ size}$$

We find that the FSB and PCIe overheads start around 6, while the memory-bus overhead starts around 12. All overheads drop slightly as the packet rate increases due to the cache that optimizes the transfer of up to four descriptors with each 64-byte transaction. As explained in [6], this overhead is mostly due to the reading/writing of packet descriptors that our naive model ignored; accounting for the overheads due to descriptors brings the memory and FSB loads close to the estimates of our naive model. Given the model's numerous simplifications, we find this an encouraging (and surprising!) indication that relatively high-level predictive models of packet-processing workloads may indeed be feasible. Moreover, as we discuss shortly, these results also point to the potential to improve performance through a more efficient descriptor architecture.

Improvements, Estimates. Our experimental results suggest a few approaches to improving performance:
Improved packet descriptor handling: We saw above that handling packet descriptors imposes an inordinate per-packet overhead particularly for small packet sizes; a simple remedy might be to have a single descriptor summarize multiple—up to a parameter k—packets. Moreover, handling packets and descriptors separately poses significant additional stress on the memory bus; a possible solution here would be to rearchitect NIC firmware to integrate pack-

[3]Note that this might point to the need to extend our naive model to consider the load on the FSB address and data bus independently (rather than as a single FSB load estimate).

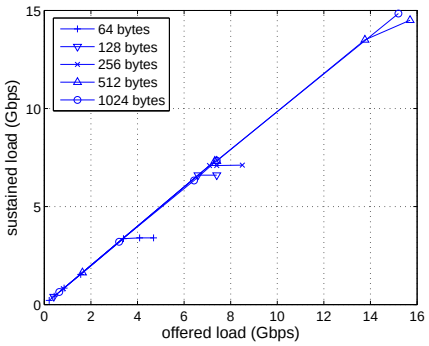

Figure 3: ***Forwarding rate under increasing load for different packet sizes.***

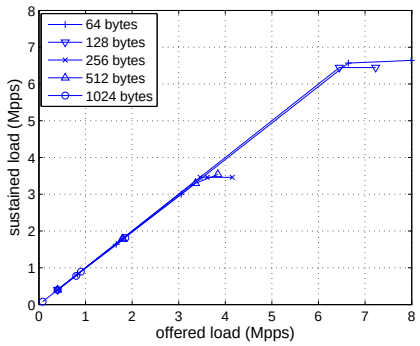

Figure 4: ***Forwarding rate under increasing load for different packet sizes – in pps.***

ets and their descriptors.

Direct I/O: A second technology, this one on the hardware front, that could further reduce bus overheads is Direct Cache Access (DCA) [13] and similar I/O acceleration technologies. In DCA, incoming packets from the NICs are directly placed into the processor caches by snooping the DMA transfer from NIC to memory.

Multi-processor mesh architectures: Perhaps the most promising avenue for improvement—and one that comes "for free"—is to use emerging servers based on multi-processor mesh architectures [4]. Because the mesh architecture replaces the FSB (our current bottleneck) with multiple inter-CPU links, we expect a significant performance improvement by just transitioning to this newer architecture. Specifically, our analysis in [6] shows that we can expect at least a 4x improvement in performance relative to current servers, and this without the above optimizations to packet descriptors.

In summary, current shared-bus servers scale to min-sized packet forwarding rates of 4.4Gbps and we estimate that future mesh-based architectures will scale to over 10Gbps. Moreover, as reasoned in [6], modifying NIC firmware to allow amortized packet descriptors stand to improve these rates to 10Gbps for shared-bus servers and over 40Gbps for mesh-based servers. This suggests that—particularly with a little care—modern server platforms might in fact scale to the higher rates typically associated with specialized network equipment. Admittedly, however, these estimates rely on significant extrapolation and much work remains to experimentally validate our reasoning.

3. SCALING SWITCHING

The switching problem arises because multiple input ports may simultaneously receive packets for the same output port at a rate that exceeds its capacity. Switching, thus, involves selecting which input sends packets to which output port at each point in time and, consequently, which packets are dropped. This must be achieved in a manner that is capable of attaining 100% throughput, services input ports fairly, and does not reorder packets.

High-speed routers meet these goals by using a switch fabric configured by a centralized scheduler, typically implemented in ASICs. As we cannot hope to scale a PC to the speeds of centralized schedulers, we look for alternative switching architectures. Borrowing from the literature on parallel interconnects, we consider interconnecting PCs to form a switching or sorting network, where scheduling is implicit in the routing through the network. Effectively, this transforms the problem of devising scheduling algorithms to one of designing an interconnect topology and routing algorithm to meet the above throughput and fairness requirements. In this paper, we focus on formulating the switching problem as it applies to our cluster-based router architecture and identifying solution options, but defer an in-depth exploration of the design space to future work.

3.1 Switching Problem

A sorting network is typically characterized by:

- a topology T defined by a set of nodes N' connected by a set of links L. Traffic originates/departs at a set of input/output nodes N where $N \subseteq N'$.
- a routing algorithm R that selects paths through T

The literature on parallel interconnects offers a variety of topologies and routing algorithms that have been applied to hardware switch fabrics, supercomputers, L2 switches [9], optical routers [14] and even Distributed Hash Tables (DHTs). The key distinctions for our cluster-based router architecture stem from our goal of building a switching network with traditional throughput and fairness guarantees in software running over commodity PCs. This distinction introduces some non-trivial constraints on the interconnect topology and routing:

Non-determinism: As our "switch fabric" is a collection of servers that implement switching in software using general-purpose OSes, we cannot assume all nodes in the fabric will operate in unison, nor precisely predict the rate of operation at any given node.

Low degree: The per-node connectivity in our switching network is limited by the number of interface slots on a typical server board. Since this number of slots is unlikely to grow significantly, we require interconnect topologies where the per-node degree is sub-linear (*e.g.*, constant, $O(\log N)$, $O(\sqrt{N})$).[4]

Limited node and link speeds: We assume nodes can only process packets at rates of upto O(R), the external line rate. Also, the speed of the inter-server links that form the internals of the switching network is bounded by O(R). These constraints follow from our discussion in Section 2 that even scaling to the external line rate is challenging.

The switching problem is then defined as follows: given our requirements (overall routing capacity, 100% throughput, fairness) and constraints (on fan-out, link and processor speeds), find a solution that minimizes the capital costs of a topology T as determined by the number and capacity of nodes N' and the number and capacity of internal links (*i.e.*, network interfaces).

3.2 Switching Solutions

There are traditionally two broad classes of routing algorithms: deterministic and load-balanced. With deterministic routing, all traffic between a given input and output follow a single predetermined path through the fabric. This offers simple, shortest-path routing and no packet reordering. However achieving 100% throughput for non-uniform traffic patterns requires that interconnect links run with significant speedups relative to the incoming line rate, which violates our third constraint.

[4]Using additional L2 switches can alleviate this issue to some extent, but is expensive for higher N.

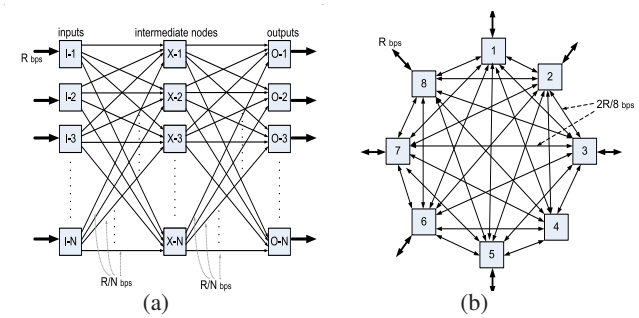

Figure 5: (a) A Valiant load-balanced mesh, (b) Physical topology of an 8-port Valiant mesh.

By contrast, load-balanced routing algorithms exploit a topology's path diversity by splitting traffic along multiple source-destination paths. This more efficiently uses the interconnect's overall capacity and, thus, requires lower-capacity internal links. One of the most promising load-balanced routing algorithms is due to Valiant [8, 23]. In a nutshell, Valiant routing proceeds in two phases: a packet from source s to destination d is first routed from s to a randomly selected intermediate node r and then from r to d. This simple strategy guarantees 100% throughput for a broad class of input traffic. Intuitively, this is because, at each phase of Valiant's approach, traffic appears to be a uniform random workload and can hence fully exploit the capacity of the interconnect at the cost of doubling the traffic load.

Figure 5(a) shows a Valiant load-balanced mesh architecture using two fully connected meshes with internal links of capacity R/N. Incoming *packets* at $I_1 - I_N$ are sent to intermediate nodes $X_1 - X_N$ in a round-robin fashion, where packets are output-queued to the appropriate output $O_1 - O_N$ based on their IP destination. In practice, this architecture could be implemented as a single mesh where a single physical node implements the logical functionality of all three stages and the internal mesh links run at $2R/N$. Figure 5(b) shows this single-mesh for an 8-port router.

There are two issues in directly applying Valiant's approach to our switching problem. The first is that load balancing introduces the possibility of reordering packets within the fabric. In the context of an optical router architecture, Keslassy *et al.* [14] present a novel algorithm that avoids packet reordering by bounding the difference in queue lengths at intermediate mesh nodes; unfortunately, their algorithm relies on predictable, deterministic packet processing by all nodes in the mesh which we cannot achieve in our software environment. Instead, we are considering an approach that exploits packet-flow diversity to achieve load-balance while avoiding reordering. In Valiant's scheme, selecting an intermediate node randomly for every packet in the first phase is ideal for load balancing but can lead to reordered packets. On the other hand, selecting an intermediate node based on the flow (*i.e.*, source/destination IP and port numbers) avoids reordering but can lead to load imbalance. We want to explore an approach in which very short bursts of packets from the same flow that arrive within a time window Δ, which we call a *mini-flow*, are sent to the same intermediate node. This scheme allows us to tradeoff load balancing (and hence the "uniformization") of traffic for bounded packet reordering. We plan to analyze the properties of this mini-flow based scheme and evaluate it using packet-level traces and prototype measurements.

The second issue is that Valiant's approach applied to a mesh topology can be problematic at large N due to our constraints on fan-out. However, one can apply the same load-balanced approach to different topologies leading to different tradeoffs in the per-node degree and link speeds. For example, one could implement Valiant's load-balanced routing over a constant-degree topology such as a torus. For a d-dimension torus with N nodes, one can show that Valiant load-balancing requires an internal node capacity of $\frac{dN^{1/d}R}{2}$ to achieve 100% throughput.[5] Table 1 summarizes the properties of the different switching networks we have investigated. They exhibit different options in the design space, *e.g.*, trading off node-processing capacity for fewer per-node links or the number of nodes for lower node-processing capacity. These examples, whose details we omit due to space constraints, serve to illustrate that multiple feasible solutions exist, each with different tradeoffs, and we leave a rigorous characterization of the design space, costs, and tradeoffs to future work.

network valiant-	N' (total # nodes)	links/ node	link cap.	node cap.
mesh	N	N	$\frac{2R}{N}$	2R
compact(2)	N=k^2	$2(k-1)$	$\frac{4R}{2k-1}$	$4R$
compact(3)	$k(k+1)$	k	$\frac{6R}{k}$	$6R$
embed. butterfly	N	$\log_2 N$	R	$2R\log_2 N$
d-torus	N	$2d$	$\frac{RN^{1/d}}{4}$	$\frac{dRN^{1/d}}{2}$
debruijn	N	2	$R\log_2 N$	$2R\log_2 N$
butterfly	$2N\log_2 N$	2	$\frac{R}{2}$	R

Table 1: *Comparison of different switching networks. N is the number of nodes and R is the external line rate.*

4. RELATED WORK

Over the years, researchers have experimented with a variety of router architectures. Partridge *et al.* [17] built a high-performance software router using a combination of multiple line cards and forwarding engine cards plugged into a high-speed switch in a custom configuration. This was the first multigigabit router with scalability to Gbps line rates using conventional CPUs. Similar architectures were adopted by multiple follow-on efforts, with many using network processors to scale to higher speeds [3, 19, 21]. In recent work, Turner *et al.* describe a Supercharged Planetlab Platform [22] for high-performance overlays that combines general-purpose processors with network processors (for slow and fast path processing respectively) interconnected by a 10Gb Ethernet switch that provides cheaper interconnection (compared to a schedulable crossbar) at the cost of sacrificing 100% throughput and fairness guarantees; they achieve forwarding rates of up to 5Gbps for 130B packets. Mudigonda *et al.* present a novel *malleable* processor architecture that seeks high-performance and programmability through processor-level support for the dynamic reconfiguration of cache capacity and number of hardware threads [16].

Our effort differs primarily in our focus on using only conventional PCs, general-purpose operating systems, and doing so for both high-speed packet processing and switching. More generally though, there are multiple dimensions along which one could compare router architectures—performance, programmability, cost, power, *etc.*—with different architectures offering different tradeoffs. The contribution of this paper is to suggest a new, purely PC-based architecture to those above, but we leave a comprehensive router-architecture comparison to future work. The results of such a study would shed light on the necessity of special-purpose architectures (such as NPs) for network-centric workloads or, alternatively, on the architectural modifications needed to improve the packet-processing capability of PCs.

The Click and XORP routers offer modular software-router architectures, but do not consider scaling beyond a single server platform; through our cluster-based approach we hope to extend their applicability to higher speeds. Finally, there is a large body of work on benchmarking and optimizing network-centric applications on PCs that we hope to leverage in our prototype.

[5] Using a torus gives rise to the question of "incomplete" tori; *i.e.*, for N ports, there may be no integer d such that $N^{1/d}$ is integral. A possibility here is to allow non-uniform partitioning of a d-dimensional torus akin to how DHTs handle a similar issue.

5. CONCLUSION, FUTURE DIRECTIONS

We call for research on building high-speed software routers using low-cost, commodity hardware. We argue the feasibility of scaling server-class PCs in a cluster-based architecture to achieve high-speed packet processing and switching, provided software is developed carefully enough to exploit hardware capabilities. To put our conjectures to the test, we are currently building a 40Gbps prototype using a cluster of 4 servers interconnected by a Valiant mesh (feasible due to the low port count) and interfaced to a control server running XORP.

Our argument has focused on data-plane performance, yet any serious proposal for a new high-speed router architecture would require analysis of features such as power and cooling, footprint, and configuration. Moreover, there is the bigger question of what the new capabilities and resources of such an architecture would enable—from shorter time to market and easily upgradable network equipment to new network services and architectures.

6. REFERENCES

[1] Intel Xeon Processor 5000 Sequence. `http://www.intel.com/products/processor/xeon5000`.

[2] Next-Generation Intel Microarchitecture. `http://www.intel.com/technology/architecture-silicon/next-gen`.

[3] The push of network processing to the top of the pyramid. Will Eatherton, Keynote at ANCS 2005. Presentation available on conference website.

[4] Intel Demonstrates Industry's First 32nm Chip and Next-Generation Nehalem Microprocessor Architecture. Intel News Release., Sept. 2007. `http://www.intel.com/pressroom/archive/releases/20070918corp_a.htm`.

[5] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing router buffers. In *Proc. of SIGCOMM*, 2004.

[6] K. Argyraki, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. Understanding the Packet Forwarding Capability of General-Purpose Processors. Technical Report IRB-TR-08-44, Intel Research Berkeley, May 2008.

[7] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick. The Landscape of Parallel Computing Research: A View from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, Berkeley, CA, Dec. 2006.

[8] C.S.Chang, D.S.Lee, and Y. Jou. Load-balanced Birkhoff-von Neumann switches, Part I: one-stage buffering. *Computer Communications*, 25:611–622, 2002.

[9] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Switches*. Morgan Kaufmann, 2004.

[10] W. Fang and L. Peterson. Inter-AS traffic patterns and their implications. In *Proc. of Global Internet*, 1999.

[11] P. Gupta, S. Lin, and N. McKeown. Routing lookups in hardware at memory access speeds. In *Proc. of IEEE Infocom*, San Francisco, CA, Mar. 1998.

[12] M. Handley. Why the Internet only just works. *BT Technology Journal*, 24, 2006.

[13] R. Huggahalli, R. Iyer, and S. Tetrick. Direct Cache Access for High Bandwidth Network I/O. In *Proc. of ISCA*, 2005.

[14] I. Keslassy, S.-T. Chuang, K. Yu, D. Miller, M. Horowitz, O. Solgaard, and N. McKeown. Scaling Internet routers using optics. In *Proc. of SIGCOMM*, 2003.

[15] E. Kohler, R. Morris, B. Chen, J. Jannotti, and F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, Aug. 2000.

[16] J. Mudigonda, H. Vin, and S. W. Keckler. Reconciling performance and programmability in networking systems. In *Proc. of SIGCOMM*, 2007.

[17] C. Partridge et al. A 50-Gb/s IP Router. *IEEE/ACM Transactions on Networking*, 6(3), June 1998.

[18] L. Peterson, S. Shenker, and J. Turner. Overcoming the Internet impasse through virtualization. In *Third Workshop on Hot Topics in Networks (HotNets-III)*, Nov. 2004.

[19] L. L. Peterson, S. Karlin, and K. Li. OS Support for General-Purpose Routers. In *Proc. of the IEEE Workshop on Hot Topics in Operating Systems*, 1999.

[20] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang. BGP routing stability of popular destinations. In *Proc. of IMC*, 2002.

[21] T. Spalink, S. Karlin, L. Peterson, and Y. Gottlieb. Building a Robust Software-Based Router Using Network Processors. In *Proc. of the 18th ACM SOSP*, 2001.

[22] J. Turner et al. Supercharging planetlab – a high performance, multi-application, overlay network platform. In *Proc. of SIGCOMM*, 2007.

[23] L. Valiant and G. Brebner. Universal schemes for parallel communication. In *Proc. of the ACM STOC*, June 1981.

[24] B. Veal and A. Foong. Performance scalability of a multi-core web server. In *Proc. of ACM ANCS*, Dec. 2007.