Analysis of a Massive Parallel Gigabit Switch built with Commodity Hardware: Can Google Route?

Guido Appenzeller and Matthew Holliman

EE384Y Final Project

ABSTRACT

In this paper, we describe a distributed routing architecture based on the use of commodity components. We evaluate the architecture of the components, discuss their aggregation into a larger system and evaluate its performance.

1. INTRODUCTION AND MOTIVATION

The current design paradigm for a high-end router is based on a centralized switching fabric built from highly optimized, custom hardware. However, the rapid and continuing decline in prices for PC hardware, specifically for both CPUs and Gigabit Ethernet cards, makes it seem possible to consider a different approach. In particular, it seems feasible now to construct a *distributed router* using a relatively large number of commodity PCs and network interfaces (NICs).

The potential benefits of such a distributed architecture have been proven previously for other computation- and bandwidth-intensive applications, examples of which include Google for search and data retrieval and Beowulf Clusters for scientific computing. A brief survey of prices for Gigabit switching and routing hardware suggests that such an approach could be equally attractive for building routers.

Fig. **??** shows a plot of the cost per Gigabit of switching capacity plotted against the total aggregate capacity of the switch. A single Gigabit Ethernet interface today is available for as low as $35. Small switches of up to 12 or 24 ports have a count of about $100 per Gigabit switching capacity [**?**]. Larger switches however have much higher per port cost [**?**]. While this data is admittedly incomplete and pricing high-end switches is non-trivial, there is evidence that the price increase is much faster than linear. At the same time it should be possible to build a large $N \times N$ switch out of in the order of $N \log N$ components. The cost of the components for building large switches out of smaller ones should thus be smaller than the price of today's large switches.

Currently the price of a stripped-down PC is approximately $250, and copper Gigabit Ethernet cards are available for as low as $35. The transfer rate achievable over a 66 MHz, 64-bit PCI bus approaches 4 Gbit/s.

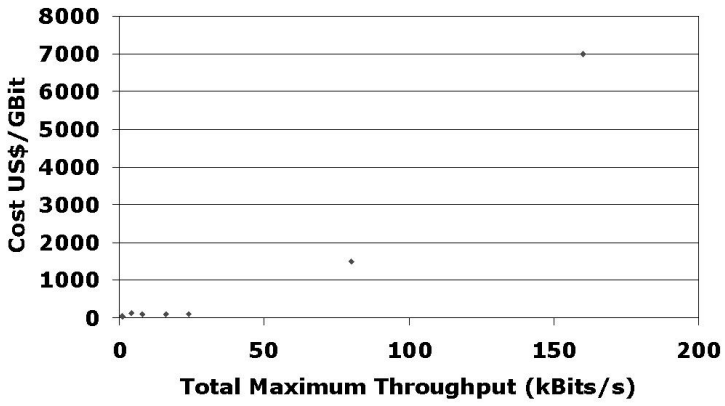

Figure 1. Cost of switches per gigabit switching capactiy for different total aggregate switch capacities. Even with the few data points it is evident that cost per GByte is much higher for high-end switches.

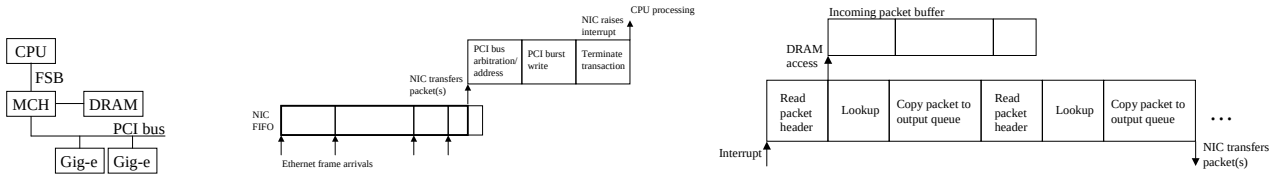

Figure 2. (a) System architecture. (b) Data flow from the NIC during packet arrivals. (c) Data and computational flow in CPU during packet processing.

Even a 33 MHz, 32-bit PCI bus on low-end hardware is capable in principle of supporting Gigabit traffic. Thus, based on back-of-envelope calculations, it seems possible in principle to build a 3×3 or potentially 4×4 switch that can route traffic at 1 GBit/s for $400. These building blocks can be connected to form larger, distributed switching fabrics. A "fabric" of one hundred or so such building blocks has properties that are very different from traditional routers. For example, the fabric features very high aggregate memory bandwidth (over 100 GB/s, even using commodity hardware) and total computational power available (over 100,000 MIPS).

While not necessarily practical as described here, we believe the basic ideas should be implementable, e.g. in VLSI or by building it out of the chipsets currently used for smaller routers. Furthermore, we believe that as additional functionality is integrated into future routers and switches, there will be an increasing focus on software and a corresponding commoditization of hardware, which could lead to techniques such as those described here being of greater interest.

In the remainder of the paper, we propose a system architecture for the purpose of distributed routing. There are two obvious aspects to consider when designing such a system: (i) the design of an individual node, taking into account hardware/software interaction, and (ii) the resulting implications for overall cluster architecture. Section **??** addresses the first of these issues, describing how we can build a 2×2 full-duplex or 4×4 half-duplex Layer 3 Gigabit switch with PC hardware. Specifically, our model predicts that PCI bandwidth, memory bandwidth, and interrupt overheads are the primary bottlenecks in the platform and that these determine both the software model we must use and the hardware on which the software can run in order to switch at line rates. Section **??** discusses how we build larger switching fabrics out of individual components. Section **??** discusses some characteristics of our router, including its stability under admissible traffic and fault tolerance. Finally, Section **??** concludes the paper.

2. COMPONENT ARCHITECTURE

In the following discussion, we consider a stream of arriving packets, each of header length H and payload length P bytes. Fig. **??**(a) shows the relevant portion of the PC system architecture to consider. Packets arrive over the PCI bus and are buffered in DRAM, read by the CPU and written back to DRAM, before departing over the PCI bus. Fig. **??**(b) and (c) depicts the resulting data and computational flow in the NIC and CPU for packet arrivals. There are three obvious potential bottlenecks in the system: (i) the PCI bus, (ii) the memory subsystem, and (iii) the computational capacity itself. We now consider each in turn to show that it is feasible to route Gigabit traffic on PC hardware.

2.1. PCI bus

Assuming each NIC has a FIFO of size F bytes for both reads and writes and the ability to transfer up to N unique IP datagrams per PCI burst transaction, for an $S \times S$ switch of line rate R, the required PCI bus width W_{PCI} (in bits) and clock frequency f_{PCI} are related to the line rate R by

$$R \leq \frac{16 \cdot \min(F, N(H+P))}{2S \cdot [L_{PCI} + 8\min(F, N(H+P))/W_{PCI}] \cdot f_{PCI}}, \quad (1)$$

where L_{PCI} is arbitrary/address latency for the PCI bus (i.e., fixed overhead), in terms of bus clock cycles. This assumes that the NIC transfers incoming packets using a burst transaction such as PCI's "Memory Write Invalidate."

It is well documented that PCI performance degrades quickly for short transactions and that the theoretical maximum bandwidth of $B_{PCI} = f_{PCI} W_{PCI}$ is rarely achieved in real systems [**?**, **?**]. However, in our system, the only devices contending for the PCI bus are the S NICs. Thus bus contention in this case does not reduce the effective throughput of the system relative to the prediction of Eq. **??**, since useful work is still being done, and so the bound imposed by R above on bus performance is sufficient.

One would expect that to implement a full-duplex 2×2 (or half-duplex 4×4) Gigabit switch on PC hardware, 66 MHz 64-bit PCI (with a theoretical maximum bandwidth of around 4 Gb/s) or better would be required. Our model confirms this, and as Fig. **??**(a) shows, also implies that moreover such a switch is constrained in the length of bus transfers in order to meet throughput requirements. Specifically, either the switch is limited to switching at Gigabit rates only for packets of length 767 bytes or more, which is clearly not desirable in practical scenarios, or coalescing of smaller frames to issue burst transactions of this length is strictly required in the NIC. Many current NICs support interrupt coalescing as a standard optimization, so the latter requirement is feasible. Emerging interconnect technologies such as PCI-X or 3GIO would tend towards relaxing these restrictions.

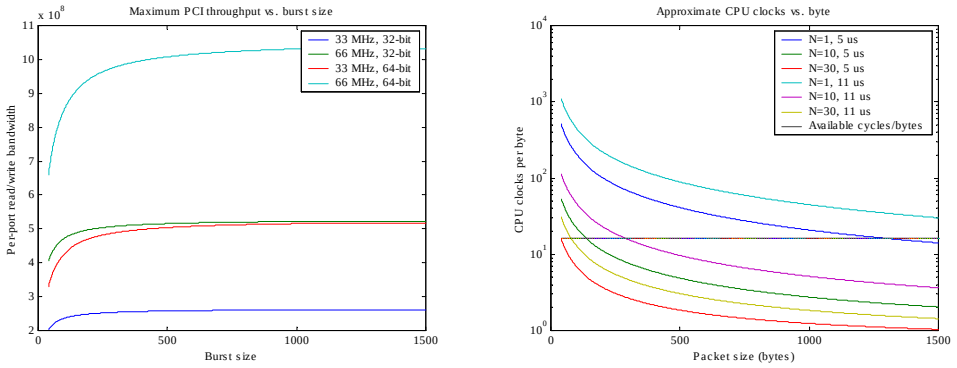

Figure 3. (a) Maximum per-port PCI read/write bandwidth (full-duplex) vs. packet size (assumes 3 cycle bus arbitration latency). (b) Estimated CPU clock cycles required per byte vs. packet size (assumes 2 GHz Pentium 4). N is number of packets coalesced per interrupt, time is interrupt latency.

2.2. Memory subsystem

2.2.1. Memory bandwidth

After the packet lookup, when the packet must be dispatched to the correct NIC, the router can either forward immediately or wait to aggregate more data for the outgoing link. As Fig. **??** indicates, our software model assumes that after a packet is processed, it is queued in a separate output queue for whichever link it is to be transferred over. This requires an extra memory copy beyond the minimum requirement. However, the PCI bus is not pipelined, and address and data are multiplexed on the bus. Furthermore, memory bandwidth is typically an order of magnitude higher than PCI bandwidth in current systems. Thus, assuming the NIC has the capability of handling multiple IP datagrams within a single transaction, framing each separately,* the right trade-off appears to be towards lower PCI bus overheads at the expense of higher memory traffic.

Under this model, and assuming that memory traffic due to lookup cache misses is negligible compared to memory traffic needed to support packet data copies, the total memory bandwidth required in the system is lower bounded by

$$B_{mem} \geq 4RS.$$

For a 2×2 switch operating at $R = 1$ Gb/s, this would result in a memory bandwidth requirement of approximately 1 GB/s. When considering which hardware to target, this would make the use of a Pentium III-class machine (1 GB/s peak bandwidth) infeasible, thus resulting in a requirement for a Pentium 4 (3.2 GB/s peak bandwidth using RDRAM) or equivalent system.

*Given the increasing sophistication of Gigabit-ethernet adaptors, which frequently include functionality such as hardware offloading of IP and TCP checksums and interrupt coalescing, we believe this is plausible.

2.2.2. Cache

The working set of the lookup component corresponds to the lookup table (approximately 150 to 200 KB [?]), plus $N(H+P)$ bytes for N incoming packets. An L2 cache of 256 KB is sufficient for this workload.

2.3. CPU

As Fig. ?? indicates, the CPU time t_{CPU} required to handle N packets arriving from a NIC is divided between notification overhead (one interrupt per PCI read or write transaction if using an interrupt-based model), packet header accesses, lookups, and output queuing, i.e. $t_{CPU} = t_{overhead} + t_{read} + N(t_{lookup} + t_{copy})$.

In the following, we assume the use of the lookup algorithm of [?], and thus approximate $t_{lookup} \leq 100$ clocks. Under the assumption that the router is running on a Pentium 4, sequential accesses to the incoming packet buffer invoke the hardware prefetcher. Consequently, memory latency impacts only the first access to the packet buffer. Thus $t_{read} \approx L_{RAM} + Hf_{CPU}/B_{mem} + (N-1)L_{L2}$, where L_{RAM} is CPU clocks due to memory latency and L_{L2} is latency of L2 cache accesses. Under the same assumption, output queuing clock cycles correspond to copying of data from L2 via a non-temporal burst write to memory, and are thus approximated by $t_{copy} \approx (H+P)f_{CPU}/B_{mem}$.

On current hardware, assuming 1500-byte packets, this would result in a requirement of on the order of approximately 1100 to 1200 clock cycles to process the packet, which could be handled by a processor running at approximately 100 MHz. The inclusion of additional functionality such as statistics and routing table updates would increase the computational requirement, but nonetheless packet processing and routing itself clearly would not be a significant bottleneck in current systems.

Interrupt processing and context switch overheads depend on the software model used in the router. At a minimum, fixed hardware interrupt costs on the x86 architecture are on the order of 5 μs [?, ?], implying that a rate of 200,000 interrupts per second would saturate the CPU, independent of processor frequency. The inclusion of operating system overheads in systems such as Windows 95 have been estimated as resulting in interrupt latencies of 11 μs or greater [?]. For these two cases (i.e., the best-case assumption that interrupt latency is 5 μs, so processing takes place within the interrupt service routine itself, and the usual Linux/Windows device driver model, with an approximate latency of 11 μs), Fig. ?? shows the expected impact of packet size and interrupt coalescing on performance for a 2 GHz Pentium 4 ($t_{overhead} = 2t_{int}$). As can be seen, for the worst-case arrivals of minimum-sized (40-byte) packets, even with the routing software running directly inside the interrupt handler, the CPU could only keep up if 40 such packets are coalesced and delivered per interrupt, thus requiring a FIFO of size at least 1600 bytes in the NIC. Implementing the packet lookup in a deferred procedure call, as would be the normal practice in both Linux and Windows, would require that 88 such packets be coalesced, thus requiring a FIFO of size at least 3520 bytes in the NIC.

Assuming the routing software is the only application running on the machine and that all pages are locked in memory so that page faults are not possible, we can avoid these significant overheads by disabling interrupts and polling for I/O completion instead. In this case, the forwarding software runs underneath the O/S scheduler.

3. CLUSTER ARCHITECTURE

¿From the preceding discussion, we see that with current PC hardware we can build either a 2×2 half-duplex or 4×4 full-duplex Gigabit switch. However, in many applications, higher capacity (either through higher line rates or a larger number of ports) would be desirable. Increasing switching capacity in the PC is not feasible due to fundamental system bottlenecks in current PC hardware as discussed above. Thus, to construct a high-capacity switch, we build a cluster out of smaller switches, as shown in Fig. ??(a).

The specific arrangement we use is a Clos network to build a larger switching fabric out of a collection of smaller ones. As an example, a 16×16 switch with 8 Gb/s aggregate capacity can be built out of twelve 4×4 switches, as shown in Fig. ??(b), thus requiring 12 PCs and 48 network cards. Similarly, a 256×256 switch with 128 Gb/s aggregate capacity can be built out of 576 PCs and 2304 network cards (i.e., forty-eight 16×16 switches).

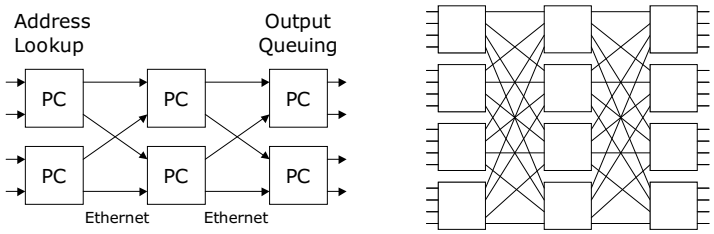

Figure 4: (a) Cluster architecture. (b) 16x16 switch built from 4x4 switching components.

3.1. Scheduling

For an $n = k$ Clos network, a dynamic matching is required for 100% throughput. For large n, e.g. for a 256×256 switch, this is computationally expensive. Furthermore, the distributed nature of the switching fabric makes any kind of centralized switching decision impractical. Coordination between nodes to communicate traffic information from the inputs to a central scheduler would imply additional latency in the forwarding path, and would require additional computational and data path bandwidth in individual nodes.

Consequently, we adopt a decentralized randomized scheduler. Specifically, nodes in the first stage of the switch dispatch packets in round-robin fashion to nodes in the second stage [**?**]. Similarly, middle-stage switches again schedule packets to outputs in round-robin fashion. The scheduling policies in both stages are easily implemented on individual nodes without external coordination.

We also observe that in contrast to Clos networks built out of circuits, intermediate nodes here have the capability to buffer data. Thus the existence of a full matching from input to output is not necessary at the time of scheduling, since intermediate nodes can resend packets if there is a collision.

4. DISCUSSION

4.1. Stability

We consider here the stability of the switch, showing that it provides 100% throughput for any admissible traffic.

To see this, we first consider the middle stage, for which the packet arrival distribution is iid uniform under the first-stage packet forwarding algorithm described above. The middle stage is an input-queued router with k inputs and k VOQs (e.g., $k = 4$) and round-robin scheduling, which is known to provide 100% throughput under iid uniform traffic. Thus the system up through the middle stage is stable.

We now consider the final stage, for which no blocking can occur and thus 100% throughput is achieved if at least one VOQ (i, j) is non-empty for all outputs j. The final stage thus corresponds to an output-queued router with k inputs, which provides 100% throughput by definition. As a result, the system itself also has 100% throughput for any admissible traffic.

4.2. Scalability

Using the proposed method we can recursively build larger $N \times N$ switches using smaller switches. The number $C(N)$ of switches we need for this is given by the recursion:

$$C(N^2) = 3N \cdot C(N).$$

Solving the recursion under the condition that the basic unit is a 4×4 yields:

$$\begin{aligned} \text{Define } C(4) &= 1. \\ C(N) &= \frac{N}{4} 3^{\log_2 \log_4 N} \\ &= \frac{N}{4} \log_4 N \left(\frac{3}{2}\right)^{\log_2 \log_4 N} \text{ components.} \end{aligned}$$

One key question here is whether it is possible to find a better structure to build a switch that would require less components. While this could be possible, we can prove that a multi-stage Clos Network is very close to the optimal solution if we model the switching process as a sorting problem.

Mapping N inputs to N outputs using a sequence of binary (2×2) switches is equivalent to sorting N numbers using a binary comparison operator. It is widely known that any sorting algorithm that is based on binary comparisons will require a minimum of $N \log N$ comparisons. The proof (e.g. [**?**]) notes that the $N!$ permutations of the input sequence can be found in a binary search tree of height $O(N \log N)$ with a small constant > 1. With less than $N \log N$ binary decisions it is not possible to identify the correct leaf.

The basic component of our system is a 4×4 switch. It can do four 4-ary comparisons in one operation. Our proof follows the proof for binary sorting. To find the number of switches necessary to switch N inputs to N outputs we consider the search tree for all possible permutations of inputs. This tree has $N!$ leaves. A 4×4 switch can "sort" four inputs in a single operation; the tree is thus a 4-ary tree with a height of $N \log_4 N$. However our 4×4 switch can perform 4 of these 4-ary comparisons in parallel. Search time is thus only a quarter of the tree height or $\frac{N}{4} \log_4 N$.

The difference between the optimal solution and our solution is a factor of $\left(\frac{3}{2}\right)^{\log_2 \log_4 N}$. This difference is very small (e.g. less than five for $N = 1,000,000$). The extra $O(\log N)$ factor in our complexity is introduced due to the additional (third) stage in the network.

4.3. Fault tolerance and reliability

One potential benefit of adopting a cluster approach is robustness against individual component failures. If a single NIC fails, 1 out of N possible paths through the network is disrupted, i.e., the achievable throughput is reduced by a factor of $1 - 1/N$. On the other hand, if an entire node fails, 4 out of N possible paths through the network are disrupted, i.e., the achievable throughput is reduced by a factor of $1 - 4/N$.

We observe that as the size of the switch grows, the relative effect of individual node failures compared to overall switch capacity diminishes as one would expect. There are no particular hot spots that could cause a large reduction in throughput.

On the negative side the average time to failure of a PC can be as low as 2 years. For our switch this would mean that PC's would have to be replaced with a rate of up to one per day.

4.4. Other requirements

We should note that our switch does not fulfill a number of requirements that are essential to make it attractive to customers that operate network backbones. These are:

- Low Energy Consumption - Our switch would require up to 50 kW of energy, this is more than is permissible for many locations.
- Small Space Requirements - 20–30 racks is substantially more than commercial routers require.
- Maintaining Packet Order - We do not address packet re-ordering that will happen in the first two stages. We can compensate for this in the output buffer however, at the expense of some additional latency. (Under the assumption that packets must be buffered to provide 100% throughput in individual components, as discussed in Sec. **??**, this buffering may come for free).

5. CONCLUSIONS

We have described how to build a scalable Layer 3 Gigabit switch out of commodity hardware (PCs and widely available Gigabit Ethernet cards). Our study shows that it is possible to route even worst-case traffic at 1 Gb/s on current PCs with an appropriate software model (forwarding implemented underneath the O/S scheduler to avoid interrupt and context switch latency) and hardware (1 GB/s memory bandwidth, 0.5 GB/s interconnect bandwidth). A cluster built out of $\Theta(N \log^2 N)$ such nodes can be used as an $N \times N$ switch. The cluster is cheaper than existing commercial switches that have a similar switching capacity and robust against individual component failures. While it is not deployable in its current form, we believe it is a promising architectural approach that warrants further research.

REFERENCES

1. M. L. Loeb, A. J. Rindos, W. G. Holland, and S. P. Woolet, "Gigabit ethernet PCI adaptor performance," in *IEEE Network*, vol. 15, issue 2, Mar./Apr. 2001.
2. D. Robinson, P. Lysaght, G. McGregor, and H. Dick, "Performance evaluation of a full-speed PCI initiator and target subsystem using FPGAs," in *Field Programmable Logic and Applications*, London, England, Sept. 1997.
3. M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, "Small forwarding tables for fast routing lookups," *Proc. ACM SIGCOMM 1997 Conference on applications, technologies, architectures and protocols for computer communications*, Cannes, France, Sept. 1997.
4. C. S. Chang, D. S. Lee, and Y. S. Jou, "Load balanced Birkhoff-von Neumann switches, part I: one-stage buffering," to appear in special issue of *Computer Communications* on "Current issues in terabit switching."
5. *http://www.mathworks.com/access/helpdesk/help/toolbox/xpc/ch_advanced_tutorial9.shtml*
6. *http://www.compuware.co.jp/drivercentral/resources/inter_lat.asp*
7. *http://www.research.solidum.com/papers/ols1999/top.html*
8. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, 1990.
9. MySimon Internet Comparison Shopping Engine. *http://www.mysimon.com*
10. CISCO web site. *http://www.cisco.com*