Internet-in-a-Box: Emulating Datacenter Network Architectures using FPGAs

Jonathan D. Ellithorpe
Computer Science Division
EECS Department
UC Berkeley
Berkeley, CA 94720-1776
jde@cs.berkeley.edu

Zhangxi Tan
Computer Science Division
EECS Department
UC Berkeley
Berkeley, CA 94720-1776
xtan@cs.berkeley.edu

Randy H. Katz
Computer Science Division
EECS Department
UC Berkeley
Berkeley, CA 94720-1776
randy@eecs.berkeley.edu

ABSTRACT
In this paper we describe the Internet-in-a-Box datacenter network emulator, an FPGA-based tool for researchers to rapidly experiment with O(10,000) node datacenter network architectures. Our basic approach to emulation involves constructing a model of the target architecture by composing simplified hardware models of key datacenter building blocks, including switches, routers, links, and servers. Since models in our system are implemented in programmable hardware, designers have full control over emulated buffer sizes, line rates, topologies, and many other network properties. Full system control also gives researchers a significant degree of system visibility. Additionally, because our node model emulates servers using a full SPARC v8 ISA compatible processor, each node in the network is capable of running real applications. This allows researchers to study a network under complex real-world workloads at a scale that matches that of a large datacenter today. Moreover, because the system is a private testbed, experiments can be deterministic and therefore reproduced by other researchers. Lastly, the system is cost effective for designers, and we show that using FPGA technology on the market today we can actually emulate a network of 256-nodes for about $2,000.

Categories and Subject Descriptors
C.3 Special-purpose and application-based systems

General Terms
Design, Experimentation, Performance

Keywords
Datacenter Networking, Hardware Modeling

1. INTRODUCTION
In recent years, datacenters have been growing rapidly to scales of up to 100,000 servers [1]. Many key technologies make this possible, including modular datacenter design and server virtualization. The changes in scale enabled by these technologies draws great attention to datacenter networking architecture. To explore this large design space, researchers typically use software simulation on virtualized servers, which can only reach a scale of O(10)~O(100) nodes [2][3]. In addition to having such limited scale, these approaches often fail to represent the true timing characteristics of the target network architectures, thus leading to less creditable results. Recently, however, cloud computing vendors have offered pay-per-use services to enable users to share their datacenter infrastructure at an O(1,000) node scale, e.g., Amazon EC2, Microsoft Azure and Google App Engine [4][5][6]. Such services, however, provide almost no visibility into the network infrastructure and provide no mechanisms for experimenting with new networking elements or devices. Also, it is expensive to run continuous experiments on thousands of nodes in a cloud.

To address the above limitations, we propose an inexpensive and reconfigurable emulation testbed for datacenter network research, called "Internet-in-a-Box", that is scalable to O(10,000) nodes using an array of FPGAs. Instead of fully implementing the target system, we build several smaller models targeting key datacenter components and compose them together in hardware. The way it works is analogous to a software based computer architecture simulator. With the latest Xilinx Virtex 5 FPGA, we can build a laptop adapter powered emulation system capable of emulating 256 nodes on a single chip at over 100MHz for about $2,000. A 10,000 node system can then be constructed from a rack of multi-FPGA boards, e.g., the BEE3 [7] system. Additionally, each node in the network is able to run real applications and OSes. The performance may be slower than a real implementation, but is still several orders of magnitude faster compared to event-driven software simulators. Furthermore, logging can be done at the hardware level, giving researchers clear visibility into the network for little or no performance penalty.

2. EMULATION METHODOLOGY
FPGAs and similar programmable ASICs have been widely used in the field of scalable system design verification, such as Cadence Palladium [8] and MentorGraphics Veloce [9]. These multi-million dollar boxes allow designers to directly implement a fully functional model of the target chip and run it at a frequency of around 1~2 MHz. We define the Target here to mean the system we want to emulate, while the Host refers to its physical representation on FPGAs. Unlike those systems, however, Internet-in-a-Box is designed to emulate a much larger target system, and therefore we do not implement a fully functional target. Neither do we implement a complex and monolithic model.
of the target. Rather, we have developed simplified hardware models of key datacenter building blocks, including switches, routers, links, and servers, and allow a designer or researcher to compose them together to then construct a model of the desired network architecture. In this paper we describe our node and switch model, and show how they can be used to emulate a target system consisting of a 64 node rack of machines connected to a top of rack switch. Before going through this example and our current system architecture, however, we must first describe two important concepts that define our basic approach to building hardware models and emulating the target system: a) the functional and timing split model-building methodology, and b) model virtualization.

2.1 Functional and Timing Split

Every model is composed of two parts: a functional model and a timing model. Both models are implemented in hardware. A functional model is composed of the hardware necessary to implement the basic functionality of the target, regardless of timing or performance. The timing model then orchestrates the functional model to execute operations in time. For example, say that our target is a single cycle access SRAM buffer. We can use a slower DRAM buffer as the functional model for the target in the host, and design the timing model to tick time as if we were using an SRAM (one tick for each access). The advantages of the functional and timing split approach to hardware modeling, when compared to implementing a fully functional target or a monolithic model of the target with inseparable functional and timing parts, are the following:

1) Reducing the hardware cost on FPGAs compared to a full system implementation. Because the functional model alone lacks any notion of timing, we can neglect any performance optimizations in the target (as in the previous example).

2) Timing models can be interchanged, allowing a designer to change system characteristics like link speeds and delays without modifying the functional model.

3) The timing model only encodes the timing characteristics of the target, and can be as simple as a set of counters, which can be seen from the example in Section 3.

2.2 Virtualization

One design goal of Internet-in-a-Box is to maximize emulation throughput per FPGA. One naïve way to do this is to replicate our models to fill up the chip. Our previous work has shown that this is suboptimal [10], and that by virtualizing a host processor the overall emulation throughput in MIPS/FPGA is twice that of using the naïve model replication method. By "virtualization" we mean the use of multiple hardware contexts sharing a model's core data-path in time (analogous to hardware multithreading in modern processors). The increase in emulation throughput as a result of virtualization is due to its ability to hide emulation latencies and maximize memory bandwidth utilization. Therefore, we introduce the idea to our hardware models to improve the emulation throughput. Leveraging the timing model, then, we can patch the serialization introduced by virtualization and preserve the parallel nature of the target. Our result is that we are able to increase emulation capacity to 256 nodes on a single Xilinx Virtex 5 LX110T FPGA, as shown in Section 4.1.

Figure 1. Mapping a target system into our emulation architecture. (a) The target is a 64 node datacenter rack with a 64-port output buffered layer-2 switch. (b) Diagram describing the logical setup and componentry of our emulation system. (c) Mapping the logical emulation setup (b) into our emulation architecture.
3. EMULATION ARCHITECTURE

In this section we walk through and describe our emulation architecture by mapping a target system into it, as illustrated in Figure 1. Our target in this example is a 64 node datacenter rack equipped with a 64-port, output buffered layer-2 switch. We first translate the target system into abstract emulation components as in (b), and then into a detailed emulation architecture in (c). In this example, our abstract emulation setup consists of 64 node models, a switch model, a scheduler, and a logger. The node and switch models are split into their timing and functional components, as described in Section 2. Model synchronization is done by the scheduler, which maintains a bound on the spread in emulation progress between models through regular communication with and control of the timing models. Lastly, the logger is responsible for monitoring interesting places in the system and streaming event records to off-chip storage. The following sections describe the node and switch model architectures, and how they are used to emulate the target system.

3.1 Node Model

Our node model targets a single datacenter server, and as illustrated in Figure 1(c) is actually composed of two more specific models: one for the node's processor and one for the node's Network Interface Card (NIC). The processor's functional model implements the 32-bit SPARC v8 ISA, and is a single issue in-order pipeline that runs at over 100MHz. The model has been verified against the certification test suite donated by SPARC International, and can therefore run unmodified SPARC binaries. A single functional model pipeline is virtualized using host multithreading, which refers to the usage of N hardware threads to implement N functional models. We have chosen N to be 64 for FPGA hardware mapping efficiency. Therefore, we can use a single pipeline to emulate the 64 nodes in the target system and run real applications on each of them. If more nodes are to be emulated, we would replicate the multithreaded processor pipeline appropriately. The processor timing model tracks the number of completed target cycles for each processor functional model, and statically schedules the threads in round-robin order for design simplicity. In the event that the functional models get out of sync, the timing model can inject nop instructions into the faster threads' instruction stream while letting the slower threads catch up.

The second component of the node model is a NIC which has been virtualized, similar to the processor, to support one NIC for every thread. Our NIC model supports sending and receiving of 64-bit layer-2 packet descriptors, illustrated in Figure 2. The packet payload is stored in memory, and can be looked up using the TAG field by any component in the system (e.g., by the receiving node model to transfer the full packet into his address space).

Our use of packet descriptors is motivated by the limits of FPGA memory resources (less than 1MB of on-chip memory for an LX110T FPGA). Upon receipt of a packet from the processor, the NIC's functional model observes the source node and places that packet into the node's transmit queue. Similarly, packets destined for node X from the network are stored in node X's receive queue. Because we have separated the functional and timing models, it is not necessary for us to implement 64 independent queues. Instead we have implemented each set of 64 queues using a single-port memory and access them serially to save FPGA resources. We let the timing model control the completion of target cycles to reflect the parallel and independent nature of the queues in the target system.

The timing model for the NIC is illustrated in Figure 1(c) and determines at what target cycle the packet at the head of each TX queue should be transmitted. It does this by using its Packet Delay Counters which count down the number of target cycles for transmission. The timing model therefore has the ability to control the line rate of each port. For example, if the NIC's timing model specifies a 1Gbps link and 1ns target cycles, a 64B packet translates into a 512-cycle delay in the target. When the counter reaches zero the packet is sent and a new value is calculated and loaded for the next packet in the queue, if any. Similar to the queues described above, the Packet Delay Counters are implemented using a single-port memory to save FPGA resources. Thus, in 64 host cycles the timing model will have checked, decremented, or loaded 64 counter values one after the other.

3.2 Switch Model

Our switch model targets a 64-port, 1Gbps, output buffered layer-2 switch, and has a very similar architecture to that of the NIC model. The functional model is composed of 64 queues implemented in a single-port memory, and forwarding logic to direct incoming packets to the correct output queue. The queues may be sized appropriately to model, in approximation, the size of the target buffers. We say approximate because the switch model stores a fixed number of packets, not bytes. The timing model controls the transmission of these packets across a single on-chip link that is multiplexed in time to emulate 64 physical links in the target. In our current implementation the switch stores 64 packets per output buffer using on-chip SRAM. If more buffer storage is needed than is afforded by on-chip resources then external DRAM can also be used. In this case the timing model can slow the passage of target cycles for the switch in the event of a high-latency access to DRAM.

4. DISCUSSION

4.1 Resource Consumption

Table 1 outlines the resource consumption for our system on a Virtex5 LX110T FPGA. Given these resource requirements, we can instantiate 4 processor pipelines, 4 NICs, and 5 switches each with a 7,680 packet buffering capacity on an LX110T FPGA. Since each packet descriptor represents a real packet of size between 64B and 1500B, 7,680 packets translates to between 0.5MB and 11MB of emulated capacity.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Reg. Bits</th>
<th>LUTs</th>
<th>BRAM (18k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>4886 (7.1%)</td>
<td>2839 (4.1%)</td>
<td>30 (10.1%)</td>
</tr>
<tr>
<td>Switch</td>
<td>53 (0.1%)</td>
<td>137 (0.2%)</td>
<td>30 (10.1%)</td>
</tr>
<tr>
<td>NIC</td>
<td>77 (0.1%)</td>
<td>250 (0.4%)</td>
<td>4 (1.4%)</td>
</tr>
<tr>
<td>Mem. Ctrl.</td>
<td>2472 (3.6%)</td>
<td>1975 (2.9%)</td>
<td>10 (3.4%)</td>
</tr>
</tbody>
</table>

As mentioned before, if more memory is needed then DRAM can also be used. A single $2,000 FPGA platform [11] is therefore capable of emulating a 256-node system with a network of 5 switches.
Looking at Table 1, it is clear that our limiting resource is on-chip memory. Thus, projecting forward to the largest 40nm Xilinx Virtex6 FPGA (SX475T), we will have 6x more resources to either spend on more models or increase the complexity of our models. In terms of on-chip packet-buffering capacity in the network, that translates to about 500,000 in-flight packets.

4.2 Slowdown
Our emulation system runs slower than the target system in terms of wall clock time. There are two contributing factors to this slowdown: a) FPGA clock speeds, and b) virtualization. Since the FPGA runs at ~100MHz, emulating a 1GHz target system contributes 10x to our slowdown. Also, because we virtualize our hardware models into 64 time slices, there is an additional slowdown factor of 64. However, host multithreading on the processor can hide emulation latencies and offset this slowdown. In fact, recent results have shown that our processor model is able to achieve more than 60 MIPS in the host running at 100 MHz and all while the timing model synchronizes threads on every target cycle. This translates to approximately 1 MIPS per thread.

If the target, then, is a 1 GHz processor with a CPI of 2, our slowdown factor is approximately 530, which is less than our upper bound on slowdown (640) and is attributable to the benefits of virtualization. Our actual slowdown factor has yet to be determined, since our processor currently runs at more than 130 MHz and a more permissive timing model may not synchronize threads every target cycle, reducing the synchronization overhead.

4.3 Scaling with Multiple FPGAs
Although our discussion thus far has focused on a single FPGA, partitioning the emulation across multiple FPGAs is fairly straightforward. This is because the functional and timing model split methodology makes each model inherently asynchronous and loosely coupled to the rest of the system. Furthermore the host multithreading technique will help to hide the inter-chip communication latencies.

4.4 Use for Energy Modeling
In our system it is easy to monitor any component and record any event. Therefore, energy consumption can be estimated by gathering the appropriate data and plugging it into the right power models. Coupled with various other system measurements, it is possible to generate a complete system picture and easily identify places for potential optimizations.

5. FUTURE WORK
Our future work primarily involves continuing to develop the system’s functionality and scaling across many FPGAs. Validating our switch model using real workloads is also an important next step, as well as understanding the performance bottlenecks in the system when scaling. Finally, we would like to use our system to validate the datacenter networking architecture research of others and explore ideas of our own. At present we are working towards using the system to study the TCP Incast problem [12]. In particular, we are using a ported TCP implementation to allow nodes to establish connections in a many-to-one fashion over a single switch, and seeing how the resulting throughput collapse unfolds at both the hardware and software levels. Since we have full control of the hardware, we may then investigate the effect of various switch architectures and features on the system’s behavior.

6. CONCLUSION
In this paper, we have described the Internet-in-a-Box datacenter network emulator and demonstrated how to map a rack of 64 machines with a 64-port rack switch into our emulation architecture on a single FPGA. Additionally, we have shown that our system is capable of emulating 256 nodes connected in a network of 5 switches, also on a single FPGA system costing $2,000. The relative cost of ownership for such a system compared to a 256-node cluster makes our platform attractive for researchers. Moreover, every researcher can own one on his or her desktop for themselves as a private research testbed, as opposed to time sharing hundreds of machines in a large machine room. We believe that Internet-in-a-Box will be a powerful CAD tool for exploring datacenter networking architectures in the future.

7. REFERENCES