Large-Scale Network Simulation Scalability and an FPGA-based Network Simulator

Stanley Bak

Abstract

Network algorithms are deployed on large networks, and proper algorithm evaluation is necessary to avoid large-scale outages or performance bottlenecks. However, evaluating a network algorithm in a simulator results in significant run times which scale poorly as we increase the number of routers. An alternative network simulator design is desired which scales significantly better than current simulators.

We evaluate the scalability of NS2, a popular network simulator, by generating topologies of large networks and show an `O(N log(N))` scalability where `N` is the number of routers. Then, we propose and implement a network simulator based on reconfigurable hardware and FPGAs, which is shown to scale `O(log(N))`. However, through implementation we demonstrate drawbacks of our FPGA-simulator, particularly area constraints offered by current generations of FPGAs.

1. Introduction

Network algorithms require significant work to evaluate. Several techniques exist to make this easier and to be able to detect protocol problems before large-scale deployment including network emulation and simulation. Network emulation requires large amounts of resources for large scale evaluation, and network simulation also scales poorly.

Current network simulators exist which are capable of simulating a given topology and flow scenario. However, the problem with some network simulators, such as NS2 [1], is that they are single threaded, so large-scale simulations take extremely long times to simulate. The problem has been partially addressed by using parallel network simulation such as SSFNet [2]. In these simulations, however, the bandwidth between simulation nodes becomes the bottleneck leading the simulation designer to require partitioning the network in ways to make the simulation perform more efficiently. Even multiple threads on a multi-core processor suffer from communication issues since the parallel execution threads must share information.

This communication bottleneck can be addressed by performing the entire simulation on custom hardware. Field Programmable Gate Arrays (FPGAs) can implement custom hardware specifically designed to be a network simulator. In this way, both the CPU exhaustion and the communication limitation problems can be solved. However, FPGAs can execute arbitrary hardware, which makes programming them difficult. The main contributions of this paper are

- A theoretic and empirical evaluation of the scalability of NS2, the most popular network simulator
- The design and implementation of a better-scaling, massively-parallel FPGA-based Network Simulator
- An evaluation of the FPGA Network simulator's scalability, and discussion of associated issues

This paper is divided into three main sections. First, in Section 2 we describe our simulation scenario. The scalability of any simulator depends on the parameters of the simulation, thus care must be taken to ensure the simulation parameters are realistic. Next, in Section 3, The Network Simulator (NS2) is described and evaluated. Expected scalability results are presented, along with an evaluation using the generated topologies. Next, in Section 4, a design is presented for a massively parallel network simulator and then implemented on FPGA hardware. The new network simulator is evaluated for scalability. As a result of implementation, a number of practical issues arise for simulation on FPGAS, which are also discussed.

2. Simulation Scenario

Since the run time of a simulator depends on the network being simulated, particular care is taken to develop a realistic simulation scenario. The important

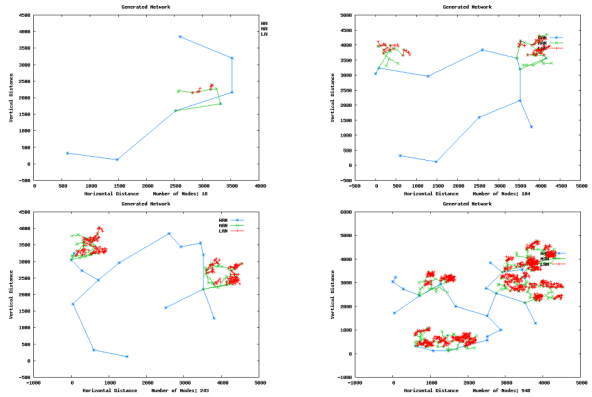

Figure 1: We evaluate the scalability of network simulators using topologies generated by the Tiers topology generator.

parts of the simulation are the network topology and the packet traffic.

For our topology, we use the Tiers topology generator [3]. This program generates topologies with three tiers, a wide area network which contains several metropolitan area networks, which each contain several local area networks. By modifying parameters to the Tiers program, we are able to generate topologies with increasing numbers of routers. The Tiers generator also provides us with location information for the routers. This location information is used to determine the delays for the links within our simulations, with routers that are further away having higher delays than routers which are close. Four sample generated topologies are shown in Figure 1.

We generate traffic between routers by having each router chose a random destination router for which to send data. Constant bandwidth flows are created 1 second into the simulation and stopped 50 seconds after the start of the simulation. The simulation ends after 55 seconds. The flows each send one packet every second. This flow scenario is probably simpler compared to that of a typical simulation. However, our intent is to show the scalability of the simulator, not that complex algorithms can be implemented on FGPA hardware (which is known to be true). This fixed flow scenario is also easier to reason about in terms of theoretic scalability, as is done in later sections. By limiting our router complexity in this way, we can more rapidly implement and evaluate the FPGA network simulator, although other algorithms such as TCP could potentially be added within the same simulator design.

The end-to-end process of generating a topology and running it through both simulators is shown in Figure 2. The topology is first generated by Tiers. We then created a C++ program, `t2n`, to parse the Tiers

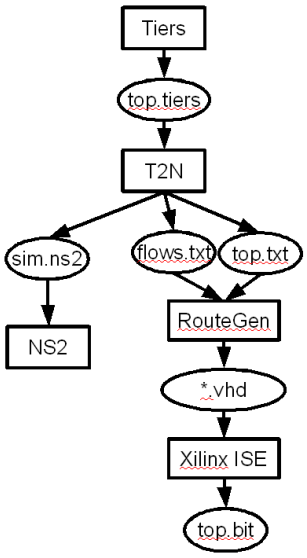

Figure 2: The process of generating and simulating a network topology involves a number of steps.

topology, create random flows, and generate an NS2 tcl script which can be directly fed into NS. The `t2n` program also outputs two files, `top.txt` and `flows.txt`, which describe the topology, link latencies, and flow information. These files are fed into the the `RouteGen` Java program that we developed, which outputs a number of VHDL source files. These VHDL source files are then turned into bitstreams using the Xilinx ISE tool.

3. The Network Simulator (NS2)

The Network Simulator (NS2), is the most popular current network simulator. In order to demonstrate the scalability of NS2, in Section 3.1 we examine the program's design and infer the scalability we expect. Then in Section 3.2, we use the Tiers topology generator and generate increasingly large network topologies and measure the performance obtained by NS2.

3.1. Expected Scalability

NS2 is a single-threaded network simulator that maintains a priority queue of events. The earliest event in the system is processed at each time step, potentially generating more events that are put into the event queue. For example, a packet traveling between routers would receive an updated event time based on the delay between the routers, and move to the next router. The processing that occurs at the next router would be pushed until further in the simulation.

Given this design of network simulator, we can predict the scalability of NS2. The computation time

scales proportionally to the number of events. The number of events is a product of the number of flows in the system and the number of hops per flow. In our simulation scenario, each new router generates a new flow, so the number of flows scales `O(N)`. The number of hops per flow also increases as we increase the number of routers, although at a slower rate. Since our network is hierarchical, we expect the path length to scale similar to that of a tree, or `O(log(N))`. Therefore, for our simulation scenario the expected scalability is `O(N log(N))`.

3.2. Measured Scalability

In order to verify our predicted scalability, we ran several tests with NS2 using the topology and flow generation outlined in Section 2. The results are displayed in Figure 3, and show a super-linear increase in the computation time as we increase the number of routers. We consider this a validation of our predicted scalability result, although there are a few other factors that may affect the run time of the simulation including the effects of cache, and logging.

One factor that may affect simulation time as we scale up is the amount of memory that is used. Smaller simulations will result in more cache hits and therefore may receive an associated speedup. Here, even through the number of events still scales `O(N log(N))`, the events get processed at different rates depending on the simulation size. We do not measure the effects of caching on our simulations, although we consider them to be small. If we used even larger topologies, where the disk would have to be used to store memory, we predict the effects may be more significant.

Another factor that may affect simulation time is logging. For this reason we chose to only log data on a single edge router which receives packets from a single other router. In this way, the amount of data logged is minimal, and does not increase for larger topologies. Therefore, the effects of logging only add a constant overhead to the simulations, and do not contribute to the scalability.

4. FPGA Network Simulator

To improve the scalability of network simulation, we develop a network simulator targeted at reprogrammable logic on an FPGA instead of at an executable for a processor. By developing on this platform, we are able to take advantage of massive fine-grained parallelism offered by hardware programming and FPGAs. However, we must switch from an event-based design like that of NS2 to a design more suited for parallel execution. The proposed network simulator design is presented in Section 4.1. Then, the network simulator is evaluated in Section 4.2. Finally, some drawbacks of FPGA network simulation are discussed in Section 4.3.

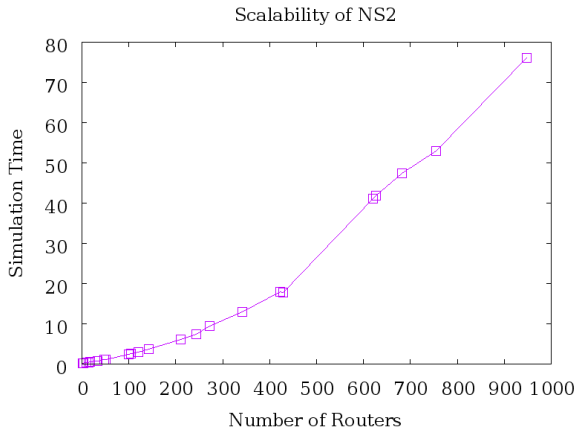

Figure 3: The measured scalability of the NS2 network simulator matches the predicted `O(N log(N))` pattern. The simulation time is in seconds.

4.1. FPGA Network Simulator Design

In order to construct a network simulator on an FPGA, we first create a computational model which we can design our simulator in and can be mapped to an FPGA. We use specific FPGA elements in order to construct our simplified model in order to maximize parallelism (and therefore performance). Particularly, we notice that for tasks that require more than a few registers of memory, a block ram on FPGA must be accessed. Typically, FPGAs have 100-200 block rams on the chip which each have on the order of tens of kilobytes of memory. Each of these can be accessed in parallel in a single clock cycle. With each of these block rams, we associate one computational element (router).

These computational elements share a global clock, and can push data (packets) onto other computational elements. In order to limit one computational element pushing data into another one at a time (for serial processing and storage), a mutex hardware block is used. To maintain coordination, a global scheduler receives ready signals from each computational element, and will set the proceed signal to true when all computational elements are ready. For debugging and analysis, we also include a serial port output module that can be associated with a single router. A logical block outline of the simulator with three routers present is shown in Figure 4. Currently, constant bandwidth computational elements are used to do our evaluation. How-

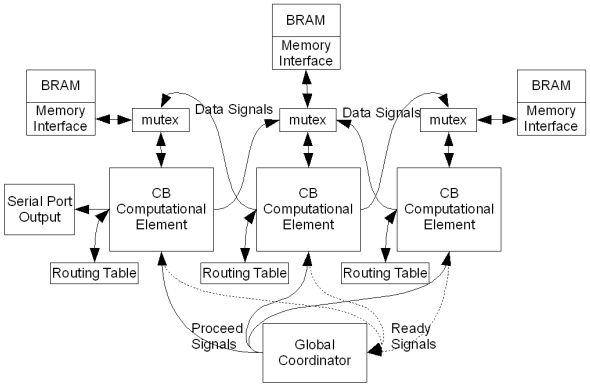

Figure 4: Here, the logical outline of our FPGA-based network simulator contains three routers.

```
r -t 1.04516 -s 0 -d 1
r -t 2.04516 -s 0 -d 1
r -t 3.04516 -s 0 -d 1
r -t 4.04516 -s 0 -d 1
r -t 5.04516 -s 0 -d 1

1045 1000 1    0
2045 2000 1    0
3045 3000 1    0
4045 4000 1    0
5045 5000 1    0
```

Figure 5: A trace comparison between the output of NS2(top) and our FPGA-based network simulator(bottom) demonstrates simulation correctness.

ever, in the future these could be replaced by TCP or other computational elements.

4.2. FPGA Network Simulator Evaluation

The FPGA network simulator is evaluated for both correctness, that the result obtained matches the result from NS2, and scalability, in terms of simulation time versus number of routers.

4.2.1. Correctness The correctness of the network simulator is evaluated by comparing the output of the NS2 simulator with the FPGA network simulator for simple topologies. One trace comparison is shown in Figure 5, and the simulators have matching results. The NS2 simulator packet arrival times are slightly shifted because NS2 models delays at the routers and not just link latencies. The FPGA simulator currently does not model these effects, although this can be added in the future.

4.2.2. Scalability Our FPGA-based network simulator has a different design than NS2. Although the number of events scales like NS2 $O(N \log(N))$, each

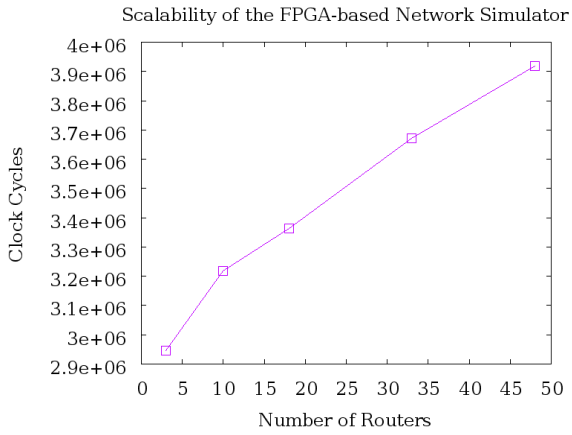

Figure 6: We ran topologies of various sizes in our FPGA network simulator to evaluate scalability.

additional router results in an additional parallel computing unit. Since N routers means N parallel computation elements, we expect an $O(\log(N))$ scalability result, assuming each computation element receives an equal share of computation. This assumption may not be valid, however, if the computation is not spread evenly throughout the network that we are simulating. This is a particular concern since we expect the core routers to be processing more traffic than edge routers. Nonetheless, adding a router and flow to the network will not always increase the computation at the core routers, which would result in a scalability of $O(N)$. Since our simulation scenario is somewhere between these two extremes, we expect the scalability to lie between an lower bound of $\Omega(\log(N))$ and an upper bound of $O(N)$.

We ran ever-larger topologies within our FPGA network simulator, and measured the scalability. The results are shown in Figure 6. The scalability empirically appears to be closer to the lower bound of $\Omega(\log(N))$, and demonstrates a large constant computation time. This constant comes from the majority of the computation time occurring when no traffic is present in the network (the simulation is sparse), and is discussed further in Section 4.3.

Run-time scalability, however, does not tell the whole story in terms of the time to perform a simulation. We address two other concerns of end-to-end simulation time, the critical path of the generated hardware, and the time it takes to perform the hardware generation.

Although the scalability of the number of clock cycles was evaluated, the duration of each clock cycle may change among the various simulations. The duration of a clock cycle depends on the critical path of the hardware, which may vary among simulation scenarios. We

recorded the critical path of the generated hardware which determines the minimum clock cycle that can be used in conjunction with the simulation. We used the Xilinx ML505 Platform [4] as the target FPGA. The results are presented in Table 1, and does not show an increasing trend with larger simulations. This can be explained because, although the amount of interconnect between modules increases with larger simulations, the length of the interconnect does not. There is some slight variation in the critical path because the hardware generation process uses heuristic algorithms such as simulated annealing, which may result in different hardware implementations between runs.

The hardware generation time is a greater concern since it consumes the majority of the end-to-end time required to perform a simulation. This process is the equivalent of compiling the simulator into hardware code. To do this we use Xilinx's 10.1 ISE software with service pack 3, running on on an Intel Core2 Quad CPU system at 2.4 GHz with 4 gigabytes of RAM and Windows Vista 32-bit. We used the default settings for synthesis which perform some analysis on the generated bitstream and provides a balanced area/execution-time trade off. The amount of time for hardware generation is shown in Table 1, and unfortunately does appear to increase for larger simulations. However, notice that this time only depends on the topology, and is independent of length of the simulation. Therefore, for long-running simulations with the same topology, this time remains constant. Additionally, by running the simulation on multiple FPGAS as we suggest Section 4.3, this process could be done in parallel for each FPGA, bounding the generation time.

4.3. FPGA Network Simulator Drawbacks

We describe three drawbacks of FPGA-based network simulation, including discrete/continuous time, sparse/dense simulations, and FPGA area constraints.

NS2 uses a continuous time scenario where time is represented by a floating-point value. Our FPGA Simulator, on the other hand, uses discrete time where there is a minimum time step. This is concerning because our FPGA Simulator may not be able to simulate the same things that NS2 can simulate. Particularly, if the delay is less than the minimum discrete interval, we can not accurately model the network. Decreasing the discrete interval can help in this matter, however it will increase the number of computation steps and therefore adversely affect simulation time.

Our simulator is also best suited for dense simulations with many events happening concurrently. If the simulation involves a small number of events, an event-based simulator such as NS2 is likely to perform well. The FPGA simulator will spend most of its time skipping eventless states. This happens to some extent in the simulations in this paper, where there are some periods of time (at the beginning of every second when traffic is generated), when lots of events occur but then after all packets arrive at their destinations many steps are skipped. This could be somewhat reduced by a design that can skip multiple discrete steps at once. This would involve changing the global coordinator in our design to a more intelligent version that would take in a "next event" time from each computational element.

Lastly, our medium-scale simulations use up a significant portion of FGPA area which limits applicability towards large-scale simulation. The limited number of concurrent BRAMs available on an FPGA could be overcome by sharing a single memory between different routers, although this would result in more limited memory constraints for each router, and would increase simulation time as mutex contention would undoubtedly rise. A more serious concern is the slice utilization on FPGAs. As shown in Table 1, the slice utilization approaches exhaustion of our Xilinx ML505 FPGA Platform as the simulations get larger. This can be overcome slightly by getting a larger FPGA which contains more slices. However, for even larger simulation we believe multiple FPGAs could coordinate on a single simulation. While the only present way to do this is manually partitioning the hardware program, we believe a more general framework could be applicable to multi-FGPA computation. The area required appears to scale linearly with the number of routers, which means that a simulation of an AS which contains 500 routers likely be achievable using less than 10 FPGAs. This sort of scalability is not the case with NS2's design, where the single thread of execution does not gain performance increases from the presence of multiple processor cores.

5. Conclusions

Network simulators must scale in order to properly evaluate large-scale effects of algorithms. By generating network topologies and flow scenarios, we were able to evaluate the scalability of both NS2, and a new network simulator targeted for FPGA execution. We verified that as the number of routers increases, NS scales `O(N log(N))`, while our FPGA-based network simulator design is closer to `O(log(N))`. Additionally, by implementing our network simulator on an FPGA, we were able to show which key challenges remain before FPGAs become a viable option for network simula-

FPGA-based Network Simulator Scalability				
Routers	Clock Cycles	Critical Path	HW Generation Time	% Slices
3	2946440	5.179ns	3:34	6
10	3217646	4.994ns	5:33	18
18	3364044	5.266ns	8:05	31
33	3670920	5.262ns	12:52	56
48	3918438	5.299ns	19:01	80

Table 1: The scalability of our FPGA simulator is shown in terms of clock cycles, critical path, hardware generation time, and area (slices).

tion, particularly, overcoming the limitations of reconfigurable area. Nonetheless, we have shown that doing network simulation on an FPGA is possible, and our proposed design is scalable.

References

[1] I. S. Institute, "The network simulator - ns-2," www.isi.edu/ nsnam/ ns/, 2008.

[2] S. R. Network, "Scalable simulation framework," www.ssfnet.org/ homePage.html, 2002.

[3] I. S. Institute, "The network simulator ns-2: Topology generation," www.isi.edu/ nsnam/ ns/ ns-topogen.html, 2008.

[4] Xilinx, "Virtex-5 lxt fpga ml505 evaluation platform," www.xilinx.com/ products/ devkits/ HW-V5-ML505-UNI-G.htm, 2008.