Abstraction-Based Performance Analysis of NoCs

Daniel E. Holcomb
UC Berkeley
holcomb@eecs.berkeley.edu

Bryan A. Brady
UC Berkeley
bbrady@eecs.berkeley.edu

Sanjit A. Seshia
UC Berkeley
sseshia@eecs.berkeley.edu

ABSTRACT

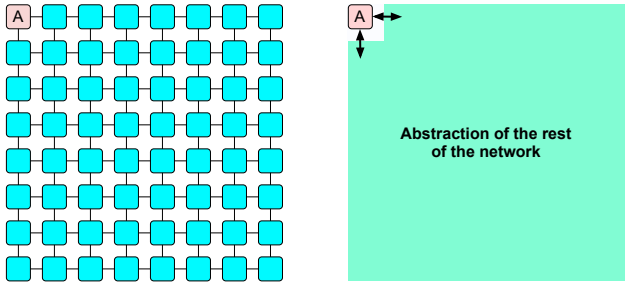
We present an approach to formally analyze quality-of-service (QoS) properties of network-on-chip (NoC) designs. To tackle industrial-scale designs, we adopt an abstraction-based approach, where only the nodes of interest in the network are precisely modeled and the rest of the network is abstracted away as sources and sinks of traffic. We give an automatic technique to infer a traffic model, comprising formal models of sources and sinks, from simulation traces derived from software benchmarks. Experimental results demonstrate that the inferred models generalize well and that our abstraction-based approach can accurately verify industrial-scale NoC designs.

1. INTRODUCTION

Network-on-chip (NoC) is a paradigm for communication within large many-core, system-on-a-chip (SoC) designs. An NoC architecture consists of a network of interconnected nodes, where each node can be a processor core or a specialized IP block with associated networking logic. Industrial examples include the Tilera TILE64™ processor. NoCs must offer quality-of-service (QoS) guarantees on the latency, jitter, and throughput of certain classes of network traffic. Meeting these constraints depends on spatial and temporal properties of traffic patterns.

Inter-node communication in an NoC is performed by the transmission of data packets through routers. Each packet is communicated as a number of smaller units called flits. Store and forward networks wait until all flits of a packet are received before forwarding any to the next node, while wormhole networks forward the individual flits of a packet as they arrive. Wormhole networks can have lower latency and use smaller buffers than store and forward [15], but a single packet can be strung out across the network blocking the forward progress of many other flows and complicating analysis [1]. Adding virtual channels allows flows to be routed around any backed-up flows.

Verifying performance (QoS) properties on NoC designs is challenging. To illustrate this point, consider the 8×8 grid of interconnected nodes shown in Fig. 1(a), where each node represents a router and network interface logic (e.g., connecting the router with a core or memory element). Consider a specific node in the grid labeled `A`. Suppose that we want to verify the property that every packet traveling through `A` spends no more than 42 cycles within `A`. One option is to perform simulations, perhaps derived from software benchmarks. While software-derived testbenches are a good way to characterize workloads for NoCs, simulations can only prove the presence of performance bugs, not their absence. An alternative approach is to use a formal verification method such as model checking [8]. The property above can be formalized in temporal logic, and the overall network can be modeled as a synchronous composition of 64 finite-state machines, one for each node. However, in our experience, even a simple router (node) has over 1000 state variables. Models with tens of thousands of state variables are beyond the capacity of current formal verification tools.



(a) 8×8 network (b) Abstraction

Figure 1: Challenge of performance verification. We wish to verify a bound on the latency through node `A`.

The obvious alternative approach to apply formal methods to this problem is to use *abstraction*. As depicted in Fig. 1(b), one can abstract all nodes in the network other than `A` into an environment model `E`. The traditional approach to generating an environment abstraction is to liberally use *non-determinism*: for example, `E` can be a state machine that non-deterministically decides at each cycle whether to send a packet to `A`. Such non-deterministic behavior is often restricted using additional constraints. However, while such modeling is useful for verifying "logical correctness" properties such as absence of deadlock, it does not work well for proving performance constraints. Consider proving the latency property as above. With a completely non-deterministic model of `E` that makes no assumptions about typical traffic patterns, no useful latency bound can be proved. In the worst case, the environment will configure all packets to require the same output port. This unlikely situation will lead to an overly pessimistic latency bound on `A`. The question is: what sorts of constraints on `E` would be reasonable?

In this paper, we present TITAN, a new abstraction-based approach for performance verification of NoC designs. TITAN [1] leverages the presence of testbenches derived from software benchmarks by generating formal models of network traffic from simulation traces. The inferred traffic models are guaranteed to be over-approximations of the simulation traces they are derived from, meaning that they can generate not only the packet sequences in the simulation traces, but also other sequences that have similar traffic characteristics. The generated traffic models have several applications: (i) they enable the use of formal techniques like model checking to prove performance (QoS) properties of NoCs; (ii) they can be viewed as formalizing assumptions about traffic patterns under which QoS guarantees hold, and (iii) they can be used for diagnosing the cause of poor performance observed on new software benchmarks.

To summarize, the main contributions of this paper are:

- A new formal traffic model suitable for performance analysis of NoCs (Sec. 4);
- A technique for inferring traffic models from simulation traces (Sec. 5), and
- Experimental results demonstrating that formal verification based on the inferred traffic models can generate more accurate results

[1]TITAN stands for Trace-Inferred Traffic-Abstractions for Network-on-chip verification

DAC 2011, June 5-10, 2011, San Diego, California, USA.

than traditional non-deterministic models with equal efficiency (Sec. 6). Our results are obtained for an industrial-scale NoC design [17] using simulation data derived from PARSEC benchmarks [16], with verification by UCLID [4], which uses model checking based on SMT solving [2].

Before we present our approach, TITAN, in more detail, we first give a brief overview in Sec. 2. Basic terminology and preliminaries are covered in Sec. 3, and related work is discussed in Sec. 7.

2. OVERVIEW

Consider the problem of finding an upper bound on the latency for sending a packet from an input channel to an output channel of a router node R in a larger network $\mathcal{N}$. As an example, consider the network $\mathcal{N}$ in Figure 2 consisting of three routers (R_0, R_1, R_2) and an environment. The environment represents the network interface connections from processing elements to their associated router nodes. In this example it is assumed that round-robin arbitration is used, every packet has exactly 3 flits (head H, body B, and tail T), and that the network interface connections are *eager* (that is, they are always able to accept packets). Furthermore, the minimum latency for any packet traveling through any of the router nodes is two cycles. On the first cycle, the packet is stored in an input buffer associated with the respective input channel to the router and a request is sent to the channel allocator. In the best case, the allocator grants access to an output channel on the following cycle. Then, in the third cycle, the packet is written into an input buffer of a neighboring router (assuming that there are buffer slots available). Table 1 shows traces for the three options for the verification of performance properties of network $\mathcal{N}$. The *Type* column separates these cases:

1. *Simulation (Sim):* a simulation trace of a single traffic pattern;
2. *Non-deterministic model (ND):* the worst-case behavior associated with fully non-deterministic modeling, and
3. *Inferred traffic model (*TITAN*):* a single traffic pattern contained in a TITAN-inferred (aka, regulated) traffic model.

The *Channel* column lists specific network channels. For example, $R_0 \rightsquigarrow R_1$ is the connection between an output of router R_0 and an input of R_1. The channels not listed are assumed to have no network traffic. The remaining columns show what happened in the network on a given cycle. For example, H_1 is listed in *cycle 1* for $R_0 \rightsquigarrow R_1$ in the various verification modes. This should be interpreted as "in cycle 1, router R_0 sent flit H_1 to router R_1, and H_1 now resides in the input buffer of R_1". If no flit is listed in a particular location, then nothing was sent over the network in that particular cycle.

In all of the traces described in this example, a total of 4 packets (12 flits) are sent to router R_1 (two each from R_0 and R_2). It is important to note that in the simulation-based verification example, the QoS property is verified for a **single** trace. Whereas, in both *non-deterministic model* and *inferred traffic model* cases, the traces shown are only sample traces in the **set** of traces contained in the traffic models. The trace shown in the simulation-based verification is contained in the traffic model for both the non-deterministic and inferred-traffic model, however, the pathological example shown in the non-deterministic model is **not** contained in the inferred-traffic model.

Simulation: In the simulation trace listed in the *Sim* section of Table 1, the packets are staggered in such a way that buffer size does not affect the latency (i.e., the buffers never fill up). The maximum latency for this trace is 5 and occurs for flits H_4, B_4, and T_4. While the situation described in this case is possible, it is only a single instance of network-traffic in $\mathcal{N}$.

Non-deterministic model: The trace listed in the *ND* section of Table 1 shows the worst-case situation that will occur when the traffic is modelled non-deterministically. In this situation, channels $R_0 \rightsquigarrow R_1$ and $R_2 \rightsquigarrow R_1$ flood R_1 with flits. This causes the input buffer in R_1 associated with inputs from R_2 to fill up before any of the packets are granted access to the output channel $R_1 \rightsquigarrow Env$. This is the cause for the worst-case latency of 11 cycles. H_4 is stored in R_1 on cycle 4, but it is not granted access to the output channel $R_1 \rightsquigarrow Env$ until cycle 15. Due to the round-robin based arbitration logic, flit H_3, which is stored in R_1 at the same time as flit H_4, gets access before H_4. An important point to reiterate: although pathological cases such as this may be possible in real life, they may not represent the software benchmarks of interest.

Inferred traffic model: Let $\mathcal{T}$ be the traffic model inferred with TITAN. The trace listed in the TITAN section of Table 1 shows one of many traffic-patterns associated with $\mathcal{T}$. In this case, the worst-case latency occurs for flits H_4, B_4, and T_4, but the latency is 8 cycles. In addition to the traffic-pattern shown for this case, the traffic-pattern shown in the *Sim* section is also contained in $\mathcal{T}$. However, the case listed in section *ND* is not contained in $\mathcal{T}$ due to the constraints imposed on the non-deterministic source models (as described later in Section 4). As you can see from this example, traffic models inferred with TITAN are more general than the specific traffic patterns seen during simulation, but not so general that the latency bound is artificially large.

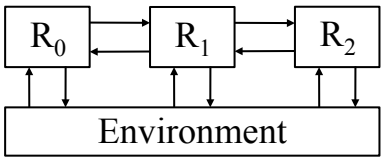


Figure 2: Simple network. Three router network $\mathcal{N}$. Each router is connected to its neighboring router(s) and the environment.

3. PRELIMINARIES

3.1 Basic Definitions

A micro-architectural model of an NoC design $\mathcal{N}$ is a tuple $\langle \mathcal{I}, \mathcal{O}, \mathcal{C}, Init \rangle$ where

- $\mathcal{I}$ is a finite set of input signals or *sources*;
- $\mathcal{O}$ is a finite set of output signals or *sinks*;
- $\mathcal{C}$ is a finite set of NoC components, such as routers, and
- $Init$ is a set of initial states.

For example, consider the NoC design $\mathcal{N}$ to be 8×8 grid in Fig. 1(a). Associated with each router $r \in \mathcal{C}$ is a location (x, y) where $1 \leq x, y \leq 8$. Each router is connected to neighboring routers in each Manhattan direction (N,S,E,W) and its own network interface (NI). The set of input sources $\mathcal{I}$ to this network are the incoming network interfaces into each router from its associated core. Similarly, the set of outputs are the outgoing interfaces from the router to its associated core.

If we consider instead the NoC design to be a single router `A`, located at position $(1, 1)$ in the grid of Fig. 1(a), there is a single component (namely, `A` itself) and there are 3 sources and sinks – one source/sink pair for the network interface associated with `A`, and two source/sink pairs for the channels connecting `A` with its neighboring routers at locations $(1, 2)$ and $(2, 1)$.

A *traffic pattern*, P_i, is associated with each input (source) $i \in \mathcal{I}$, and denotes *what* packets are injected into the router and *when*. Over a fixed number of cycles N, P_i is a sequence of pairs $\langle p_{i1}, p_{i2}, \ldots, p_{iN} \rangle$ where $p_{ij} = (b_{ij}, bv_{ij})$. Source i sends a packet into the network for all cycles j such that b_{ij} is **true**. The data value that source i injects on cycle j is bv_{ij} (when $b_{ij} = \mathbf{false}$ the value of bv_{ij} is irrelevant). The traffic patterns are generated by the environment of the NoC design $\mathcal{N}$.

Each output (sink) $o \in \mathcal{O}$ also has an associated traffic pattern P_o. P_o is a sequence of pairs $\langle p_{o1}, p_{o2}, \ldots, p_{oN} \rangle$ where $p_{oj} = (b_{oj}, bv_{oj})$. Sink o consumes a packet on all cycles j such that b_{oj} is **true**. Sink traffic patterns differ from those of sources in that only the b_{oj} variables are generated by the environment. The data

Type	Max Latency	Channel	Cycle 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Sim	5	$R_0 \rightsquigarrow R_1$	H_1	B_1	T_1				H_3	B_3	T_3								
		$R_2 \rightsquigarrow R_1$				H_2	B_2	T_2				$\mathbf{H}_4$	$\mathbf{B}_4$	$\mathbf{T}_4$					
		$R_1 \rightsquigarrow$ Env			H_1	B_1	T_1		H_2	B_2	T_2		H_3	B_3	T_3		$\mathbf{H}_4$	$\mathbf{B}_4$	$\mathbf{T}_4$
ND	11	$R_0 \rightsquigarrow R_1$	H_1	B_1	T_1	H_3	B_3	T_3											
		$R_2 \rightsquigarrow R_1$	H_2	B_2	T_2	$\mathbf{H}_4$				B_4	T_4								
		$R_1 \rightsquigarrow$ Env			H_1	B_1	T_1		H_2	B_2	T_2		H_3	B_3	T_3		$\mathbf{H}_4$	B_4	T_4
TITAN	8	$R_0 \rightsquigarrow R_1$	H_1	B_1	T_1			H_3	B_3	T_3									
		$R_2 \rightsquigarrow R_1$		H_2	B_2	T_2			$\mathbf{H}_4$	$\mathbf{B}_4$	$\mathbf{T}_4$								
		$R_1 \rightsquigarrow$ Env			H_1	B_1	T_1		H_2	B_2	T_2		H_3	B_3	T_3		$\mathbf{H}_4$	$\mathbf{B}_4$	$\mathbf{T}_4$

Table 1: Simulation traces/formal witnesses for network $\mathcal{N}$ *Type* denotes the various verification options. *Max Latency* is the maximum latency for any flit in the trace shown. *Sim:* simulation-based verification; *ND:* fully non-deterministic traffic model; TITAN*:* TITAN-inferred traffic model. *Channel:* a communication channel within the network $\mathcal{N}$. $R_0 \rightsquigarrow R_1$ denotes the channel from router R_0 to R_1.

values bv_{oj} at the sink are passed in the opposite direction, from the network to the environment.

An input sequence for a simulation is a vector of traffic patterns of the same length N, one for each source and sink. We present in the next section a formal model of traffic whose behaviors are a set of traffic patterns with similar characteristics as those obtained from software-derived simulation traces.

3.2 Performance Properties

While any performance property that can be formulated as a safety property is of relevance in this paper, we focus on two in particular: *throughput* and *latency* bounds.

A throughput constraint, ϕ_{thru}, is parameterized by the number of packets, n_p, to be sent over the network and the total number of cycles n_c. A throughput constraint $\phi_{thru}(n_p, n_c)$ is **true** if and only if n_p packets injected into the network are ejected at the correct destination in n_c cycles or less. A latency constraint, $\phi_{latency}$, is parameterized by a packet p and the maximum number of cycles allowed for packet transit, c. Suppose packet p is sent into the NoC on cycle n_s reaches its destination on cycle n_e. Then $\phi_{latency}(p, c)$ is **true** if and only if $n_e - n_s < c$.

Observe that due to the fixed bound on the number of cycles, both properties can be verified by bounded model checking, either from an initial state, or from a general state using techniques such as k-induction.

We are also interested in computing the tightest latency or throughput bound, not just in answering a YES/NO question about whether a given bound holds. In this case, bounded model checking must be combined with an optimization loop that performs, for example, binary search over the value of the objective function (latency or throughput).

4. FORMAL TRAFFIC MODEL

Modeling sources and sinks as generating packets completely non-deterministically can allow unrealistic scenarios. For example, in an N-cycle execution, the behaviors of a completely non-deterministic source include the pattern $p_{ij} = (\mathbf{true}, x)$ for all $i \in \mathcal{I}$ and $0 \leq j \leq N$. This source would attempt to send packets on every cycle, leading to overly pessimistic verification results.

We disallow such pathological worst-case traffic by introducing a formal model of traffic channels that we call a *bounded channel regulator*, where a *channel* is any link between components in a network. This model is inspired by the regulator model described by Cruz [9]. We make important changes that make it suitable for formal verification. Although we discuss in Sec. 5 how to generate such models from traces, the model introduced in this section can be useful even for manual modeling of traffic in NoC designs. We elaborate on this model below after introducing some notation.

We present our regulators following the XMAS design paradigm [7]. XMAS provides a compact notation to describe fundamental building blocks of NoC designs, such as sources, sinks, buffers, etc. Due to lack of space, we provide here only brief descriptions of the XMAS components we need to model regulators. It is important to note that the components described here are slightly modified version of the components described in [7], and are supplemented by additional components for modelling crossbars, and arbitrary sequential elements used to model arbitration. Specifically, we will use the following three XMAS components:

1. *A FIFO Buffer:* parameterized by its size k;
2. *Source:* an interface between the environment and network, a source generates and attempts to send packets as prescribed by its traffic pattern;
3. *Sink:* an interface between the environment and network, a sink attempts to consume packets as prescribed by its traffic pattern;

4.1 Traffic Regulators

A *bounded channel regulator* $\mathcal{T}_\mathcal{R}$ is a monitor on a channel that checks three constraints: the *rate* ρ, *burstiness* σ, and bound B on the traffic that traverses the channel. Figure 3 illustrates a bounded channel regulator. The buffer of size σ begins filled with tokens, and one token is removed whenever a head flit passes through the channel being monitored. Src_ρ adds a token to the buffer once every ρ cycles, unless the buffer is already full. During a simulation, Src_ρ becomes inactive once it has produced a total of $B - \sigma$ tokens. If a head flit ever passes through the channel when the regulator queue is empty, then a Boolean flag a_i is set to signal that the channel traffic does not conform to the constraints of regulator i. We refer to a regulator with rate ρ, burstiness σ, and bound B as $\mathcal{T}_\mathcal{R}(\rho, \sigma, B)$.

The regulator can be applied to any channel in a network, such as a channel originating from a completely non-deterministic source Src_{ND} that represents traffic from an abstracted upstream router. Combining Src_{ND} with the regulator provides a way to evaluate when the non-deterministic source is exhibiting behavior that is overly-adversarial relative to the router that it models. A regulator can also be used in conjunction with an eager sink Snk_{Eager} to model traffic to an abstracted downstream router. In this context, since the sink is eager, the regulator is indirectly monitoring whether the upstream non-deterministic sources are targeting the eager sink in an overly-adversarial manner.

Using traffic regulators to identify overly-adversarial behaviors gives a way to eliminate these pathological cases from consideration when proving performance properties. Specifically, the network traffic is not overly-adversarial whenever no channel regulators have flagged their channel as being overly-adversarial. This condition is denoted as a traffic model $\mathcal{T} \equiv \left(\bigwedge_{i \in \mathcal{I} \uplus \mathcal{O}} \neg a_i \right)$, To verify that some property ϕ holds for all non-overly-adversarial traffic, the property $\mathcal{T} \implies \phi$ is checked. Note that the inputs $\mathcal{I}$ and outputs $\mathcal{O}$ of a given traffic model $\mathcal{T}$ depend on what part of the network is being abstracted.

4.2 Extensions to Traffic Model

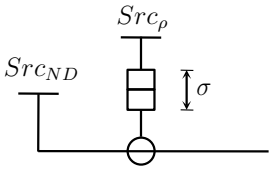


Figure 3: Traffic regulator. The bounded channel regulator $\mathcal{T}_\mathcal{R}(\rho, \sigma, B)$ models all traffic patterns with average rate constrained by ρ, burstiness by σ, and total number of packets bounded by B.

In addition to the single-channel regulators described in the previous section, multi-channel regulators can be used to further constrain network traffic. For example, a regulator that monitors all of the input channels of a router `A` can signal whether the sum of all input traffic to `A` exceeds constraints ρ,σ and B. Such a regulator is useful to create a traffic model that disallows all 5 inputs from sending a packet at the same time, but does not disallow any individual input from sending a packet. Stated differently, such a regulator can create a traffic model with bounded correlation between channels.
Furthermore, the traffic model can include constraints that are not based on regulators. Boolean constraints are placed on the destination addresses of traffic on particular channels. This rules out cases where sources representing abstracted routers generate impossible traffic, such as traffic that must be routed back in the direction from which it came.

5. TRAFFIC MODEL INFERENCE

As noted earlier, NoC designers might wish to optimize their designs for particular *kinds* of software benchmarks from a set of application domains. The PARSEC benchmarks [16] are one such example. However, it is desirable to optimize the design not just for the specific programs in the benchmark suite, but for all programs that generate "similar" traffic. The channel regulator model presented in the preceding section is our attempt to formalize this notion of "similarity". This section describes how such a model is inferred from simulation data.
Specifically, the model inference problem we address in this section is as follows:

> Given a finite set $\mathcal{P}_i$ of traffic patterns for a channel i in an NoC design, construct a regulator modeling i that generates a superset of $\mathcal{P}_i$ with the same burstiness and rate characteristics.

The set $\mathcal{P}_i$ is generated by projecting the traffic of channel i from simulation data. We first describe this step in Sec. 5.1. Next, in Sec. 5.2, we describe how we infer the bounded source regulator model for a single pattern from $\mathcal{P}_i$.

5.1 Patterns from Simulations

Each software benchmark defines a specific input sequence for simulating the entire NoC design. However, for abstraction-based verification, we are only interested in analyzing a particular portion of the network.
This requires a model of the traffic that the environment will inject into it, and receive from it. For example, when the network is a grid of routers (nodes), the simulation input sequence involves injecting traffic through the network interface ports of each node. However, if the portion to be analyzed is a single node n_i (or subset of nodes), the environment injects traffic into the network interface port of n_i, and injects traffic via each neighboring input channel (N,S,E,W) of n_i. We wish to extract n_i from the network and analyze it in isolation by replacing its connections to neighbors and the network interface port by non-deterministic sources and greedy sinks. The traffic model is imposed on top of this non-deterministic environment model to ensure that the environment is not overly-pessimistic.
For this purpose, given a input sequence for the entire NoC, we simulate the NoC with that input sequence and record the sequence of packets generated at the input/output ports of n_i. The sequence of packets at these channels is used to infer a traffic model.

5.2 Model Inference from a Simulation Trace

The inferred model must achieve two opposing goals: it must conservatively model *all* portions of the simulation traces, and simultaneously be tight enough to prove performance properties than are not provable under a purely non-deterministic traffic model. Our approach for inferring such a model consists of two parts; finding the *maximum packet density* for every size window of packets in the simulation trace, and *inferring regulator parameters* that model the trace conservatively. For a single channel, we use an example to demonstrate how to infer a model that can be used in a 30-cycle symbolic simulation. The RTL simulation trace for the channel is assumed to have packets arriving at the channel on cycles $2, 14, 21, 37, 41, 99$.

1. We first determine the maximum packet density from the simulation trace. More precisely, we compute pairs $(y(t), t)$, where $y(t)$ is a number of packets, and t is the shortest duration in the trace over which $y(t)$ packets are received. For this example, the worst cases are to receive 1 packet in 1 cycle, 2 packets in 5 cycles, 3 packets in 20 cycles, or 4 packets in 28 cycles. A model must consider all such cases that can occur within the bound used in symbolic simulation. In Fig. 4, these points are denoted by the symbol 'x', and $y(t)$ is considered to be the function that interpolates between these points with straight line segments.
2. Next, a $\mathcal{T}_\mathcal{R}(\rho, \sigma, B)$ regulator is inferred that is conservative with respect to $y(t)$. As described in 4.1, the regulator models an average rate ρ, a maximum burst size σ, and a total number of packets B, with the parameters inferred as follows:
   1. The burst size σ is chosen to be the maximum number of consecutively-arriving packets (e.g. their head flits arrive 3 cycles apart). For this example, $\sigma = 1$.
   2. B is set to be the maximum number packets to ever arrive within a duration equal to symbolic simulation depth. More formally, $B = \max_{t<30} y(t)$; for the example, $B = 4$.
   3. Finally, ρ is chosen to make the regulator as tight as possible while still being a conservative abstraction of the starting trace. Specifically, given the already-determined σ and B, this corresponds to choosing the maximum ρ such that the regulator $\mathcal{T}_\mathcal{R}(\rho, \sigma, B)$ yields a function $\hat{y}(t)$ which bounds the $y(t)$ for the trace. For this example, $\rho = 5$. We compute $\hat{y}(t)$ as the line that passes through $(0, \sigma)$ and which has the smallest slope so that it lies above the $y(t)$ for the trace.

The preceding steps have inferred the regulator $\mathcal{T}_\mathcal{R}(5, 1, 4)$ from the example simulation trace. Figure 4 shows a dashed line represents the maximum channel traffic modeled by this regulator. This line is the continuous function $\hat{y}(t) = \max\left(\sigma + \frac{t}{\rho}, 4\right)$; note that the exact constraint imposed by the regulator is actually a step function along the same line, with each step corresponding to a token being added to the regulator's queue. All traffic patterns that fall below the line are modeled by our regulator, and those above it (e.g. sending 3 packets within 5 cycles) are not. Note that the inferred regulator is just one of many that conservatively model $y(t)$. Figure 4 shows that a less general, but still conservative, model is the conjunction $\mathcal{T} = \mathcal{T}_\mathcal{R}(5, 1, 4) \wedge \mathcal{T}_\mathcal{R}(14, 2, 4)$.
The steps above easily generalize to the case of inferring a model that holds across multiple channels, or for inferring a model from multiple simulation traces. For a multiple channel regulator (Sec. 4.2), the only difference is that the RTL simulation data is the union of the simulation data across all channels of interest. To infer a channel model from multiple simulation traces, the function $y(t)$ from each simulation is combined into a global $y(t)$ by taking the pointwise maximum; the model is then inferred from this global worst-case.

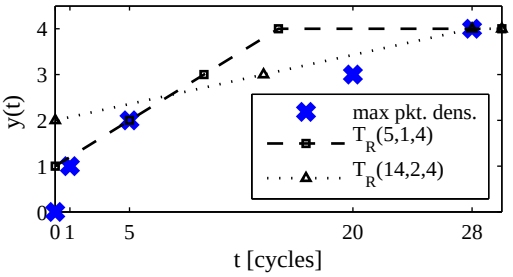


Figure 4: Two possible inferred traffic models for the same simulation data. Both regulators are conservative with respect to all parts of the simulation trace. Note that each rules out some behaviors that the other does not. Our heuristic would infer regulator $\mathcal{T}_{\mathcal{R}}(5, 1, 4)$.

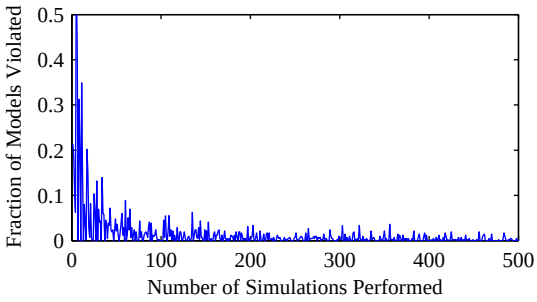


Figure 5: Convergence of inferred traffic models. Traffic models inferred from hundreds of RTL simulations tend to be valid with respect to further simulations.

Node	*Runtimes in Seconds*				Min. Latency Proved
	Symb. Sim.	Dec. Proc. Encod.	SAT Solv.	Total	
ND	19.3	173.7	1399.1	1592.1	16
0,0	16.1	134.3	806.1	956.5	13
1,1	23.2	243.2	3412.1	3678.4	15
2,2	26.8	288.5	1712.4	2027.7	16
3,3	21.6	214.8	3318.5	3554.8	15
4,4	28.5	289.4	2292.0	2609.9	16
5,5	23.3	241.2	3603.6	3868.1	15
6,6	21.6	217.4	4385.8	4624.8	15
7,7	17.5	134.4	671.7	823.6	13

Table 2: Proving latency bounds using TITAN **models** Results for a node with non-deterministic traffic, and for 8 nodes using inferred TITAN traffic models from an 8x8 network. The runtime is broken down into symbolic simulation, UCLID's decision procedure encoding, and SAT solving. The final column give the tightest latency bound that can be proved. In some cases, TITAN models lead to tighter provable bounds than those achieved using pure non-determinism.

6. EXPERIMENTAL RESULTS

TITAN is evaluated on an 8×8 mesh of nodes, where each node is a CMP router with 5 input ports and 5 output ports [17]. Each input port has a single buffer with a depth of 8 slots. The traffic injected into the network during RTL simulation is generated from architectural simulation [21] of the PARSEC benchmark suite [16]. The architectural simulation traces are provided at the packet-level; for RTL simulation, each packet is assumed to be a single head, body, and tail flit injected on the same cycle. RTL simulation is performed using Icarus Verilog.

6.1 Inferring Traffic Models

Each trace of architectural simulation of the PARSEC benchmarks contains hundreds of thousands of packets injected over a span of millions of cycles, making exhaustive RTL simulation a costly proposition. Instead of exhaustive simulation, TITAN infers formal models from RTL simulation of a small, random subset of the traces. For each node, 11 regulator models are inferred. The 11 regulators are one for the traffic entering on each input channel, one for the traffic exiting on each output channel, and one for the sum of the traffic entering on all 5 input channels.

The TITAN models are inferred from 500 RTL simulations, where each simulation is a randomly chosen sample of 5000 cycles from a randomly chosen PARSEC benchmark. It is then evaluated whether the regulator models inferred from the first $i-1$ traces remain valid for the i^{th} trace; i.e., whether the i^{th} trace is a behavior of the models generated from the first $i-1$ traces. Figure 5 shows the fraction of all models that are valid for the first $i-1$ simulations but not for the i^{th} simulation. As the number of simulations performed approaches 500, the vast majority of inferred models remain valid with respect to further RTL simulations. This gives some confidence that the models are largely representative of those that might be inferred from RTL simulation of the exhaustive set of traces.

6.2 Latency Verification

By restricting verification to behaviors that conform to inferred traffic models, it is possible to prove tighter latency bounds than can be proven using full non-determinism. To verify a latency property ϕ on a single node in the network under an inferred traffic model $\mathcal{T}$, the approach is as follows. All 5 input channels are replaced by non-deterministic sources, and all 5 output channels are replaced by eager sinks. The traffic model $\mathcal{T}$ comprises Boolean constraints, as well as regulator constraints on each input channel, each output channel, and a cumulative regulator constraint over all 5 input channels. The node is then symbolically simulated for 20 cycles, and the validity of formula $\mathcal{T} \implies \phi$ is checked. A result of valid means that it is not possible to exceed the latency bound while also conforming to the traffic model $\mathcal{T}$. The minimum latency for which the property holds is found using iterated SMT solving with UCLID, with a different latency bound checked each time.

This approach is evaluated on the 8 nodes along a diagonal of the 8x8 network. Each node has its own traffic model $\mathcal{T}$. Table 2 shows the smallest latency bound that can be proved for each node. The total runtimes for each node are also given, and are broken up into the constituent parts of symbolic simulation, decision procedure encoding, and SAT-solving using MiniSat v2 [11]. The first node in the table imposes no traffic model on its non-deterministic sources, so the formula to be checked is $\mathtt{true} \implies \phi$. In a 20 cycle symbolic simulation, a counterexample can be found to disprove all latency bounds up to 15 cycles, and a latency bound of 16 cycles is found to be valid; note that a much larger bound would be discovered using a greater simulation depth. For some nodes in the network, using inferred traffic models allows a tighter latency bound to be proved. For example, on each of the nodes at the corners of the network (0,0 and 7,7) it is possible to prove a latency bound of 13 cycles. In the case of these corner nodes, the tighter bounds are due largely to the fact that 2 of the 5 input and output channels are completely unused. This information is captured in an inferred traffic model $\mathcal{T}$, but is lost with a purely non-deterministic traffic model. For some other nodes in the network, the inferred traffic models are too conservative to prove any latency bound that is tighter than that achieved with purely non-deterministic traffic.

7. RELATED WORK

NoC designs can be modeled in various ways. Synchronous dataflow (SDF) graphs [13] are suited to modeling networks with regular traffic such as multimedia applications. Network calculus is suited

to reasoning about bursty traffic, as in the Internet. Statistical models are suited to average-case analysis. There is much crossover between the approaches.

SDF: While several efforts have used SDF to model NoCs (e.g. [19]), such modeling is limited to only a subset of NoCs. Assumptions of periodic sources and data-independent routing make SDFs well-suited to modeling multimedia NoCs, but not for general-purpose chip multiprocessor (CMP) NoCs. In a CMP, the injected traffic at each node can vary in burst size, have irregular periods, and choose destinations non-uniformly over time. Additionally, due to the lack of support for conditionals, SDFs are not expressive enough to model NoC designs with detailed routing and arbitrary logic.

Network Calculus: The analysis of general-purpose CMPs is typically based on simulation or probabilistic reasoning. Cruz [9, 10] presented network calculus as an efficient way to compute delay bounds in a FIFO network with sources satisfying burstiness constraints. Stochastic automata networks (SAN) [18] have also been used to model network traffic in SoCs [14]. While SANs allow for efficient reasoning about average case results, they are not suitable for worst-case analysis, as we address. Addressing limitations in the probabilistic analysis of stochastic models, adversarial queuing theory has been proposed [3]. Networks with arbitrary aggregate multiplexing may not satisfy the global FIFO property that is a precondition for tight bounds in network calculus [20].

Statistical modeling: Various statistical traffic models have been used to characterize network traffic. Varatkar and Marculescu show that NoC traffic for a multimedia application is temporally self-similar, and characterize it using a Hurst parameter [22]. Soteriou et al. [21] model traffic flow as a 3-tuple consisting of a Hurst parameter and parameters for hop distance and spatial injection distribution. However, these statistical models are not a formal/conservative representation of the underlying traffic, and designing according to such models can be misleading [12].

Formal Methods: The abstraction of a latency-insensitive system [5] has been explored as a verification aid. A latency-insensitive system assumes only that component delays are multiples of a shared clock, and can be functionally verified independent of the component delays. This abstraction is a useful way for propagating global invariants that can greatly speed up RTL model checking of logical properties of networks [7], and can make certain properties inductive [6]. However, such formal modeling has not been demonstrated to be useful for performance (QoS) properties.

8. CONCLUSION

In this paper, we proposed a new formalism to model traffic in an NoC design. We also gave an automatic technique to infer such a traffic model from simulation traces derived from software benchmarks. We have demonstrated that our abstraction-based approach can efficiently and accurately verify select nodes in an industrial-scale 8×8 NoC design. Apart from the application to formal verification, it would be interesting to explore other applications of our proposed model such as performance diagnosis.

Acknowledgments. This research was supported in part by SRC contract 2045.001, an Alfred P. Sloan Research Fellowship, and the Gigascale Systems Research Center, one of six research centers funded under the Focus Center Research Program (FCRP), a Semiconductor Research Corporation entity.

9. REFERENCES

[1] V. Adve and M. Vernon. Performance analysis of mesh interconnection networks with deterministic routing. *Parallel and Distributed Systems*, Jan 1994.

[2] C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability*, volume 4, chapter 8. IOS Press, 2009.

[3] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson. Adversarial queuing theory. *Journal of the ACM*, 48(1):13–38, 2001.

[4] R. E. Bryant, S. K. Lahiri, and S. A. Seshia. Modeling and verifying systems using a logic of counter arithmetic with lambda expressions and uninterpreted functions. In *Proc. Computer-Aided Verification (CAV'02)*, LNCS 2404, pages 78–92, July 2002.

[5] L. Carloni, K. McMillan, and A. Sangiovanni-Vincentelli. Theory of latency-insensitive design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(9):1059 –1076, Sep 2001.

[6] S. Chatterjee and M. Kishinevsky. Automatic generation of inductive invariants from high-level microarchitectural models of communication fabrics. In *Proc. Computer Aided Verification*, 2010.

[7] S. Chatterjee, M. Kishinevsky, and U. Ogras. Quick formal modeling of communication fabrics to enable verification. In *High Level Design Validation and Test Workshop*, 2010.

[8] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2000.

[9] R. Cruz. A calculus for network delay, part I. network elements in isolation. *IEEE Transactions on Information theory*, 37(1):114 –131, Jan 1991.

[10] R. Cruz. A calculus for network delay, part II: Network analysis. *IEEE Transactions on Information theory*, 37(1):132–141, 1991.

[11] N. Eén and N. Sörensson. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing*, pages 333–336. Springer, 2004.

[12] A. Kahng, B. Lin, K. Samadi, and R. S. Ramanujam. Trace-driven optimization of networks-on-chip configurations. *Design Automation Conferene*, 2010.

[13] E. Lee and D. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[14] R. Marculescu, U. Ogras, and N. Zamora. Computation and communication refinement for multiprocessor soc design: A system-level perspective. *ACM Transactions on Design Automation of Electronic Systems*, 11(3):592, 2006.

[15] L. Ni and P. McKinley. A survey of wormhole routing techniques in direct networks. *COMPUTER,*, Jan 1993.

[16] PARSEC Benchmark Suite. Available at `http://parsec.cs.princeton.edu/`.

[17] L.-S. Peh. *Flow Control and Micro-Architectural Mechanisms for Extending the Performance of Interconnection Networks*. PhD thesis, Stanford University, August 2001.

[18] B. Plateau and K. Atif. Stochastic automata network of modeling parallel systems. *IEEE Transactions on Software Engineering*, 17(10):1093–1108, 1991.

[19] P. Poplavko, T. Basten, M. Bekooij, J. L. van Meerbergen, and B. Mesman. Task-level timing models for guaranteed performance in multiprocessor networks-on-chip. In *CASES*, pages 63–72, 2003.

[20] G. Rizzo and J. LeBoudec. "Pay bursts only once" does not hold for non-FIFO guaranteed rate nodes. *Performance Evaluation*, Jan 2005.

[21] V. Soteriou, H. Wang, and L.-S. Peh. A statistical traffic model for on-chip interconnection networks. In *MASCOTS*, pages 104–116, 2006.

[22] G. Varatkar and R. Marculescu. Traffic analysis for on-chip networks design of multimedia applications. In *Design Automation Conference*, pages 795–800, 2002.