Use Cases for Compositional Modeling and Analysis of Equation-based Constrained Flow Networks*

Nate Soule   Azer Bestavros   Vatche Ishakian   Assaf Kfoury   Andrei Lapets

August 26, 2011

Abstract

Numerous domains exist in which systems can be modeled as networks with constraints that regulate the flow of traffic. Smart grids, vehicular road travel, computer networks, and cloud-based resource distribution, among others all have natural representations in this manner. As these systems grow in size and complexity, analysis and certification of safety invariants becomes increasingly costly. The NetSketch formalism and toolset introduce a lightweight framework for constraint-based modeling and analysis of such flow networks. NetSketch offers a processing method based on type-theoretic notions that enables large scale safety verification by allowing for compositional, as opposed to whole-system, analysis. Furthermore, by applying types to the modeled networks, analysis of composite modules containing incomplete or underspecified components can be conducted. Here we describe various use cases for such modeling tasks, and walk through the development of appropriate NetSketch models.

I Introduction

This document outlines several use cases for the NetSketch formalism. NetSketch provides a framework for the modeling and analysis of constrained flow networks. By employing type theoretic notions to sub-graphs of a network, even large models developed in NetSketch can be efficiently analyzed for satisfaction of safety properties. The formalism, coupled with an implementation provide a lightweight, and user friendly, yet powerful analysis engine that can be applied to many areas of computer science, engineering, and beyond. A detailed description of the formalism [2], and a review of the implementation [4] can be found in related papers.

The remainder of this document is structured around domains and associated use cases. Each domain provides a brief background, followed by a list of use cases - each of which presents:

- A description of the actor in the use case
- The actor's goal
- A URL pointing to related executable models
- The details of a particular problem instance
- An example of solving the problem instance using NetSketch

*This work is supported in part by NSF awards CNS-0952145, CCF-0820138, CSR-0720604, CNS-1012798, and EFRI-0735974.

Each use case is necessarily (for illustrative purposes) a small example of what would likely be much larger and more complex modeling scenarios. One of NetSketch's core strengths is its ability to scale analyses to large models. These uses cases are intended to show the type, and ease of modeling and analysis in NetSketch, and make evident how these actions would scale along with the model.

Wherever possible the examples use linear constraints to model the problem domain so as to be executable in the current version of the implementation[1]. Where clear benefit from the use of nonlinear constraints was evident, this restriction was abandoned, and is explicitly stated.

II Domain: Cloud Service Level Agreements (SLA)

Background

The Amazon Elastic Compute Cloud (EC2) provides compute capacity to end users via virtual machines. Users select from a predefined set of virtual machine instance types depending on their processing needs. Each instance type is bound by a given minimum specification, and associated pricing model. An EC2 standard "small" instance type, for example, guarantees 1.7 GB of memory, a virtual single core processor capable of providing 1 EC2 Compute Unit (ECU), and 160 GB of local storage. One EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. Note that the Amazon cloud environment is used here to provide concrete examples, however the same prinicples described apply equally well to other cloud based frameworks.

Use Case 1

Actor: Cloud Infrastructure Provider

Goal: Find configurations that meet the service level agreement (SLA) requirements of a given user VM instance type.

URL: http://csr.bu.edu/netsketch/NetSketchClient.html?loadModel=Cloud Producer

Problem Description: A great many configurations of software and phsyical hardware can meet the specifications listed for any given EC2 instance type, and from a technical perspective Amazon need not restrict itself to a particular set of real or virtual components (consistency among components for maintenance and cost drivers is of course a motivating factor in any business model). Thus when designing new configurations to back a given instance type a great deal of lattitude in selection exists. Each potential configuration must be analyzed to ensure that it provides the minimum resources outlined in the instance description. Further, it is in the best interest of Amazon to maximize the financial income from the hardware and software they provide, and thus they must also strive to ensure they are not over provisioning the systems (by either minimizing cost via changing/reducing the provided hardware, or by maximizing profit by increasing the number of virtual instances running on that hardware). Thus relatively tight bounds must be adheared to.

[1]The NetSketch formalism takes a constraint language as a parameter. The current tool is constructed to work with linear constraints. The implementation can be accessed via a link in the NetSketch section of the iBench website at: http://www.cs.bu.edu/groups/ibench

This problem can be approached from two opposite directions. A set number of user instances could be selected, and then a configuration searched for that will satisfy these instances. Alternatively a configuration could be established and analyzed to determine the number and type of user instances that could be safely supported by such a configuration.

Example Here we will walk through an example of the problem type defined above. For the sake of brevity this example will be kept to a small size, though the concepts described extend to more elaborate and complex scenarios (indeed one of NetSketch's core strengths is the ability to analyze large systems). In this example a cloud infrastructure provider would like to test if a particular hardware configuration can support three instances of virtual machines: two of type "Small" and one of type "Large". The requirements for these instances are outlined in Figure 1[2].

Instance Type	**CPU (ECU)**	**RAM (GB)**	**Storage (GB)**
Small	1	1.7	160
Large	4	7.5	850

Figure 1: Instance specifications

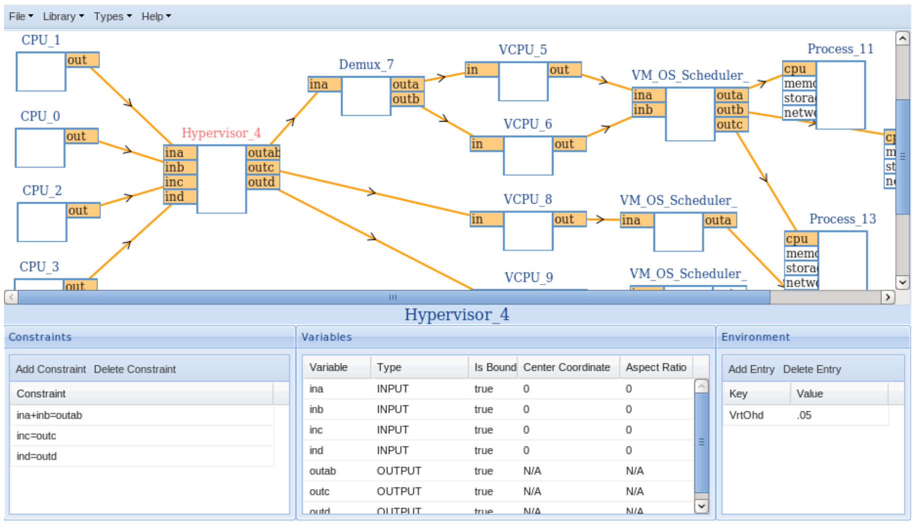
Here we will concentrate on modeling the CPU requirements. RAM, disk storage, and other resource requirements (*e.g.* network) would be modeled similarly and could be done within the same NetSketch model. Linear constraints are used here, though if real-time processes were to be included in the model, nonlinear constraints may be better suited to the problem. For an example of what form periodic nonlinear constraints might take, see Section III.

To begin we define a phsyical machine on which the virtual machines will run. As we are concentrating on processor requirements we define this machine by its CPU's. Within the NetSketch environment we access the menu `Library` → `Cloud Infrastructure`, and select `Phsyical CPU`. Here we repeat this process 4 times to define a 4-processor machine. In the typical case today these processors are all uniform, however as described in work from the Barrelfish project [3] heterogeneous systems may well provide a more suitable way to meet future processing demands. To demonstrate that homogeneity is not a requirement of the modeling system we will define a heterogeneous system.

The decision of units in Netsketch is left to the modeler. Here, for simplicity, we will work directly with Amazon's Elastic Compute Unit, the ECU. Each processor has a port representing the cycles it produces (in terms of ECU). The variable out represents this port. We define CPU_0 and CPU_1 to have $\mathsf{out} = 2$, and CPU_2, and CPU_3 to have $\mathsf{out} = 1$. We introduce a node representing the hypervisor, which will serve to regulate (schedule) which preocesses receive which shares of the total processing power of the physical machine. We select the `Mux-Demux` component from the `Cloud Infrastructure` library. This is a parameterized library element and thus we are prompted to provide the number of inputs, and the number of outputs (with the system building the appropriate constraints based on our answers)[3]. We have 4 processors that we wish to divide among 3 virtual machines, thus we select 4 and 3 respectively. As depicted in Figure 2 we will modify the constraints

[2]The requirements were selected for the purpose of example from the Amazon instance types listing [1], which may change over time.

[3]The ability to parameterize is not a hardcoded feature of this library element. NetSketch library elements use functions as first class descriptive elements when defining a library model. Thus when the number or relationship of inputs and outputs may vary based on user input this is simply encoded in the function describing the component's inputs, outputs, and/or constraints.



Constraint
ina+inb=outab
inc=outc
ind=outd

Variable	Type	Is Bound	Center Coordinate	Aspect Ratio
ina	INPUT	true	0	0
inb	INPUT	true	0	0
inc	INPUT	true	0	0
ind	INPUT	true	0	0
outab	OUTPUT	true	N/A	N/A
outc	OUTPUT	true	N/A	N/A
outd	OUTPUT	true	N/A	N/A

Key	Value
VrtOhd	.05

Figure 2: NetSketch model of a cloud configuration depicting both infrastructure provider, and cloud consumer components.

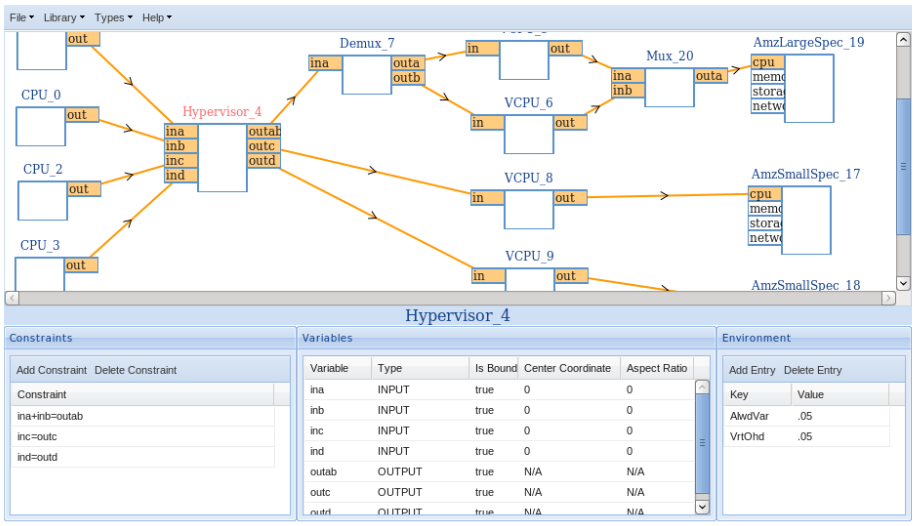
of this node to capture the notion that two of our virtual processors are pinned to physical processors, while the remaining two virtual processors can float between the remaining physical processors.

The input ports of the Hypervisor node are labelled ina, inb, inc, ind and the output ports outab, outc, outd. We'll route the 2 2-ECU processors to the large instance virtual machine, which requires 4 ECU's, and pin each of the 1-ECU processors to the two small instances. To do this we modify the constraints of Hypervisor to be:

$$\text{ina} + \text{inb} = \text{outab}$$
$$\text{inc} = \text{outc}$$
$$\text{ind} = \text{outd}$$

We desire the output outab to be routed to a single machine, but one with two virtual processors. We model this by again using the `Mux-Demux` parameterized library component to create a 1 input, 2 output node. This is connected to the outab port of the Hypervisor. We then create two virtual CPU's by adding them from the `Cloud Consumer` library, and connect them to the outputs of the demultiplexer. These library components introduce a new global environment constant, VrtOhd to represent the virtualization overhead cost. The constraints thus regulate the virtual processor to output the cycles it receives from the physical processor, minus a percentage described by VrtOhd. The two processors for the small instance virtual machines are added to the model in a similar way, but connected directly to the hypervisor via outc and outd respectively. In this way they have each been pinned to a single physical processor.

Figure 2 depicts the above scenario, with additional model elements representing virtual machine operating system schedulers connected to the virtual processors, and these schedulers connected to particular processes. This system could then be analyzed for satisfaction of the processes' requirements given the physical and virtual structure described. This, however, represents the modeling activities of two separate groups: cloud infrastructre providers, and cloud infrastructure consumers. A more practical representation is portrayed in Figure 3. Here library components representing the specifications of small and large instances are placed on the canvas, replacing the individual



Constraint
ina+inb=outab
inc=outc
ind=outd

Variable	Type	Is Bound	Center Coordinate	Aspect Ratio
ina	INPUT	true	0	0
inb	INPUT	true	0	0
inc	INPUT	true	0	0
ind	INPUT	true	0	0
outab	OUTPUT	true	N/A	N/A
outc	OUTPUT	true	N/A	N/A
outd	OUTPUT	true	N/A	N/A

Key	Value
AlwdVar	.05
VrtOhd	.05

Figure 3: NetSketch model of a cloud configuration for verifying infrastructure meets the requirements of an instance type.

processes with a higher level of abstraction. These components contain constraints that require their inputs to meet the minimum requirements of their respective VM instance types, and are thus connected to the output from the 3 virtual machines.

Performing type inference on the entire network will tell us if the system meets its requirements. If no type can be inferred then no range of values on any of the unbound ports can lead to guaranteed safe operation (here safe meaning the SLA is met). As described the system may not actually be safe. The virtualization overhead represented in the processor constraints on each virtual processor means that the output cycles from these nodes will always be less than the input cycles. Thus while a total of 6 ECU's are generated by the physical processors, and a total of 6 are required, the system leaks some of this processing power to virtualization maintenance tasks. In this case this may actually be acceptable, as an ECU is defined as providing the compute power of a 1.0 - 1.2 Ghz CPU, and thus provides a small amount of allowed variance. In Figure 3 this is captured via the AlwdVar environment variable, which has been referenced in each of the instance specification nodes.

Inferring a type for the entire system is not unreasonable in this small example, however doing so essentially reduces the analysis to work on whole-system rather than compositional principles. The true power of NetSketch comes from its ability to use types to allow for greater scalability. As you extend this example to a larger more realistic network, the model in Figure 3 would likely be one of many substructures in the entire cloud infrasutrcture. This substructure could be given a type and thus easily reused and composed with other substructures. Formations of disk arrays, network devices/connections, compute nodes, etc. could thus be connected in various ways and recursively typed at various levels of abstraction (cores $\rightarrow$ processors $\rightarrow$ servers $\rightarrow$ racks $\rightarrow$ data centers, etc) and tested to determine how many instances they support, or how well they support a given instance set. When operating on typed models, even very large networks can be analyzed efficiently, allowing for many what-if scenarios to be tested.

Use Case 2

Actor: Cloud Consumer

Goal: Determine the cheapest instance type that will support application requirements.

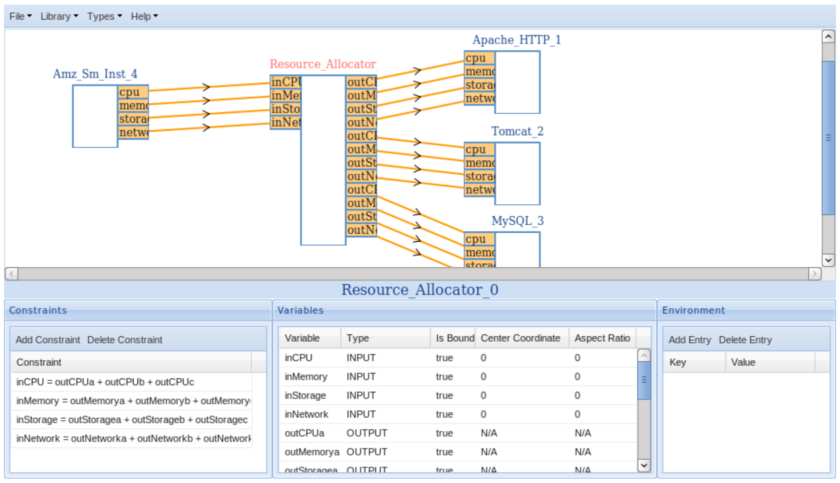
URL: http://csr.bu.edu/netsketch/NetSketchClient.html?loadModel=Cloud Consumer

Problem Description: A user desiring to deploy their web application to Amazon's cloud environment must select an instance type from a variety of options. Each option provides different compute capacities, and different pricing. The user wishes to minimize the cost of this cloud deployment while also guaranteeing that their minimum requirements for the web application will be satisifed. The user knows the processes required for their application, and the resource requirments of these processes. They wish to determine the most cost effective, safe, option for their instance selection.

Example Here an example of a model describing cloud consumer instance type selection will be presented. For illustrative purposes only a small model will be described, but the designs documented here extended to larger and more complex scenarios.

We will construct a model for a simple web application consisting of 3 core processes/applications:

1. An HTTP server - Apache HTTP Server
2. A Java Web Container - Tomcat
3. A RDBMS - MySQL



Constraint
inCPU = outCPUa + outCPUb + outCPUc
inMemory = outMemorya + outMemoryb + outMemory
inStorage = outStoragea + outStorageb + outStoragec
inNetwork = outNetworka + outNetworkb + outNetwork

Variable	Type	Is Bound	Center Coordinate	Aspect Ratio
inCPU	INPUT	true	0	0
inMemory	INPUT	true	0	0
inStorage	INPUT	true	0	0
inNetwork	INPUT	true	0	0
outCPUa	OUTPUT	true	N/A	N/A
outMemorya	OUTPUT	true	N/A	N/A
outStoragea	OUTPUT	true	N/A	N/A

Key	Value

Figure 4: NetSketch model of a cloud deployment for use in inferring the most appropriate instance type.

As in the previous example we will draw largely from NetSketch's library components, modifying where required[4]. We begin by creating an instance of the `Resource Allocator` component. This

[4]NetSketch modules may also be created from scratch and manually given constraints/ports.

will allow us to model receiving compute capacity, as well as RAM, disk storage, and network bandwidth on the input ports, and scheduling/routing that capacity to 1 or more processes. Since the library cannot predict the number of attached processes to model, the `Resource Allocator` is a parameterized component. Upon selecting it from the NetSketch library we are prompted to provide the number of processes that we wish to connect to this new component. For this example we select 3 in order to represent the processes listed above.

Next we create nodes to represent processes by selecting `Cloud Consumer` → `Process` from the library menu. We do this 3 times, giving each process a descriptive name. The `Resource Allocator` component has 4 outputs for each of the 3 processes representing CPU, RAM, disk storage, and network bandwidth. Connecting these outputs to the corresponding inputs on each process effectively allows routing of each commodity through the allocator.

We now specify the minimum requirements that each of our processes needs in order to operate safely. Here safety may be defined as allowing the processes to execute, or more likely allowing the processes to operate at the speed desired under the load expected (*i.e.*, MySQL may be able to operate given some minimum amount of RAM, CPU, disk, and network bandwidth, but may need higher levels of some or all of those resources to operate at acceptable performance levels given a particular maximum load).

For this example we specify the minimum requirements as shown in Figure 5. No requirements of network bandwidth are listed here as no streaming or intensive throughput use cases are being modeled for this particular application.

Process	**CPU (ECU)**	**RAM (GB)**	**Storage (GB)**	**Network (Mb/s)**
Apache HTTP Server	0.2	205	3.5	0
Tomcat	0.25	185	1.8	0
MySQL	0.4	300	4	0

Figure 5: Instance specifications

Figure 4 depicts this model connected to an Amazon Small Instance. In this way type inference can be used to determine if the model is safe (*i.e.* the small instance provides enough resource capacity to meet the minimum requirements of the processes). Alternatively the model without the Amazon small instance present could have a type inferred and then this type could be compared to the available instance types to find the best match. A potential extension to the tool could automate this by allowing the user to select an objective function for finding the best match (*i.e.* smallest encompassing bounds, etc) given a set of candidates.

III Domain: System Design

Background

In the design of systems various components are composed and connected to allow data and processing to flow in a desired manner. Each component will produce, consume, or store data - or potentially will exhibit a combination of these behaviors. Examples of such systems can be found in operating system design, enterprise application design, among others.

Use Case 1

Actor System Designer

Goal Determine required maximum queue depths for a system of processes connected via data flowing through queues.

URL: http://csr.bu.edu/netsketch/NetSketchClient.html?loadModel=Processes And Queues

Problem Description A system architect has designed a series of processes that will interact via passing messages through queues. The maximum rate at which each process will produce messages and the minimum at which each will consume messages is known. A certain amount of resources must be allocated to each queue. The designers desire to limit this resource allocation to the minimum requried to support the expected maximum usage of the system. To do so they must establish the maximum queue depth given the stated rates of consumption and production of messages.

Example In this example a system of 6 processes communicating via 2 queues is modeled and analyzed to determine safe maximum queue depths. To best capture the operation of this system, nonlinear constraints are used. This form of constraints fits well within the NetSketch formalism, but is not executable in the current implementation.

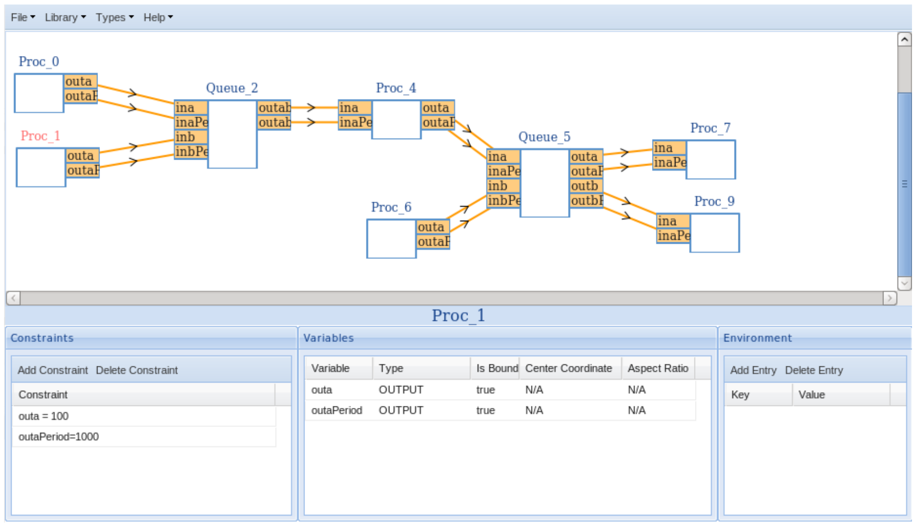
In this model the message production and consumption are periodic. Each process produces/consumes m messages every t milliseconds. m and t are modelled explicitly as outputs or inputs corresponding to production and consumption respectively. A single process could have multiple streams of messages being produced and/or consumed, and thus may have a series of ports $m_0...m_n$ and $t_o...t_n$. Queues have similar input and output ports for allowing messages to flow into, and out of the queue.

In this model we use two library elements, `System Process` and `Queue` from the `Systems` library. Processes and queues are added to the canvas as shown in Figure 6. Processes may have simple constraints which explicitly describe their input or output rates. $\mathsf{Proc_1}$ in Figure 6 for example has outa (the number of messages) bound to 100, and $\mathsf{outaPeriod}$ (the period) bound to 1000, indicating that this process produces 100 messages every 1000 milliseconds.

While a given queue may accumulate messages over the short term, the system designer wants to ensure that in the long term all messages put on the queue are eventually processed. This property is modeled, for $\mathsf{Queue\_2}$ for example as:

$$\frac{\mathsf{ina}}{\mathsf{inaPeriod}} + \frac{\mathsf{inb}}{\mathsf{inbPeriod}} \leq \frac{\mathsf{outab}}{\mathsf{outabPeriod}}$$

The designer may further include a constraint to dictate that the depth of a given queue never goes beyond a certain threshold. $\mathsf{Queue\_5}$ for example may contain this constraint:



Constraint
outa = 100
outaPeriod=1000

Variable	Type	Is Bound	Center Coordinate	Aspect Ratio
outa	OUTPUT	true	N/A	N/A
outaPeriod	OUTPUT	true	N/A	N/A

Key	Value

Figure 6: NetSketch model of a system of processes communicating via queue based message passing.

$$
\begin{aligned}
500 \geq\ & \\
& \mathsf{max}(\mathsf{ina} + \mathsf{inb}, \\
& \quad ((\mathsf{floor}(\tfrac{\mathsf{outaPeriod}}{\mathsf{inaPeriod}}) * \mathsf{ina}) + (\mathsf{floor}(\tfrac{\mathsf{outaPeriod}}{\mathsf{inbPeriod}}) * \mathsf{inb})) - ((\mathsf{floor}(\tfrac{\mathsf{outaPeriod}-1}{\mathsf{outaPeriod}}) * \mathsf{outa}) + (\mathsf{floor}(\tfrac{\mathsf{outaPeriod}-1}{\mathsf{outbPeriod}}) * \mathsf{outb})), \\
& \quad ((\mathsf{floor}(\tfrac{\mathsf{outbPeriod}}{\mathsf{inaPeriod}}) * \mathsf{ina}) + (\mathsf{floor}(\tfrac{\mathsf{outbPeriod}}{\mathsf{inbPeriod}}) * \mathsf{inb})) - ((\mathsf{floor}(\tfrac{\mathsf{outbPeriod}-1}{\mathsf{outaPeriod}}) * \mathsf{outa}) + (\mathsf{floor}(\tfrac{\mathsf{outbPeriod}-1}{\mathsf{outbPeriod}}) * \mathsf{outb})))
\end{aligned}
$$

which uses a ternary max() function to ensure that for the system to be safe it must have a message depth of no more than 500. Alternatively, as the case is in this example, the designer may ask the NetSketch system to determine for them what the maximum safe depth is. One way to accomplish this is by adding a new output port to the queue representing the queue depth. This port, maxDepth would be regulated by a constraint of:

$$
\begin{aligned}
\mathsf{maxDepth} =\ & \\
& \mathsf{max}(\mathsf{ina} + \mathsf{inb}, \\
& \quad ((\mathsf{floor}(\tfrac{\mathsf{outaPeriod}}{\mathsf{inaPeriod}}) * \mathsf{ina}) + (\mathsf{floor}(\tfrac{\mathsf{outaPeriod}}{\mathsf{inbPeriod}}) * \mathsf{inb})) - ((\mathsf{floor}(\tfrac{\mathsf{outaPeriod}-1}{\mathsf{outaPeriod}}) * \mathsf{outa}) + (\mathsf{floor}(\tfrac{\mathsf{outaPeriod}-1}{\mathsf{outbPeriod}}) * \mathsf{outb})), \\
& \quad ((\mathsf{floor}(\tfrac{\mathsf{outbPeriod}}{\mathsf{inaPeriod}}) * \mathsf{ina}) + (\mathsf{floor}(\tfrac{\mathsf{outbPeriod}}{\mathsf{inbPeriod}}) * \mathsf{inb})) - ((\mathsf{floor}(\tfrac{\mathsf{outbPeriod}-1}{\mathsf{outaPeriod}}) * \mathsf{outa}) + (\mathsf{floor}(\tfrac{\mathsf{outbPeriod}-1}{\mathsf{outbPeriod}}) * \mathsf{outb})))
\end{aligned}
$$

In this way the designer can then infer a type for the system, including this unbound output port, and thus determine the safe range of values for the queue depth. An extension of the current implementation to allow for internal variables would provide an alternative mechanism for defining and inferring the maximum queue depth.

IV Summary

NetSketch's use of compositional analysis techniques brings reasoning about large scale versions of these and other scenarios that may have been formerly intractable into the realm of real world practicality. The NetSketch toolset exposes this power via a lightweight, user friendly mechanism. The uses cases presented here illustrate just a sample of the domains and scenarios that can be naturally represented as equation based constrained flow networks, and thus modeled and analysed in the NetSketch framework. Ongoing work on the system is exploring more expressive constraints, more precise and manageable types, further user enhancements to the tool, as well as integrations with other equation based modeling solutions. All of these enhancements will highlight further use cases and domains that can benefit from NetSketch analysis.

References

[1] Amazon. Amazon EC2 instance types.
http://aws.amazon.com/ec2/instance-types/, 2011.

[2] Azer Bestavros, Assaf Kfoury, Andrei Lapets, and Michael Ocean. Safe Compositional Network Sketches: Formalism. Technical report, Department of Computer Science, Boston University, Boston, MA, USA, 2009. Tech. Rep. BUCS-TR-2009-029, October 1, 2009.

[3] Adrian Schpbach, Simon Peter, Andrew Baumann, Timothy Roscoe, Paul Barham, Tim Harris, and Rebecca Isaacs. Embracing diversity in the barrelfish manycore operating system. In *Proceedings of the Workshop on Managed Many-Core Systems*, June 2008.

[4] Nate Soule, Azer Bestavros, Assaf Kfoury, and Andrei Lapets. Safe Compositional Equation-based Modeling of Constrained Flow Networks. Technical report, Department of Computer Science, Boston University, Boston, MA, USA, 2011. Tech. Rep. BUCS-TR-2011-014, June 5, 2011.