Scalable Simulation of Complex Network Routing Policies

Andrew I Stone
Colorado State University
stonea@cs.colostate.edu

Steven DiBenedetto
Colorado State University
dibenede@cs.colostate.edu

Michelle Mills Strout
Colorado State University
mstrout@cs.colostate.edu

Daniel Massey
Colorado State University
massey@cs.colostate.edu

ABSTRACT

Modern routing protocols for the internet implement complex policies that take more into account than just path length. However, current routing protocol simulators are limited to either working with hard-coded policies or working on small networks (1000 nodes or less). It is currently not possible to ask questions about how the routing tables will change on all of the autonomous systems (e.g., AT&T, Sprint, etc.) in the internet, given a change in the routing protocol. This paper presents a routing policy simulation framework that enables such simulations to be done on resources that are readily available to researchers, such as a small set of typical desktops. We base the policy simulation framework on the Routing Algebra Meta-Language (RAML), which is a formal framework for specifying routing policies. Our theoretical contributions include proving that the signatures and the meet operation induced by the preference operator in RAML define a semilattice and that routing policy simulation frameworks are analogous to data-flow analysis frameworks.

The main problem we address is that direct implementation of routing policy simulation has scaling issues due to the $O(n^2)$ memory requirements for routing tables. However, due to properties of routing algebras specified in RAML, we are able to segment the simulation problem into multiple runs that propagate route information for subsets of the network on each run. This strategy enables us to perform a simulation that does not exceed system memory on typical desktops and enables the 43 minute, parallel simulation of a real network topology (33k nodes) and an approximation of the common BGP protocol.

Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: Routing protocols;
D.3.3 [**Language Constructs and Features**]: Frameworks;
I.6.8 [**Types of Simulation**]: Parallel

CF'10, May 17–19, 2010, Bertinoro, Italy.

General Terms

Algorithms, Languages, Performance

Keywords

Metarouting, Simulation, Routing, Parallel, Performance, Data-flow analysis

1. INTRODUCTION

Network routing algorithms such as those used in the internet are a combination of protocol mechanisms and routing policies. A *mechanism* dictates how route information is propagated through the network, and a *policy* indicates how a router selects amongst possible routes to a particular destination node. In early routing protocol design the focus was on mechanism, with simple route selection policies such as shortest paths. However, policy has become a critical and often not well understood component of routing. ISPs no longer choose routes solely on shortest path [21], but instead makes decisions based on a variety of factors such as economics, path length, load, and security.

Policies can be complex and hard to specify, let alone implement. In order to truly understand modern routing policies one needs a way of formalizing them as well as a way to simulate different polices on different network topologies. Furthermore, these topologies must be on the order of tens of thousands of nodes since many factors only arise at such scales. The network research community face the challenge of both how to specify, and how to simulate, routing policies.

In this paper, we present a scalable, routing policy simulation framework that supports a subset of the routing algebra metalanguage (RAML) formalization proposed by Griffin and Sobrinho [7]. RAML provides the formalization for specifying policies (i.e., network protocol programming model), and we provide a framework for efficiently simulating routing policies on very large topologies. Current network protocol simulators work well with small networks [3], and some simulators are able to scale when hard-coded for specific policies [18, 22], however *no previous simulator has provided programmable routing policy simulation on very large graphs.*

A routing policy simulation framework enables networking researchers to ask the following questions: (1) what routes will ISPs choose based on routing policy, (2) how do routing tables change with changes in topology, and (3) what changes occur if the policy is changed (slightly or dramatically). These questions are simple and fundamental, but nevertheless difficult to answer with the current set of net-

work simulation tools. We have already used the routing policy simulation framework presented here to determine how close simulations of BGP on a couple of empirically determined topologies match the actual routing table entries [4].

The main problems that must be overcome to enable scalable routing protocol simulation on realistic network topologies are the $O(N^2)$ memory requirement for the routing tables and long simulation times. Our implementation is able to keep the memory requirement to $O(N)$ and use the same approach for reducing memory to easily parallelize the simulation. A technique we call *segmentation* reduces the size of routing tables at each node in the network simulation to a constant number of entries, one route for each node in the subset of currently originating routes. Additionally, segmentation presents us with tasks that can be run concurrently with almost linear speedup on a set of workstations in an active undergraduate computer science lab[1]. This speedup shows that our tool can answer the networking questions we have outlined in a practical time-frame, using hardware that is accessible.

In this paper, we make the following contributions:

- We present a routing algebra simulation framework that enables metarouting specifications using the Routing Algebra Meta Language (RAML) specified in [7].
- We compare the routing algebra simulation framework with data-flow analysis frameworks and prove that the route signatures and preference operator in RAML form a semilattice.
- We present a segmentation algorithm for performing the routing policy simulation that reduces the simulation memory requirements from $O(N^2)$ to $O(N)$ and enables easy parallelization.
- We show that it is possible strictly bound the number of visits to each node that the simulator must make to converge on a routing solution; however this technique does not improve performance on real network topologies.
- We show experimental results based on a number of simulation variants and parameters including segmentation size, use of a priority queue to tightly bound number of node visits, multicore parallelization, and distributed parallelization. The best configuration to simulate a BGP-like protocol on a thirty-three thousand node graph derived from internet measurement takes forty-three minutes when run across ten workstations in an active undergraduate lab.

The rest of this paper is organized as follows: in Section 2 we review routing protocols, present the analogy between RAML and data-flow analysis frameworks, and present the RAML-based simulation framework we have implemented. In Section 3, we detail and evaluate the performance aspects of the simulation tool including the segmentation algorithm that enables scalable simulations. We present related work in Section 4, and then end the paper with concluding remarks in Section 5.

[1]Our experiments were all run during the last week of a semester when every machine in the lab was being used interactively and in batch mode by students finishing course projects.

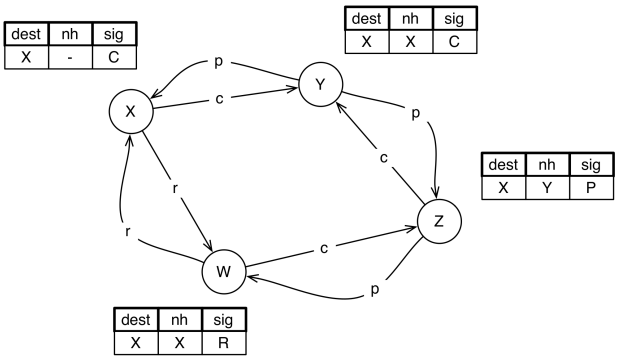

Figure 1: In this routing example, each node in the graph only has a routing table entry for the destination X. If node Z wants to send a packet to node X, then it will send the packet to the next hop Y.

2. ROUTING POLICY SIMULATION FRAMEWORK

In this section, we review metarouting [7]. Routing algebras provide the formal underpinnings for metarouting and share many concepts with data-flow analysis frameworks. Using the data-flow analysis analogy, we are able to show that the routing algebras, which were originally based on semiring algebras, have evolved to be based on meet semilattices. Finally, we leverage the theory to provide a simulation framework that enables efficient routing algebra specification and simulation.

2.1 Review of Metarouting

Metarouting [7] is a declarative programming model for specifying routing protocols. A routing protocol consists of a *mechanism* that dictates how route information is propagated through the network and a *policy* that indicates how a node selects amongst possible routes to a particular destination node and how propagation of a route over a link affects that route's preference in the selection process. The main routing protocol used between autonomous systems (e.g., AT&T, Sprint, etc.) is called BGP (Border Gateway Protocol). BGP uses a routing *mechanism* called path vector, where nodes send and receive routing information to and from their neighbors. In path vector, nodes do not have topology information about the full network. Instead, each node only knows its set of neighbors. A *routing solution* consists of a routing table at each node that indicates which neighbor should be the next hop for a packet addressed to any node in the network.

The routing policy for BGP is complex, so we only use the local preference aspect of BGP to illustrate how routes can be propagated in a network. Local preference is a policy in which a provider is willing to propagate routes its customers and peers, but not to any of its own providers. Figure 1 contains a four-node network with nodes X, Y, Z, and W. The edges in the graph are labeled with policy information that contributes to the local preference aspect of the BGP routing policy. In Figure 1, we are propagating routes to the node X. Note that each node contains a routing table with one entry indicating the route when the destination

is node X and a signature associated with that route. The *signature* indicates a route preference. The signature capital C indicates a customer route, and it is the most preferred. R indicates a peer route, and P indicates a provider route. The preference ordering is $C \prec R \prec P \prec \phi$, with $A \prec B$ indicating that A is more preferred than B.

The route propagation starts at node X itself, which pushes the route (X,-,C) over its outgoing edges. When a route traverses an edge, the label on that edge is applied to the route signature to produce a new route signature. When the label little c is applied to the signature capital C, the signature stays the same as can be seen with the route (X,X,C) at Y. When the label little r is applied to signature capital C, the signature becomes R as can be seen with the route (X,X,R) at node W. By assigning the integer value 1 to capital C and little c, 2 to capital R and little r, and 3 to capital P and little p, we can express the local preference label operation as a function, $f_l(x) = (\text{ if } (x \leq l) \text{ return } l \text{ else return } \phi)$, where l is the numeric value for the label and x is the numeric value for the input signature. ϕ is a special signature that indicates there is no path to the destination node.

In Figure 1, the Z node must select between two possible routes to X. However, since the signature R propagated across the c edge goes to ϕ, the route from node Y with signature $f_p(C) = P$ is more preferred.

In general, a routing algebra as defined by Griffin and Sohbrino in the Metarouting paper [7] can be specified with the tuple:

$$A = (\Sigma, \preceq, L, \oplus, \mathcal{O}),$$

where Σ is the set of possible signatures, $\preceq$ is a preference operator[2] between signatures such that $s_1 \preceq s_2$ indicates that s_1 is more preferred than s_2, L is a set of policy labels that can be associated with route propagation directional edges in the network topology graph, $\oplus$ is the label operator that given a label and a signature returns a new signature, and $\mathcal{O}$ is the set of signatures that can be assigned to a route at the route's originating node. The Routing Algebra Meta-Language (RAML) specified in [7] aims to compose algebras to create new algebras to model various routing policies. BGP is approximated as a routing algebra in [7].

Sobrinho [20] showed that any strictly monotonic algebra will converge [20]. A strictly monotonic algebra is one where the signature returned from a label application is always less preferred than the input signature,

$$\sigma \prec l \oplus \sigma.$$

Sobrinho [20] uses this property and the assumption of a vectoring mechanism for the propagation of routes to prove that the routes determined by the simulator are not necessarily globally optimal, that the iterative simulation algorithm results in the same route signatures that a networking running the same routing policy would, and that the routing policy will result in deterministic routes if the next hop identifier is incorporated into the signatures. In summary, strictly monotonic algebras that include the next hop in the signature will result in a single routing solution on a static network.

[2]Note that *local preference* is full routing policy algebra and that the *preference operator* is an operator that compares between roue signatures. The intersecting use of the word "preference" is incidental.

2.2 Comparison with Data-Flow Analysis Frameworks

Griffin compares the formalization of routing policies in RAML to the formalization of programming language syntax with context free grammars [1]. Using that analogy, the routing policy framework we present here would be akin to YACC [12], which provides parser implementations given a grammar specification. However, we find that another analogy is even more appropriate. Specifically, routing policy frameworks have much in common with data-flow frameworks, and we leverage the similarities and differences between the two formalizations to describe routing policy frameworks to a programming languages research audience, show that routing algebras are based on semilattices, and in Section 3 argue the correctness of the segmentation algorithm, which enables scalable and efficient routing policy simulation.

Data-flow analysis is a technique for analyzing programs at compile time to determine attributes such as when variables are live, dependence relationships between statements, etc. Data-flow analysis problems can be formalized with a meet semilattice and properties on a set of transfer functions that map to and from the semilattice set [14, 15, 2, 13]. A data-flow analysis framework can be specified with the following tuple:

$$DFA = (D, V, \wedge, F),$$

where D is the direction of the data-flow analysis, V is the set of possible data-flow values, $\wedge$ is the meet operator for combining more than one data-flow value, and F is a set of transfer functions that model the effect of program statements on the data-flow values [2]. The analogy between routing algebras and data-flow analysis problems is that a route signature is like a data-flow value, the route preference operator is like the partial ordering induced by the meet operator, and the policy labels and label operator are like the statements and their associated transfer functions.

Routing algebras were originally based on semiring algebras [17], but Griffin and Sobrinho [7] argue that the requirement that labels and signatures come from the same set is too restrictive for expressing more general routing policies. We observe that the set of signatures and the meet operation induced by the preference operator have the same characteristics as a semilattice.

THEOREM 1. *A signature set Σ, a weight function $w()$ that maps each signature to an integer, and the $min()$ function define a semilattice.*

Proof: In a semilattice, the meet operator must be idempotent, commutative, and associative. In RAML, the meet operator is the *min* function. The *min* function is used to select amongst a set of signatures. Each possible route signature is mapped to an integer weight with a function w such that if $w(s_1) \leq w(s_2)$ then s_1 is preferred to s_2, $s_1 \preceq s_2$. The semilattice set of values can be the integers in the range of the weight function given the domain of signatures. The min function is idempotent ($min(x, x) = x$), commutative ($min(x, y) = min(y, x)$), and associative ($min(x, min(y, z)) = min(min(x, y), z)$).

A semilattice must also have a top element $\top$ such that for all x in the set, x meet $\top$ equals x. The set of signatures

always includes an element ϕ such that for all other signatures s, $s \prec \phi$. Therefore the weight function w will always map the signature ϕ to a larger integer than all of the other signatures and ϕ is $\top$. ▌

We leverage the analogy with DFA by developing a routing policy simulation framework similar to the many data-flow analysis frameworks that have been developed [8, 11, 16]. In a data-flow analysis framework, a program analysis can be specified by providing some specification for the possible data-flow values, the meet operator, and the transfer functions (one for each statement type). In our routing policy framework, the user specifies the possible signature values, the preference operation between signatures (which in turn induces a meet operator), and a label operation per label type.

2.3 Specifying a Routing Algebra

We implement the routing policy framework within a tool called the MetaRouting Simulator (MR.Sim). The MR.Sim tool iteratively propagates routes given a routing algebra and a network topology with policy labels associated with directed edges between nodes. In MR.Sim, a routing algebra is defined in C++ by subclassing signature, preference operator, label, and label operator base classes and defining the required interface for each of the algebra components. Users provide MR.Sim with network topologies and parameters used in calculating policy labels via a text file. This section details the process of using the MR.Sim tool to model an example routing policy that is similar to BGP [3].

Three of the main components of BGP are local preference, path, and breaking route ties with the router id. Local preference formalizes the idea that an internet service provider is willing to propagate routes for its customers, but not necessarily for its peers or its own provider. The path component is a list of nodes in the route path. The last component is the router id. We elide other aspects of BGP to simplify the example.

To model the three main components we specify three corresponding algebras from [7]:

1. FM(LP(3));
2. SIMSEQ(0,N-1), where N is the number of nodes in the graph; and
3. RouteID.

The BGP-like algebra is then defined as a composition of the three simpler algebras.

With MR.Sim, routing algebras are specified by subclassing a set of C++ classes. These classes reflect the formal definition of a routing algebra. Specifically, these classes are `Signature`, `Label`, `LabelOperator`, and `PreferenceRelation`. MR.Sim includes a number of concrete instances of these algebra component classes so that algebras do not need to be written from scratch. For example, the FM(LP(3)) algebra leverages the `IntSignature`, `IntLabel`, `FMIntOp`, `LPInt` classes. Lines 1, 4, 6, and 10 in Figure 2 show the specification of an FM(LP(3)) algebra in C++.

In [7], a Routing Algebra Meta-Language (RAML) was defined with various operators that compose algebras into more complex algebras. For example, the BGP-like algebra

[3]The Border Gateway Protocol (BGP) is currently used to propagate routing information in the internet.

```
1   FMIntOp fmOp;
2   SimSeqOp seqOp;
3   LpOp lpOp;
4   LPInt prefRel;
5   SeqPref seqPref;
6   BGPRelationshipReader relationshipReader;
7   RouteIDReader routeIDReader;
8
9   // create the three algebras BGP is composed of
10  Algebra algFMLP(
        "FMLP", &fmOp, &prefRel, &relationshipReader);
11  Algebra algAdd(
        "SIMSEQ", &seqOp, &seqPref, &routeIdReader);
12  Algebra algRouteID(
        "RouteID", &lpOp, &prefRel, &routeIDReader);
13
14  // calculate the composed routing algebra (the BGP
15  // policy).  First parameter specifies number of
16  // algebras to compose
17  Algebra bgp = Algebra::lexProduct(
        3, &algFMLP, &algAdd, &algRouteID);
```

Figure 2: Modeling BGP-like with MR.Sim's policy implementation framework.

is the lexical-product composition of three simpler algebras. In general, the lexical product of two algebras

$$(\Sigma_A, \preceq_A, L_A, \oplus_A, \mathcal{O}_A) \otimes (\Sigma_B, \preceq_B, L_B, \oplus_B, \mathcal{O}_B)$$

is defined to be a new algebra

$$(\Sigma_{A\otimes B}, \preceq_{A\otimes B}, L_{A\otimes B}, \oplus_{A\otimes B}, \mathcal{O}_{A\otimes B}),$$

where the algebra components are defined as follows:

$$\Sigma_{A\otimes B} = (\Sigma_A - \phi) \times (\Sigma_B - \phi) \cup \{\phi\}.$$

The preference operator in the new algebra $\preceq_{A\otimes B}$ is defined as

$$\langle \sigma_A, \sigma_B \rangle \preceq \langle \beta_A, \beta_B \rangle \stackrel{def}{=} (\sigma_A \prec_A \sigma_B) \ or \\ ((\sigma_A \sim_A \beta_A) \ and \ (\sigma_B \preceq_B \beta_B)).$$

The set of labels in the new algebra $L_{A\otimes B}$ is defined as $L_A \times L_B$. The label operator in the new algebra $\oplus_{A\otimes B}$ is a new function where

$$\langle \lambda_A, \lambda_B \rangle \oplus \langle \sigma_A, \sigma_B \rangle \stackrel{def}{=} \langle \lambda_A \oplus_A \sigma_A, \lambda_B \oplus_B \sigma_B \rangle,$$

and

$$\mathcal{O}_{A\otimes B} = \mathcal{O}_A \times \mathcal{O}_B.$$

The lexical product of n algebras is a generalized form of the definition above. The MR.Sim tool implements a function to compute such a composition, an application of which can be seen on line 17 of the code in Figure 2.

One aspect of each routing algebra is the specific policy labels associated with each directional edge in the network topology graph.

- In the FM(LP(3)) algebra, edges are labeled according to a peering relationship, which is defined as either a customer to provider relationship (c), a provider to customer relationship (p), or a peer to peer relationship (r). In FM(LP(3)) routes that go through c links are preferred over those that go through r or p links.
- In the SIMSEQ(0,N-1) and RouteID algebras edges are labeled according to the source RouterID.

- The SIMSEQ algebra keeps track of a route's path, and shorter paths are preferred to longer ones.

The RouteID algebra is used for tie-breaking between routes that have otherwise equally preferred paths.

Network topologies and parameters that are used to calculate edge labels are specified to MR.Sim via a `.map` text file. In our example BGP-like algebra, labels are a 3-tuple composed of labels from the FM(LP(3)), SIMSEQ(0,N-1), and RouteID algebras. Each line of a `.map` file specifies an edge and the parameters that are used to compute that edge's label. For the BGP-like example, an input `.map` file might include the following lines:

```
1 2 relationship=c weight=1 routeID=1
2 1 relationship=p weight=1 routeID=2
```

The above two lines model two autonomous systems (AS) connected to one another. In this example AS 1 is a customer of AS 2. The weight parameter is used by a different algebra ADD(0,N-1), which is not part of BGP. The router id label is the AS identifier for the source node in each edge.

In general, the lines in the `.map` file are formatted as

```
AS0 AS1 <label parameters>
```

This specifies that there is a connection going from the source AS0 to the target AS1 and that this connection has the given label parameters. The format of the `<label parameters>` field fits the grammar

$$\langle \text{label parameters} \rangle \Rightarrow \mathit{parameter} = \mathit{value}$$
$$\mid \mathit{parameter} = \mathit{value}\ \langle \text{label parameters} \rangle\,,$$

which indicates that the label parameter consists of one or more parameter/value pairs. Labels for different algebras are calculated from different parameters.

The calculation of labels for a given algebra is performed by an instance of a `NetworkReader` object. The MR.Sim framework includes several such readers. Example of using such readers and tying them to algebras is shown on lines 6, 7, 10, 11, and 12 of the source code in Figure 2. The flexible format of label parameters in the `.map` files makes it possible for one file to include the label information many different algebras.

To perform a simulation with the MR.Sim framework a user will instantiate an instance of a `NetworkGraph` object and call a provided method to read in a network topology (`.map`) file. Once a `NetworkGraph` is read in, users can use the framework to construct an algebra, such as in Figure 2, and pass the graph along with the algebra to a `simulate` function that performs the simulation. Resulting routing tables can be output to file by calling an `outputTables` function.

The current implementation of MR.Sim includes definitions for the basic routing algebras: FM(LP(3)), ADD(0,N-1), RouteID, and SIMSEQ(0,N-1), and the ability to take their lexical product in any combination. Experimenting with new algebras, either by constructing new base algebras or changing the order of composed algebras, is quite simple in the provided tool. Future work includes implementing other RAML features such as the scoped product and disjunction composition operators, adding optional attributes, programmable labels, and adding in other functions to manipulate algebras [7].

3. SIMULATOR PERFORMANCE ENHANCEMENTS

The goal of our simulator is to produce converged routing tables for a large-scale simulated network graph using a realistic simulated policy. Furthermore, we want this simulation to converge on an answer overnight and be able to do so using the type of machines available to networking researchers at academic institutions. In this section, we examine algorithmic and parallelization strategies that enable us to achieve this goal and improve simulator performance. We observe the effects of these steps on three generated network graphs, and one large-scale network graph that was experimentally generated to model the topology of the internet [24]. We experiment with two policies, (1) a simple ADD algebra, essentially shortest path, and (2) the BGP-like policy discussed in Section 2.

The main issues with performance are (1) memory scaling problems and (2) bounding the number of node visits. We make the memory requirements scale at $O(N)$ instead of $O(N^2)$ with a segmentation algorithm. We use a priority queue to strictly bound the number of node and edge visits to one, which results in improved performance for some graphs and not for others. We also show parallel scalability results on multicore and loosely-coupled, cluster platforms. Amongst the possible optimization choices we tested, the best configuration was to use the segmented algorithm, without a priority queue, on a cluster platform.

3.1 Problem Segmentation for Reduced Memory Usage

Routing simulation is complicated by the memory requirements of routing tables. Since every node requires a routing entry for every other node there is an $O(n^2)$ memory requirement for these entries. On large scale network graphs this memory requirement is prohibitively expensive, thus performing a single simulation to calculate route-to information for all nodes is not feasible.

Fortunately, due to how the routing problem is structured, it is possible to do a segmented simulation, which only calculates the route entries for a subset of the graph at a time. A routing solution is a mapping of nodes to routes and signatures at each other node in the graph. The route to one node does not depend on the route to another node. As such the routing problem is like a separable data-flow analysis, where the routes to an individual node can be computed disjointly from all other node routes Our simulator enables users to specify the size of the subset of nodes to compute routes for, and it performs several simulations in succession to compute route-to entries for this subset over the entire network graph. After each of the simulations the partial result is output to a file so that route entries need not be held in memory. Since the subset size is a constant this reduces the memory requirements to $O(n)$ and makes simulation of large-network graphs tractable.

The size of these subsets will have an impact on performance; smaller subset sizes require greater numbers of simulations while larger requires a greater deal of memory. To help figure out the nature of this apparent tradeoff, Figure 3 shows the memory and time requirements for simulation with various subset sizes. The algebra used in these simulations is the ADD algebra. Interestingly in the second graph the best performance occurs with a subset size of one.

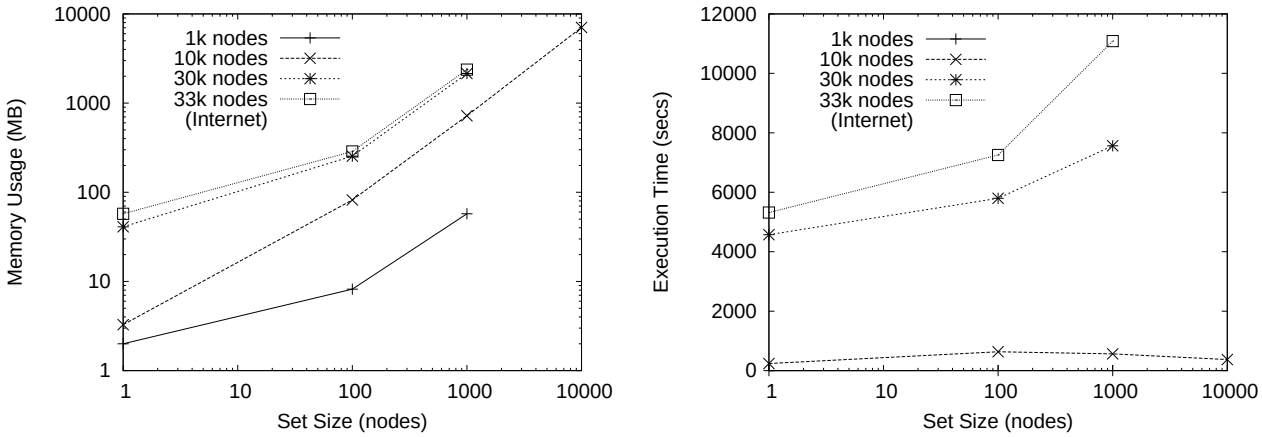

Figure 3: Subset size versus memory requirements and execution times. All runs use Add algebra. Execution time for 1000 node example is less than 3 seconds regardless of subset size.

A segment size of one requires the least amount of memory, thus enabling the greatest portion of the graph to reside in cache. For all subsequent experiments we use a subset size of one.

3.2 Bounding Node Visits

The simulation algorithm for propagating routes to one destination puts nodes in the network on the queue as their route signatures improve. A visit to a node consists of applying the outgoing edge label operations to the route signatures at the node and updating target nodes if their corresponding route signatures become more preferred. The complexity of the simulation algorithm is $O((N+E)H)$, where N is the number of nodes in the network, E is the number of edges, and H is the height of the signature semilattice.

For the BGP-like algebra, H is bound by $O(N)$ because of the path component. This leads to a complexity of $O((N+E)N)$ for propagating the routes to a single node and a complexity of $O((N+E)N^2)$ for the full simulation. The main unknown is how many times will each node need to be visited before it converges to a signature for each route.

By putting nodes on a priority queue with the more preferred signatures being higher priority, the propagation for a segment size of one takes $O((N+E)lgN)$ time and the full simulation takes $O((N+E)NlgN)$ because each node and edge will only be visited once. This works because all of the nodes can be divided into three sets: (1) nodes that have been visited and consequently are reachable, (2) nodes that are reachable, but have not been visited, and (3) nodes that are not reachable yet and have a ϕ signature. Of all the nodes that are reachable, we should visit those nodes with the most preferred signature. There is no way their signature will improve because for them to get a new signature, we would have to do at least one label application from all of the nodes that are reachable. Due to the strict monotonicity property $s \prec l \oplus s$, the signature can only become less preferred and therefore the set of reachable nodes with the most preferred signatures at this point cannot be propagated a better signature.

The $O(lgN)$ is due to the push, pop, and update complexity for a binomial priority queue implementation. Each time we visit a node we are popping that node off the queue. Each time we apply a label operator, we may be pushing a node on the queue or updating the priority of a node on the queue.

The graph in Figure 4 shows the execution times of simulating the BGP-like policy using a priority queue and regular (non-priority) queue. Symbols are placed on top of each bar to show the relative number of node visits that occur. For some graphs the number of visits for the priority queue version is much lower than the non priority queue version. For these graphs using a priority queue improve performance. When the number of node visits is closer, the overhead of the priority queue results in better performance for the original simulation algorithm.

3.3 Multicore Parallelization

Another benefit of segmentation is that it is amenable to parallelization. The individual segmented simulations can be performed independently. Table 1 shows the speedup that is achieved on on a multicore machine (four 2.0 GHz Intel Xeon cores with 2GB RAM). Due to the fact that much of the execution time is spent waiting for L2 cache misses, and the fact that multiple cores of this machine share the L2 cache, ideal speedup is not possible unless the number of cache misses per thread scales down as the number of threads increases. This would be the case if the first and last columns of the table matched. Nevertheless, there is some benefit to running multiple threads of the simulation on a multicore machine.

3.4 Distributed Parallelization

Due to a shared L2 cache ideal speedup was not achieved on our multicore machines. However, such speedup is possible on a distributed memory system. We implemented an MPI version of the simulator that can be dispatched to run on any number of machines. Figure 5 shows the speedup we have achieved by dispatching the simulation on varying numbers of machines in an active undergraduate computer science lab. For comparison, an additional line shows the speedup achieved running multiple threads on a single multicore machine (from the results in Figure 1). All runs used a segment size of one, all topologies used the priority queue version except for the internet topology. The systems in

Table 1: Scalability of 10k node graph, with BGP algebra, on a multicore platform

			L2 data cache misses (billions)		
Threads	Execution time	Speedup	total	per thread	serial total / per thread
1	3307.26	1	29.49	29.49	1
2	2324.89	1.42	37.4	18.7	1.58
3	2156.54	1.53	52.21	17.4	1.69
4	1334.85	2.48	41.21	10.3	2.86

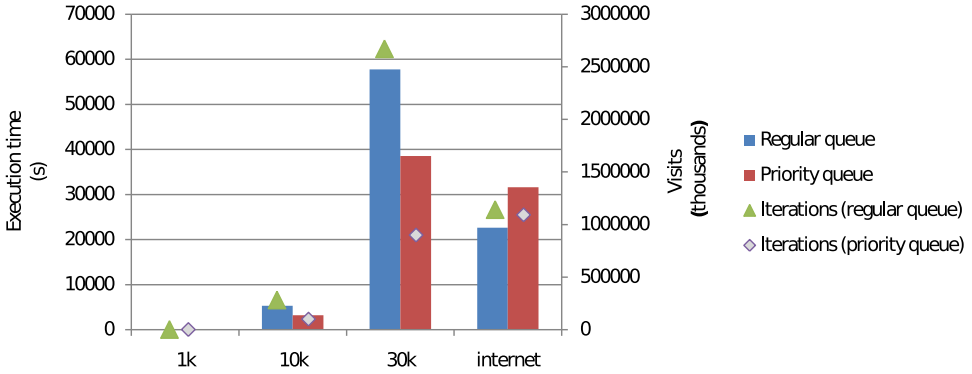

Figure 4: Execution times of simulations with and without priority queue. The triangles and diamonds mark the number of node visits each simulation incurs (on axis on right). When there is a dramatic difference in node visits for the version with the priority queue vs. the version without the priority queue implementation is more efficient, the less substantial difference for the internet modeled network causes the non-priority queue version to be more efficient. The 1k node graph takes 22 secs without using a priority queue and 16 secs with the priority queue.

the lab are all the same (8 core, 3.0GHz Intel Xeon). Despite many other active processors on these machines, we have achieved good scalability. The benefit of using such parallelism is clear when examining the times to complete the internet modeled simulation, which takes 377 minutes (6.28 hours) when using one machine, and 43 minutes (0.71 hours) when using ten. The serial times of the 1k, 10k, and 30k topologies are 12.81, 3193, and 38521 seconds respectively. The times for the 1k, 10k, and 30k topologies when run across ten machines are 1.38, 348, and 4455 seconds. In theory even greater speeds could be achieved for all topologies by using more machines.

4. RELATED WORK

Our usage of Metarouting concepts allows us to scale route policy simulations to internet-sized topologies. In [22], Wojciechowski was also able to simulate internet topologies with BGPSIM. However, BGPSIM is restricted to simulating BGP, while our simulator is capable of evaluating any policy that may be expressed by Metarouting.

The work of Feamster and Rexford [5] is similar in spirit to our own. Policy configuration requires knowledge of what impact changes will bring. To do so, fast calculations which may omit information such as message passing or convergence times are necessary. While Feamster and Rexford solve this problem with algorithms designed to handle BGP, our usage of Metarouting and its proven algebra properties provide us a more general approach.

Other work exists to look at the parallelization of network

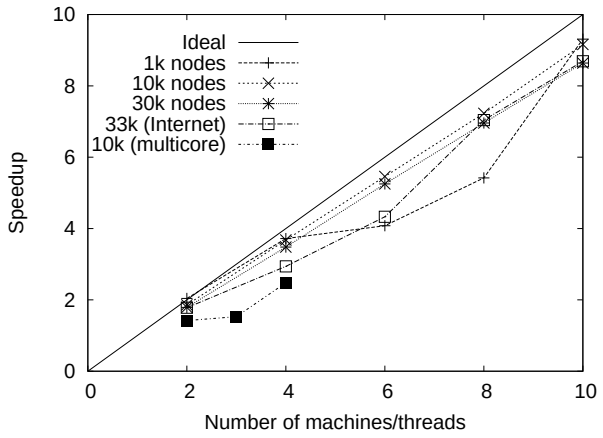

Figure 5: Speedup achieved on simulation of BGP algebra on a distributed parallel system. One line shows speedup achieve using a multicore machine, for this line the horizontal axis is number of threads. Machines are 3.0GHz, 8 core, Intel Xeon's (for distributed results only one core is used). The serial times of the 1k, 10k, 30k, and 33k internet topologies are 12.81, 3193, 38521, 22624 seconds respectively.

routing simulation, but such work focuses less on solving the converged routing problem and more on gathering information at a packet-level [19, 6, 23]. These simulations also require a greater degree of synchronization than MR.Sim. categories: conservative synchronization and optimistic synchronization. Conservative synchronization requires a barrier to be placed at all points where a race conditions could occur. Optimistic synchronization does not do so, but requires simulators to either employ state saving or reverse computation techniques to roll-back in case of an error. The MR.Sim tool avoids these issues through its segmented algorithm, which does not require synchronization.

One common issue in routing simulations is dealing with the quadratic growth of routing table size as number of nodes increases. In MR.Sim we avoid this issue through the segmented algorithm which only calculates a constant portion of each node's routing table at a time. Others have looked at techniques for reducing the size of routing tables [10, 9]; however these techniques can only optimize portions of tables where routes are connected in a spanning tree, and they work best if routing tables do not change frequently. Such changes are frequent in simulations that solve the routing problem.

5. CONCLUSIONS

We have presented an implementation framework based on RAML. Our simulator, MR.Sim, is able to solve the routing problem for protocols specified in this framework on network graphs specified in text files. Our simulator is able to scale to large scale network graphs (10K - 100K nodes). The memory requirements for storing complete routing tables during simulation is prohibitive, thus we have developed a method of simulation that segments the problem into several runs that can be run sequentially or in parallel. We have found that the ideal segmentation set size is one element. We have developed a versions of the simulator exploit parallelism on multicore and cluster platforms and have been able to conduct a simulation of a BGP-like policy on a topology modeled after the internet.

There are many ways our work can be expanded. Future work includes: (1) looking at methods to aid in load balancing and therefore achieve further performance in multicore machines and clusters, (2) expanding the policy implementation framework to include more of the features of RAML (such as new ways of composing algebras), (3) using the tool to study various networking questions related to the affects of topology and policy, and (4) further studying the connection between data-flow analysis and metarouting. Many performance optimizations for data-flow analysis frameworks have been developed over the years: interval analysis, bit-vectors, etc. Although some of these approaches require properties that label operators and network topologies do not have (i.e., interval analysis is a no-go due to non-reducibility in the graph and lack of DFA monotonicity), other performance optimizations could be developed to further improve the performance of the routing simulation.

6. REFERENCES

[1] *Metarouting website*, `http://www.cl.cam.ac.uk/~tgg22/metarouting/`.

[2] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman, *Compilers: Principles, techniques, and tools, second edition*, Pearson Addison Wesley, Boston, MA, USA, 2007.

[3] Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, , and Haobo Yu, *Advances in network simulation*, 2000.

[4] Steve DiBenedetto, Andrew Stone, Michelle Strout, and Dan Massey, *Simulating Internet Scale Topologies With Metarouting*, Tech. Report CS-10-103, Colorado State University, March 2010.

[5] Nick Feamster and Jennifer Rexford, *Network-wide prediction of bgp routes*, IEEE/ACM Trans. Netw. **15** (2007), no. 2, 253–266.

[6] Richard M. Fujimoto, Kalyan Perumalla, Alfred Park, Hao Wu, Mostafa H. Ammar, and George F. Riley, *Large-scale network simulation âĂŞ how big? how fast*, In Symposium on Modeling, Analysis and Simulation of Computer Telecommunication Systems (MASCOTS), 2003.

[7] Timothy G. Griffin and João Luís Sobrinho, *Metarouting*, Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM) (New York, NY, USA), ACM, 2005, pp. 1–12.

[8] Mary Hall, John Mellor-crummey, Rene Rodriguez, Mary W. Hall, John M. Mellor-crummey, Alan Carle, and Alan Carle, *Fiat: A framework for interprocedural analysis and transformation*, In Proceedings of the Sixth Workshop on Languages and Compilers for Parallel Computing, Springer-Verlag, 1993, pp. 522–545.

[9] Zhiyu Hao, Xiaochun Yun, and Hongli Zhang, *An efficient routing mechanism in network simulation*, PADS '06: Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation (Washington, DC, USA), IEEE Computer Society, 2006, pp. 150–157.

[10] Akihito Hiromori, Hirozumi Yamaguchi, Keiichi Yasumoto, Teruo Higashino, and Kenichi Taniguchi, *Reducing the size of routing tables for large-scale network simulation*, PADS '03: Proceedings of the seventeenth workshop on Parallel and distributed simulation (Washington, DC, USA), IEEE Computer Society, 2003, p. 115.

[11] G. Holloway and A. Dimock, *The machine-suif bit-vector data-flow analysis library*, 1998.

[12] S. C. Johnson, *Yacc - yet another compiler compiler*, Tech. Report 39, AT&T Bell Laboratories, Murray Hill, NJ, 1975.

[13] Uday P. Khedker, Amitabha Sanyal, and Bageshri Karkare, *Data flow analysis theory and practice*, CRC Press, Taylor & Francis Group, 2009.

[14] G. A. Kildall, *A unified approach to global program optimization*, ACM Symposium on Principles of Programming Languages, October 1973, pp. 194–206.

[15] T. J. Marlowe and B. G. Ryder, *Properties of data flow frameworks: a unified model*, Acta Informatica **28** (1990), no. 2, 121–163.

[16] Florian Martin, *PAG – an efficient program analyzer generator*, International Journal on Software Tools for Technology Transfer **2** (1998), no. 1, 46–67.

[17] Mehryar Mohri, *Semiring frameworks and algorithms for shortest-distance problems*, Journal of Automata, Languages and Combinatorics **7** (2002), no. 3, 321–350.

[18] B. Quoitin and S. Tandel, *Modeling the routing of an autonomous system with c-bgp*, 2005.

[19] George F. Riley, Richard M. Fujimoto, and Mostafa H. Ammar, *A generic framework for parallelization of network simulations*, in Proceedings of the Seventh International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 1999, pp. 128–135.

[20] João Luís Sobrinho *Network routing with path vector protocols: Theory and applications*, Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM) (New York, NY, USA), ACM, 2003, pp. 49–60.

[21] Feng Wang and Lixin Gao, *Inferring and characterizing internet routing policies*, 2003.

[22] Maciej Wojciechowski, *Border gateway protocol modeling and simulation*, Master's thesis, University of Warsaw and VU University Amsterdam, 2008.

[23] Garrett R. Yaun, Harshad L. Bhutada, Christopher D. Carothers, Murat Yuksel, and Shivkumar Kalyanaraman, *Large-scale network simulation techniques: Examples of tcp and ospf models*, SIGCOMM Computer Communication Review **33** (2003), 2003.

[24] Beichuan Zhang, Raymond Liu, Daniel Massey, and Lixia Zhang, *Collecting the internet as-level topology*, SIGCOMM Comput. Commun. Rev. **35** (2005), no. 1, 53–61.