Minimizing Routing State for Light-weight Network Simulation

Polly Huang
Swiss Federal Institute of Technology
huang@tik.ee.ethz.ch

John Heidemann
USC/Information Science Institute
johnh@isi.edu

Abstract

We introduce a routing mechanism, referred to as algorithmic routing. It is a viable routing alternative for network simulations with minimal space complexity – $O(N)$. In theory and for simulations size of the Internet, algorithmic routing has the potential of reducing memory requirement by several orders of magnitude. In practice and through ns-2 simulations on random topologies, we find memory consumption of algorithmic routing exhibits a similar scaling property. However, routes computed by algorithmic routing are not all the shortest. Although we find the relative difference is below 10% for more than 80% of the routes, we are cautious about its applicability to general network simulations. With further discussion on impacts of the distortion, we derive a set of guidelines and recommend users to apply this technique only when suitable.

1 Introduction

One major bottleneck in large-scale network simulation is the memory requirement for routing states. As of January 2000, we observe at least 284,805 routers in the Internet [1] and some 6,474 of them situated in the backbone [2]. Assuming each routing entry consumes 4 bytes of memory, network simulations of this size will require a minimum of 360GB memory for the Dijkstra all-pair shortest path routing, with space complexity $O(N^2)$ [3]. Hierarchical routing scales in $O(NlogN)$ for balanced networks with $logN$ levels of hierarchy [4]. The Internet however runs in a two-level hierarchy where the backbone runs one routing protocol, e.g, BGP [5], also of space complexity $O(N^2)$, and each local domain runs another routing protocol, e.g., OSPF [6] or RIP [7]. In this case, the memory requirement to maintain routing states for the backbone dominates the total consumption and it is approximately 200MB.

To further minimize memory requirement for routing, we propose an $O(N)$ routing mechanism. This mechanism was inspired by work of Raman et. al. [8], in which the authors evaluated scalability and behavior of a reliable multicast mechanism in large-scale binary trees. For their simulations, they used a simple algorithm to *compute* next hop for any source and destination pair without maintaining a routing table, thus the name – *algorithmic routing*. We extend the algorithm to k-ary trees. Then to map an arbitrary network topology into a k-ary tree, we adopt the Breath First Search (BFS) algorithm [4] and re-assign node addresses in an *orderly* fashion. This order then allows simple next hop computation without maintaining a routing table. The $O(N)$ cost in space actually comes from maintaining the address mapping between the original topology and the corresponding k-ary tree; thus reducing the memory consumption of a Internet-scale simulation to 120KB. Despite being effective in reducing memory requirement, routes computed by algorithmic routing may not be all the shortest. Consider ring components in the networks. In algorithmic routing, certain links in these ring components will be ignored as if they do not exist. This artifact results in sub-optimal routes.

To evaluate algorithmic routing in practice, we implement the mechanism on *ns-2* [9]. With a set of transit-stub topologies, randomly generated by GT-ITM [10], we compare memory and run-time consumption of algorithmic routing to other mechanisms. We also quantify the degree of sub-optimality. To ensure simulation results reflect true behavior of Internet, we discuss limitations of algorithmic routing for general network simulations and derive guidelines to determine scenarios that are suitable (or not suitable) to simulate with algorithmic routing.

In short, the contribution of this work is (a) a generalization of Raman et al.'s work to arbitrary topologies, (b) an evaluation of the memory and time savings and potential inaccuracies of our approach, and (c) recommendations to simulation studies that algorithmic routing is applicable.

2 Background and related work

We provide in this section background on Internet routing, routing abstraction in network simulation and related work in scalable simulation techniques.

2.1 Routing in the Internet

The physical Internet is a collection of routers interconnected by links. A contiguous collection of routers under one administrative authority is called an administrative domain (or *Autonomous System*). OSPF [6] and RIP [7] are two popular protocols to route packets within an administrative domain. They are similar in that both protocols converge to shortest routes for all source and destination pairs, but also different in that each OSPF router floods the domain with states of its neighboring links (referred to as *link state* protocol) whereas each RIP router distributes to neighboring routers its vector of distances to every other routers in the domain (referred to as *distance vector* protocol).

The domain-wide flooding and the $O(N)$ routing table per router do not scale to the entire Internet. Thus, BGP [5] is introduced to route packets across domains. In a nutshell, BGP routers aggregate routing information per domain and exchange this per-domain information with the neighboring domains, in a fashion similar to distance vector routing. This information aggregation alleviates the scaling problems with message flooding and routing table size. BGP in principle converges to shortest routes unless specific paths are given based on domain policy (thus also known as *path vector* routing).

2.2 Routing in network simulation

Some routing protocols in simulators implement details of route exchange. However many simulators can also compute routes in a centralized fashion [11] when details of message exchange are not crucial and memory or computation resource is scarce. Flat and hierarchical routing in *ns-2* [9] are two examples of such abstraction technique for light-weight network simulations.

Flat routing in *ns-2* performs the Dijkstra shortest route computation. In that, each node maintains one adjacency state to each other node in the topology. Hierarchical routing in *ns-2* is similar to the flat routing in a sense that it also does Dijkstra-style iteration by walking through all nodes and eventually settling on the shortest routes. Thus, this particular form of hierarchical routing also yields shortest routes for all source and destination pairs. The difference is that, if there are three levels of hierarchy, each node maintains one adjacency state to each other node (level-1) in a local cluster, to each other cluster (level-2) in a domain, and to each other domain (level-3) in the topology. This form of hierarchical routing is different from that of the Internet where the backbone and local domains compute shortest routes independently. There is sometimes an inter-operating protocol, e.g., IBGP [12], operating among border routers within a domain. These IBGP routers are both backbone-level and local routers and have both backbone-level and local routing information. They correspond to this cluster level (level-2) in the hierarchical routing in *ns-2*. Below we analyze various forms of routing by their space and time complexity.

The flat routing generates a routing table that each node has the next hop and cost information to every other node. Its space complexity is $O(N^2)$. Each node in *ns-2* hierarchical routing maintains routes to nodes within a local cluster, to other clusters within the domain, and to other domains within the topology. This results in average $O(kNlog_kN)$ space complexity, where k is the rank (i.e., number of subdomains per domain) and log_kN is the height of the hierarchy. Given that Internet in this routing sense is 3-level high (i.e., $log_kN = 3$), k equals $\sqrt[3]{N}$. Thus the average space complexity for hierarchical routing in *ns-2* is $O(N\sqrt[3]{N})$. As for Internet hierarchical routing, when ignoring IBGP specific states, the space complexity is $O(kN)$, where each node in a k-node group ($\frac{N}{k}$ of these groups) maintains only routing states to other $k-1$ local nodes. $O(kN)$ equals $O(N\sqrt[3]{N})$ with a fixed 3-level hierarchy. The $O(N)$ space complexity of algorithmic routing comes from maintaining mapping between the original topology and the corresponding k-ary tree. This will be clear in Section 3 when algorithmic routing is explained in detail. Table 1 summarizes the space complexity.

Dijkstra Flat	$O(N^2)$
Dijkstra Hierarchical	$O(N\sqrt[3]{N})$
Internet Hierarchical	$O(N\sqrt[3]{N})$
Algorithmic Routing	$O(N)$

Table 1. Space complexity

Computational complexity for flat routing, which adopts the Dijkstra algorithm, is $O(N^3)$. In that, each node (N) searches for a best route to each other node (N) through existing routes known by these other nodes (N). Computational complexity for hierarchical routing in *ns-2* is $O(N^2\sqrt[3]{N})$. In that, each node (N) searches for a best route to each of the local nodes, clusters, and domains ($\sqrt[3]{N}$) through existing routes in each other node (N) known to these local nodes, clusters and domains. As for Internet hierarchical routing, when ignoring IBGP specific computation, the time complexity is $O(k^2N)$, where each group of k nodes ($\frac{N}{k}$ of them) does a $O(k^3)$ Dijkstra computation. $O(k^2N)$ equals $O(N\sqrt[3]{N^2})$ with a fixed 3-level hierarchy. Computational complexity for algorithmic routing is $O(N)$. This is because the BFS algorithm and address reassignment are both $O(N)$ processes. The computation cost of $O(logN)$ per route lookup is shifted to the traffic generation phase and is not included in the route establishment phase. Table 2 summarizes the time complexity.

2.3 Scalable simulation technique

Parallelism can improve simulation scale in ratio to the number of machines added, but this linear growth is not sufficient to add several orders of magnitude scaling needed. There is increasing interest on taking a complimentary solution. Just as a custom simulator includes only details nec-

	Per Setup	Per Lookup
Dijkstra Flat	$O(N^3)$	$O(1)$
Dijkstra Hierarchical	$O(N^2\sqrt[3]{N})$	$O(1)$
Internet Hierarchical	$O(N\sqrt[3]{N^2})$	$O(1)$
Algorithmic Routing	$O(N)$	$O(logN)$

Table 2. Time complexity

essary for the task at hand, a general simulator can support configurable levels of details for different simulation studies. This is also referred to as the *selective abstraction* approach. Although abstract simulations are often more efficient, they are not identical to more detailed ones. It is critical that when proposing abstraction techniques one also explores the potential impact or distortion these abstraction techniques can introduce, so the users can avoid applying techniques that would lead to invalid conclusions.

Most abstraction techniques focus on enabling simulations with a large amount of traffic, by aggregating individual packets into coarser-grained packet trains [13, 14, 15] or fluid flows [16]. We are interested in resolving the scaling problem in the routing element of network simulations. Riley et. al. [17] proposed to compute route on demand when a new packet is generated. Routes are cached per source and destination pair to prevent same routes from being computed over and over again. This mechanism reduces time and memory consumption significantly at route setup phase because it does not compute routing table at all. However, when traffic starts, it involves an $O(N)$ time consumption per route computation and potentially $O(N^2)$ memory consumption if traffic needs to be generated across all source and destination pairs. We show in the remaining part of the paper that our abstraction technique, algorithmic routing, effectively reduces memory consumption to $O(N)$ independent of traffic pattern. Algorithmic routing also reduces time consumption to $O(N)$ for one-time route setup and $O(logN)$ per route lookup. We discuss as well simulation scenarios that are suitable or not suitable for applying this particular abstraction technique.

3 Algorithmic routing

There are three components in algorithmic routing. The first component involves a simple algorithm to compute next hop in a binary tree and its extension to a more general k-ary tree. The second component looks to map an arbitrary network topology into a k-ary tree which in turn can be used for algorithmic lookup. The last component defines a three-step procedure for general route lookup. We elaborate details of the three components in the subsections to come.

3.1 Route lookup in k-ary tree

The first component is best explained with a binary tree. Figure 1 gives an example of a binary tree (see left plot) and the formula for address assignment (see right plot). For any source A, a destination B can fall in three possible regions. See Figure 2. Node $2A+1$, $2A+2$, or $(A-1)/2$ would be the next hop depending on which region B resides.

We can find the next hop from A to B by walking B to the root, a process of $O(log_2 N)$. If we pass by $2A+1$, the next hop is $2A+1$. If we pass by $2A+2$, the next hop is $2A+2$. If we reach the root without passing by any of the two, the next hop is $(A-1)/2$. Walking from B to the root is a recursive $(B-1)/2$ computation in a binary tree.

This algorithm can be extended to k-ary tree as follows:

```
next_hop (A -> B):
    while (B > 0)
        B_parent = (B-1)/k
        if (B_parent == A) return B
        B = B_parent
    return (A-1)/k
```

With average computational complexity $O(log_k N)$, this simple algorithm let us *compute* next hop for any source and destination pair without maintaining a routing table, and it works as long as the topology is a k-ary tree with regular address assignment. This is not the case for most network topology of interest. Network topology is often arbitrary. We address this problem by applying a tree search algorithm on any arbitrary topology to obtain the corresponding k-ary tree.

3.2 Tree mapping

To enable algorithmic lookup, we use a tree search algorithm to define the k-ary tree equivalent for an arbitrary topology. We choose Breath First Search (BFS) for the reason that it produces least height trees, i.e., routes in a least-height tree are likely shorter. This is crucial and will become clear in Section 3.4 and 4.4 when discussing sub-optimal routes as results of algorithmic routing. The tree mapping starts with the lowest addressed node as the root of BFS. BFS algorithm traverses all the immediately connected nodes, and then recursively traverses connected nodes at the next level until all the nodes are visited. While the original topology is converted into such a tree, its rank (maximal number of leaves per node) can be recorded and used as the value of k for the k-ary tree. Subsequently, the node addresses are re-assigned to be a k-ary tree with possibly some empty leaves. See Figure 3 for an example. As a result of this tree mapping process, we obtain a one-to-one mapping of the node addresses between the original topology and mapped k-ary tree. This mapping is in the order of $O(N)$ and is in fact the dominant factor in the entire memory consumption for algorithmic routing. The BFS and ad-

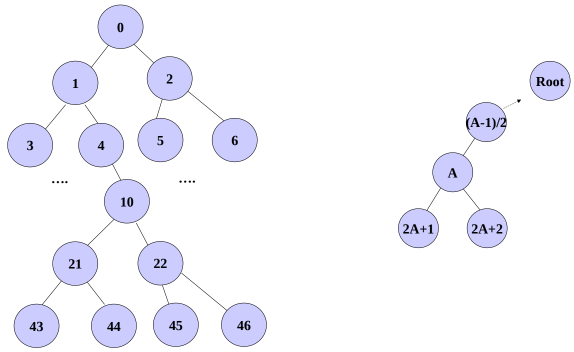

Figure. 1. An example binary tree (left), and node address assignment (right)

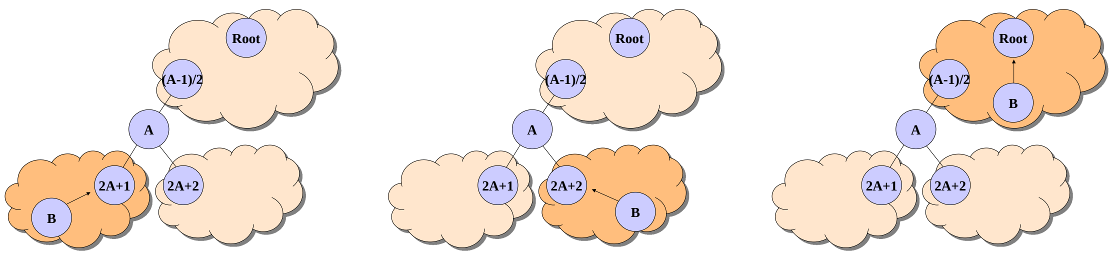

Figure. 2. B in the left-child domain (left), B in the right-child domain (middle), and B in the parent domain (right)

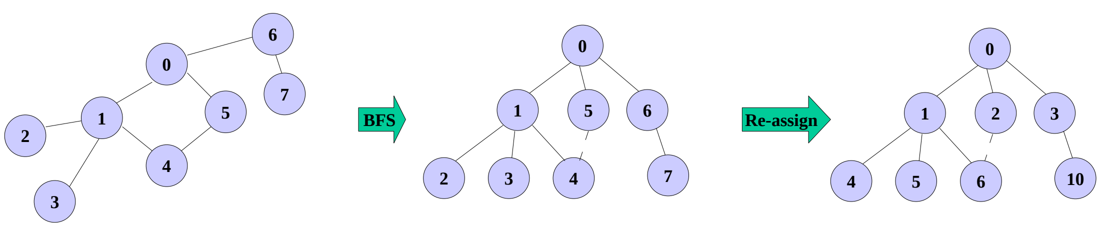

Figure. 3. An example arbitrary topology (left), after BFS mapping (middle), and after re-assigning node addresses (right)

dress re-assignment is $O(N)$ in computational complexity.

3.3 Route lookup in general topology

With the k-ary tree mapping and lookup algorithm in hand, general next hop lookup from node A_{orig} to B_{orig} is a three-step procedure. First, find the corresponding addresses of node A_{orig} and B_{orig} to the k-ary tree, say A_{tree} and B_{tree}. Second, compute the next hop node from A_{tree} to B_{tree}, say C_{tree}. Third, find the corresponding address of C_{tree} from the k-ary tree to the original topology, C_{orig}. Through the process, we identify the next hop C_{orig} for any source A_{orig} and destination B_{orig} pair in an arbitrary topology.

Combining the three components, we derive the algorithmic routing mechanism which is $O(N)$ in space complexity, and in computational complexity, $O(N)$ per simulation setup and $O(log_k N)$ per route lookup. Algorithmic routing also yields sub-optimal routes. That is, routes are not all the shortest. Next, we explain the sub-optimal route problem and its implication to general network simulation.

3.4 Sub-optimal route

In a tree topology, there is exactly one route for any source and destination pair whereas in a topology containing cycles (or is cyclic), there exist multiple routes. During the tree mapping process, certain links in the original topology are ignored. For example, link 4-5 in Figure 3. These links are those used to be part of a cycle in the original topology. Thus, in case the tree mapping happens to cut those links on the shortest routes for certain source and destination pairs, for these source-destination pairs algorithmic routing will yield sub-optimal routes. For example, in Figure 3 the shortest route from node 4 to 5 with algorithmic routing is via node 1, not directly to node 5 anymore.

In other words, algorithmic routing may not capture simulation relevance to some of the links. In particular, for data transfer in between source and destination pairs that links on the original shortest routes are cut, the network delay may be longer. Thus, when studying quality of service issues that deal with strict end-to-end delay guarantee, simulations using algorithmic routing could result in over-conservative design. Over-conservative designs may leave the network under-utilized but end-to-end delay guarantee would be met. In a sense, although simulations using algorithmic routing do not yield precise results, they are reasonable as worst-case analysis.

In addition, in algorithmic routing every node has exactly one route to every other node. One can expect a higher degree of traffic concentration and sometimes unintended network congestion at the root of the mapped k-ary tree. Thus when studying congestion control protocols, simulations using algorithmic routing could result in over-estimation of congestion level and users may end up with back-off mechanisms that are over-sensitive. Again, although simulations using algorithmic routing may not be precise, they give reasonable worst-case results.

For special cases where there is only one sender in the simulated scenario, we can start BFS tree search at the sender as the root of the k-ary tree. In this case all routes will be exactly the same as those in the shortest path computation. For simulations with few senders, multiple k-ary trees can be maintained, one per sender. In this case, memory consumption will be slightly higher depending on the number of senders there are, i.e, $O(sN)$, where s is the number of senders in the simulations, but simulations using algorithmic routing are exactly the same as those using shortest path routing.

3.5 Summary

We have described the major components of algorithmic routing. Instead of a routing table. this routing mechanism maintains address mapping between the general topology and corresponding k-ary tree and results in $O(N)$ space complexity. Although algorithmic routing introduces an $O(log_k N)$ computational cost per route lookup, for a sequential simulator such as *ns-2*, it would be a reasonable trade-off to be able to run the simulations at all with limited memory per machine.

Algorithmic routing also introduces distortion where routes for certain source and destination pairs can be sub-optimal. While this distortion may affect simulation studies on end-to-end delay estimation and congestion control, the fact that this sub-optimal route problem consistently over-estimates the level of end-to-end delay and congestion makes simulations using algorithmic routing reasonable worse-case analysis. For special cases where there are one or few sources, algorithmic routing gives exactly the same results as the shortest path routing.

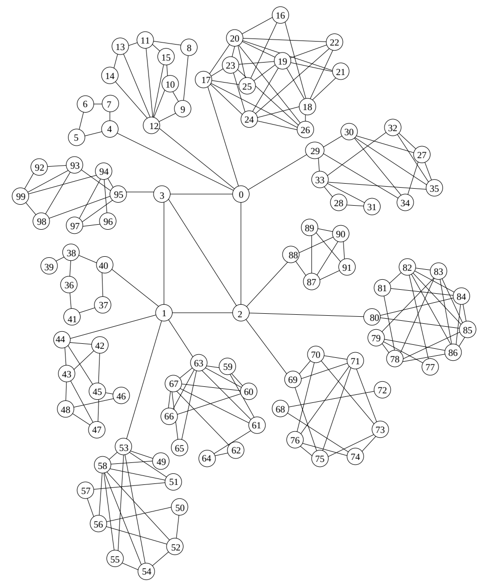

Figure. 4. An example GT-ITM transit-stub topology with 100 nodes

4 Evaluation

We implement algorithmic routing on *ns-2* which has already a set of implementation for flat (or shortest path) and hierarchical routing. Using random topologies generated by GT-ITM, we evaluate memory and time usage of the three routing mechanisms, as well as degree of distortion. Details of the simulation setup and analysis are provided below.

4.1 Methodology

In this subsection, we describe implementation of the three routing mechanisms in *ns-2*, GT-ITM random topologies, measurement metrics, simulation scenarios, and software/hardware platform of our experiments. In *ns-2*, the flat routing is implemented in the `C++` space whereas a major portion of hierarchical routing is implemented in the `OTcl` [18] space. We choose to implement algorithmic routing in `OTcl` space due to development time constraint.

To analyze how each of these routing mechanisms scales to the size of network topology, we generate random topologies with increasing sizes, from 100 nodes to 500 nodes. For each size, we simulate one random topology. We are particularly interested in the transit-stub type (Figure 4) in GT-ITM because it attempts to model the hierarchical structure

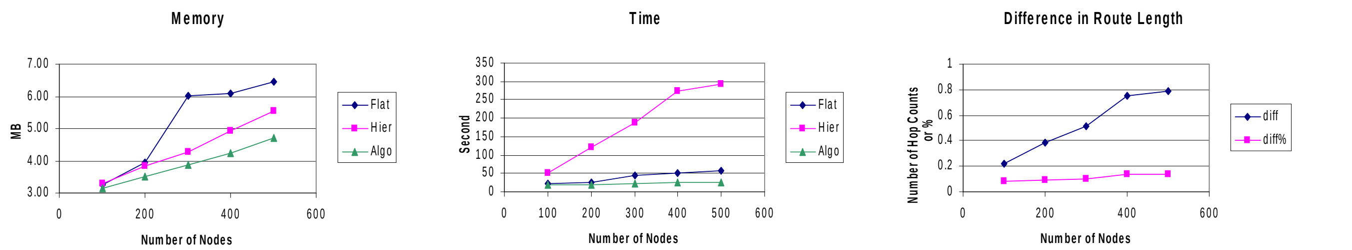

Figure. 5. Comparison: memory (left), run-time (middle), and distortion (right)

of Internet. We change parameters proportionally to generate transit-stub topologies in different sizes while holding the average connectivity constant, approximately 1.77.

At the end of the simulation, we measure the total memory and run-time consumption. To quantify the effect of sub-optimal routes, we measure path length for all source and destination pairs and compute the relative difference of algorithmic routing to the other two routing mechanisms.

All simulations end after route computation is completed, i.e., no traffic is generated. To highlight the memory and run-time cost in the routing component of simulation, we conduct all simulations in session mode [11], where node and link structures are reduced to conserve memory. All memory and time consumption data are collected by simulating the scenarios on a PC with a Pentium II 450MHz CPU, 1 GB RAM, and running FreeBSD 3.0.

Characteristics of the GT-ITM topology is critical in this study, given that different graph constructs could result in different performance and distortion measurement. Previous work [19] has unveiled that topology of the Internet backbone exhibits power-law decay. Analysis from preliminary measurement observed [20] similar power-law decay in router-level Internet topology. However, it is identified [21, 22] that GT-ITM random topologies do not show clear power-law decay. Although the evidences suggest that GT-ITM random topology model might be inadequate, it is not clear yet if the measurement of router-level Internet topology does cover a reasonable base (difficult to validate as well). It is under investigation whether the router-level Internet topology inherits the power-law properties from its backbone component. Even if it is so, simulated topology must contain a large number of nodes and links (1000s of them) to bring out the power-law statistical properties. Thus, for small- to median-sized topology, GT-ITM is deemed still a reasonable tool [23].

4.2 Memory usage

Left plot in Figure 5 shows the memory usage for one simulation. We find results of memory consumption deterministic after repeating the simulations. As we increase number of nodes included in the topologies, the memory requirement for flat, hierarchical and algorithmic routing mechanisms increase as well. Although the expected growth of flat routing should be proportional to N^2, it experiences a faster than N^2 jump every 2^k nodes, and then flattens out between 2^k and 2^{k+1}. This is due to the memory allocation policy in FreeBSD's `C` library which interacts with *ns-2*'s flat routing. The number of entries per node in the routing table is always 2^k, where k is an integer. When the number of nodes increases to $2^k + 1$, *ns-2* will allocate another 2^k entries. In this case, $2^k - 1$ entries are not used. This means the memory requirement increases by $(2^k)^2$, i.e, 4^k, at these jump points. This *power of 2* artifact contributes to the jumping-4^k-and-then-flattening-out behavior. We expect to see another jump between node 500 and 600 (jumping from 512 entries to 1024 entries). Memory requirement for hierarchical routing increases in a significantly slower rate roughly in the order of $N\sqrt[3]{N}$ (Table 1). For algorithmic routing, it is even lower in the order of N.

4.3 Time usage

Middle plot in Figure 5 shows surprising results that flat routing, supposedly the least scalable routing mechanism, runs faster than hierarchical routing and only slightly slower than algorithmic routing. By investigating this unexpected phenomenon, we discover that simulation speed is closely related to two important factors – analytical complexity and programming language efficiency. Sometimes, language efficiency can be as crucial as analytical complexity to determine simulation speed. From the analytical results, flat routing should scale the worst in the order of $O(N^3)$, hierarchical routing second in $O(N^2\sqrt[3]{N})$, and algorithmic routing the best in $O(N)$ (Table 2). Simulation results (see Figure 5) for hierarchical and algorithmic routing agree reasonably well with the analytical results. However, flat routing scales surprisingly better than expected, much better than hierarchical routing and only slight worse than algorithmic routing. This is because that flat routing's Dijkstra

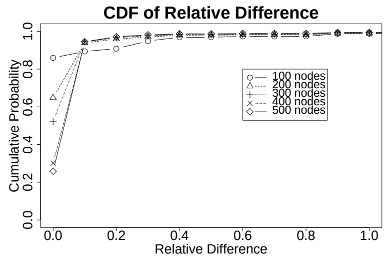

Figure. 6. Distribution of relative difference

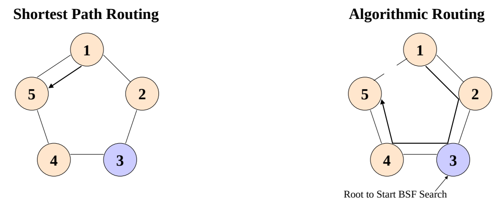

Figure. 7. An example of route distortion by algorithmic routing

algorithm is implemented in `C++`, a much more efficient language than `OTcl`, in which hierarchical and algorithmic routing mechanisms are implemented. In this case, efficiency of programming language overtakes and dominates simulation performance. We plan to investigate further the effect of programming language efficiency by porting the current `OTcl` implementation to `C++`.

4.4 Distortion

By mapping arbitrary topologies into trees, we ignore certain links that otherwise are part of the cycles in the original topology. This results in sub-optimal routes (Section 3.4). For each transit-stub topology, we run the same simulation using flat, hierarchical and algorithmic routing. For each source and destination pair, we compute the path length difference in hop counts. Relative difference indicated in percentage is the ratio of the difference to the path length in flat routing (or shortest path routing). Due to the nature of the hierarchical routing in *ns-2*, i.e., routes are always the shortest, we do not observe any difference in route length between the flat and hierarchical routing. Thus, we compare only routes of algorithmic routing to flat routing

Right plot in Figure 5 depicts the average absolute (`diff`) and relative difference (`diff%`). Each data point is obtained by averaging over absolute or relative differences of all source and destination pairs. The average relative difference increases slowly within the range of 8-14%. Table 3 shows that most routes have the same length (low median) and that few routes have very large differences (large maximum, up to 700%). Although median seems to increase when topology size increases, Figure 6 shows that independent of topology size increase, a high percentage (80-90%) of routes stay within 10% difference.

We hypothesize that the relative difference in path length is related to the size and the amount of ring components (or cycles) in a topology. To be more precise, if the sizes of the ring components are large, the maximal and average relative differences will be large; if the amount of ring components increases, the average relative difference will also increase. Consider a ring topology with 5 nodes. See Figure 7. The shortest path from node 1 to 5 is to go directly through link 1-5 (Figure 7, left). After being mapped to a string, the shortest path from node 1 to 5 becomes the path through link 1-2, 2-3, 3-4, and 4-5 (Figure 7, right). The route length difference is 3 hops and the relative difference is 300%. One can imagine that the maximal relative difference would be determined by the largest ring component in the topology.

Additionally, if there are more ring components, there will be more routes having different lengths which brings up the average difference and possibly also the median of differences. With preliminary analysis [24], we identify a sound property on the average relative difference of algorithmic to shortest routes in ring topology. That is, for a ring of N nodes, the average relative difference is exactly $\frac{k}{2k+1}$, where k is the largest integer smaller than $\frac{N-1}{2}$. From the formula, we obtain a theoretical upper bound of average relative difference for rings – 50%. The formula also indicates that a small ring of three nodes introduces 33% of relative differences by itself. For the transit-stub topologies we experimented, there must be a substantial amount of routes remaining the same or not much different to take the average down to 8-14%. This explains in Figure 6 that 80-90% of routes fall within 10% relative difference.

Our analysis confirms that algorithmic routing yields sub-optimal routes. However, the degree of sub-optimality (i.e., relative difference) for most routes is small, which means a good portion of communication over the topology is minorly affected. The difference could be negligible for worst-case analysis. The average relative difference potentially allows users to compensate the measured end-to-end delay results from simulations using algorithmic routing. Further analysis has to be done to verify that our hypothesis of larger or more ring components results in larger route length difference. Experiments need to be conducted to understand more on the effect of algorithmic routing to actual Internet topology and to quantify as well the relative difference to Internet-like hierarchical routing which may involve non-trivial (sometimes complicated) domain policy. We address these issues in the future work in Section 6.

4.5 Summary

We implement algorithmic routing on *ns-2* which has already a set of implementation for flat (or shortest path) and hierarchical routing. Using random topologies generated

Number of Nodes	Mean %	Std Dev %	Median %	Max %
100	7.954	29.817	0	400
200	9.236	28.395	0	600
300	10.314	25.888	0	700
400	13.368	24.323	12.5	600
500	13.539	21.610	12.5	700

Table 3. Statistics of relative difference

by GT-ITM, we evaluate memory and time usage of algorithmic routing to flat and hierarchical routing. This set of results confirm that the three routing mechanisms scale approximately in the order of O(N^2), O($N\sqrt[3]{N}$), and O(N) in terms of memory consumption which demonstrate that algorithmic routing helps achieving the objective of minimizing the routing states for light-weight network simulations. Our time consumption results confirm that the hierarchical and algorithmic routing mechanisms scale in the order of O($N^2\sqrt[3]{N}$) and O(N) respectively. The O(N^3) flat routing surprisingly out-performs hierarchical routing. This is due to the fact that flat routing in *ns-2* is implemented in `C++` whereas the other two in `OTcl`, and in this case, efficiency of programming language becomes the dominant factor.

To quantify degree of distortion, we again experiment with GT-ITM generated topology and compare path lengths for all source and destination pairs in algorithmic and shortest path routing. This set of results show that there are occasional large relative difference for few source and destination pairs, but mass majority of the relative differences are small. Depending on the problem in study, these differences might be negligible (e.g., worst-case analysis) or simply not have any impact at all (e.g., single- or few-sources analysis). Given our preliminary analysis on graphs with tendency of higher degree of distortion, we recommend users to avoid especially applying the algorithmic routing technique on networks that contain large ring-like components.

5 Conclusion

We have described a routing mechanism whereby large-scale networks can be simulated with only an $O(N)$ amount of states. When applying to networks size and structure of the Internet, algorithmic routing has the potential of reducing memory consumption by three orders of magnitude to hierarchical routing and six orders of magnitude to flat routing. However, simulations using algorithmic routing are not identical to those using hierarchical or flat routing. Just as other abstraction techniques distort simulation results one way or another, we identify a bounded amount of distortion in route length by algorithmic routing. Through analysis and simulation, we come to these conclusions:

- Algorithmic routing does scale in $O(N)$ in both memory and time consumption.
- Algorithmic routing has an extra $O(logN)$ computational cost per route lookup.
- Average degree of distortion in route length is small (8-14%).
- For mass majority of the routes (80%+), relative differences remain small (10%).

Although there is an $O(logN)$ computational cost per route lookup, algorithmic routing is still desirable for sequential simulators where large-scale simulations may not run at all with a limited amount of memory. With elegant tree addressing scheme [25], this lookup can be done in $O(1)$ time. The last two bullets imply that the degree of distortion scales well to topology size and it is likely that most communication is done through routes that are only slightly different.

For simulation scenarios that have single or few senders, multiple k-ary tree mappings can be maintained, one per sender, so that routes are always the shortest. If the simulated topologies are trees, algorithmic routing yields exactly the same results. Simulations using algorithmic routing consistently over-estimate metrics such as end-to-end delay and congestion level. Therefore, simulations using algorithmic routing qualify as worst-case analysis. With our simulation and analytical results on the degree of distortion, one can compensate the overall simulation results proportionally for more precise understanding of the problem in study. In fact, there exist spanning tree algorithms [26] that compute in polynomial time and give bounds to the degree of distortion (i.e., average relative difference). In short, algorithmic routing is suitable for any of the following cases.

- simulated scenarios contain few senders,
- simulated topologies are trees, or
- simulation objective is to assess the worst-case performance.

We would recommend avoid using algorithmic routing for cases that satisfy any two of the following conditions.

- simulated scenarios contain many senders,
- simulated topologies contain large or many ring-like components.
- results have to be precise

6 Future work

We propose to improve algorithmic routing in three ways. The first proposal is inspired by the fact that single-sender simulations can start tree mapping from the sender node to avoid distortion at all. One can maintain one tree mapping per sender for few-sender simulations or per node on a large ring component (or alternatively nodes on opposite sides of a large ring). This results in $O(sN)$ memory consumption but is a reasonable alternative to trade off memory for higher degree of accuracy, given s remains small. The second proposal aims to reduce the maximal relative difference. Given that every node has information of directly connected links anyway, we propose to perform a 2-pass route lookup where we check first if the destination is directly connected to current node. In case not, we then look up using algorithmic routing. Thirdly, we expect a lower degree of distortion with a slightly modified tree search algorithm. BFS algorithm results in least-height tree which is already a desirable property to avoid sub-optimal routes, i.e., routes tend to be shorter in general. Current implementation of BFS is in its most naive form. For nodes at the same level in tree, including the root at the highest level, we search based on node address. We propose to give higher precedence to nodes with higher degree of connectivity and search nodes at the same level based on the precedency. This will bring nodes connecting to high-connectivity nodes higher in the tree and reduce average route difference.

We plan to incorporate these changes in the near future when we port the current `OTcl` implementation to `C++`. For the long-term future, formal analysis need to be done to verify the hypothesis of larger or more ring components results in larger route difference. The hierarchical structure of Internet and its, sometimes, policy-based routing are currently subjects under investigation [27] Once these become clear, experiments can be conducted to understand the effect of algorithmic routing to actual Internet topology and routes.

Acknowledgments

We would like to thank Deborah Estrin, Suchitra Raman, Kannan Varadhan, Yuri Pryadk, and Padmaparna Haldar for their discussions in the early stages of this work. We would also like to thank Lee Breslau and Thomas Erlebach for their suggestions to further improve the quality and performance of the algorithm.

References

[1] SCAN, "SCAN+LUCENT Internet Map," http://http://www.isi.edu/scan/mercator/maps.html/, 1999.

[2] NLANR, "NLANR AS Map Archives," http://moat.nlanr.net/Routing/rawdata/, 2000.

[3] E. W. Dijkstra, "A Note on Two Problems in Connection with Graphs," *Numerical Mathematics*, vol. 1, pp. 269–271, 1959.

[4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, McGraw-Hill, New York, 1989.

[5] Y. Rekhter and T. Li, "A Border Gateway Protocol 4 (BGP-4)," RFC 1771, Internet Request For Comments, Mar. 1995.

[6] J. Moy, "OSPF Version 2," RFC 2328, Internet Request For Comments, Apr. 1998.

[7] C. L. Hedrick, "Routing Information Protocol," RFC 1058, Internet Request For Comments, June 1988.

[8] S. Raman, S. Shenker, and S. McCanne, "Asymptotic Scaling Behavior of Global Recovery in SRM," in *Proceedings of Performance'98/ACM Sigmetrics'98*, June 1998.

[9] L. Breslau, D. Estrin, K. Fall, S. Floyd, A. Helmy, J. Heidemann, P. Huang, S. McCanne, K. Varadhan, H. Yu, Y. Xu, and VINT Project, "Advances in network simulation," *IEEE Computer*, vol. 33, no. 5, pp. 59–67, May 2000.

[10] E. Zegura, K. Calvert, and M. Donahoo, "A Quantitative Comparison of Graph-based Models for Internet Topology," *ACM/IEEE Transactions on Networking*, vol. 5, no. 6, Dec. 1997.

[11] P. Huang, D. Estrin, and J. Heidemann, "Enabling Large-scale simulations: selective abstraction approach to the study of multicast protocols," in *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, Montreal, Canada, July 1998, pp. 241–248, IEEE.

[12] T. Bates and R. Chandra, "BGP Route Reflection - An alternative to full mesh IBGP," RFC 1966, Internet Request For Comments, June 1996.

[13] J. Ahn and P. B. Danzig, "Speedup vs. simulation granularity," *IEEE/ACM Transactions on Networking*, vol. 4, no. 5, pp. 743–757, Oct. 1996.

[14] P. Huang and J. Heidemann, "Capturing TCP Burstiness for Lightweight Simulation," in *Proceedings of the SCS Multiconference on Distributed Simulation*, Phoenix, Arizona, USA, January 2001, Society for Computer Simulation.

[15] Y. Guo, W. Gong, and D. Towsley, "Time-stepped Hybrid Simulation (TSHS) for Large Scale Networks," in *Proceedings of the IEEE Infocom*, March 2000.

[16] V. Misra, W.-B. Gong, and D. Towsley, "Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED," in *Proceedings of the ACM SIGCOMM*, Stockholm, Sweden, Aug. 2000, pp. 151–160.

[17] G. F. Riley, M. H. Ammar, and R. Fujimoto, "Stateless Routing in Network Simulations," in *In Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, San Francisco, CA, USA, August 2000.

[18] D. Wetherall, "Object-oriented Tcl," ftp://ftp.tns.lcs.mit.edu/pub/otcl/doc/tutorial.html.

[19] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On Power-Law Relationships of the Internet Topology," in *Proceedings of the ACM SIGCOMM*, Cambridge MA, Aug. 1999.

[20] R. Govindan and H. Tangmunarunkit, "Heuristics for Internet Map Discovery," in *Proc. IEEE INFOCOM*, 2000.

[21] A. Medina, I. Matta, and J. Byers, "On the origin of power-laws in Internet topologies," *ACM Computer Communication Review*, Apr. 2000.

[22] C. Jin, Q. Chen, and S. Jamin, "Inet: Internet topology generator," Tech. Rep. CSE-TR-433-00, University of Michigan, 2000.

[23] E. Zegura, "Thoughts on Router-level Topology Modeling," The end2end-interest Mailing List Archives, January 2001.

[24] P. Huang and J. Heidemann, "Minimizing Routing State for Lightweight Network Simulation," Tech. Rep. TIK-Nr.106, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology, Zurich, 2001.

[25] B. Schieber and U. Vishkin, "On Finding Lowest Common Ancestors: Simplification and Parallelization," *SIAM J. Comput.*, vol. 17, no. 6, pp. 1253–1262, 1988.

[26] B. Y. Wu, G. Lancia, V. Bafna, K.-M. Chao, R. Ravi, and C. Y. Tang, "A Polynomial Time Approximation Scheme for Minimum Routing Cost Spanning Trees," in *Proc. Ninth Annual ACM-SIAM Symposim on Discrete Algorithms (SODA'98)*. 1998, pp. 21–32, ACM.

[27] H. Tangmunarunkit, R. Govindan, S. Shenker, and D. Estrin, "The Impact of Routing Policy on Internet Paths," in *Proc. IEEE INFOCOM*, Alaska, USA, April 2001.